Songle Sync: A Large-Scale Web-based Platform for Controlling Various Devices in Synchronization with Music

Jun Kato, Masa Ogata, Takahiro Inoue, Masataka Goto
National Institute of Advanced Industrial Science and Technology (AIST), Tsukuba, Japan
{jun.kato,masa.ogata,takahiro.inoue,m.goto}@aist.go.jp

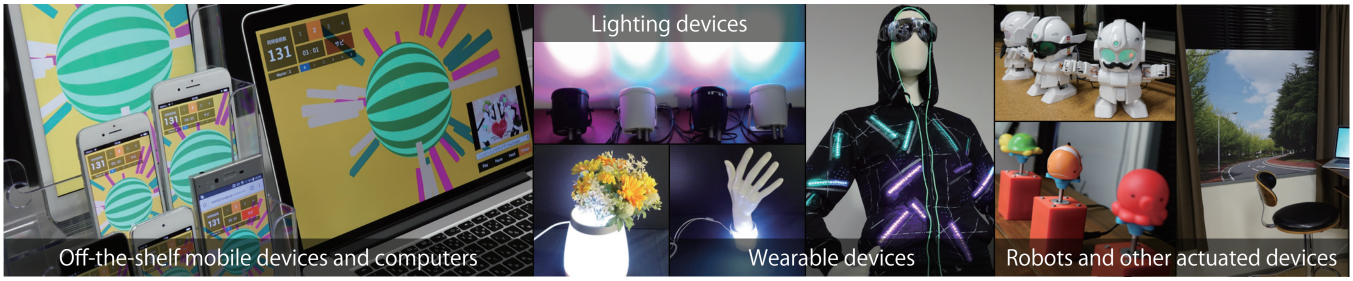

Lighting devices
Off-the-shelf mobile devices and computers
Wearable devices
Robots and other actuated devices

Figure 1: A variety of Internet-connected devices can be synchronized with music playback on the Songle Sync platform.

ABSTRACT

This paper presents Songle Sync, a web-based platform on which hundreds of Internet-connected devices — including smartphones, computers, and other physical computing devices — can be controlled to synchronize with music playback. It uses music-understanding technologies to dynamically synthesize music-driven multimedia performances from a musical piece of choice. To simultaneously control hundreds of devices, a conventional architecture keeps always-on connections between them. However, it does not scale and suffers from latency and jitter issues when there are various devices with potentially unstable networks. We address this with a novel autonomous control architecture in which each device is notified of forthcoming musical events (e.g., beats and chorus sections) to automatically drive various changes in multimedia performances. Moreover, we provide a development kit of an event-driven multimedia framework for JavaScript, example programs, and an interactive tutorial. To evaluate the platform, we compared latencies, jitters, and amounts of network traffic between ours and the conventional architecture. To examine use cases in the wild, we deployed the platform to drive over a hundred of a variety of devices. We also developed a web browser-based application for a multimedia performance with music playback. It provided audiences of hundreds with a bring-your-own-device experience of synchronized animations on smartphones. In addition, the development kit was used in a two-day hackathon. We report lessons learned from these studies and discuss the future of the Internet of Musical Things.

CCS CONCEPTS

• **Human-centered computing** → **Web-based interaction**; • **Applied computing** → **Sound and music computing**; • **Software and its engineering** → *Application specific development environments*;

KEYWORDS

Internet of Musical Things, music synchronization, multimedia control, application programming interface

ACM Reference Format:
Jun Kato, Masa Ogata, Takahiro Inoue, and Masataka Goto. 2018. Songle Sync: A Large-Scale Web-based Platform for Controlling Various Devices in Synchronization with Music. In *2018 ACM Multimedia Conference (MM '18), October 22–26, 2018, Seoul, Republic of Korea.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3240508.3240619

1 INTRODUCTION

This paper presents a platform with which a developer can easily make lots of various kinds of devices behave in synchronization with playback of a musical piece of choice. Such a platform is useful at any occasion with music to augment the scenery so that the user can feel more sense of music and more united with neighbors. For instance, an end-user can play a favorite musical piece on smartphone whose playback status and timeline are shared in real time with the smartphones of neighboring friends. The smartphone screens show synchronized graphic animations, making a sense of unity among people in place. Live event audiences of hundreds can enjoy visual performances synchronized with musical pieces playing at the venue, either on the huge onstage screen or on their own smartphones, augmenting their acoustic experience with the synchronized visual sensations. Furthermore, at any place with background music, such as a shopping mall or restaurant, the music can drive connected screens and lightings on the walls or floors, dancing robots, visitors' smartphones, etc.

MM '18, October 22–26, 2018, Seoul, Republic of Korea

ACM ISBN 978-1-4503-5665-7/18/10…$15.00
https://doi.org/10.1145/3240508.3240619

While there have been prior attempts to synchronize multiple devices with music playback, the aforementioned applications could have not been implemented due to three difficulties. First, regarding the *scalability*, it is still not easy to control in synchronization with music playback a lot of off-the-shelf devices like smartphones and tiny computers, which are not always connected to a reliably fast and stable network. Second, regarding the *variety*, existing applications are for a homogeneous hardware setup consisting of a single device type and for a specific musical piece. There is no unified way to synchronize a variety of devices with the music playback. Multimedia performances need to be prepared on a musical piece-basis, meaning that event types (e.g., musical beats) and their timings need to be manually annotated beforehand. Accordingly, there is no way to dynamically reflect the end-user's preference on the choice of musical pieces. Third, given these *scalability* and *variety* difficulties, the *programming experience* — how to develop an open platform that enables easy development of scalable music-driven multimedia applications — has not been investigated.

To address these difficulties, this paper proposes Songle Sync, a web-based platform capable of controlling hundreds of a variety of Internet-connected devices to synchronize with the playback of musical pieces. The following sections are organized as follows. **Section 2** examines existing music-driven applications as well as frameworks for developing such applications to highlight the characteristics of our platform. **Section 3** introduces its two concrete usage scenarios and clarifies the technical requirements. **Section 4** explains the platform that addresses the difficulties regarding the *scalability* and *variety*. We propose a novel autonomous control architecture that utilizes intermittent compact communications over the Internet, enabling scalable and stable real-time control of hundreds of devices. In addition, our platform is built on top of standard web technologies and can control a variety of devices in a unified way. Our platform also utilizes Songle [6] that provides semantic information of musical pieces on the web (musical beats, chords, and so on.) This eliminates the need to manually annotate each musical piece and furthermore allows the end-user to enjoy applications with a favorite musical piece. **Section 5** describes the *programming experience* — any developer can develop scalable music-driven applications with help of its musical-event-driven API, example programs, and tutorials. Finally, **Section 6** reports performance evaluations as well as deployments in the wild. Songle Sync was publicly released on August 2, 2017[1] and have been used to drive over a hundred of a variety of devices, to provide to audiences of at least 275 people a bring-your-own-device (BYOD) experience of a 40-minute multimedia performance with music playback, and in a two-day hackathon. We report lessons learned and discuss the future of the Internet of Musical Things (IoMT).

2 RELATED WORK

2.1 Music-Driven Applications

Multimedia performances synchronized with music playback are appealing to audiences and have attracted much attention from both industry and academia. For instance, technologies for making three-dimensional computer graphics models move in accordance with the beat timings [5, 21] have been developed and deployed industrially. Technologies for authoring a video in which a video clip or photo transforms into another at appropriate timings of the background music [1, 10, 22] and for showing computer graphics animations based on the lyrics text [3, 4, 12] have been proposed.

There is much potential in the large-scale music-driven multimedia performances. Existing systems are capable of changing wristband colors [26] by using a wireless signal, changing ear hat colors [20] by using infrared signals, and flashing phone screens [18] by using inaudible sound emitted from speakers. The Smartphone Orchestra [25] allows participants to bring their smartphones, each of which plays a unique part of the musical piece on its web browser filled with a single color. When the smartphones are placed in one place, they create a unified musical scenery.

These systems typically need dedicated devices or applications to be installed prior to the multimedia performance. In contrast, our platform aims to support a heterogeneous hardware setup of smartphones, tablets, personal computers, and the Internet of Things (IoT) devices, i.e., physical computing devices connected to the Internet. In the case of browser-based Songle Sync applications, the user can join the performance just by visiting the website.

2.2 Music-Driven Development Frameworks

In response to high demands for music-driven applications, application programming interfaces (APIs) and frameworks for the purpose have been proposed.

While most of the APIs focus on providing manually annotated metadata such as that available at Gracenote [8], some of them are based on automatic analysis. For instance, Spotify APIs provide semantic information of the musical elements in the musical pieces on the service [24] with help of the technologies of Echo Nest [23]. Songle Widget [7] is an event-driven multimedia framework for the development of web browser-based applications based on semantic information of musical pieces on the web. Amalia.js [9] is a metadata-enriched HTML5 video player that enables plugin development for visualizing time-series data with help of timing events emitted from the player. Our platform borrows a certain design rationale such as event-driven APIs and differs from them in that ours controls hundreds of various Internet-connected devices.

Synchronicity between multimedia (audio, visual, haptic, etc.) streams played on multiple devices has been extensively studied [16]. IMSync [17] took human factors into consideration and proposed a strategy to keep devices coordinated. A sync server was implemented to which multimedia devices are connected through WebSockets. The use of standard web technologies for synchronous application frameworks has become common such as those for synchronizing video playback [11], multimedia conferencing [27], and remote collaborations (TogetherJS [14]). Note that, in most cases, there is an assumption that streams are data-intensive or contain user interactions and the whole data cannot be transferred beforehand. This results in the always-on architecture in which the client devices always need to be stably connected to the server. In contrast, our applications are driven by the semantic information of musical pieces, which is lightweight and can be initially transferred to the client devices. This is the key enabler for our autonomous control architecture.

[1] Songle Sync. http://api.songle.jp/sync

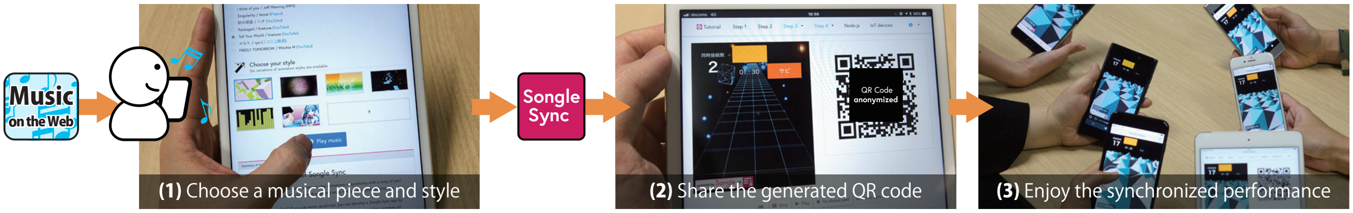

Music on the Web
Songle Sync
QR Code anonymized
(1) Choose a musical piece and style
(2) Share the generated QR code
(3) Enjoy the synchronized performance

Figure 2: An example usage scenario of the Songle Sync platform: synchronizing smartphones with the music playback.

3 GOAL: MUSIC-DRIVEN MULTIMEDIA PERFORMANCE AT SCALE

As explained above, our goal is to implement an open platform that enables easy development of music-driven applications for augmenting music listening experience with a lot of various synchronized devices. The resulting multimedia applications can provide the end-user with more sense of music and unity with other people than simply playing music does.

This section shows two concrete usage scenarios of the Songle Sync platform to help introduce its goal as well as its technical requirements. The first scenario is supported by the actual tutorial website (**Figure** 2, **Section 5.3**) serving as a brief introduction to the platform. The second is implemented and tested with over a hundred devices (**Figure 1**, **Section 6.2**).

3.1 Instant BYOD Experience for Smartphones

The first scenario allows the user to choose a musical piece available on the web, play it on a smartphone, and invite friends to bring their smartphones whose web browsers are instantly synchronized with the music playback. There is no existing platform that allows such an instant bring-your-own-device (BYOD) experience without installing dedicated applications.

3.1.1 Choosing a Musical Piece and Animation Style. The user uses a web browser to choose a musical piece on the web (an MP3 file or page on a music and video sharing service such as YouTube, Piapro, and Niconico) and an animation style from a list (**Figure** 2).

Then a music player appears and starts streaming the musical piece. Next to the player is computer graphics animation, which is synchronized with the beat timings, and changes visual patterns when the playback enters the chorus section. The user can enjoy music acoustically and visually at the same time.

3.1.2 Sharing the URL and/or QR Code. When the application shows the music player, it also generates a URL to share with other people. It displays the URL in a text box and a clickable button to share the URL via social networking services such as Twitter. Additionally, it displays the QR Code that navigates to the URL.

The user can invite as many remote or neighbor friends as desired (see Section 5.1 for the limitation) to access the shared URL with the web browser on their smartphones.

3.1.3 Synchronizing Smartphones with Music over the Internet. When the friends access the URL by clicking a link or capturing the QR Code, the same application launches in their web browser. They can choose whether they need sound (when playing remotely) or not (when sitting next to the user).

All the application instances are connected to the Songle Sync server so that the playback status and timeline are synchronized. When the user changes a musical piece to play, pauses the playback, or seeks to a different position in the musical piece, the friends' smartphones follow the changes. A friend can join and leave the session at any time. While the synchronization of the remote audio playback might need some time because audio streaming usually requires a certain amount of data buffering, the changes in the playback status are shared almost instantly.

3.2 Various Devices Synchronized with Music

There are an increasing number and variety of microcontrollers and tiny computers that can connect to the Internet. They can be used to build various kinds of Internet of Things (IoT) devices, including but not limited to smart light bulbs, robots, and remote-controllable curtains. The second usage scenario allows a heterogeneous hardware setup, consisting of such devices, to synchronize with a single musical piece. Such a setup has not been supported by the existing systems.

All devices shown in **Figure 1** are placed in a demonstration room and run JavaScript programs to connect to the Songle Sync server. Several laptop computers are used to control music playback, one of which is connected to speakers and plays sound. When a musical piece is playing on the computer, all devices and web browsers react to musical events such as hitting each beat and entering the chorus section. This heterogeneous setup as a whole creates a unique scenery synchronized with music and provides an immersive music listening experience to people in the room.

3.3 Technical Requirements

To achieve the scenarios, there are three technical requirements we considered when designing the platform. First, the platform should support a dynamic and heterogeneous hardware setup allowing various kinds of devices to join and leave the performance at any time. These days, smartphones and many microcontrollers and tiny computers can connect to the Internet, and we therefore chose not to develop our own network protocol stack but to build the platform on the standard web technologies. The devices can either run JavaScript programs (e.g., smartphones, Intel Edison, and Raspberry Pi) or be controlled by a computer running JavaScript-based processes (Node.js) with the help of middleware (e.g., Johnny-Five). We therefore chose to provide a JavaScript library for application development on the platform.

Second, the platform should enable scalable and stable control of hundreds of devices that do not always have fast and stable network connections. The platform should be usable under such challenging

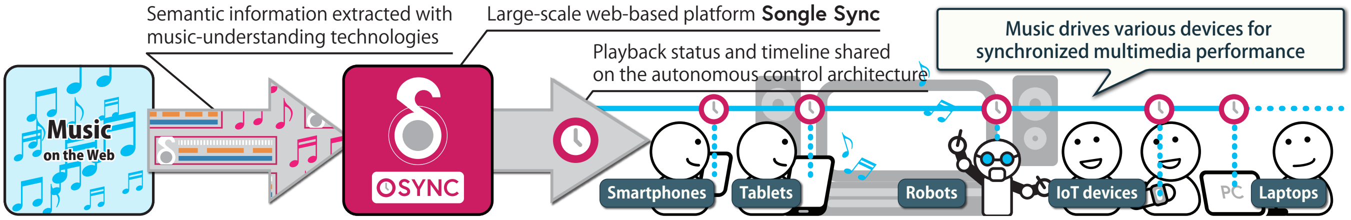

Semantic information extracted with music-understanding technologies
Large-scale web-based platform Songle Sync
Playback status and timeline shared on the autonomous control architecture
Music drives various devices for synchronized multimedia performance
Music on the Web
SYNC
Smartphones
Tablets
Robots
IoT devices
PC
Laptops

Figure 3: An overview of the Songle Sync platform capable of synchronizing hundreds of devices to the music playback.

networking environments. The communication protocol between the platform and devices needs to be compact. More importantly, the latency and jitter of the devices should be small (<100 milliseconds, a few frames at 30 fps) or the performance will not look synchronized with the music. Addressing this issue is not straightforward and will be discussed in detail in Section 3.

Third, the platform should allow easy application development. The developers should not need to annotate the music manually, which takes a considerable amount of time. Otherwise, the end-user is forced to choose a musical piece from the manually annotated ones. Our platform therefore depends on a public web service called Songle [6] that has analyzed more than a million musical pieces on the web. In addition, we borrowed the design of the event-driven application programming interface from Songle's development framework called Songle Widget [7]. It allows one to write code that triggers performances on a device upon musical events such as musical beats and changes in chords. Since Songle Widget was designed for programming a single device and does not support synchronizing hundreds of devices, we built our own multimedia framework to eliminate the need to write low-level code to manage synchronization between the devices as explained in **Section 5**.

4 SONGLE SYNC

This section explains how we met the technical requirements and implemented a scalable and stable platform for controlling hundreds of a variety of devices in synchronization with music (**Figure 3**). First, the issues with the existing approach is presented. Second, a novel autonomous control architecture is explained. Third, its scalability features are described in detail.

4.1 Issues with Always-on Connections

To control hundreds of devices in synchronization with music, it is common to implement an architecture that keeps always-on network connections between the "master" and "slave" nodes. Each master node has control over the music playback and emits a command every time it wants to change the state of the slave nodes. All slave nodes keep listening for commands from the master nodes so that they timely react to the musical events (e.g., draw graphics, actuate motions, and flash lights).

Proprietary systems used to synchronously change the color of wristbands in a concert hall [26] and other examples mentioned in the Related Work section are based on this architecture. The commands are transmitted from a master node to all slave nodes through wireless signals from an antenna, inaudible sound from a loud speaker, and WebSocket connections between the server and a smartphone application. To control lighting in a nightclub, concert hall, or arena, a digital communication protocol called DMX512 [19] is often used, and it is also based on this architecture.

While widely adopted for synchronizing a number of devices, the conventional always-on architecture requires low-latency and low-jitter connections between the master and slave nodes. Otherwise, either the latency or the jitter, or both, will result in the slave nodes being out of synchronization with the music playback. This would not be an issue when the system is a homogeneous hardware and network setup — i.e., consisting of the same kind of slave devices and the same kind of master devices connected through similar network configurations with low latencies and jitters. In contrast, the Songle Sync platform aims at supporting a heterogeneous hardware setup under a variety of networking environments and therefore cannot adopt the always-on architecture. Within the same Songle Sync application, devices would have a variety of computing capabilities and be connected through various network configurations.

4.2 Autonomous Control Architecture

To address the shortcomings of the always-on architecture, we propose an autonomous control architecture in which every node behaves autonomously without always-on connections. Instead of establishing direct connections between the master and slave nodes, the proposed architecture adds the platform server in the middle of the communication. Implementation-wise, master and slave nodes are treated identically by the server except for the master nodes capable of the playback control.

As explained in the next subsection, both master and slave nodes synchronize their clocks upon connection to the server and try to keep their clocks synchronized. And as explained in **Section 4.2.2**, when a master node specifies a musical piece to play, all nodes receive the global event timeline containing musical events and their timing information. The playback status is managed by the server, can be controlled by master nodes, and is shared with all nodes. With the synchronized clocks, the global event timeline, and the shared playback status, each node knows when to do what. These communications between the server and nodes happen intermittently. Even in extreme cases, the slave nodes that have completed the initial synchronization behave appropriately until the end of the musical piece even if they get disconnected from the server.

4.2.1 Clock Sync: Sharing Time. When a master or slave node establishes the connection to the platform server, it stores the timestamp on the local clock in memory (departure time t_d). Then the server responds with the timestamp on the server clock (server time t_s). When the node receives the server response, it stores the timestamp on the local clock (arrival time t_a) and calculates the average of the departure and arrival time (average time).

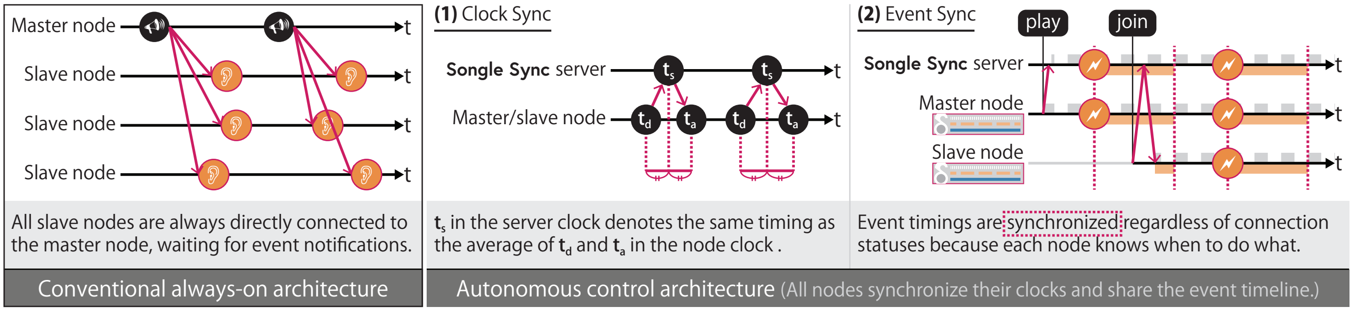

Master node
Slave node
Slave node
Slave node
t
All slave nodes are always directly connected to the master node, waiting for event notifications.
Conventional always-on architecture
(1) Clock Sync
Songle Sync server
Master/slave node
t_s
t_d
t_a
t_s in the server clock denotes the same timing as the average of t_d and t_a in the node clock .
(2) Event Sync
play
join
Songle Sync server
Master node
Slave node
Event timings are synchronized regardless of connection statuses because each node knows when to do what.
Autonomous control architecture (All nodes synchronize their clocks and share the event timeline.)

Figure 4: Comparison between the conventional always-on architecture and the proposed autonomous control architecture.

Under an ideal circumstance in which (a) there is no time difference between the server and node and (b) the outbound and inbound trip times are the same, the average time should be the same as the server time. In reality, the difference between the outbound and inbound trip times is small enough to be ignored, and the time difference between the server and node is calculated as $t_s - \frac{t_d+t_a}{2}$). To eliminate the effect of network jitters, we conduct robust estimation by repeating this process for several times (five times in the current implementation) and taking the median value as the time difference. This procedure is very similar to the Network Time Protocol (NTP) [2], and the major difference is that ours uses the median value as the time difference while the NTP uses the mean value. We use the median value since we assume relatively larger jitters in the communication compared to the ordinary use cases of the NTP.

Additionally, after the initial synchronization process, the node runs the same process periodically with an interval that can be specified by the application developer. Previous timestamps are not cleared but accumulated to calculate the median value, providing a more stable result. In this way, even when the network connection is unstable or when there is a local clock drift in the node (which is often the case for microcontrollers and web browsers on low-powered smartphones), the local clock can continually catch up with the global server clock.

4.2.2 Event Sync: Sharing Timeline and Playback Status. When a master node issues a command to set a musical piece to play, all connected nodes are notified and retrieve from the server all the musical elements and their timing information in the musical piece, including music structure, beat structure (beats and downbeats), and chords. When a new node connects to the server while the musical piece is already set and playing ("join" in **Figure 4**), it retrieves the same information and starts synchronizing with the other nodes. With the complete musical event information ready on each node, the node can prepare the performance of the musical piece on its own, instead of keeping asking master nodes.

Whenever after the musical piece to play is set on the server ("play" in **Figure 4**), a master node can issue a command to start playing it, pause it, or change the playing position. All nodes intermittently acquire the playback status by polling requests to the server, and therefore, the changes in the playback status are gradually shared among all nodes. In practice, the polling interval can be specified by the developer but is typically set to several seconds, and all nodes catch up with the server in a similar time order. The transferred playback status is very compact, consisting of whether or not the music is being played and the timestamp when the music playback is started on the global server clock.

4.3 Scalability Features

As explained above, the proposed architecture requires only intermittent communications between the server and nodes. Each master or slave node can join and leave the session at any time and, consequently, all the communications do not need to happen at the same time, while those in the always-on architecture do.

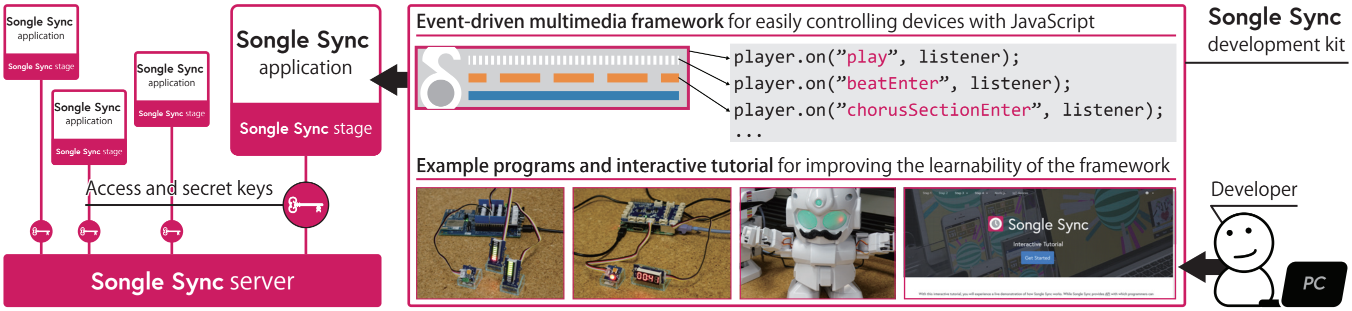
These characteristics already contribute to the scalable and stable control of the devices with the platform. Furthermore, there are several other features making it even more scalable.

4.3.1 Configurable Parameters for Synchronizations. Regarding the intermittent communications, the application developers can configure two parameters for each Songle Sync-based application to control the communication frequency. The first parameter defines the interval between the Clock Sync, and the second one defines the interval between the Event Sync. Depending on the needs of the application, the developer can experiment and find an appropriate set of parameters to reduce the network traffic and help increase the scalability.

When the interval between the Clock Sync is wider, the risk of out-of-sync caused by local clock drift and network congestion increases. This would not be an issue when the application can accept occasional time drifts and the nodes are expected to count time precisely (which is the case for most devices, except for some microcontrollers).

When the interval between the Event Sync is wider, the risk of delayed slave responses to the master's command increases. This would not be an issue when the master issues no additional command after starting the music playback.

4.3.2 Intermittent and Compact Data Transfer. Implementation-wise, the communications between the server and nodes are always handled by standard HTTP requests from the nodes to the server. Once the server responds to a request, it always disconnects the connection to release the socket. If it did not use such short-lived HTTP connections but keep-alive ones or other always-on protocols such as WebSocket, each connection would keep a file descriptor opened in the server operating system, resulting in resource outage when there are tens of thousands of the nodes connected to the server. The current implementation also supports a round-robin DNS for



```
player.on(”play”, listener);
player.on(”beatEnter”, listener);
player.on(”chorusSectionEnter”, listener);
...
```

Figure 5: The development kit for Songle Sync applications, each of which is identified with the keys and runs independently.

load-balancing the HTTP requests to the multiple synchronized Songle Sync server capable of hosting more nodes.

The above-mentioned efforts for intermittent communication also contribute to reducing the network traffic. Note that the playback status response from the server does not contain any time-varying values such as the current server time. Consequently, when there is no change in the playback status, the response data always remains the same, increasing the chance of hitting the cache and reducing the response traffic. Additionally, provided that the communications are HTTP-based, it is also possible to compress the response body in the gzip format to further reduce the traffic.

5 SOFTWARE DEVELOPMENT KIT

As a web-based platform, Songle Sync allows developers to create their own JavaScript-based applications (**Figure 5**). This section briefly introduces the programming experience on the platform.

5.1 Stages with Access and Secret Keys

There are potentially many developers who create hundreds of Songle Sync applications running simultaneously on the platform. Each application can be launched by many users on hundreds of a variety of synchronized devices and needs to be independent from other applications. The platform therefore introduces the notion of a **stage** to which each master and slave node is connected. Each **stage** on the server has its own status information consisting of the musical piece URL and playback status (including the timestamp on the server when the musical piece started).

To connect to a **stage** as a master or slave node, a JavaScript program needs a short text token named the **access key**. The master node additionally needs a **secret key**, with which the node issues a command to control the playback status. These keys are generated by encoding a universally unique identifier (UUID) when the developer creates a **stage** on the Songle Sync website and are initially visible only to the developer.

The **access key** is always required to join the session and is not considered secret information. The developer can easily create a URL that contains the access key as a query string or a QR code representing the URL, both of which are useful for advertising the web application. In contrast, the **secret key** is used to control the playback status and should usually be shared only among the developers, performers, or organizers of multimedia performances. If somebody else gains the **secret key**, the status of the application can be controlled in an unintended way, which could be dangerous when using physical computing devices.

5.2 Event-driven Multimedia Framework

To enable easy development of music-driven applications, the platform provides an event-driven multimedia framework for JavaScript. It is designed from the following perspectives.

First, the framework should prevent developers from writing boilerplate code to manage timings to react to interesting musical events. For instance, the developer might want to make objects bounce to the beat or to produce a more prominent visual effect when the chorus section starts playing. Developing music-driven applications requires coding a lot of such musical event-driven behaviours, and the event-driven application programming interface (API) facilitates this coding. While the idea was previously explored in Songle Widget [7], then the focus was on controlling a single browser-based application. In contrast, ours focuses on controlling hundreds of devices and is designed with portability and scalability in mind. While it can be loaded via a `<script>` tag in a browser-based application as in Songle Widget, it can also be loaded by a `require` function in a Node.js-based application that runs on microcontrollers and tiny computers. Furthermore, to reduce the network traffic, our API takes a modular approach that allows loading only some of the musical elements, not all of them as in the previous work.

Second, the framework should eliminate the need to write low-level code to handle connections between the server and master and slave nodes. As the event-driven API was previously proved to be effective, we aimed at keeping its simplicity while allowing device control at scale. Note that the Songle Sync API can be used to control a single device without synchronizing it with others. Meanwhile, the same application starts running on hundreds of the devices when a single line that passes an access key and optionally a secret key to the API is inserted.

5.3 Example Programs and Interactive Tutorial

To demonstrate the use cases of the multimedia framework and explain how the applications can be built, example programs for a variety of environments are developed and open-sourced in GitHub, including those for web browsers on a computer and smartphone and those for microcontrollers and tiny computers. Additionally, a web-based five-step tutorial is provided. This tutorial is built on top of insights gained from prior study on online coding tutorials [13].

With its clear structure consisting of five steps, it accepts learners with a wide range of knowledge on programming and involves their active engagement.

The first two steps are for both developers and end-users. The first step, "Experience Songle Sync," allows the user to choose an animation style from the list and to enjoy computer graphics animation synchronized with the music. When the same page is opened with another browser window or a smartphone, all the animations are synchronized. A master node repeating playback of a single musical piece runs on the tutorial server, allowing the user to experience Songle Sync at any time. The second step, "Enjoy Songle Sync with a Musical Piece of Choice," was described in **Section 3.1**.

The rest of the steps of the tutorial are intended for developers and each page contains in-page code editors with runnable example programs. By following the instructions and optionally editing programs, such as by changing musical piece URLs and access keys, the developers can learn and experiment with the API without leaving the tutorial. The tutorial is also integrated with GitHub and capable of publishing the edited source code as GitHub Gist.

6 EVALUATION

We referred to the literature [15] to design the following three evaluation studies: (1) performance evaluation to compare our framework against the baseline method of the always-on architecture, (2) deployment in the wild to test the practical and real usage and examine the "ceiling" of the framework, and (3) use in a hackathon event to test the "threshold" of the framework.

6.1 Performance Evaluations

6.1.1 Realistic Performance Measurement. To evaluate the performance limitation of our implementation, we measured network traffic, latencies, and jitters using existing musical pieces and off-the-shelf smartphones.

First, we measured network traffic of a web browser-based Songle Sync application that renders computer graphics animation and is used in the tutorial website. It initially loaded static files required to run the application (~500 kB), ran Clock Sync, and received the musical elements and their timing information (~30 kB). Later, it periodically ran Clock Sync and Event Sync (~5 kB/minute). If the same content were pre-rendered and provided as video streaming, there would be no initial load but the application would stably consume at least 1 Mbps (7.5 MB/minute). The total traffic is far more than ours.

Second, we used a high-speed camera to measure jitters — time differences between the slave nodes in a Wi-Fi-connected iPad mini 4 and six 4G-LTE-connected smartphones (three iPhone 7 Plus, iPhone 7, Sony Xperia XZ running Android, and NuAns NEO running Windows 10 Mobile). The observed jitters were less than 100 ms and none of the smartphones experienced significant latency or jitter that could affect the multimedia performance, regardless of connection types (Wi-Fi or LTE). This encouraging result will be verified in the wild setting (**Section 6.2**.)

6.1.2 Comparison with Always-on Architecture. We emulated an unstable network configuration with the Linux tc command controlling the netem kernel module and tested the performance of our architecture and the always-on architecture.

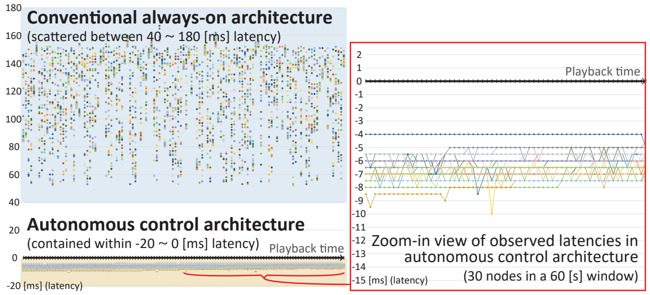


Figure 6: Results of the performance evaluations.

For our architecture, we created Docker containers of the Songle Sync server, a master node, and a slave node. For the always-on architecture, we created two containers of a simple WebSocket server (i.e., a master node) and client (i.e., a slave node). In both cases, connections between the container instances go through the emulated network. We spawned 30 slave nodes and played a musical piece for a minute to measure latencies and jitters.

As a result, we confirmed that our architecture outperforms the always-on architecture (**Figure 6**). With 100 ± 30 ms of emulated network latency, our architecture experienced -6.4 ms of average latency and each node did not experience noticeable time drifts. Note that ours allows negative latencies — nodes could trigger events slightly earlier than planned. In contrast, the always-on architecture experienced 116 ms of average latency and each node had a varied latency along time affected by the network jitter.

6.2 Deployments in the Wild

To confirm that the Songle Sync platform functions properly with hundreds of a variety of connected devices and in a challenging networking environment, we developed and ran the Songle Sync applications for the following two stages.

First, we conducted a demonstration experiment with the devices (**Figure 1**, **Section 3**). All the devices were connected to a single stage and behaved in synchronization with the musical piece playing on the speakers in the demonstration room. As a result, the server never became overloaded. Analysis of the collected log showed that over 110 devices were stably connected to the platform and kept synchronized with musical pieces.

Second, we developed a web-based Songle Sync application that dynamically renders the computer graphics animation in synchronization with a 40-minute music playback. This was used in a musical performance (**Figure 7**), where the audiences launch the application with the web browsers on their smartphones and enjoy the animation, which was shown on both the smartphone screens and the large screen in the front. Prior to the performance, we distributed 900 postcards with the printed QR code that navigates to the Songle Sync application website. As a result, more than 600 users used the application, at least 275 using it at the same time without any troubles, and such users included not only the original audience of the performance but also the visitors around the performance area. Many users launched the application before the performance started, which showed a standby screen and loaded the Songle Sync library in the background. This allowed to run the Clock Sync for multiple times before the actual performance.

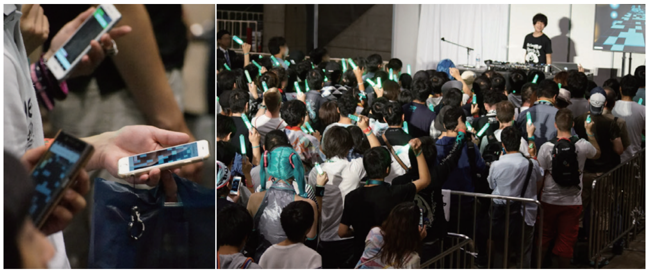


Figure 7: Songle Sync in the live multimedia performance.

6.3 Development Kit Usability

To confirm that our platform is usable by third-party developers, we conducted a two-day hackathon event with twenty-four university students. They formed six groups, each of which consisted of four students with diverse grades and diverse expertise in programming. On the first day, we spent an hour explaining the overview of the Songle Sync platform. Then for 3 hours we ran an idea-thon in which student groups discussed and proposed their ideas of the Songle Sync applications. For the rest 8 hours, the students prototyped their applications and prepared their presentations. During the implementation, we were open to any technical questions including JavaScript basics and kit usage.

As a result, all groups used example programs and/or tutorials to learn the platform usage, which was confirmed by monitoring the GitHub repository forks and Gist publications. Every student group succeeded in prototyping a working Songle Sync application. For instance, one group created a music game application for three or more players wearing headphones. All of them but one listen to the same musical piece and the goal of the outlier is not to be discovered by the others while the others try discovering the outlier. Implementation-wise, all smartphones are synchronized through Songle Sync and the outlier's musical piece is muted and replaced by the playback of a different piece.

7 DISCUSSION

7.1 Internet of Musical Things (IoMT)

Our goal of providing Songle Sync as an open platform is to allow many developers to create its applications and to investigate the potential of augmenting various sceneries with music and music-driven multimedia performances. Throughout deployments in the wild and a hackathon, we have gained initial feedback about our progress toward that goal. We have several use cases of Songle Sync in mind, the first of which was already experimented with.

First, as shown in **Section 6.2**, live performances with music can be augmented with music-driven devices. While we initially tested the use of Songle Sync during the performance, we also deployed a Songle Sync application before another performance, while the audiences of hundreds were waiting for the main performance in the concert hall. They were not bored but could use their smartphones to feel unity with others in the audience even before the performance. Second, at shopping malls, restaurants, and wherever else background music is used, developers can deploy Songle Sync-based applications to provide a more unified experience to visitors. Large screens, lightings, customer service robots, and any other computing devices can be driven by the background music. Third, a whole city can be connected to Songle Sync where streetlights are synchronized with music, buildings are lightened up with synchronized projection mappings, and visitors can join the synchronized session through QR codes shown at various places.

We foresee a future in which, with the help of application developers, objects of various scale — from a tiny LED to a whole city — can be connected to the Internet and synchronized with music. We call such networked objects the "Internet of Musical Things (IoMT)" and believe that Songle Sync can leverage fruitful relationships between Internet of Things and musical pieces.

7.2 Lessons Learned

As explained in **Section 6.1**, we confirmed that the proposed autonomous control architecture outperforms the conventional always-on architecture. It is feasible to use the proposed Songle Sync platform for building music-driven applications at the scale needed to realize the Internet of Musical Things (IoMT).

In the hackathon, the end-to-end support of Songle Sync for application development was appreciated. The tutorial allows prototyping without leaving web browsers, and example programs show the platform's actual use. The application can initially run on a laptop with a single musical piece and then be easily scaled up to run on hundreds of smartphones and other devices with a musical piece of choice. There is no need to prepare low-level code to synchronize devices, a server program to connect the devices, and annotations of musical pieces. Such completeness as a platform is sometimes underestimated in research but is critical in the wild.

In the live performances, we noticed that the users far from the main screen and out of the performance area tended to use the application to complement the synchronized visual experience. Some of the users sometimes seemed to be bored because there was no interactivity. This smartphone application would be more appealing if tapping the screens affected the animations. Such bi-directional communication has seldom been implemented in conventional applications of the always-on architecture, since communications are usually uni-directional (from master nodes to slave nodes). Songle Sync, however, is technically capable of bi-directional communication. Its future version should allow aggregating feedback from the slave nodes and processing it on the master nodes for interactive multimedia performances.

8 CONCLUSION

This paper proposed a web-based platform for controlling a lot of various devices in synchronization with music and evaluated its performance. We envision the "Internet of Musical Things (IoMT)" era in which the developer can easily make a lot of surrounding devices provide synchronized multimedia performances and the user can feel more sense of music and unity with the others. The platform is already available on the web with the development kit.

ACKNOWLEDGMENTS

We thank Keisuke Ishida, Rie Tanaka, Shuhei Tsuchida, and Hiromi Nakamura for their support in the demonstration experiment. This work was supported in part by JST ACCEL Grant Number JPMJAC1602, Japan.

REFERENCES

[1] Jonathan Foote, Matthew Cooper, and Andreas Girgensohn. 2002. Creating Music Videos Using Automatic Media Analysis. In *Proceedings of the Tenth ACM International Conference on Multimedia (Multimedia '02)*. ACM, New York, NY, USA, 553–560. https://doi.org/10.1145/641007.641119

[2] Network Time Foundation. 2014. ntp.org: Home of the Network Time Protocol. http://www.ntp.org.

[3] Shintaro Funasawa, Hiromi Ishizaki, Keiichiro Hoashi, Yasuhiro Takishima, and Jiro Katto. 2010. Automated Music Slideshow Generation Using Web Images Based on Lyrics. In *Proceedings of the 11th International Society of Music Information Retrieval Conference (ISMIR '10)*. 63–68.

[4] Gijs Geleijnse, Dragan Sekulovski, Jan Korst, Steffen Pauws, Bram Kater, and Fabio Vignoli. 2008. Enriching Music with Synchronized Lyrics, Images and Colored Lights. In *Proceedings of the 1st International Conference on Ambient Media and Systems (Ambi-Sys '08)*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, Article 1, 8 pages. http://dl.acm.org/citation.cfm?id=1363163.1363164

[5] Masataka Goto and Yoichi Muraoka. 1994. A Beat Tracking System for Acoustic Signals of Music. In *Proceedings of the Second ACM International Conference on Multimedia (Multimedia '94)*. ACM, New York, NY, USA, 365–372. https://doi.org/10.1145/192593.192700

[6] Masataka Goto, Kazuyoshi Yoshii, Hiromasa Fujihara, Matthias Mauch, and Tomoyasu Nakano. 2011. Songle: A Web Service for Active Music Listening Improved by User Contributions.. In *Proceedings of the 12th International Society of Music Information Retrieval Conference (ISMIR '11)*. 311–316.

[7] Masataka Goto, Kazuyoshi Yoshii, and Tomoyasu Nakano. 2015. Songle Widget: Making Animation and Physical Devices Synchronized with Music Videos on the Web. In *Proceedings of the 2015 IEEE International Symposium on Multimedia (ISM '15)*. IEEE, 85–88.

[8] Gracenote. 2018. Gracenote Developer Music + Auto APIs. https://developer.gracenote.com.

[9] Nicolas Hervé, Pierre Letessier, Mathieu Derval, and Hakim Nabi. 2015. Amalia.Js: An Open-Source Metadata Driven HTML5 Multimedia Player. In *Proceedings of the 23rd ACM International Conference on Multimedia (MM '15)*. ACM, New York, NY, USA, 709–712. https://doi.org/10.1145/2733373.2807406

[10] Xian-Sheng Hua, Lie Lu, and Hong-Jiang Zhang. 2004. Automatic Music Video Generation Based on Temporal Pattern Analysis. In *Proceedings of the 12th Annual ACM International Conference on Multimedia (Multimedia '04)*. ACM, New York, NY, USA, 472–475. https://doi.org/10.1145/1027527.1027641

[11] Volker Jung, Stefan Pham, and Stefan Kaiser. 2014. A Web-based Media Synchronization Framework for MPEG-DASH. In *Proceedings of the 2014 IEEE International Conference on Multimedia and Expo Workshops (ICMEW '14)*. IEEE, 1–2. https://doi.org/10.1109/ICMEW.2014.6890620

[12] Jun Kato, Tomoyasu Nakano, and Masataka Goto. 2015. TextAlive: Integrated Design Environment for Kinetic Typography. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. ACM, New York, NY, USA, 3403–3412. https://doi.org/10.1145/2702123.2702140

[13] Ada S. Kim and Andrew J. Ko. 2017. A Pedagogical Analysis of Online Coding Tutorials. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17)*. ACM, New York, NY, USA, 321–326. https://doi.org/10.1145/3017680.3017728

[14] Mozilla Labs. 2018. TogetherJS. https://togetherjs.com.

[15] David Ledo, Steven Houben, Jo Vermeulen, Nicolai Marquardt, Lora Oehlberg, and Saul Greenberg. 2018. Evaluation Strategies for HCI Toolkit Research. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA, Article 36, 17 pages. https://doi.org/10.1145/3173574.3173610

[16] Mario Montagud, Pablo Cesar, Fernando Boronat, and Jack Jansen (Eds.). 2018. *Mediasync - HandBook on Multimedia synchronization*. Springer. https://doi.org/10.1007/978-3-319-65840-7

[17] Mu Mu, Lyndon Fawcett, Jamie Bird, Jamie Jellicoe, Steven Simpson, Hans Stokking, and Nicholas Race. 2017. Closing the Gap: Human Factors in Cross-Device Media Synchronization. *IEEE Journal of Selected Topics in Signal Processing* 11, 1 (Feb 2017), 180–195. https://doi.org/10.1109/JSTSP.2016.2638358

[18] Offline. 2017. Offline - formerly Wham City Lights. http://offli.ne.

[19] ESTA Technical Standards Program. 2009. New Edition of DMX512-A Is Now Available. http://tsp.esta.org/tsp/news/newsdetails.php?newsID=291.

[20] Disneyland Resort. 2018. Made with Magic - Mickey Ears and More. https://disneyland.disney.go.com/entertainment/made-with-magic/.

[21] Danielle Sauer and Yee-Hong Yang. 2009. Music-driven Character Animation. *ACM Transaction on Multimedia Computing, Communications, and Applications* 5, 4, Article 27 (Nov. 2009), 16 pages. https://doi.org/10.1145/1596990.1596991

[22] David A. Shamma, Bryan Pardo, and Kristian J. Hammond. 2005. MusicStory: A Personalized Music Video Creator. In *Proceedings of the 13th Annual ACM International Conference on Multimedia (Multimedia '05)*. ACM, New York, NY, USA, 563–566. https://doi.org/10.1145/1101149.1101278

[23] Spotify. 2018. Spotfiy Echo Nest API – Developer. http://static.echonest.com/enspex/.

[24] Spotify. 2018. Spotify – Developer. https://developer.spotify.com.

[25] WildVreemd. 2016. The Smartphone Orchestra. https://smartphoneorchestra.com.

[26] Xylobands. 2012. Xylobands LED Wearable Wristbands. http://xylobands.com/xylobands.

[27] Adham Zeidan, Armin Lehmann, and Ulrich Trick. 2014. WebRTC Enabled Multimedia Conferencing and Collaboration Solution. In *Proceedings of the World Telecommunications Congress 2014 (WTC '14)*. 1–6.