A Serverless Signaling Scheme for WebRTC Using Bluetooth LE

Takuto Hashibe, Makoto Murakoshi, Teruaki Kitasuka, and Toru Nakanishi
Hiroshima University
Graduate School of Advanced Science and Engineering
Hiroshima, Japan
m235586@hiroshima-u.ac.jp, m226181@hiroshima-u.ac.jp, kitasuka@hiroshima-u.ac.jp, t-nakanishi@hiroshima-u.ac.jp

***Abstract*— In recent years, the widespread use of smartphones has increased the demand for image file sharing tools. Image files are highly private data that often show people's faces, and it is desirable that they be shared securely. Therefore, the final objective of this study is to implement an application that can share image files between iOS and Android devices via a secure P2P communication path using the WebRTC technology. In this paper, as a preliminary step to the final objective, we propose a serverless signaling scheme for WebRTC using BLE. In this scheme, the role of exchanging terminal information, which was played by WebSocket communication via a signaling server, is replaced by the BLE standard specifications of advertise, scan, and connect. In addition, the role of exchanging SDP and ICE Candidates is replaced by bidirectional communication using BLE write. We also verified that P2P communication was established by the serverless signaling and that image files could actually be shared between Android smartphones.**

***Index Terms*—WebRTC, BLE, Serverless Signaling**

I. Introduction

A. Research background

Smartphones are widely popular in Japan, with iOS and Android accounting for most of the OS in use. According to Statcounter GlobalStats [1], as of December 2022, iOS and Android account for 99% of the mobile operating system market share in Japan. Thus, most Japanese use one of these two OSs for their smartphones. In daily life, we see many scenes of people taking two-shot or group photos, and the following three are representative methods of sharing the photos taken.

- AirDrop
- Nearby Share
- LINE

AirDrop and Nearby Share can share images directly between devices without a server, but cannot be used across operating systems between iOS and Android. Unlike these two methods, LINE can share photos between iOS and Android devices across operating systems, but it cannot be used directly between devices, so photos are shared via a server.

As of August 2022, LINE uses a unique end-to-end encryption protocol called "Letter Sealing" for text message and location data communications, and messages cannot be decrypted by the server [2]. However, for image and video file communications, only the communication layer is encrypted, and there is no guarantee that the image and video files stored on the server are protected.

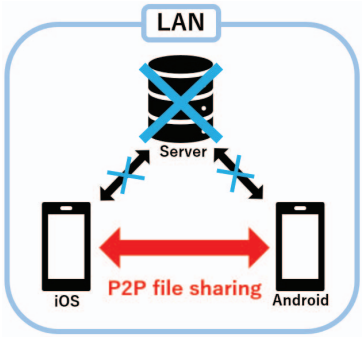


Fig. 1. Final objective

B. Research Objective

As shown in Fig. 1, the final objective of this research is to implement an application that straddles the iOS and Android operating systems and allows users to share image files with people nearby easily and over a secure communication channel, similar to AirDrop. Image files are highly private information that often contain people's faces. Since such information is often exchanged, it is necessary to use secure communication channel. In this research, P2P (Peer-to-Peer) communication is used as a secure communication channel. P2P communication is direct communication between terminals without a server, and is secure because it reduces the risk of eavesdropping. In this research, we are considering establishing a P2P communication path between nearby peers in a local network as a use case for users, and we believe that if this is achieved, easy sharing similar to AirDrop will be possible. To realize this cross-OS P2P communication function, WebRTC technology is adopted in this research.

There is another way to send files using bluetooth serial port profile, but it could not be used because iOS does not support this profile. [3] There are other bidirectional communication technologies such as WebSocket, but WebSocket excels in client-server bidirectional communication, while WebRTC excels in p2p bidirectional communication. In the future, when extending the functionality, communication between terminals

2832-1324/23/$31.00 ©2023 IEEE
DOI 10.1109/CANDARW60564.2023.00057

connected to different networks can be envisioned. In such cases, WebRTC is easier to extend because it has a mechanism for crossing NATs in p2p communication. For these reasons, we chose to use WebRTC for file sharing.

As we will discuss in Chapter IV, we have confirmed that p2p data communication between iOS and Android is possible with the WebRTC demo application [4] that we refer to. In other words, although signaling is implemented between Android devices in this paper, the possibility of data communication between iOS and Android is guaranteed when implemented on iOS as well.

WebRTC generally requires a signaling server to be set up for P2P connection between terminals. If WebRTC can be serverless, the risk of application functionality shutdown due to sudden server down can be ignored, thereby improving usability. Therefore, this study proposes a method and algorithm for serverless WebRTC between mobile terminals using BLE. Since BLE has been installed in many mobile terminals in recent years and can be used in many terminals, it was adopted as the communication method for serverless signaling.

In this paper, the technology that forms the basis of this research is explained in Chapter II. In Chapter III, we explain the general method and the proposed method, and in Chapter IV, we describe the design of the application. Furthermore, in Chapter V, we will check the operation of the actual application, and in Chapter VI, we will conclude.

II. Background Technology

A. WebRTC

WebRTC stands for Web Real Time Communication [5].WebRTC is a technology that enables web applications and web browsers to send and receive arbitrary data directly between browsers without the need for an intermediary. The architecture of WebRTC is divided into two major phases: the signaling phase and the data transfer phase [6].

1) Signaling Phase: In the signaling phase, an operation called signaling is performed. Signaling is the process of making P2P communication possible between two peers. SDP and ICE protocols are used in the signaling phase.

SDP stands for Session Description Protocol [7]. SDP is a format that stores information on the media description part and the session description part. Specifically, it stores information such as the codec information for audio and video, and the transfer rate for data communication. By exchanging this information, the terminals can select the codecs that can be used by each other and establish a P2P communication path.

ICE stands for Interactive Connectivity Establishment Protocol [8], which provides a mechanism to automatically search for available communication paths under various user environments (whether they are the same network or not) and set the optimal communication path.

The current WebRTC uses the Offer-Answer model as the negotiation pattern. This is explained in detail in the example of sender Alice and recipient Bob. Alice sends an SDP Offer to Bob through the signaling server, which contains information on her available codecs, etc. When Bob receives the SDP Offer, then creates an SDP Answer containing information on codecs that he can receive, and sends SDP Answer back to Alice through the signaling server. In this way, both parties determine the available formats, etc.

When both peers have finished exchanging SDPs, they then send ICE candidates to each other according to the ICE protocol. This is information containing possible communication paths, which are shared with the peer as soon as they are found. Then, among the multiple candidates, the route with the smallest network overhead is preferentially selected.

2) Data Transfer Phase: After the signaling phase, the data transfer phase begins. For data transfer, WebRTC provides two types of channels: a media channel and a data channel. In the media channel, video and audio stream data is communicated. In the data channel, any data can be exchanged. The two types of data available in the data channel are strings and binary data, which can be sent and received in both directions. We use only data channel, since the purpose of this research is to transfer image files. Therefore, the media channel is not used in this research.

B. BLE

BLE stands for Bluetooth Low Energy and was first introduced by the Bluetooth Special Interest Group (SIG) in 2010 as part of the Bluetooth 4.0 specification [9], defining the overall architecture and implementation details of BLE. As the name suggests, BLE is a standard dedicated to low power consumption and low cost, and is not compatible with the previous Bluetooth standard (Bluetooth classic). The BLE architecture is described below.

central and Peripheral are the roles in BLE communication. BLE provides communication between these two parties. The central is responsible for controlling the communication, and generally a smartphone plays this role. Peripherals, on the other hand, communicate in response to requests from the central, such as BLE beacons. Peripherals are defined by services and characteristics. A service represents a functional unit, and a characteristic exists within it. The characteristic contains the data that is actually used. Central devices communicate by reading and writing data by specifying these services and characteristics.

The flow of BLE communication is as follows. First, the peripheral device periodically broadcasts information about itself to make the central device aware of its presence. This operation is called advertise. Then, the central equipment scans the transmitted information to recognize peripheral devices in the vicinity and searches for devices with which it can communicate. Data communication begins when the central device finishes its search, selects a connection target, and issues a request to start the connection. At this time, the peripheral device stops advertising and moves into the data communication phase with the connected device.

III. Design of BLE Signaling

This section describes both general and proposed signaling design. The changes on the proposed signaling is also

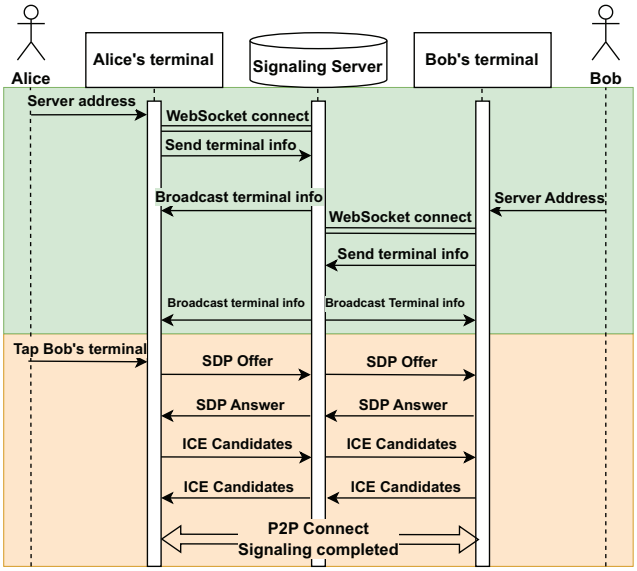


Fig. 2. General Signaling with a Server

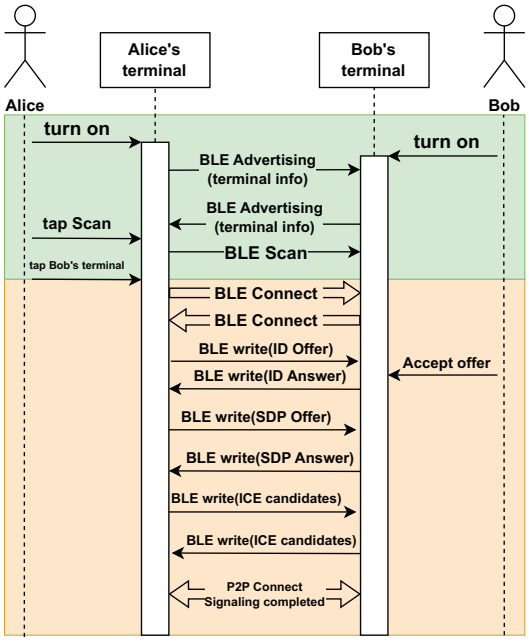


Fig. 3. Proposed BLE Signaling

described. Fig. 2 and Fig. 3 show sequence diagrams summarizing each method. These sequence diagrams show the exchange of terminal information in the yellow-green area and the exchange of SDP and ICE Candidates in the orange area.

A. General Signaling with a Server

The sequence diagram shown in Fig. 2 illustrates general signaling techniques using a signaling server. In Fig. 2, the general method is illustrated with the example of Alice and Bob being the sender and receiver, respectively. Both Alice and Bob's terminals connect the signaling server using WebSocket. When Alice's terminal connects to the server, it sends information about its own terminal to the server. Then, the server broadcasts Alice's information to the all connected terminals. When Bob's terminal connects to the server, the server broadcasts as in the Alice case. This broadcast allows users to recognize the users with whom they can currently communicate. When Alice selects Bob's terminal, Alice's terminal begins signaling. Alice sends an SDP Offer to Bob via the server, and Bob returns an SDP Answer. Then, they send ICE candidates to each other and initiate P2P communication on that route. In this way, the signaling server is used as a bridge to exchange SDP and other information between two terminals before P2P communication begins.

B. Proposed BLE Signaling

In this study, we propose to use BLE for signaling instead of a signaling server. The sequence diagram shown in Fig. 3 illustrates the proposed method of WebRTC signaling using BLE.

In Fig. 3, the proposed method is illustrated with the example of Alice and Bob being the sender and receiver, respectively. When Alice and Bob launch the application on their respective terminals, each terminal starts sending the advertisements with its own terminal information. Then, when Alice's terminal scans and receive an advertisement of Bob's terminal, she can retrieve the terminal information of Bob's. Alice selects Bob and makes a BLE connection request. When Alice selects Bob as her communication counterparty, her terminal makes a BLE connection request to Bob's terminal.

In this way, the proposed method enables the exchange of terminal information through the standard BLE specifications of advertise, scan, and connect, whereas in the general method, terminal information is deliberately sent and broadcasted to the server. The above process enables BLE data communication between Bob and Alice's terminals. Since Alice's terminal plays the role of central and Bob's terminal plays the role of peripheral, Alice's terminal can read and write the characteristics of Bob's terminal, but Bob's terminal cannot actively transmit data. Therefore, immediately after Alice's terminal connects to Bob's terminal via BLE. Bob's terminal sends a connection request to Alice's terminal to connect, thereby enabling both terminals to play the roles of central and peripheral, and to send data at any desired timing. Once the BLE connection is established, the SDP and ICE candidates are sent to each other over the established connections. When sending data to the other terminal, data is sent using write, which writes data to the other terminals characteristic.

The proposed method adds two actions, ID Offer and ID Answer, which did not exist in the general signaling method. This is because a unique ID, which is an identifier of each terminal, is required when exchanging SDP and ICE Candidates, and a phase to exchange IDs is provided in order to understand each IDs. This ID is a six-digit integer ID randomly generated by the terminal itself and exchanged with the communication

TABLE I
DEVELOPMENT ENVIRONMENT

Category	Content
Terminal OS	Android 13
Terminal model	Google Pixel 7
Framework	Flutter 3.10.5
Language	Kotlin 1.8
Language	Dart 3.0.5
IDE	Android Studio Electric Eel 2022.1.1 Patch 2

partner using ID Offer and ID Answer. Another method is to use a dynamically generated MAC address as a unique ID. However, due to the BLE specification, it is not possible to obtain its own MAC address, so it is necessary to ask the communication partner to tell it [10]. The number of times data is sent and received is the same between using the MAC address as the ID and generating and using a random ID. After the exchange of SDP and ICE candidates is complete, a route with low network overhead is selected as in the general method, and P2P communication begins on that route.

IV. IMPLEMENTATION

A. Development environment

The development environment is shown in TABLE I. Since the final objective is to implement an application that can be used on both iOS and Android operating systems, we used the Flutter [11] framework and Dart language for development, which are suitable for multi-platform development. For development, we used two open source WebRTC software [4] [12]. For the application itself, we used the flutter-webrtc-demo program [4] as a reference. We also used flutter-webrtc-server [12] as the server-side program. These two programs are demo versions of WebRTC applications that use a signaling server for signaling, which We describe as the general method. We implemented serverless signaling by modifying the flutter-webrtc-demo program.

In this study, we tested with various packages that are publicly available for using BLE with Flutter, but they did not work well on Android devices. Therefore, we used the Flutter method channel API provided in Flutter. We wrote the code for BLE communication in Kotlin, and called it from the Flutter side.

We implemented serverless signaling between Android devices that did not work well with the BLE package.

The important parts of the implementation are explained in the following sections IV-B and IV-C.

B. method channel

The BLE library for the Android platform must be called in Kotlin or Java language. To call the BLE library from Flutter app, we use Flutter method channel [13] [14]. The method channel allows us to pass messages between Flutter app written in Dart and Android host written in Kotlin. In this study, the Android BluetoothLeChat sample code [15] was used as a reference when implementing BLE functionality in Kotlin. Data transfer using method channel is shown in Fig. 4. Black arrows indicate data exchange between Flutter and Kotlin through method channel. The green arrows indicate data exchange through BLE. This application uses the String type when sending SDP and other data via BLE write. The reason for using the String type is that it supports both method channel data passing and BLE data passing. When sending data from Flutter to Kotlin, the data is converted to JSON format as a String and sent.

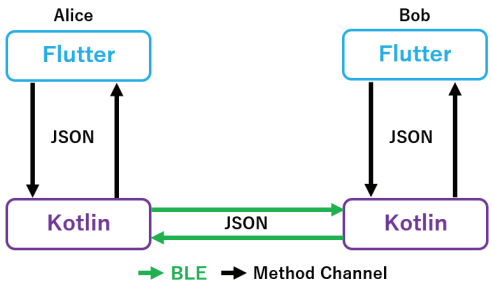


Fig. 4. Data transfer using method channel

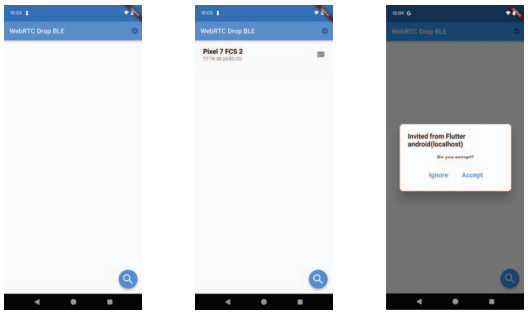
C. Data transfer rate of BLE

There is a limit to the number of bytes of data that can be recived/sent at one time in a BLE read/write operation, which is set to 20 bytes by default. When tested with the Google Pixel 7 used in this study, only 18 bytes could be transmitted at a time. In addition, the BLE specification does not allow the next read/write operation to be performed before the response is returned after one read/write operation, so an appropriate interval must be provided when sending multiple packets. In this regard, testing on the Google Pixel 7 used in this study showed that data could be received after an interval of at least 120 ms. However, the SDP Offer to be sent in the signaling performed in this study has approximately 600 characters, and it takes approximately 4 seconds to send one of these characters, which is quite slow. To solve this problem, the MTU of the BLE data communication was extended using the ATT_MTU extension protocol. When tested on the Google Pixel 7a used in this study, the MTU could be extended up to 512 bytes. This significantly increased the data transfer rate of BLE. Specifically, An SDP of approximately 600 characters takes 240 ms because the write is divided into two writes.

V. PROTOTYPE

In this section, we will check the operation of a serverless application realized by BLE with actual screenshots.

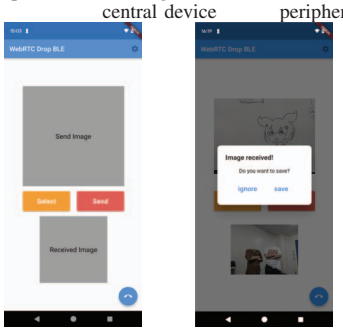
First, when the application is launched, the screen looks like Fig. 5a, and BLE advertisement starts automatically. This process takes place at both terminals, whether they are senders or receivers. Then, tapping the button with the magnifying glass icon in the lower right corner starts a scan, and when the scan is complete, the names of nearby devices that can communicate are listed and displayed as shown in Fig. 5b. When sender tap the device you want to start communicating with, the sender's terminal issues a BLE connect request to the



(a) first screen of both central and peripheral devices

(b) screen after scanning on the central device

(c) confirmation dialog of invitation on the peripheral device



(d) Image sharing screen of both central and peripheral devices

(e) Confirmation dialog to receive an image on both central and peripheral devices

Fig. 5. Screenshot of Signaling Serverless Fileshare

receiver's terminal, and when the connection is established, the receiver's terminal also issues a connect request. Then, when mutual connection is established, exchange of IDs starts. ID exchange begins with the sender terminal sending an ID Offer to the receiver terminal. This ID Offer contains a randomly issued unique ID of the terminal and the name of the terminal that sent it. When the receiving terminal receives this ID Offer, it displays the name of the terminal that sent the ID Offer as shown in Fig. 5c, and outputs a dialog asking whether to continue signaling. If the receiver presses Accept on this dialog, ID Answer is returned and signaling begins. On the other hund, if the receiver presses Ignore, the signaling is discarded.

When signaling is completed, the two terminals enter the data transfer phase and transition to the screen shown in Fig. 5d. When the terminal transitions to this screen, the P2P data channel has already been opened and binary data can be sent and received at any time. Then press the Select button to select an image from the device's gallery and press the Send button to send the image to the recipient using the P2P data channel. As you can see in Fig. 5e, we confirmed that the image files were shared over the P2P data channel.

VI. Conclusion

We proposed a serverless method of WebRTC using BLE, implemented it, and confirmed that it works. Specifically, the role of exchanging terminal information through WebSocket communication via a signaling server was replaced with advertise, scan, and connect, which are specifications of the BLE standard. In addition, the role of exchanging SDP and ICE Candidates, which is the main part of signaling, was replaced with two-way communication using BLE write, and an ID exchange phase was added in the middle of serverless signaling. In this way, we realized serverless signaling by using the BLE function that is standard in mobile terminals. By using this method, there is no risk of the application stopping due to server failure. As a result, the usability of file-sharing apps has improved. A future work is to implement BLE-based serverless signaling so that it can be used on iOS terminals as well. Since the final objective of this research is to perform file sharing across operating systems, file sharing between Android terminals at the current stage is not sufficient. In the future, we must implement BLE-based signaling on iOS terminals in the same way as in this study, and confirm whether P2P file sharing between iOS and Android is possible.

References

[1] Statcounter GlobalStats
https://gs.statcounter.com/os-market-share/mobile/japan (accessed January 12, 2023).

[2] LINE Corporation, LINE Encryption Report, December 2022
https://linecorp.com/en/security/encryption/2022h1 (accessed July 28, 2023).

[3] Apple, Bluetooth profiles that iOS and iPadOS support, May 8 2023
https://support.apple.com/en-us/HT204387 (accessed October 11, 2023).

[4] Flutter WebRTC community, flutter-webrtc-demo, Github
https://github.com/flutter-webrtc/flutter-webrtc-demo (accessed July 31, 2023).

[5] WebRTC API, Mozilla Developer Network web docs
https://developer.mozilla.org/ja/docs/Web/API/WebRTC_API (accessed July 28, 2023).

[6] Kensaku Komatsu. "What Impact Does WebRTC Has?; Innovative game changer on web business". The Journal of the Institute of Image Information and Television Engineers. vol.70, no.3, pp.215–219, March 2016. (in Japanese)
https://www.jstage.jst.go.jp/article/itej/70/3/70_215/_pdf (accessed January 12, 2023).

[7] Ali C. Begen, Paul Kyzivat, Colin Perkins, Mark J. Handley, "SDP: Session Description Protocol", RFC8866, January 2021
https://datatracker.ietf.org/doc/html/rfc8866 (accessed July 28, 2023).

[8] Ari Keränen, Christer Holmberg, Jonathan Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal", RFC8445, July 2018
https://datatracker.ietf.org/doc/html/rfc8445 (accessed July 28, 2023).

[9] Bluetooth® Low Energy Primer, Bluetooth SIG, 2023
https://www.bluetooth.com/ja-jp/bluetooth-resources/the-bluetooth-low-energy-primer/ (accessed July 28, 2023).

[10] Android developers, Android 6.0 Changes,January 2023
https://developer.android.com/about/versions/marshmallow/android-6.0-changes.html#behavior-hardware-id (accessed August 9, 2023).

[11] Flutter:
https://flutter.dev/ (accessed July 28, 2023).

[12] Flutter WebRTC community, flutter-webrtc-server, Github
https://github.com/flutter-webrtc/flutter-webrtc-server (accessed July 31, 2023).

[13] Flutter, Writing custom platform-specific code
https://docs.flutter.dev/platform-integration/platform-channels (accessed July 28, 2023).

[14] javadoc flutter MethodChannel
https://api.flutter.dev/javadoc/io/flutter/plugin/common/MethodChannel.html(accessed July 28, 2023).

[15] Android BluetoothLeChat Sample, Github
https://github.com/android/connectivity-samples/tree/main/BluetoothLeChat (accessed July 28, 2023).