Clock Synchronization

ECSE-420 Parallel Computing

Alonso Medina	260480522
Cheng Gong	260463255
Yuechuan Chen	260504371
Faraz Oman	260576984

1. Abstract

In this project, we developed an Android app that consists of a multiplayer game in which two players face off against each other. After delving into Android development in an earlier assignment, we became more interested in exploring server-side problems in an Android app. In particular, we were looking to test out different ways of maintaining synchronization in the client-server connection, by implementing a simple Android game. By focusing on the clock synchronization between client and server, we hope to achieve a reasonable fairness between players that have different network environments. A prototype server was implemented using Node.js and SocketIO to support client server communication. Furthermore, we implemented three different synchronization methods, no sync, cristian algorithm and berkeley algorithm.

Table of Contents

2. Abbreviations & Definitions

API	- Application Programming Interface.
Node.js	- Runtime environment for server-side web applications.
Socket.io	- JavaScript library that implements websocket protocol.
OS	- Operating System
IDE	- Integrated Development Environment
RTT	- Round Trip Time

3. Background

In today's world, technology is making advances at an ever-increasing rate. One of the reasons for these increases in performance are thanks to the advances in parallelism and synchronization research done by computer scientists. The extent of these advances can be seen in the widespread use of modern mobile devices. As people interact with these devices more and more frequently, and use these devices to interact with others, there has been an increasing demand for faster and more responsive gadgets.

Android:

As we reach the physical limit of computer hardware, more and more effort goes into developing new parallel algorithms and improving multicore communication efficiencies. This very trend is seen extensively in the mobile market nowadays. Chipmakers aims to maximize battery usage by reducing (per core) frequency and increasing cores per chip. This creates new sets of restraints that developers requires to face while making mobile application. Below are some of the parallel/synchronization related challenges an Android developer needs to take into account :

- Application is heavily reliant on network connection, users expects seamless browsing / gaming experience. Proper caching and reducing latency is the key to a good user experience.
- Application may also require real time responsiveness. For example, our game requires clock synchronization between the server and the client to avoid giving an unfair advantage to either players due to network latencies.
- Responsive UI requires the delegation of application logic to background threads that run on separate cores to prevent the blockage of UI animations.
- The synchronization of threads and the access of exclusive resources by those threads requires careful consideration. Solutions to the resource access problem include the resource **lock**, the **Observer pattern** (accessing system resources) , and **Message passing** via Handlers (executing a piece of code on another thread).

The purpose of the Android application is to investigate various server-client synchronization and communication approaches. In addition, we are looking forward to gain a moderate understanding of the prevalent parallelism techniques used in Android.

Node.js Server:

We have long been curious about the infamous Node.js framework that began to gain attraction back in 2011. Node.js is a server side platform built on Chrome's V8 JavaScript Engine. It takes advantage of an event-driven, non-blocking I/O model that makes it lightweight and efficient. According to Mike Pennisi, a professional blog writer who has performed extensive stress testing on both Node.js and Socket.IO, Node.js is capable of handling more than ten thousand concurrent connections with little impact on its average response time. Thanks to the non-blocking I/O operations, Node.js requires only one OS thread to run the main event loop on, thereby avoiding the possibility of deadlocking (since there are no locks) and reducing the complexity of development. To utilize the full potential of a modern multi-core computer system, Node.js also provides

means to spawn child processes and allows port sharing, as well as interprocess communication.

Our game server is built with Node.js and uses Socket.io as the means to establish realtime communication with the mobile game.

Gameplay:

The app offers a fast paced yet straightforward gameplay that encourages quick reflexes. The game consists of a single screen in which each player is presented with three buttons. The goal of the game is to press the buttons as quickly as possible without pressing any of the buttons the players pressed last, leaving only one clear choice at a time. Once a player has pressed a button, that button changes colour to indicate that it shouldn't be pressed again until the player has pressed a different one. The game ends when a player presses the incorrect button. If a player pressed the wrong button then they get eliminated, and the remaining players play until one player is left. In the case that the game reaches its time limit with neither player making a mistake then the winner is determined by whoever has scored the most points (i.e. pressed the most buttons) during the course of the game. This means that players that play too passively will lose in the score count, but players that are overly aggressive will likely make a mistake and lose, thus the player must find a good balance between the two. For an example of what the game looks like please see Figures 1 and 2 below.

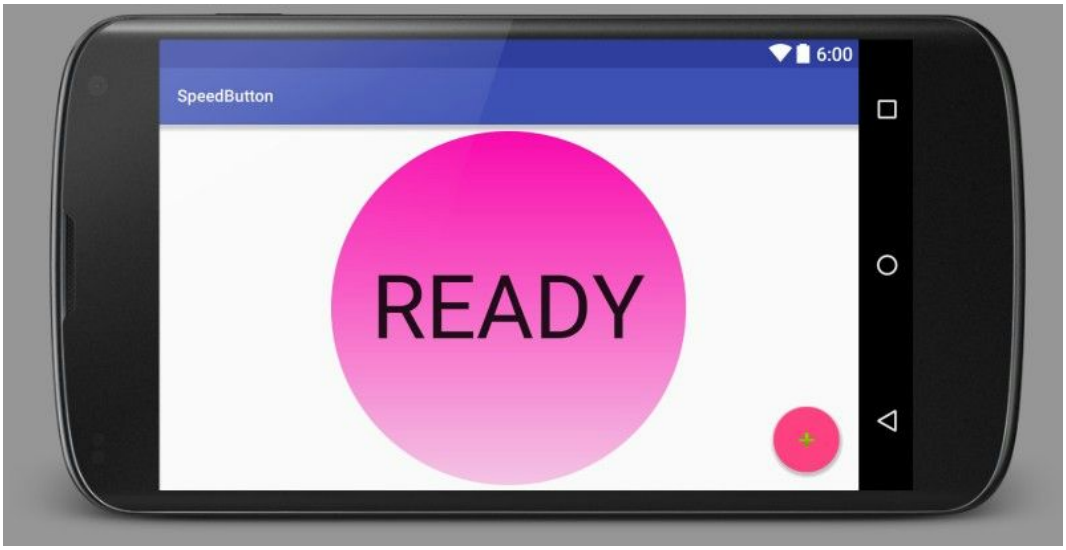

Figure 1: App main screen

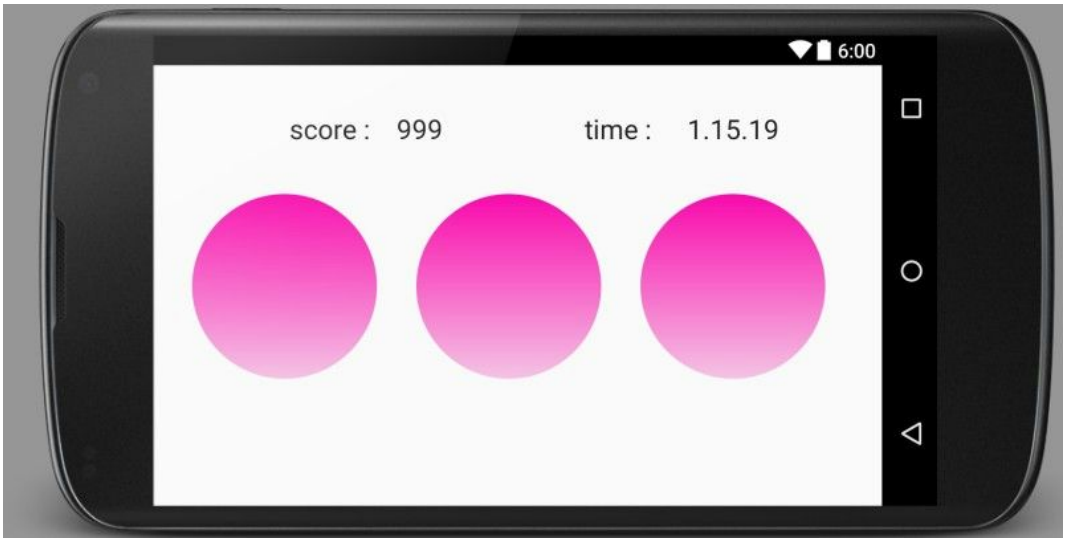

Figure 2: App game screen

This game offers several aspects which we can improve using parallel programming, such as the connection between the players as well as the connection between both the player and the server, keeping track of each player's score, etc. It should be noted that this game was developed because we needed a platform to test the different synchronization implementations on; although the game does contain parallel components, this was not the main focus of the project. Thus, the majority of the effort

went towards testing the client-server connection rather than the design of the game itself.

4. Requirements

Functional requirements:

- The app must allow the user to match versus an opponent.
- The game must have near perfect sync so that it appears perfect to the user to play the game as intended

Non-functional requirements:

- The app must be stable.
- The app must provide feedback to the user based on results of user commands.
- The app must provide feedback near instantaneously after the command was issued.
- The app must have an intuitive interface.

Constraints:

- The number of games to run at once and players who are able to play will be limited to only one game and two players in the game

5. Design

Websocket:

The communication between the server and the Android clients uses SocketIO to send each other messages such as new player has joined, synchronization, event updates and game results. Examples of how events are handled can be found below.

```
// listening on button_click event
socket.on('button_click', function(data) {
  var iButtonClicked = data.button_clicked;
  var iScore = data.score
  checkButtons(iButtonClicked, socket, iScore);
});
```

Figure 3: Server listen on button click event from clients

```
// if the click is valid, inform the opponent about the new score
socket.emit("score_update", {score : iScore, button_update : iButtonClicked});
console.log("valid click");
```

Figure 4: Server broadcast a client's score and button clicked to other clients

```
public Emitter.Listener updateEvents = new Emitter.Listener() {
    @Override
    public void call(final Object... args) {
        activity.runOnUiThread(new Runnable() {
            @Override
            public void run() {
                JSONObject data = (JSONObject) args[0];
                String result;
                try {
                    result = data.getString("message");
                    int takenButton = Integer.parseInt(result);
                    String buttonId = "button" + takenButton.toString();
                    GameButton mButton = (GameButton) activity.findViewById(R.id.buttonId);
                    Log.d("buttonTaken", buttonId);
                    mButton.setTaken(true);
                } catch (JSONException e) {
                    return;
                }
            }
        });
    }
};
```

Figure 5: Android client update button click from other players

App implementation:

The app was developed using Android studio in Java and XML. We also used Android Annotations to make the coding more streamlined. All the graphics in the game are taken from the android Drawable and Resources classes. We have three separate activities: MainActivity, RoomActivity and GameActivity. The activities are started by an intent in the order listed. In each of the activities there are buttons or shapes which will trigger a motion event if a finger touches it. When the app first enters the Game activity, it will connect with the server and compute the time offset. From there the game is then initiated.

Figure 6: App implementation diagram

Synchronization Methods Implementation:

Synchronization between the server and clients needs to be taken care of since clients may have different network environment (e.g: dial-up, cable and ADSL) and this plays a major role on the game result since it is a relatively fast-paced game.

1. **No Synchronization**

   The first server prototype has been implemented without any synchronization. Whenever a client presses a button, it sent a message to inform the server about this event. A server responds to events as FIFO fashion without taking into account of the network delay of the clients. This implementation does not ensure the fairness of the game, since a player with a really good internet connection can potentially win the game even this player click the button after another player that has a slower internet connection.

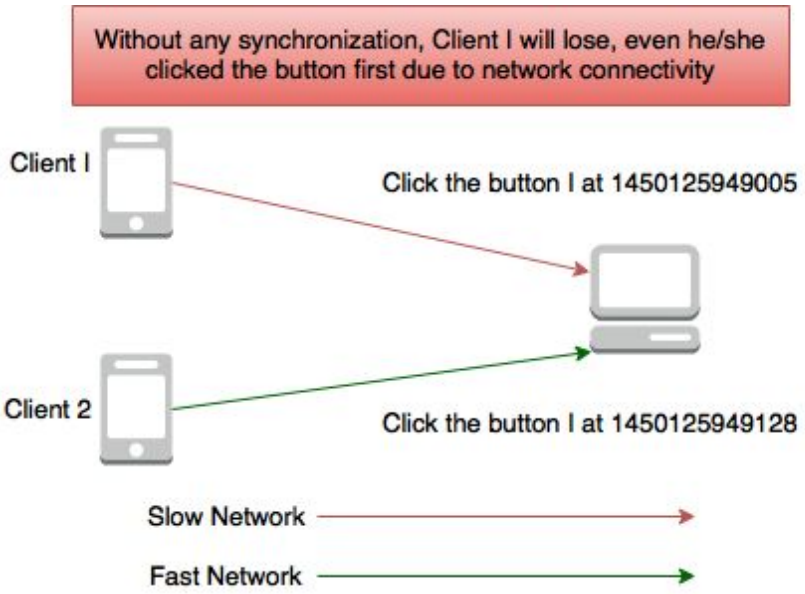

Figure 7: No Synchronization Implementation Diagram

2. **Cristian's algorithm**

Cristian algorithm was first introduced in 1989 by Flaviu Cristian. Cristian's Algorithm works between a process P, and a time server S. First, P requests the time from S. After receiving the request from P, S prepares a response and appends the time T from its own clock. Finally, P then sets its time to be T + RTT/2.

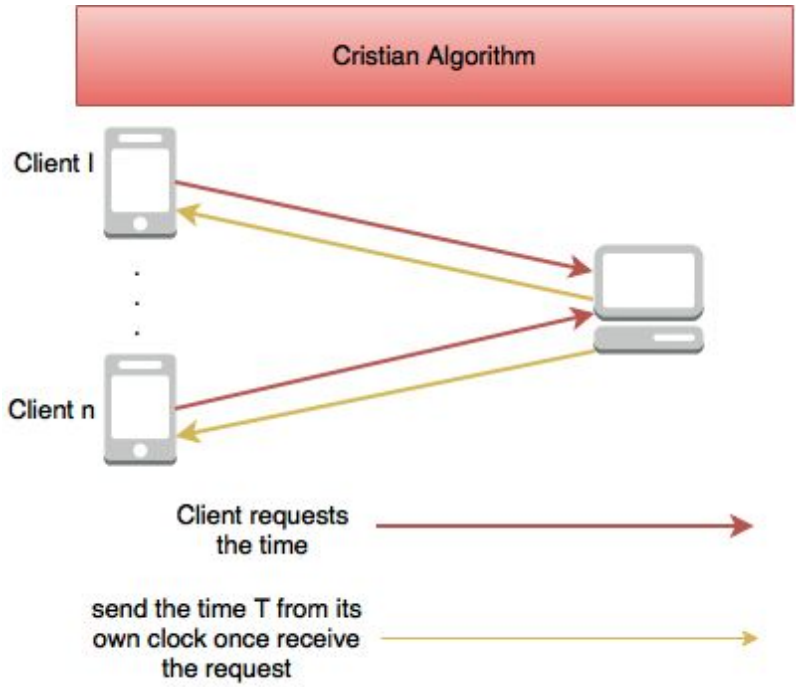

Figure 8: Cristian Algorithm Implementation Diagram

3. **Berkeley algorithm**

Developed in 1989, the algorithm is a method of clock synchronisation in distributed computing. The algorithm works in a master-slave setting where, in our case, the master is the server and the slaves are the clients. First, the master will poll all the slaves sequentially gathering the RTT's and estimating the client's time. Then the server will average all the clock times including its own and then will calculate the offset for each of the clients individually and send the calculated offset to each of the clients. The clients will then use the offset to add to its time when communicating with the server. A diagram of the algorithm can be seen below.

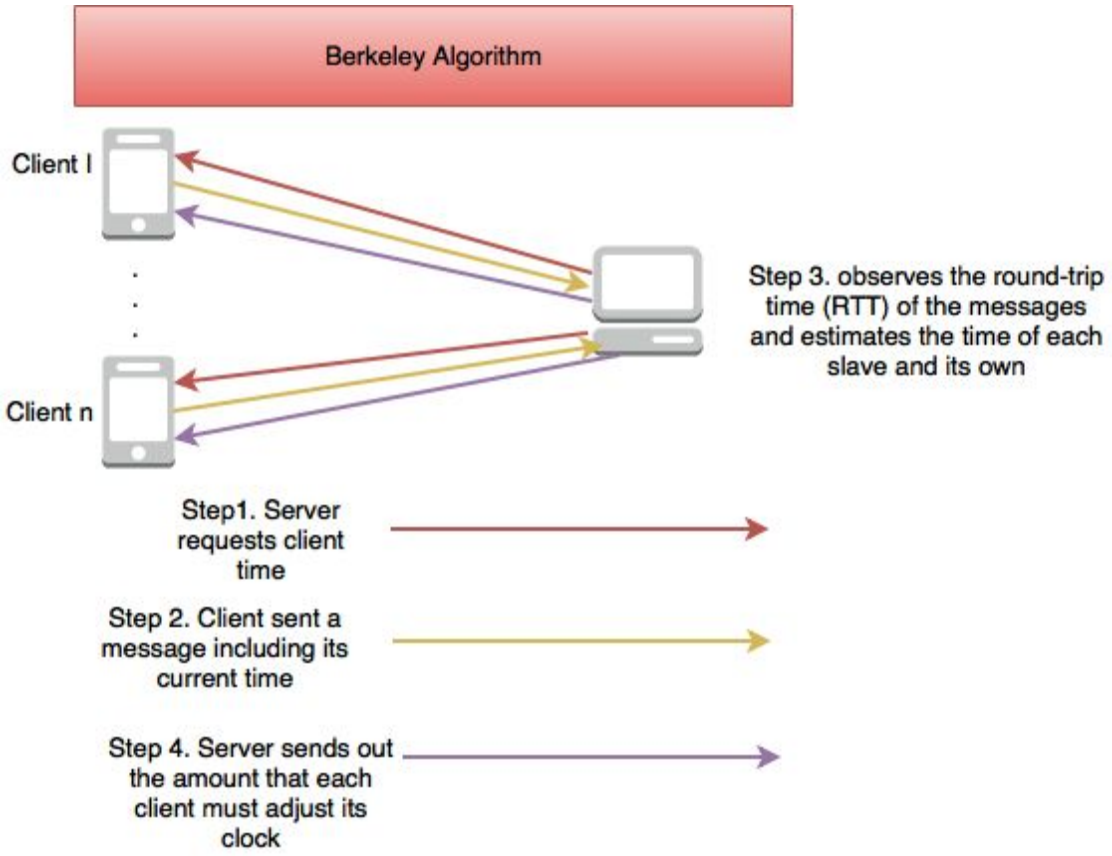

Figure 9: Berkeley Algorithm Implementation Diagram

6. Teamwork

At the beginning of the project, since we all have different schedules, we decided to divide the work so that everyone could work on the project on their own time. As such, Faraz and Yuechuan were tasked with creating the Android app and Alonso and Cheng took charge of building the Node.js server end of the project. Later on, once the servers were ready, Faraz and Yuechuan modified the android to work with the server with some assistance from Cheng and Alonso, as needed. All members contributed to this report equally.

7. Conclusion

Our project was interesting but we were only able to do so much. Given more time and knowledge we would make some improvements on the project, particularly on the synchronization. Some things we would like to improve is to periodically measure RTT and use a exponential smoothing filter to calculate value for actual RTT. Another is to average the client-server time separately from server-client time and to see what difference it makes versus only taking RTT into account. Also, we would take time drift and different clock speeds into account and use NTP for better accuracy. Lastly from the method of Berkeley and Cristian, we can see that since Berkeley gets its time from an average it would work better with more clients so Cristian's algorithm would be overall better for our purpose.

In conclusion, this project helped bring us outside the scope of the course to learn something related to, but not covered during lectures. It was interesting to expand upon things we may not have thought about during the course assignments and material. Overall, while there are certainly some improvements that could be made on our design, we feel that the project went reasonably smoothly and it was very engaging to learn how the different algorithms work, as well as implementing them in practice.

8. References

[1] N. E. Baughman and B. N. Levine, “Cheat-proof playout for centralized and distributed online games,” *Proc. IEEE INFOCOM 2001. Conf. Comput. Commun. Twent. Annu. Jt. Conf. IEEE Comput. Commun. Soc. (Cat. No.01CH37213)*, vol. 1, pp. 104–113, 2001.

[2] N. E. Baughman, M. Liberatore, and B. N. Levine, “Cheat-proof playout for centralized and peer-to-peer gaming,” *IEEE/ACM Trans. Netw.*, vol. 15, no. 1, pp. 1–13, 2007.

[3] B. Di Chen and M. Maheswaran, “A cheat controlled protocol for centralized online multiplayer games,” *Proc. ACM SIGCOMM 2004 Work. NetGames ’04 Netw. Syst. Support games - SIGCOMM 2004 Work.*, p. 139, 2004.

[4] “IDC Spins a Tale of Android vs. iOS Stats in Smartphones Only - Patently Apple.” [Online]. Available: http://www.patentlyapple.com/patently-apple/2013/10/idc-spins-a-tale-of-android-vs-ios-stats-in-smartphones-only.html. [Accessed: 15-Dec-2015].

[5] D. A. C. Varma, P. K. R. M, and P. Gopinath, “Performance Comparison of Physical Clock Synchronization Algorithms,” vol. 3, no. 5, pp. 1355–1364, 2013.

[6] “Realtime Node.js App: A Stress Testing Story - Deployment, Performance, Server Side, Testing, Web Applications - Bocoup.” [Online]. Available: https://bocoup.com/weblog/node-stress-test-analysis. [Accessed: 15-Dec-2015].

[7] “Node.js.” [Online]. Available: https://nodejs.org/en/. [Accessed: 15-Dec-2015].

[8] “Google Trends - Web Search interest: ‘nodejs’ - Worldwide, 2004 - present.” [Online]. Available: https://www.google.com/trends/explore#q=%22nodejs%22. [Accessed: 15-Dec-2015].