Collaborative Illustrator with Android Tablets Communicating through WebRTC*

Shougo INOUE¹, Nonmember and Satoshi FUJITA²(a), Member

SUMMARY In this paper, we consider the collaborative editing of two-dimensional (2D) data such as handwritten letters and illustrations. In contrast to the editing of 1D data, which is generally realized by the combination of insertion/deletion of characters, *overriding* of strokes can have a specific meaning in editing 2D data. In other words, the appearance of the resulting picture depends on the reflection order of strokes to the shared canvas in addition of the absolute coordinate of the strokes. We propose a Peer-to-Peer (P2P) collaborative drawing system consisting of several nodes with replica canvas, in which the consistency among replica canvases is maintained through data channel of WebRTC. The system supports three editing modes concerned with the reflection order of strokes generated by different users. The result of experiments indicates that the proposed system realizes a short latency of around 120 ms, which is a half of a cloud-based system implemented with Firebase Realtime Database. In addition, it realizes a smooth drawing of pictures on remote canvases with a refresh rate of 12 fps.

key words: collaborative drawing, WebRTC, Android, semantics of two-dimensional data

1. Introduction

Recently, collaborative editing has become a popular activity thanks to the working style reforms such as the encouragement of work sharing and the popularization of teleworking. Google Docs [4] is a typical collaborative editing tool operating in web browsers. This tool enables concurrent proofreading with the writing of a report, for example, since changes made on a document are notified to the other users in real-time. Although similar applications such as Office Web Apps [1] are provided by several online storage services such as OneDrive and Dropbox, since they are realized as a *cloud service*, edited documents are uploaded to the cloud server and any keystroke given by a user must travel several hundred kilometers even if the collaborators work in the same room. In 2017, GitHub released a package called Teletype for Atom to support the collaborative programming with Atom editor without relying on a cloud server [18]. This package enables the collaborative editing of a text file through data channel of WebRTC [20], while avoiding write conflicts to the shared file through specific data structure called CRDT (Conflict-free Replicated Data Type) [10], [14].

In this paper, we consider the collaborative editing of *two-dimensional* (2D) data such as handwritten letters and illustrations. Editing of 2D data is different from the editing of 1D data in the following sense: First, the semantics of 2D data is often given by the *absolute coordinate* rather than the relative positions and the shape of strokes could have a specific meaning. For example, we can represent the present time in a picture by the distance between a straight line representing the horizon and a unit circle representing the sun (consider the animation of sunrise/sunset). Second, the semantics is often given by the *overriding* of strokes placed on a 2D canvas. For example, by overriding two circles filled with some color, we can represent which circle is closer and which circle is transparent (consider a situation in which a ball is hidden by another ball and the α value of the closer ball is relatively small). In fact, overriding is different from insertion nor replacement which are commonly used in the editing of 1D data.

By considering such characteristics of 2D data, this paper considers three editing modes based on users’ strokes. Those editing modes are realized as a part of *collaborative illustrator* implemented with Android tablets. The proposed system is realized as a collection of replica canvases in which the consistency among replicas is maintained through data channel of WebRTC. In the current implementation, the drawing data is sampled with 60 fps, and is transmitted to the canvas of collaborators with an interval of 85 ms, and then drawn on remote canvases with the default refresh rate. The latency of data transmission is approximately 120 ms, which is a half of a cloud-based implementation with Firebase Realtime Database.

The remainder of this paper is organized as follows. After reviewing related work in the next section, we describe the details of the proposed system in Sects. 3 and 4, respectively. Section 5 summarizes the results of evaluations. Finally, Sect. 6 concludes the paper with future work. This paper is an extended version of a paper [9] presented at CANDAR 2019. The difference to the conference version is summarized as follows: 1) we clarify the definition and the difference of three editing modes and add concrete use cases; 2) we add the way of applying CRDT to the editing of 2D data; 3) we add details of implementation with WebRTC including ICE and signaling; and 4) we add results of experiments concerned with the adjustment of optimal

Manuscript received January 7, 2020.
Manuscript revised May 26, 2020.
Manuscript publicized August 13, 2020.
¹The authors are with the Department of Information Engineering, Graduate School of Engineering, Hiroshima University, Higashihiroshima-shi, 739–8527 Japan.
²A preliminary version was presented as “Collaborative Illustrator with Android Tablets” by the same authors in CANDAR 2019. This work was partially supported by KAKENHI, the Grant-in-Aid for Scientific Research (B), Grant Number 16H02807.
a) E-mail: fujita@se.hiroshima-u.ac.jp
DOI: 10.1587/transinf.2020PAP0009

Copyright © 2020 The Institute of Electronics, Information and Communication Engineers
transmission interval with several observations.

2. Related Work

2.1 Collaborative Drawing Services

Collaborative drawing is widely used as a tool for fun or business, and in the past few years, several companies and organizations released drawing services such as Aggie.io⁰, Drawpile¹, Sketch board², and Flock draw⁳. Those services are roughly divided into two categories, namely cloud-based file sharing and P2P-based screen sharing. A representative of the former type is Chrome Canvas and that of the latter type is pixiv Sketch LIVE⁴, where the latter realizes the screen sharing through media channel of WebRTC (note that it does not support bidirectional collaboration since it just realizes the screen sharing of a designated user with the audience). Although there is a toy system based on a Java GUI toolkit and a simple master-worker model⁵, no previous work studies the bidirectional collaborative drawing based on the P2P technology.

Such a P2P system can be realized by preparing a replica canvas on each node and by connecting those nodes through communication channels, as in Teletype for Atom. So far, many proposals have been done in the literature to maintain the consistency of replicas in P2P systems⁶,⁷,⁸,⁹,¹⁰,¹¹,¹²,¹³,¹⁴,¹⁵,¹⁶,¹⁷. The idea used in P2P systems is applied to other fields such as distributed database systems. For example, distributed Key Value Stores such as Amazon Dynamo and Oracle NoSQL Database adopt a weak consistency model for replica control called eventual consistency in which the convergence to a legitimate state occurs without strict synchronization to the global clock.

2.2 CRDT

It is known that the commutativity of operations plays a crucial role in realizing an efficient consistency maintenance. CRDT (Conflict-free Replicated Data Type) provides a methodology to keep the consistency of replicas by restricting the class of operations to commutative operations.

Early CRDTs such as WOOT¹³ and Logoot¹⁴ are originally designed for the collaborative editing of 1D data, and the design of Treedoc¹⁰,¹⁴ is directly linked to the invention of CRDT. Text editing is a combination of insertion and deletion, which is realized by a data structure called Add-Remove Partial Order¹³. By definition, a partially ordered set contains one maximal element and one minimal element, and in the context of text editing, the beginning of a file corresponds to the minimal element and the end of the file corresponds to the maximal element. Then a user inserts a new element after a designated element, and deletes a designated element from the text. If two insertions to a point in the text are conducted in parallel, we may resolve such a conflict by using time stamps attached to each operation. More concretely, if \( c \) is inserted between \( a \) and \( b \) earlier than \( d \), then the resulting text is uniquely determined as \( adcb \).

LOGOOT and Treedoc use a tree structure to efficiently realize the above idea as follows: 1) First, arrange nodes in the tree so that any two adjacent elements in the text fulfill the parent-child relationship. For example, if a given text is \( ab \), then \( a \) is a child of the root and \( b \) is a child of \( a \). 2) Insertion of element \( c \) after \( a \) is reflected to the tree by creating a new left-most child \( c \) of node \( a \). The next element of \( c \) in the text is the sibling immediately to the right of \( c \) although it is not explicitly specified in the tree. Note that the insertion of element is achieved by expanding the width of the tree, rather than extending the depth. Therefore, even if several elements are inserted to a location in parallel, they can be easily reflected to the tree structure by arranging the order of children according to the time stamps. 3) Deletion of \( a \) from the text is realized by upgrading the leftmost child of node \( a \) to the position of \( a \). Note that the upgraded node should be the parent of all children of \( a \) (except itself) in addition to its own children, which would significantly increase the number of children of the upgraded node. LOGOOT and Treedoc overcome this issue in different ways (interested reader could refer to¹⁰,¹⁴,¹⁷).

Note that the above idea can naturally be applied to the editing of 2D data consisting of Add, Stroke, Undo and Redo operations, since it can be modeled as the manipulation of a partially ordered set (in contrast to the 1D case, delete of a stroke would give a damage to the appearance of the result picture, as will be explained later).

3. Collaborative Drawing of 2D Data

3.1 A Consistency Maintenance Problem

In the proposed system, users draw pictures on a shared canvas through touch screen of Android tablets. In Android API, such a drawing is regarded as a sequence of strokes, where stroke is a sequence of points on the canvas with additional attributes such as pen type and \( \alpha \) value. Upon being generated, strokes are converted into JSON format and are broadcast to all users through data channel of WebRTC, so that generated strokes are reflected to all nodes in real-time.

Appearance of the resulting picture depends on the reflection order of strokes. More concretely, it can have a different appearance if and only if it contains a pair of strokes crossing at a point and those strokes are reflected to the canvas in different order. Thus, to maintain the consistency of appearance of the picture over all devices, the reflection order of crossing strokes should be kept identical. In the following, two strokes are said to be intersecting if their enclosing rectangles have a non-empty intersection, where enclosing rectangle is a minimum circumscribed rectangle of the stroke parallel to \( x \)- and \( y \)-axes. Since any two cross-
ing strokes intersect, in the following, we assume that each stroke intersects with at least one stroke, without loss of generality (i.e., there is no isolated vertex in the terminology of graph theory).

We assume that each stroke is associated with a unique completion time and strokes generated by a user are reflected to the canvas in an ascending order of the completion time (see Appendix for the alignment of local clocks in a distributed environment). Undo/Redo of a stroke can be done unless its next stroke is reflected and such operations are easily realized using a list of strokes and an auxiliary stack, whereas in the collaborative drawing, Undo/Redo of a stroke could be blocked by intersecting strokes generated by other users.

3.2 How to Order Strokes

Concerned with the ordering of strokes generated by different users, the proposed system supports the following three editing modes.

3.2.1 No Control Mode

The first mode orders strokes in an ascending order of the completion time similar to the single-user case. To avoid unexpected interaction of strokes, we limit the length of each stroke to 5 sec. A use case of this mode is the creation of tourist attraction map, in which a rough sketch of the map is given first, and then the details are independently drawn by each user. Collaborative note taking and real-time commentary over short movies with handwritten letters would be promising use cases of this editing mode.

3.2.2 Static Priority Mode

The second mode prioritizes users in a static manner. It is realized by assigning a virtual layer to each user and prioritizing them so that a higher priority layer is drawn later, where strokes on the same layer are prioritized as in the first mode. In practice, the priority of layers and the assignment of users to the layers are managed by an administrator, which is generally the first user who starts the drawing of the picture. As a result, it enables us to create a collaborative illustration as is intended by the administrator. The reader could imagine the drawing of a picture composed of four virtual layers corresponding to distant view, close view, mob characters, and main characters.

3.2.3 Intentional Branching Mode

The third mode allows users to manage branches due to intersecting strokes. It follows the idea of Git distributed version control system. In this mode, each user explicitly acquires the content of the shared canvas to the local device (i.e., clone), edits it locally, and records the history of changes locally (i.e., commit). The outcome of edition is explicitly uploaded to the central canvas (i.e., push), and if it conflicts with the changes made by the other users, it resolves the conflict automatically or manually (i.e., merge).

4. P2P Communication with WebRTC

In the proposed system, JSON data corresponding to strokes are exchanged among remote users through data channel of WebRTC. Figure 1 illustrates an overview of the P2P communication in the proposed system. WebRTC is an API and protocol suite defined in HTML5, and is particularly suitable for interactive applications such as video conference and video chat.

4.1 Signaling and ICE

Each node in a P2P system must know about its counterpart before starting the actual data transmission. Such a media-tion process is called signaling, which is generally realized by exchanging SDP (Session Description Protocol) packets with necessary information such as IP address, transport layer protocol, and the type and format of the communication media (through Web server, for example).

P2P communication could take different route depending on the network configuration. For example, if each node inside NAT is given a private IP address, it becomes difficult for the outside nodes to communicate with inside nodes merely with their global IP address. ICE (Interactive Connectivity Establishment) [15] adopted in WebRTC is a general framework for overcoming such a difficulty. ICE uses two dedicated servers called STUN (Session Traversal Utilities for NATs) and TURN (Traversal Using Relays around NAT). STUN server is used to judge whether a given node is inside NAT or not and to obtain the global address of the node. Global address acquired through STUN server is shared by all nodes via an adequate signaling server to start the P2P communication. If the connection from outside NAT is blocked by the firewall, ICE tries to establish a pseudo P2P communication passing through TURN server.

Fig. 1 P2P communication in the proposed system [9].
4.2 Data Channel

WebRTC provides two communication channels called media channel and data channel, where media channel is used for media streams such as audio and video and data channel is used for general data streams.

We use data channel in the proposed system. It adopts SCTP or QUIC as the transport layer protocol, and to prevent eavesdropping and falsification, DTLS is used as the datagram encryption protocol. SCTP (Stream Control Transmission Protocol) was originally designed to improve the performance of TCP and UDP, and is widely used for the communication between base stations in cellular networks. Nevertheless, it has not been a general protocol for the Internet until now, since most NAT boxes such as broadband routers prohibit connections besides TCP and UDP. WebRTC overcomes this issue by implementing SCTP over DTLS (i.e., by having NAT recognize SCTP as UDP). QUIC (Quick UDP Internet Connections) is another transport layer protocol supported by WebRTC which is being developed by Google since 2013. QUIC is expected to be a successor of TCP plus TLS. In the current implementation, we use SCTP as the transport layer protocol.

5. Evaluation

5.1 Testbed

To evaluate the performance of the proposed method, we implement a testbed consisting of Android tablets with Android 7.0. All programs are written in Java 8. CPU of each device is MediaTek MT81638 (1.3 GHz, quad core) and each device has a RAM of 2 GB. Those devices are not connected to the same access point, and the route connecting given two devices is established with the aid of signaling server beforehand (typically, one device is connected to the campus network through Wi-Fi access point and another node is connected to a cellular network through 4G base station). See Appendix for the details of the signaling server.

In the experiments, the performance of the P2P-based system is compared with a cloud-based system implemented with Firebase Realtime Database [3], which is known as a typical cloud-hosted database. Figure 2 illustrates an overview of the cloud-based system. Firebase stores JSON data to the cloud database (DB) and is synchronized with clients in real-time, in such a way that all clients share one instance of DB and automatically receive updates with the latest data through WebSocket. More concretely, JSON data corresponding to a stroke generated by a node is received by another node in the following steps: at first, the sender stores the JSON data to DB, which is detected by the listener of clients connecting to the DB. Next, upon detecting the change of DB, the listener acquires JSON data from DB.

5.2 Latency

At first, we evaluate the latency of the collaborative drawing with two users. More concretely, we measure the time period from which user A draws a point on his screen to which the drawn point is displayed on the screen of user B. Since local clocks of those nodes might not be adjusted, the measured value is compensated by RTT to an NTP server1 which is measured for each node beforehand. We repeat such a measurement of the latency 1000 times. Figure 3 shows the distribution of the resulting latency. The average latency is 124 ms in the current implementation, which is almost equal to the time required for the data transmission from A to B since parsing of strokes to/from JSON data takes about 2 ms.

For comparison, we conduct a similar experiment with

\[\text{Fig. 2} \quad \text{Cloud-based implementation of the collaborative drawing system with Firebase Realtime Database [9].}\]

\[\text{Fig. 3} \quad \text{Distribution of the latency in the P2P-based method.}\]

\[1\text{https://www.mfeed.ad.jp/en/index.html}\]

\[2\text{Since the internal system of Android cannot specify an NTP server, the synchronization with a designated NTP server is conducted in the application layer using Apache Commons Net 3.6 API.}\]
the cloud-based system. Figure 4 summarizes the results. The average latency is 236 ms, which is almost a twice of the P2P-based implementation. Such a long latency of the cloud-based system is dominated by the RTT to the cloud server, since in the current setting of Firebase, we could not change the location of the data center from us-central1 which is located in the central of US.

5.3 Smoothness of Drawing Strokes

In Android tablets, the stroke given by a user through touch screen is continuously reflected to the canvas with a rate of about 60 fps. On the other hand, if such a stroke is received from other devices, it should be detected by the listener one by one, and be reflected to the canvas of the receiver by explicitly invoking a drawing function. Since the event detection by the listener incurs a certain overhead, if the transmission interval is too short (e.g., 60 times per sec), the overall overhead of the listener becomes heavy, which occasionally causes a suspension in the drawing. On the other hand, if the transmission interval is too long (e.g., 5 times per sec), the stroke is transmitted intermittently, which again degrades the smoothness of the drawing.

We conduct the following experiment to identify an appropriate data transmission interval:

1. Fix a stroke continuously generated on device A for a given time period (e.g., 5 sec);
2. Device A sends the stroke to device B with a fixed transmission interval; e.g., if the transmission interval is 1 sec, then the stroke is transmitted as a sequence of sub-strokes of length 1 sec each;
3. We then measure the time period from which the first sub-stroke is received by B to which the drawing of the entire stroke completes.

Since the entire stroke is always drawn on the screen of device B even if the data transmission is disturbed by external events such as buffer manipulation, the resulting time period indicates how much the influence of disturbance affects the drawing experience. Such a measurement is repeated 50 times for each combination of parameters.

Figure 5 summarizes the results, where the horizontal axis is the data transmission interval and the vertical axis is the drawing time averaged over 50 trials. From the figure, it is confirmed that the drawing takes around 5 sec for small time interval; e.g., it takes 5.15 sec if the time interval is 1/1 (=1.0) sec. However, the drawing time drastically grows as the time interval reduces; e.g., if the time interval is 1/60 sec, which corresponds to the maximum frame rate of Android (i.e., 60 fps), it takes about 30 sec to complete the drawing of a stroke of 5 sec; i.e., with a stretch of six times.

The amount of increase is relatively small when the interval is moderate; e.g., it takes 5.47 sec for 1/10 and 5.98 sec for 1/12. Note that those values are consistent with the subjective observation on remote drawing; in fact, we perceived no delay for 1/12 (= 83 ms), but a delay was certainly perceived when the interval becomes less than 1/15 (= 67 ms). In addition, we confirm that the stretch of the drawing time is a constant regardless of the stroke length.

On the other hand, under a long transmission interval, the quality of collaborative drawing could degrade due to the following reasons. First, the update rate of CustomPaintView at the receiver is no larger than k times per sec when the transmission interval is set to 1/k (i.e., k transmissions per sec). In fact, the frame rate observed at the receiver is k fps at transmission interval of 1/k for each 1 ≤ k ≤ 10, which saturates at k = 10, namely, we could not achieve a frame rate higher than 10 fps by reducing the transmission interval (the reader should remind that a low frame rate such as 1 fps is completely useless as a part of real-time collaborative drawing). Second, a long transmission interval incurs additional latency to the latency evaluated in the first experiment. In fact, when the transmission interval is 1/k, the stroke is sent out from the sender after waiting for 1/2k sec on average, where 1/2k takes 50 ms, 42 ms, and 33 ms when k = 10, 12, and 15, respectively. On the other hand, from Fig. 5, the increase of a stroke of 1 sec is 84 ms, 198 ms, and 522 ms when k = 10, 12, and 15, respectively. From those results, we can conclude that the transmission rate should be
set to either 1/10 or 1/12, where the former realizes a smaller stretch with a slightly longer latency.

5.4 Scalability

Finally, we evaluate the scalability of the proposed system with respect to the number of users.

At first, we can claim that the cloud-based implementation is scalable enough since Firebase Realtime Database allows 100 users to simultaneously access the database even in the free plan, which increases up to 0.1 million users in the pay-as-you-go plan. In addition, concerned with the cloud cost, users can download up to 10 GB data from the cloud per month in the free plan, and the supplemental cost is 1 $ per GB in the pay-as-you-go plan. Since the amount of data exchanged between two devices is only few KB per sec, we can conclude that the cost of the cloud-based system is sufficiently small.

In the following, we evaluate the scalability of P2P-based implementation. We use a full mesh connection as the P2P overlay; namely, we do not use a specific server for multicasting strokes to the other nodes. The number of nodes is either 2, 6, or 11, where the case of two nodes is the same as the first experiment. We measure the time period from which one user draws a point on his screen, to which the drawn point is displayed on all devices participating in the collaborative drawing. We repeat 1000 measurements by changing the role of nodes in the multicasting. Table 1 summarizes the results. It is confirmed that the average latency is bounded by 130 ms even if the number of nodes is 11; namely, we can conclude that the proposed system realizes a collaborative drawing for small groups with a sufficiently short latency.

6. Concluding Remarks

This paper proposes a collaborative drawing system based on the P2P technology. The result of experiments indicates that it could realize a collaborative drawing with a short latency (i.e., less than 130 ms) and a reasonable refresh rate (i.e., 12 fps). Although it is less scalable than a cloud-based implementation, the proposed system reduces the latency of the cloud-based system to almost a half.

We leave following issues as future work: 1) to propose a CRDT for editing 2D data; 2) to develop concrete applications such as collaborative note taking and real-time commentary; and 3) to port the proposed system to other platforms such as iOS and Harmony OS.

References

[1] D. Chartier, “Microsoft Office will float to the cloud with Office Web,” Ars Technica, Oct. 2008.

[2] X. Chen, S. Ren, H. Wang, and X. Zhang, “SCOPE: Scalable Consistency Maintenance in Structured P2P Systems,” Proc. IEEE INFOCOM, vol.3, pp.1502–1513, 2005.

[3] Firebase, https://firebase.google.com/docs/database

[4] Google Docs, https://www.google.com/docs/about/

[5] L. Lamport, “Time, clocks, and the ordering of events in a distributed system,” Commun. ACM, vol.21, no.7, pp.558–565, 1978.

[6] J. Lan, X. Liu, P. Shenoy, and K. Ramamritham, “Consistency Maintenance in Peer-to-Peer File Sharing Networks,” Proc. 3rd IEEE Workshop on Internet Applications (WIAPP ’03), pp.90–94, 2003.

[7] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kournanov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulkil, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W.-T. Chang, and Z. Shii, “The QUIC Transport Protocol: Design and Internet-Scale Deployment,” Proc. ACM SIGCOMM ’17, pp.183–196, 2017.

[8] D. Lee, “Google’s Chrome Canvas is a handy browser app for quick doodles,” the Verge, Dec. 2018.

[9] S. Inoue and S. Fujita, “Collaborative Illustrator with Android Tablets,” Proc. CANDAR ’19), pp.208–214, 2019.

[10] M. Letia, N. Preguica, and M. Shapiro, “CRDTs: Consistency without concurrency control,” Operating Systems Review, ACM, vol.44, pp.29–34, 2009.

[11] Z. Li, G. Xie, and Z. Li, “Efficient and Scalable Consistency Maintenance for Heterogeneous Peer-to-Peer Systems,” IEEE Trans. Parallel and Dist. Sys., vol.19, no.12, pp.1695–1708, Dec. 2008.

[12] NTT Communications Corporation, “SkyWay-Enterprise Cloud WebRTC Platform,” https://webtrc.ecl.ntt.com/

[13] G. Oster, P. Urso, P. Molli, and A. Imine, “Data consistency for P2P collaborative editing,” Proc. Int. Conf. Computer-Supported Coop. Work (CSCW), ACM, pp.259–268, 2006.

[14] N. Preguica, J.M. Marques, M. Shapiro, and M. Letia, “A commutative replicated data type for cooperative editing,” Proc. Int. Conf. on Distributed Comp. Sys. (ICDCS), pp.395–403, 2009.

[15] J. Rosenberg, “Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal of Offer/Answer Protocols,” RFC 5245, 2010.

[16] H. Shen, “An efficient and adaptive decentralized file replication algorithm in P2P file sharing systems,” IEEE Trans. Parallel and Dist. Sys., vol.21, no.6, pp.827–840, June 2010.

[17] H. Shen, “IRM: Integrated File Replication and Consistency Maintenance in P2P Systems,” IEEE Trans. Parallel and Dist. Sys., vol.21, no.1, pp.100–113, Jan. 2010.

[18] N. Sobo, “Code together in real time with Teletype for Atom,” Atom Blog, Nov. 2017.

[19] K. Yamanaka, S. Kamei, and S. Fujita, “Collaborative P2P Painting on a Shared Canvas,” Proc. CSE/EUC/DCABES 2016, pp.221–224, 2016.

[20] WebRTC, https://webrtc.org/

[21] S. Weiss, P. Urso, and P. Molli, “Logoot-undo: Distributed collaborative editing system on P2P networks,” IEEE Trans. Parallel and Dist. Sys., vol.21, pp.1162–1174, Aug. 2010.

Appendix A: SkyWay

By considering the sustainability of the resulting application, we use SkyWay [12] developed by NTT Communications in the implementation with WebRTC. SkyWay provides SDK to the software developers including signaling server and STUN/TURN servers, where we use Android SDK in our implementation. With this SDK, peer objects are automatically created upon starting the application. Created object connects to the signaling server and is assigned
Lamport’s logical clocks (the local clock of process 1 advances to 6 by receiving a message containing time stamp 5 from process 2).

Appendix B: Time Alignment

In order to reflect strokes to the shared canvas in a consistent manner, time stamps attached to those strokes must be as accurate as possible. In fact, if the clock of a node significantly gains compared with the other nodes (e.g., by 10 min), a stroke issued by the node will be overridden by the other nodes even if it was actually drawn after other strokes; namely we could not keep the meaning of resulting pictures.

There are two approaches to realize such an alignment of local clocks in a distributed environment. The first one is to synchronize to a (virtual) global clock through protocols such as NTP (Network Time Protocol), while it has an apparent drawback such that it could not guarantee the accuracy for offline users. The second approach is to use logical clock instead of physical clocks. In the Lamport’s logical clocks [5], for example, the local clock of each node “ticks” upon receiving a message from other node (so that the logical time advances the time contained in the received message by one if it is larger than its local time). In our case, such an approach with logical clocks correctly works since the overriding of strokes by the users defines a partial ordering of events, although it would take a certain time before the convergence.