화면 분할 스트리밍

최종보고서

201311313 정인원
201411311 장원영
201411318 함형준

건국대학교 컴퓨터공학과

목차

1. 프로젝트 개요

본 프로젝트의 아이디어는 강의 강연, 군대 등등 여러 곳을 가면 큰 여러 개의 스크린 화면으로 하나의 강연이나 영상을 보여주는 장면에서 시작했다. 이를 통해 스마트폰 세 대를 연결해서 동영상을 재생하는 프로그램을 개발하게 되었다. 여러 개의 디스플레이를 컨트롤하기에는 안드로이드 스튜디오에서 개발하는 것이 낫다고 판단해서 개발 환경을 안드로이드 스튜디오로 정했다. 개발 언어는 안드로이드 공식 언어인 Kotlin으로 정했다. 이를 활용하는 것이 UI적인 측면에서도 Java보다 더 용이하게 사용할 수 있어서 Kotlin으로 결정했다.

-Image of Prototype

상기 이미지는 초기에 구상했던 프로토타입 모델이다. 핸드폰 3대에 하나의 영상이 비율에 맞춰 분할되는 모습이다. 이것이 우리 프로젝트의 목표에 가장 적절하다고 생각했다. 프로토타입을 작성하며 구상한 주요 기능의 성공요소를 다음과 같이 정의했다.

- 핸드폰 사이 영상의 동기화
- 비율에 맞춘 동영상 분할

위의 요소를 충족시키는 방향으로 프로젝트를 계획하고 실행에 옮겼다.

2. 요구사항분석

본 프로젝트의 요구사항은 다음과 같다.
<기능적 요구사항>

Ref.	Function	Description
1	기기연결	클라이언트 핸드폰을 서버 핸드폰에 연결
1.1	와이파이 다이렉트 연결	와이파이 다이렉트 방식으로 네트워크 연결
1.2	화면비율 정보 관리	핸드폰 기종 별 디스플레이 할 화면 비율 관리
2	동영상 선택	스트리밍 할 영상을 갤러리에서 선택
2.1	갤러리 접근	동영상을 선택하기 위해 내부저장소에 접근
2.2	동영상 선택	갤러리에서 스트리밍할 동영상 선택
2.3	스트리밍 가능여부 확인	헤더 정보를 확인해 스트리밍 가능여부 확인
2.3.1	헤더정보 변환	스트리밍 불가능한 영상의 경우 헤더를 앞으로 이동하여 파일 재생성
3	동영상 재생	3대의 핸드폰에서 영상 재생
3.1	동영상 파일 선택	갤러리 또는 파일 탐색기에서 재생할 동영상 선택
3.2	파일 패킷 전송	서버 핸드폰에서 선택된 동영상 파일을 패킷 단위로 클라이언트에 전송
3.3	수신확인 신호 전송	클라이언트에서 패킷을 받은 후 서버로 수신확인 신호 전송
3.4	영상 재생 신호 전송	서버에서 수신확인을 접수 후 클라이언트에 재생 신호 전송
3.5	화면 분할	재생해야 할 화면 비율 정보 확인
3.6	영상 재생	비율 정보를 기준으로 필요한 화면만 재생
3.7	10초 앞으로 빨리감기	영상을 10초 앞으로 빨리감기
3.8	10초 뒤로 되감기	영상을 10초 뒤로 되감기
4	동영상 일시정지	3대의 핸드폰 모두 영상 재생 일시정지
4.1	영상 정지 신호 전송	서버 핸드폰에서 클라이언트로 영상 정지 신호를 전송
4.2	수신확인 신호 전송	클라이언트에서 수신확인 신호 전송 후 클라이언트 영상 정지
4.3	영상 정지	서버에서 수신확인을 접수 후 영상 정지
5	볼륨조절	영상의 볼륨 조절(영상의 소리는 서버 핸드폰에서만 발생)
5.1	볼륨 높이기	서버 핸드폰의 볼륨 높이기
5.2	볼륨 낮추기	서버 핸드폰의 볼륨 낮추기

<비기능적 요구사항>

1.영상 재생 싱크를 맞추어야 한다.
2.스마트폰 기기간 화면비율을 맞추어야 한다.

-기능적 요구사항의 경우 선택된 파일의 헤더를 정확히 확인하고, 변환할 경우
변환된 파일이 정상적으로 실행이 되게 하는 것이 중요하다.
-비기능적 요구사항의 경우 핸드폰 3대에서 실행되는 영상이 핸드폰 기기별로 정보를
정확히 받아와 화면에 맞게 분할되어야 하며, 재생하는 동안 최대한 같은 프레임의 영상을 재생

화면 분할 스트리밍

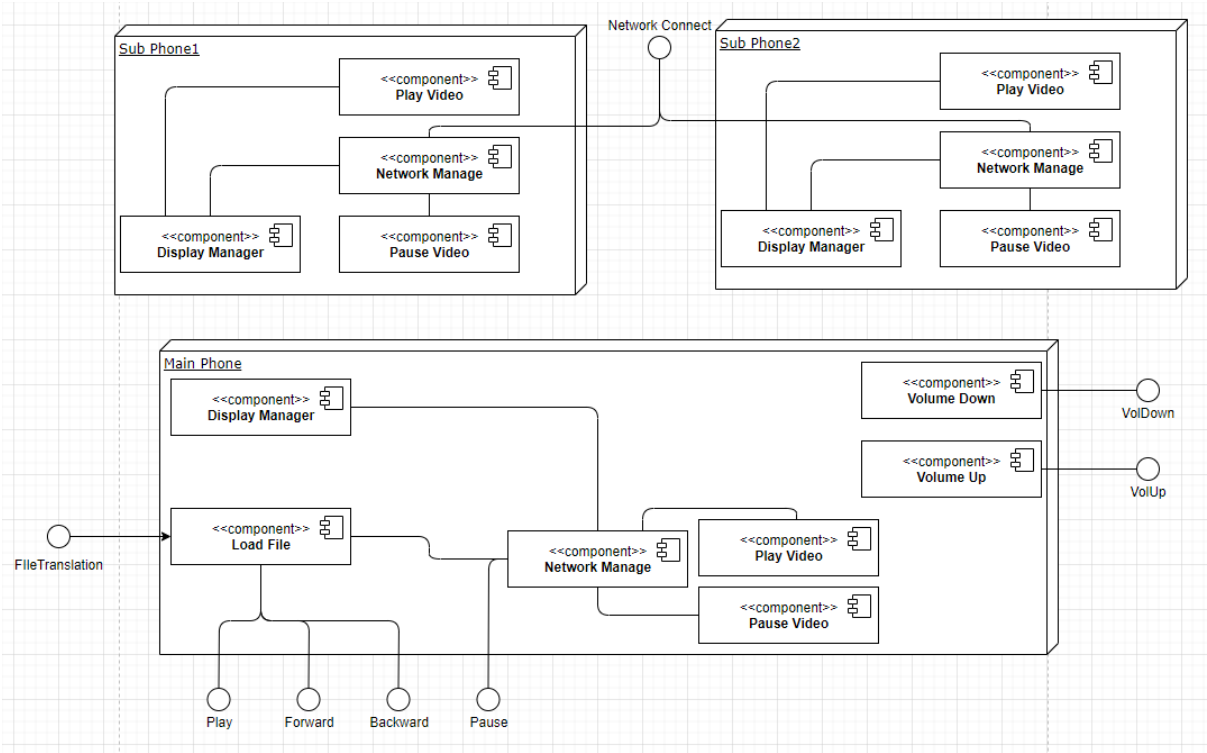
하는 것이 중요하다.

<System Test Case>

Ref.	Description
1.1	3대의 기기가 Wi-Fi Direct로 연결되는지 확인
1.2	기기들의 모델을 통해 핸드폰의 가로 길이 확인
2.1	갤러리가 열리는지 확인
2.2	선택된 영상의 경로를 출력
2.3	헤더 부분을 출력
2.3.1	생성된 파일의 헤더 정보를 출력
3.1	서버 기기에서 전송 테스트 메시지가 출력되는지 확인
3.2	각 기기에서 수신 테스트 메시지가 출력되는지 확인
3.3	각 클라이언트 기기에서 영상 재생 신호를 수신했는지 시스템 메시지 출력
3.4	화면이 각 기기에 맞춰 분할이 되는지 확인
3.5	영상이 매끄럽게 재생되는지 확인
3.6	3대의 기기 모두 10초 뒤 영상이 재생되는지 확인
3.7	3대의 기기 모두 10초 전 영상이 재생되는지 확인
4.1	영상 정지 신호 수신 후 클라이언트 기기에서 시스템 메시지 출력
4.2	수신확인 신호 전송 후 서버 기기에서 시스템 메시지 출력
4.3	3대 기기 모두 영상이 정지되는지 확인
5.1	볼륨 바가 나타나고 소리가 커지는지 확인
5.2	볼륨 바가 나타나고 소리가 작아지는지 확인

3. 디자인

<Image of Architecture Diagram>

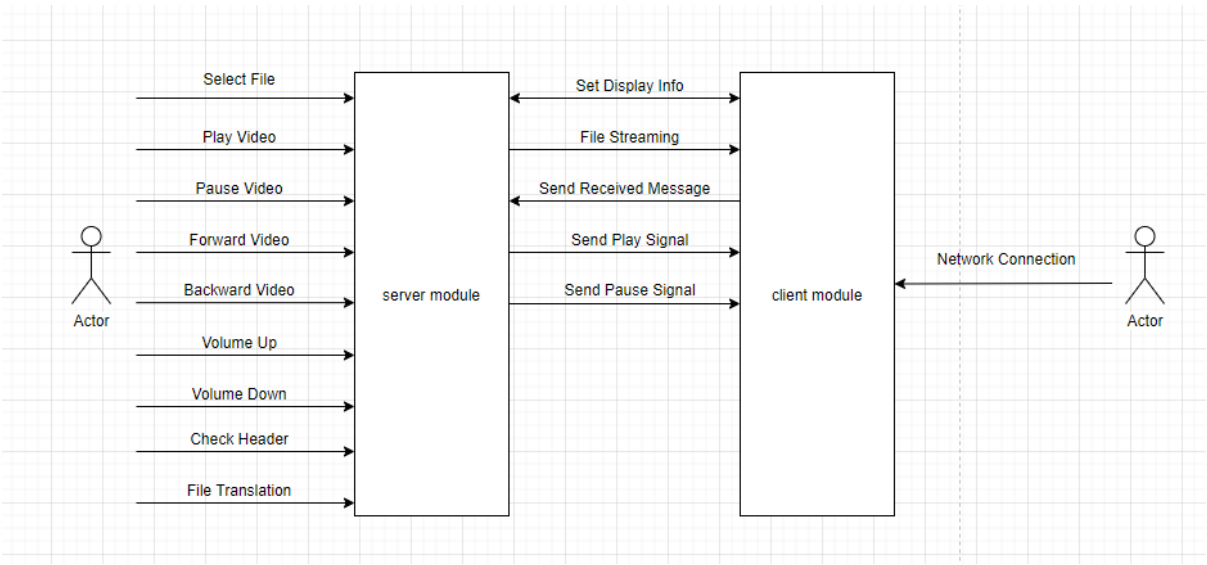


-이 앱은 서버 핸드폰, 클라이언트 핸드폰 두 개의 모듈을 가진다.

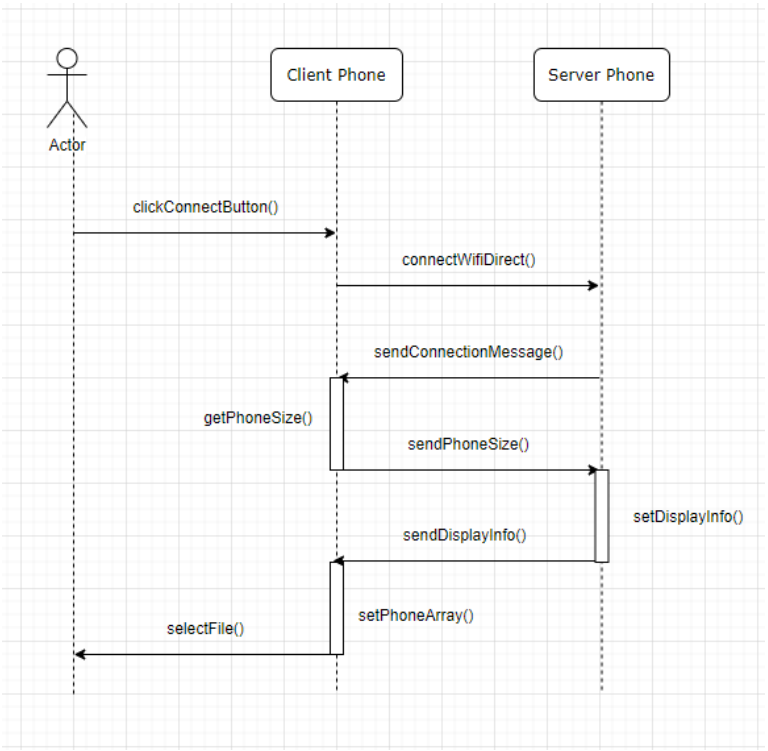
-서버 핸드폰에서는 화면분할 정보 계산, 동영상 파일 선택, 헤더 정보 확인, 파일 변환, 파일 전송, 정보 전송, 영상 재생, 볼륨 조절 등의 기능을 수행한다.

-클라이언트 핸드폰에서는 서버 핸드폰에서 전송해주는 정보들을 받아들여 영상을 재생해준다.

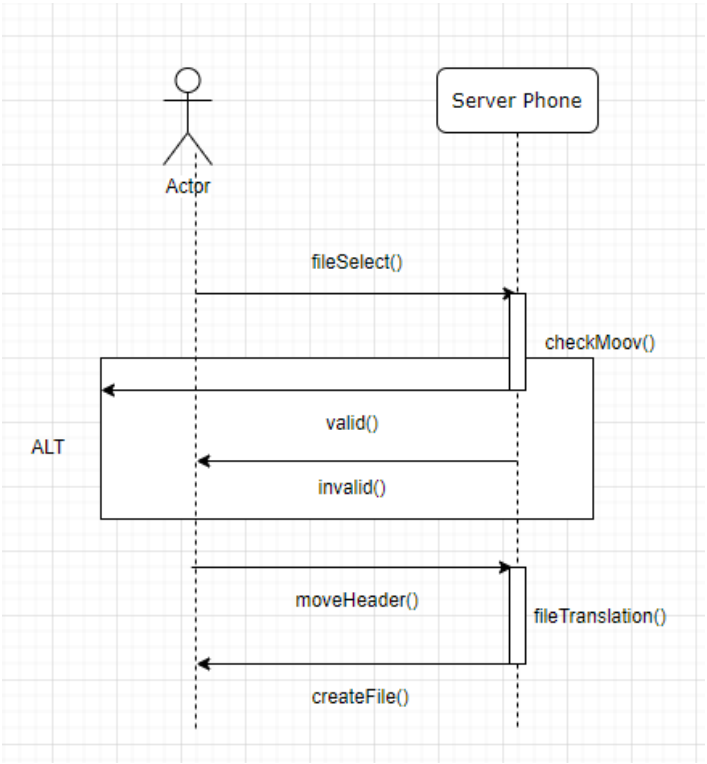
<Image of Interfaces>



<Image of Sequence Diagram>

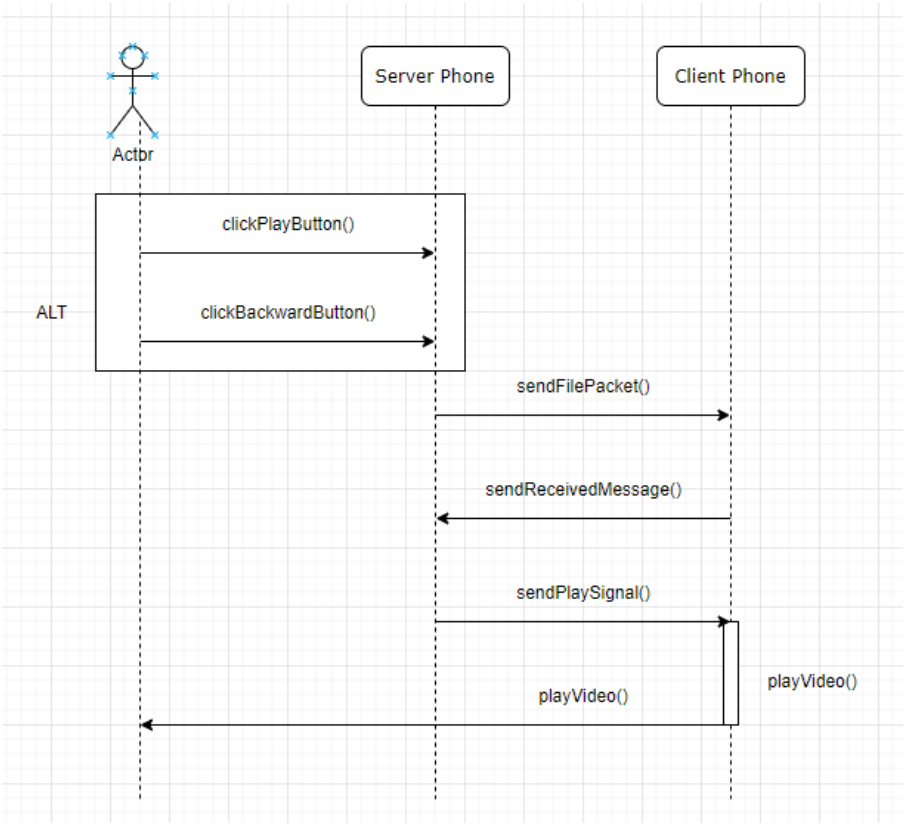


- Network Connection

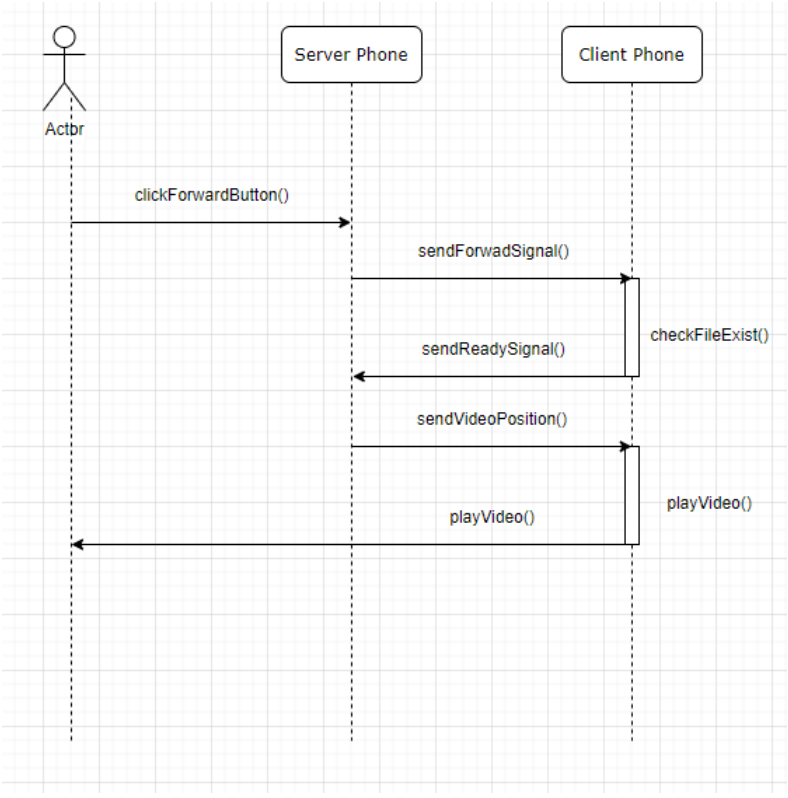


- Check Header

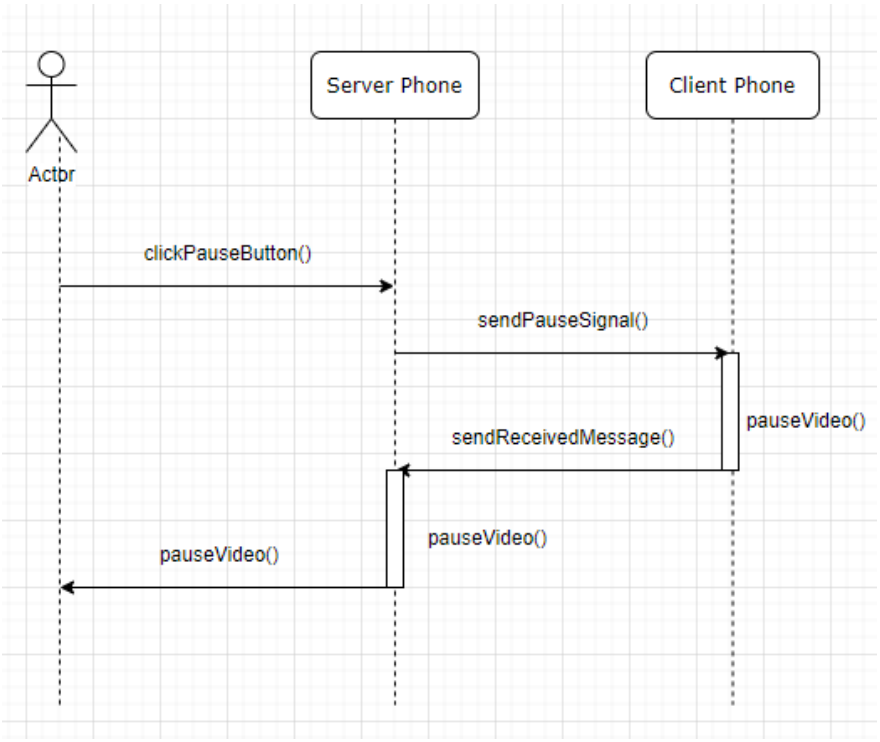
화면 분할 스트리밍



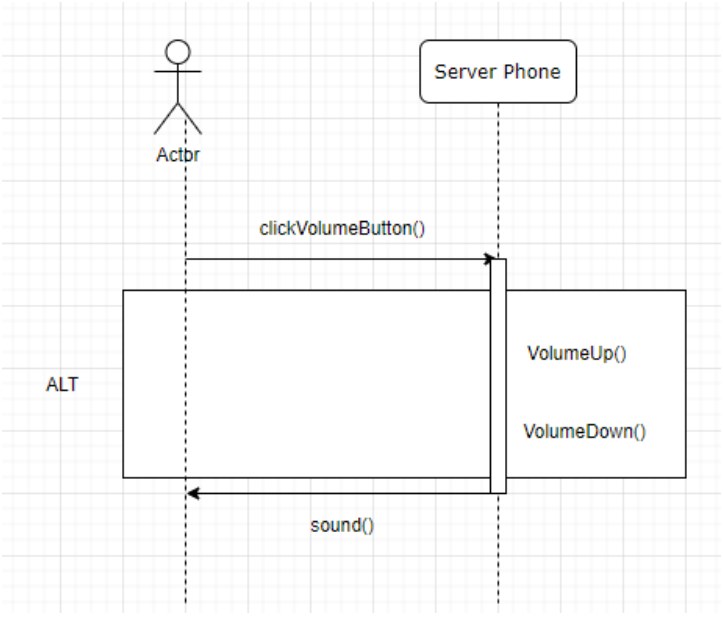
- Playing Video



-Forward Video

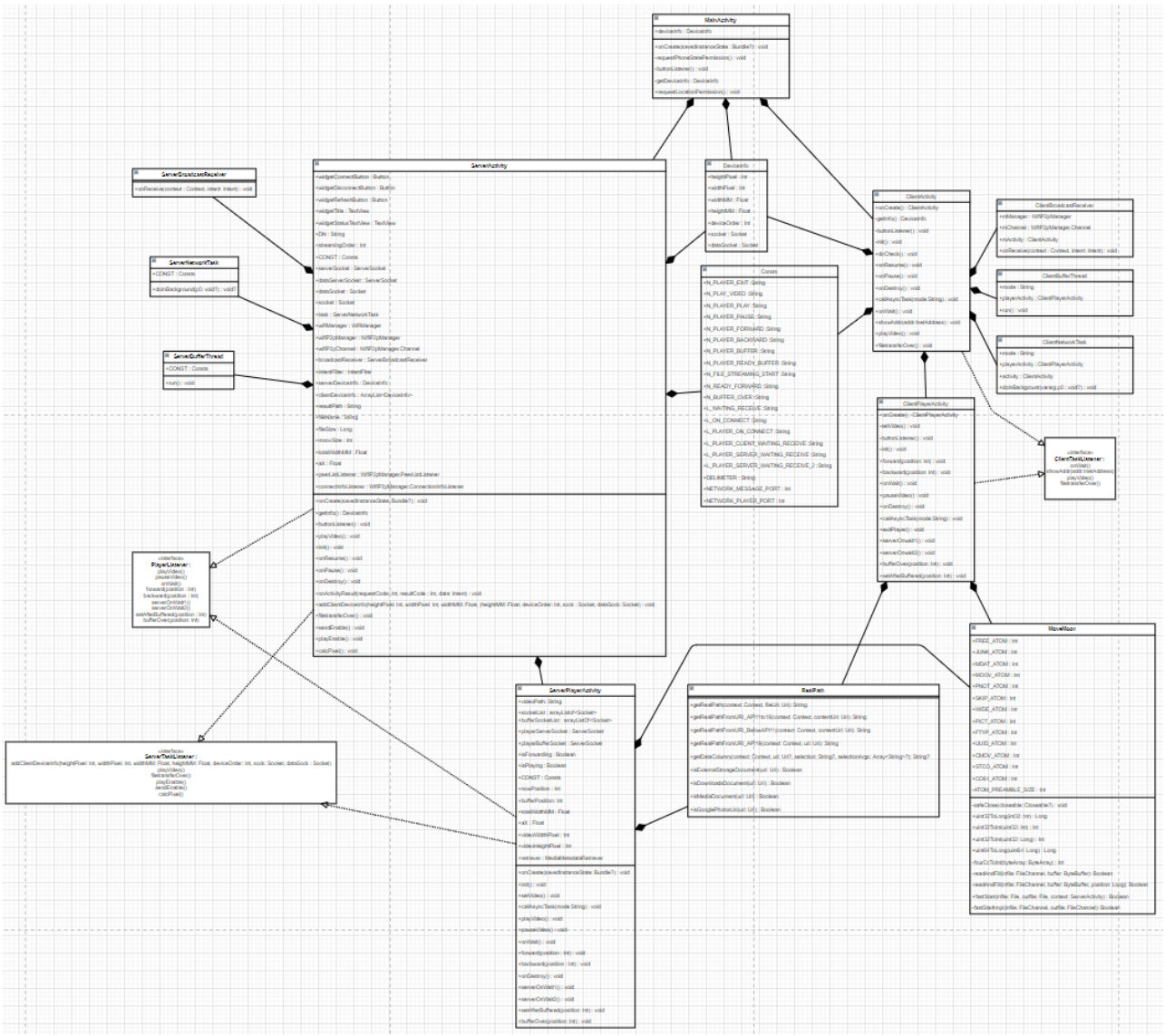


-Pause Video

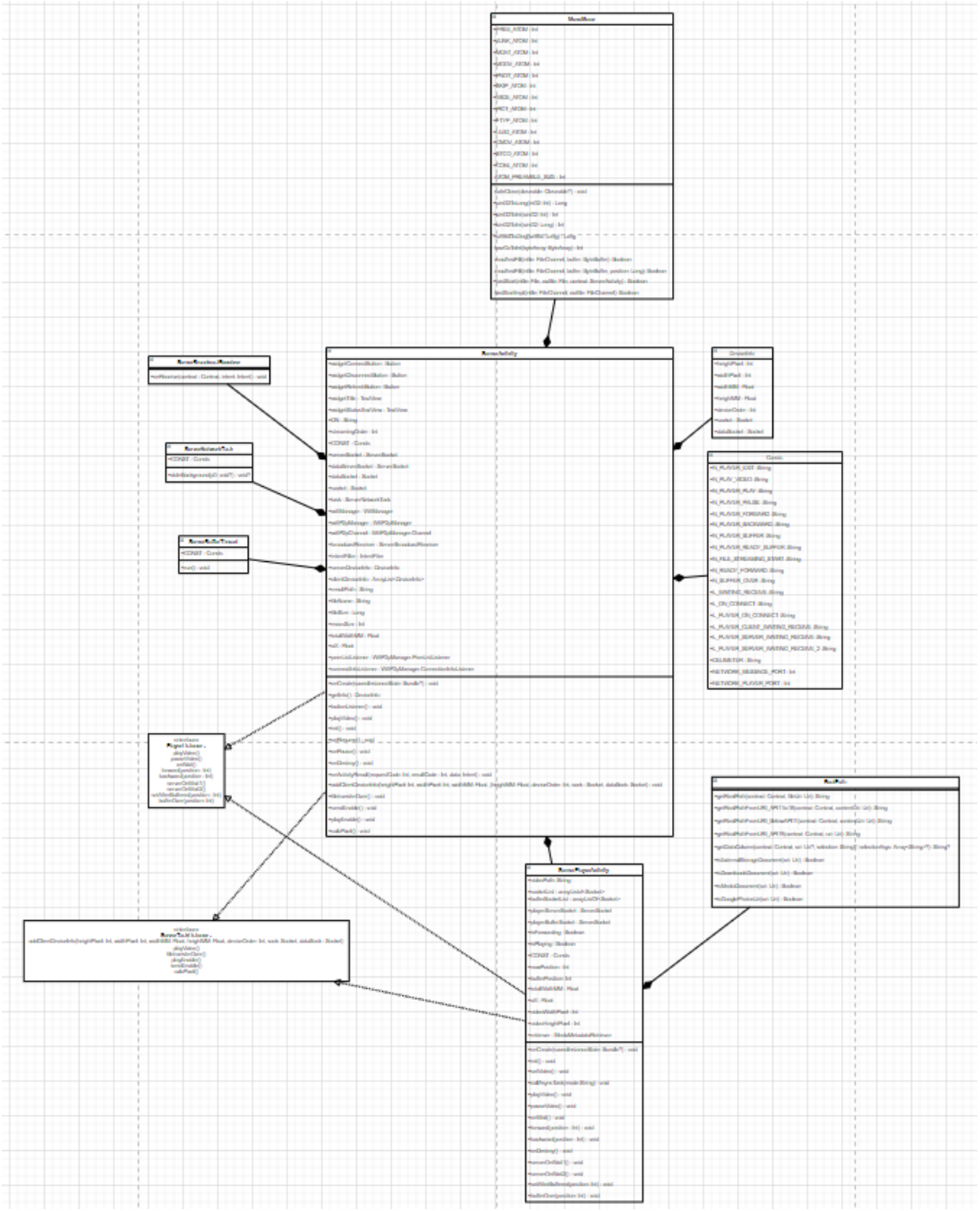


- Volume

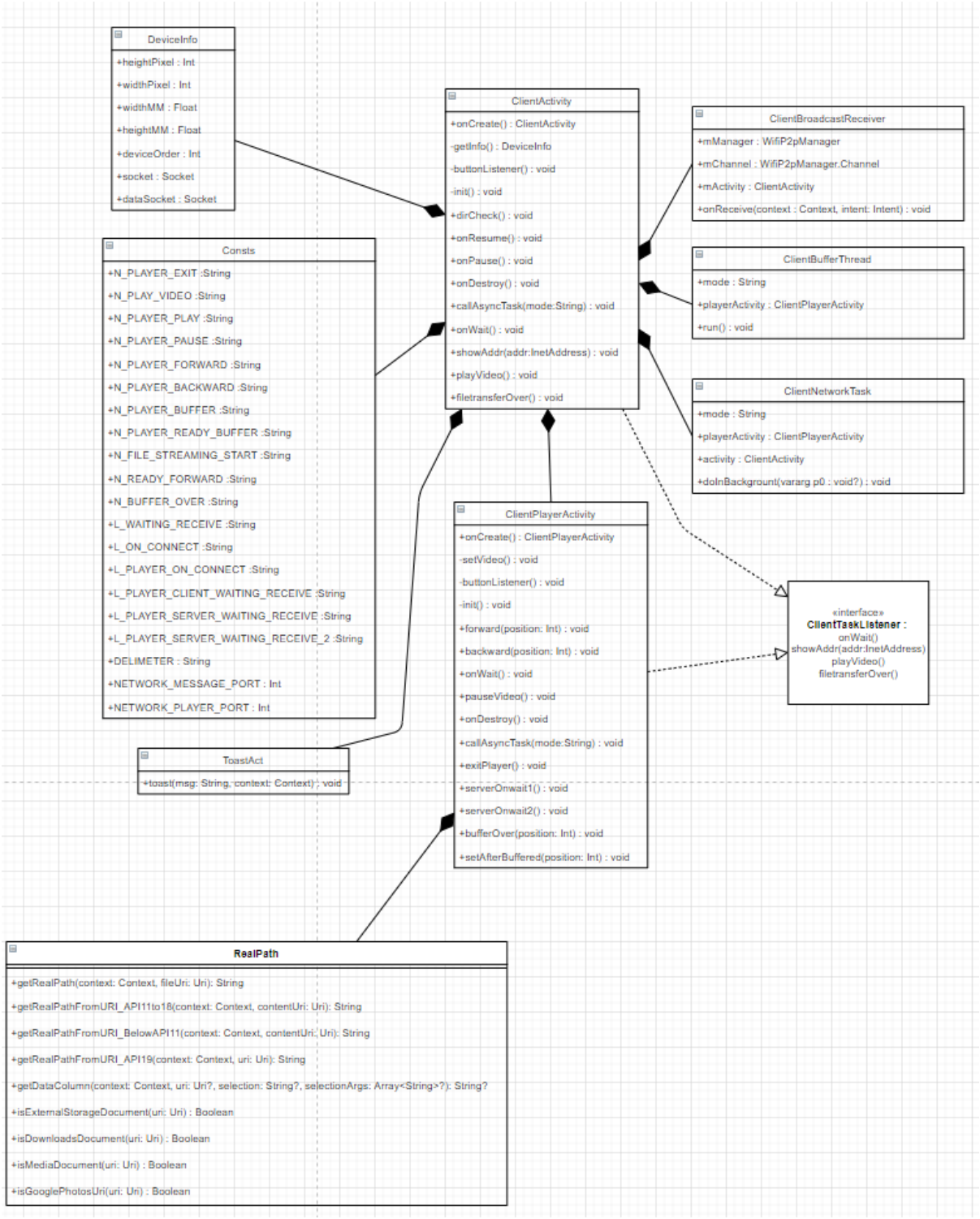
<Image of Class Diagram>



-Class Diagram



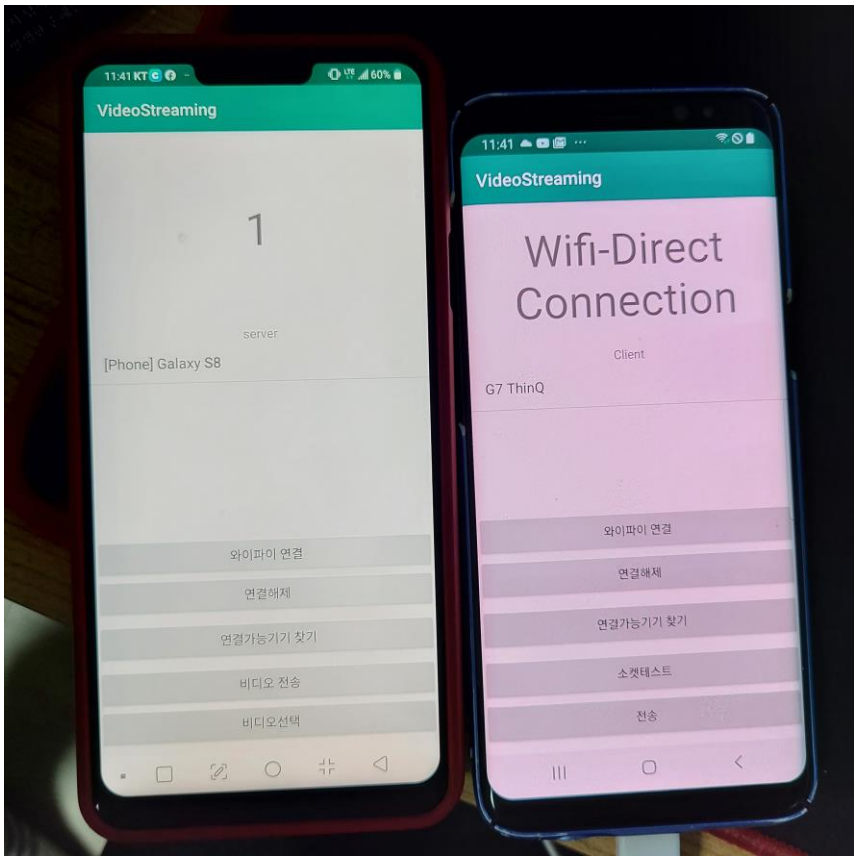
- Server Class Diagram



-Client Class Diagram

4.개발과정

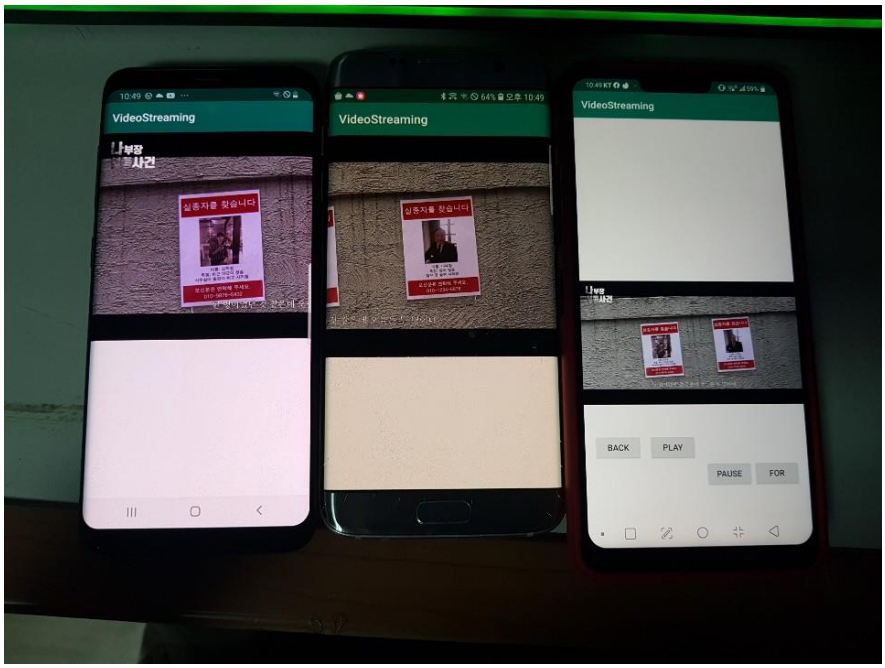
1차 구현 당시 와이파이 다이렉트를 이용한 핸드폰의 네트워크 연결에 초점을 두어 개발을 진행했다. 또, 연결 후 호스트(서버)의 네트워크 정보를 받아와 소켓을 연결하는 것 까지 구현을 완료했다. 영상 제어를 위해선 소켓 연결을 유지해야 했는데, 액티비티 전환에 따라 소켓 연결을 이어갈 수 없었다. 이 부분에서 어려움을 느끼고 2차 구현까지 대부분의 시간을 소켓 문제를 해결하는 데 사용했다. 다음은 당시 구현한 산출물의 이미지이다.



-1차 구현물 (~5.18)

2차 Iteration 직전 위 문제를 해결하고 먼저 다운로드를 통한 영상 재생을 목표로 하여 구현을 이어 나갔다. 당시 약간의 화면 분할도 구현을 했지만 특정 기기에 한하여 일부 화면만 분할이 가능한 정도였다. 완성된 기능으로는 파일 전송 및 수신, 영상 재생 및 제어가 있었다. 서버 핸드폰에서 파일을 선택하여 양쪽 클라이언트 핸드폰에 전송을 시도하면 클라이언트 핸드폰은 본 앱 전용 폴더를 생성하고 파일을 생성했다. 서버에서 영상 재생을 시도하면 해당 폴더에 접근하여 영상을 재생했다. 후에는 프로토콜에 따라 영상을 제어했다.
다음은 2차 구현 당시 산출물의 이미지이다.

화면 분할 스트리밍



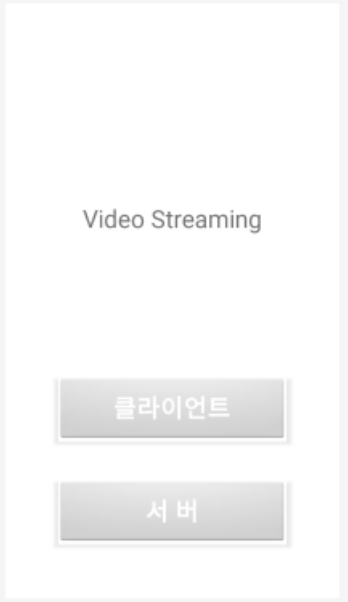
-2차 구현물(~6.4)

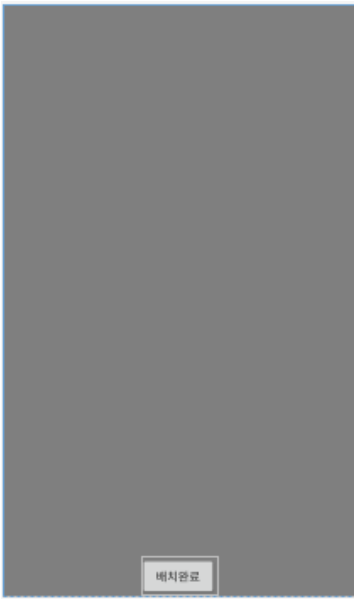
마지막으로 최종적으로 구현한 것의 이미지이다.(기능에 관한 부분은 후술)

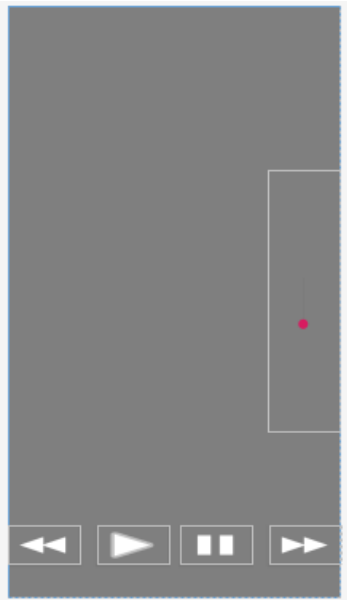
-최종 구현물(~6.18)

5.구현방법

<Image of UI Placement>







<Activity별 기능>

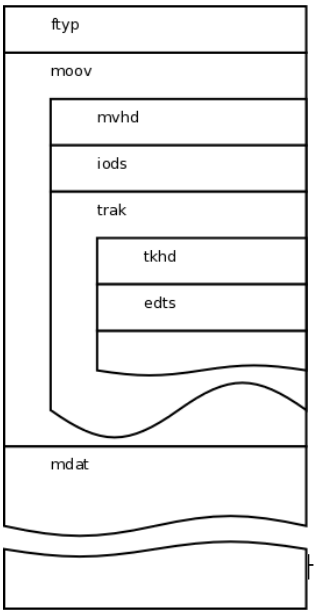
1) Main Activity
Main Activity에서는 대부분 와이파이 다이렉트, 파일 읽기, 쓰기 등의 기능을 실행하기 위한 Permission을 얻는 기능으로 이루어져 있다.
또한 화면 비율 정보를 계산하기 위해 핸드폰의 가로 세로 픽셀과 픽셀을 통해
화면의 실제 mm 길이를 계산하여 데이터 클래스에 저장한다.

2) Server Activity
Server Activity에서는 스트리밍을 위한 정보들을 계산하고, 파일 변환, 전송 등의
주요 기능들을 수행한다.

-Wifi Direct Connection
연결가능기기 찾기 버튼을 통해 client 핸드폰의 연결을 대기한다. 연결이 완료되면
Wifi Direct의 Host가 되며 Connection Status 텍스트 필드에 server가 입력된다.
이후 소켓 연결 버튼을 클릭하면 Async Task를 통해 Server Socket을 생성하며 Client 핸드폰들의 소켓 연결을 대기한다.

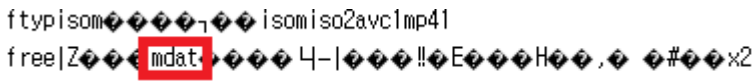
-Video Select
비디오 선택 버튼을 통해 핸드폰의 내부저장소에 접근하여 갤러리를 열어준다.
갤러리에서 파일을 선택하면 Server Activity에 선택한 파일의 경로를 결과로
전송한다. 이 경로는 resultPath 변수에 저장되며 후에 파일을 전송할 때 사용된다.

-Check Header



Video Select를 통해 받아온 경로를 이용해 파일의 헤더 구조를 파악한다.
대부분의 경우 moov 박스가 mdat 박스 앞에 존재한다. 하지만 영화나 드라마 같은 용량이 큰 파일은 mdat 박스 뒤에 moov 박스가 존재하는 경우가 종종 있다.
온전한 파일의 경우 헤더가 파일의 뒷부분에 존재하더라도 스캔을 통해 파일 헤더를 먼저 파악하지만, Progressive Download 스트리밍의 경우 헤더가 뒷부분에 있으면 미디어 데이터만 존재하고 헤더가 없는 상황이기 때문에 동영상을 재생할 수 없다. 따라서 Mpeg-4 파일의 헤더 구조를 파악한 후 moov 박스의 존재 여부를 확인한다. moov 박스가 등장하기 전에 mdat 박스(미디어 데이터 박스)가 등장한다면 영상을 전송하는 동안 재생을 할 수 없으므로 사용자에게 이를 알린다.
또한, 사용자의 필요에 따라 선택된 파일을 스트리밍 용으로 생성할 것인지 다른 영상을 선택할지 알린다. 파일을 스트리밍 용으로 생성할 경우 File Translation을 수행한다.

-File Translation
File Translation은 Check Header 이후 사용자가 파일을 스트리밍 용으로 생성하겠다고 요청할 경우 실행된다.



위 사진은 mdat 박스가 앞에 있는 경우이다. 앞에 존재하는 바이트들을 일정 크기만큼 가져와 String으로 나타냈다. 이 경우에는 mdat를 전부 다운로드 받아야 비로소 헤더를 다운로드 받을 수 있다. 게다가 mdat의 크기는 헤더의 크기에 비해 매우 크기 때문에 헤더가 없으면 사실상 스트리밍을 할 수 없다. 많은 스트리밍 동영상 플레이어들 역시 헤더가 뒤에 있는 파일은 지원하지 않는 추세이다.
따라서 mdat 박스가 앞에 있는 파일은 사용자의 요청에 따라 mdat박스와 moov박스의 위치를 바꿔 재생성 한다.

위 사진은 동일 영상 파일로 moov박스와 mdat박스의 위치를 바꾼 후 일정 바이트 크기 만큼을 String으로 나타낸 것이다. 보이는 바와 같이 위의 mdat박스가 아닌 moov박스가 나타났다.
이제 동영상 플레이어에서 이 moov박스, 헤더 정보를 파악한 뒤 미디어 데이터가 존재하는 곳 까지 영상을 재생할 수 있다.

-Display Info
소켓 연결 대기 후 Client에서 연결 요청을 통해 연결이 된다. 연결이 완료되면 각 핸드폰의 가로 세로 픽셀, 화면의 mm 길이를 받아와 데이터 클래스에 저장하고List에 add 한다. 연결이 성사된 순서대로 deviceOrder 변수에 값을 매기고 마찬가지로 데이터 클래스에 저장한다.
Server Activity에서는 이 중, mm 길이를 합하는 과정을 수행한다. 이후 Player Activity로 넘어가는 과정에서 Client 핸드폰에 합해진 mm 길이를 전송한다. 또, deviceOrder에 따라 앞선 기기의 가로 길이를 전송해준다. deviceOrder가 첫 번째라면 0을 두 번째라면 첫 번째 기기의 가로 길이(mm)를 전송해준다.

-File Transfer
소켓을 연결하고 동영상 파일까지 선택을 완료했다면 파일 전송을 시작한다.
파일 전송은 Async Task를 통한 소켓 통신으로 두 Client 핸드폰에 동시적으로 파일을 전송한다. 파일을 전송하기 전 파일 헤더의 크기와 전체 파일 크기를 전송한다. Client 핸드폰에서는 헤더의 크기를 미리 파악하고, 그 크기만큼의 패킷을 수신하면 동영상 플레이어에서 동영상 재생을 준비한다.

3) Client Activity
Client Activity는 Server Activity에서 전송해주는 신호에 따라 플레이어를 재생하거나 전체 화면 mm를 저장한다.

-Wi-fi Direct Connection
연결가능기기 찾기 버튼을 클릭하여 현재 와이파이 다이렉트를 사용중인 핸드폰을
검색한다. 원하는 핸드폰을 발견하여 클릭할 경우 peer의 권한으로 와이파이 다이렉트에 연결된다. 또 호스트의 Inet Address를 받아와 변수에 저장한 뒤 후에 소켓 연결에 사용한다.

-File Transfer
소켓을 통해 서버에서 파일을 전송 받는다. 동영상 플레이어에서 실행할 동영상을
쉽게 찾을 수 있도록 디렉토리를 생성한 뒤 그 디렉토리에 파일을 생성한다.

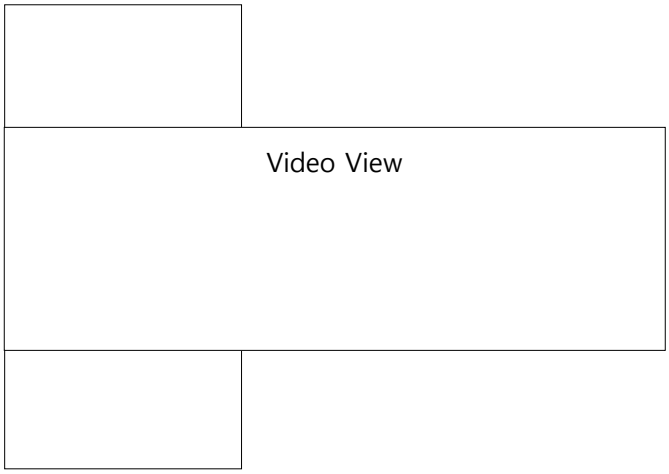
-Display Info
두 클라이언트 모두 소켓이 연결된 경우, 서버에서는 3대 핸드폰의 화면 mm를 합하여 전송해준다. 이렇게 전송 받은 길이는 변수에 저장되며 동영상 플레이어가 실행될 때 값을 전달해준다.

4) Player Activity

Server Activity에서 모든 준비를 마치고 동시 재생 버튼을 누르면 Player Activity가 실행된다. 이 Activity에서 화면 분할, 영상 제어에 따른 싱크 조절이 이루어진다.
-Network Connection
Activity가 실행되면 가장 먼저 Server Socket을 생성하여 Client들의 소켓 연결을
기다린다. 소켓 연결이 완료되기 전에는 컨트롤러들이 작동하지 않도록 설정한다.
연결이 되지 않은 경우에 컨트롤러를 작동하는 경우, 서버 핸드폰에서만 영상이 재생된다. 두 개의 클라이언트 모두 연결이 완료되면 비로소 컨트롤러가 작동하도록 한다.

-Split Display
화면분할을 위해선 다음과 같은 정보들이 필요하다.
-각 핸드폰의 해상도 (가로x세로 pixel)
-각 핸드폰의 dpi(가로, 세로)
-각 핸드폰의 가로 길이(mm)
-3대의 핸드폰의 가로 길이의 총합(mm)
-재생할 영상의 해상도
-핸드폰이 배치될 순서
위 정보들을 변수에 저장한 뒤, 동영상을 재생하는 Video View 위젯을 배치한다.
이 정보들을 토대로 Video View의 크기를 조절하고 위치를 옮겨 원하는 부분 만큼을 화면에 표시하는 방법을 사용했다.

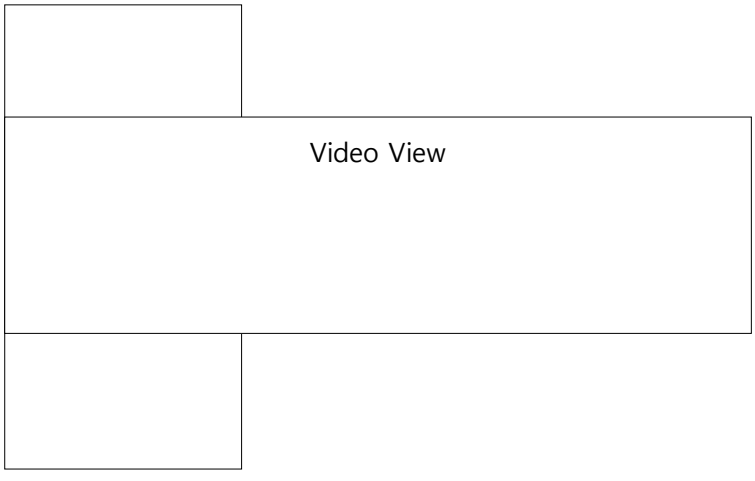
전체 크기만큼 Video View의 가로 길이 설정
-먼저 Video View의 크기를 원하는 만큼 늘려준다. 앞서 Server Activity에서 계산한 가로 길이의 총합을 파악한다. 이 값을 이용해 Video View의 가로 길이를 늘려줘야 하지만,
이 위젯의 레이아웃을 위한 인자는 pixel 값을 받아 오기 때문에 mm를 이용해서는
원하는 크기로 늘려줄 수 없다. 따라서 이 mm를 pixel로 변환해야 한다.
변환을 위해서 앞서 준비한 가로 dpi값을 이용한다. 기종별로 dpi값이 다르므로
이 변환은 서버에서 일괄적으로 수행하지 않고 핸드폰마다 영상이 재생되기 전에
각각 수행한다.
dpi는 1인치당 픽셀의 개수를 의미한다. 만약 Video View의 가로 길이를 10인치로 설정하고 싶다면 10*dpi 값을 레이아웃 인자로 전달해주면 된다.
하지만 우리는 inch가 아닌 mm를 사용했기 때문에 1인치당 픽셀의 개수가 아닌 1mm당 픽셀의 개수를 구해야 한다. 안드로이드에서 기본적으로 1mm당 픽셀의 개수를 제공하지 않기 때문에 dpi를 먼저 구해서 바꿔줘야 한다. 1인치는 약 25.4mm이다.
따라서 dpi/25.4 = 1mm당 픽셀의 개수가 된다. 이제 Video View 크기 인자에 구한 값
(가로길이 총합*dpi/25.4)을 대입해주면 된다. 그러면 핸드폰 내부의 Video View 크기는
다음과 같이 변한다.



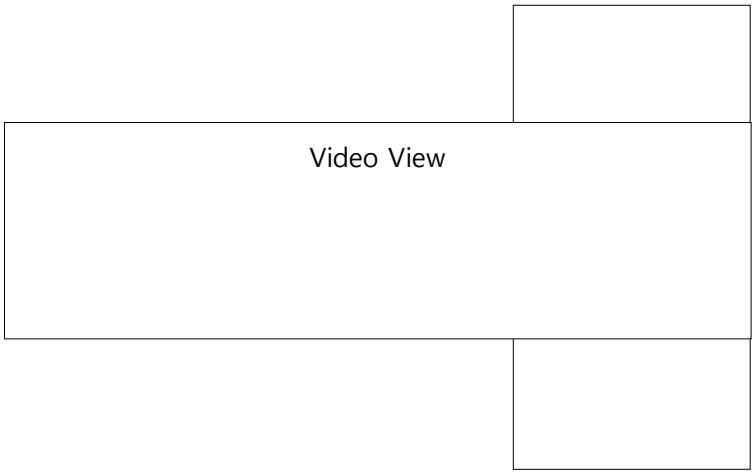
한 마디로 플레이어의 크기가 핸드폰의 크기를 넘기면서 특정 영역만을 재생하게 하는 것이다.

2. 동영상의 해상도에 맞춰 Video View의 세로 길이 설정
가로 길이를 설정했다면 세로 길이 역시 동영상의 해상도에 맞춰 설정해줘야 한다.
기본값으로 설정할 경우 동영상이 재생될 때 가로 길이도 기본값으로 재생될 수 있다.
세로 길이를 설정하기 위해선 가장 먼저 동영상의 해상도를 알아야 한다. 동영상의 해상도는 MediaMetadataRetriever를 사용해 알아냈다. 객체를 선언한 뒤 파일의 경로를 지정해주면 동영상 파일의 metadata를 추출할 수 있다.
이렇게 구한 동영상의 해상도는 Video View의 가로 세로 비율을 맞춰 세로 길이 설정에 사용한다. 예를 들어, 선택된 동영상의 해상도가 1920*1080라면, 가로와 세로의 비율이 16:9가 된다. 만약 핸드폰의 가로 길이 총합이 160mm라면 Video View의 세로 길이는 90mm로 설정하면 된다.
한 마디로, (가로 길이 총합*세로 해상도/가로 해상도)*세로dpi/25.4가 Video View의 세로 길이가 되는 것이다.

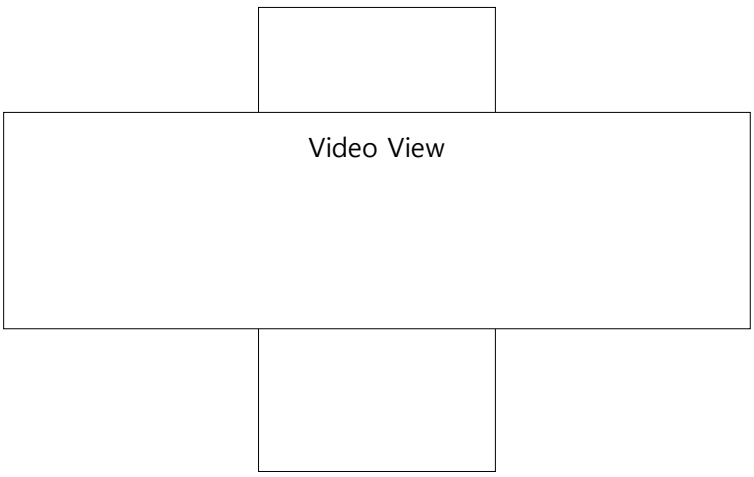
3. 순서에 맞게 특정 영역으로 이동
해상도에 맞게 Video View의 크기를 설정했다면 Video View는 다음과 같이 배치된다.



이 배치는 클라이언트 핸드폰 중 가장 왼쪽에 배치되는 핸드폰이 가져야 할 Video View의 위치이다. 서버 핸드폰은 오른쪽에 배치되어야 하기 때문에 다음과 같은 Video View의 배치를 요구한다.



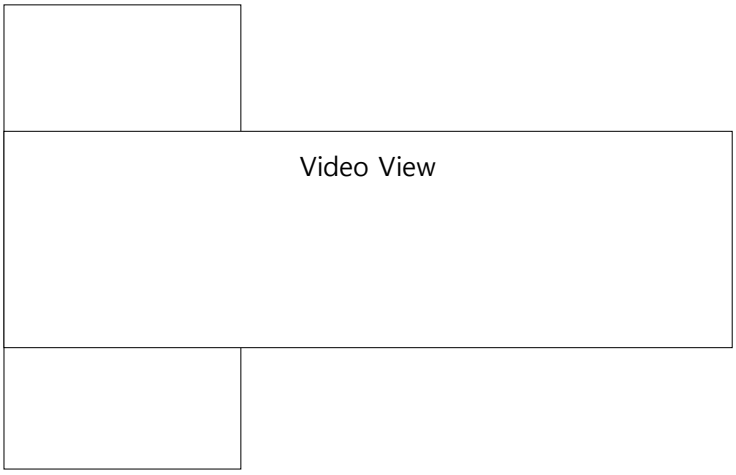
따라서 Video View를 –X 방향으로 보내야 한다. 크기는 핸드폰의 가로 길이 총합이므로 클라이언트 핸드폰 두 대의 가로 길이만큼 이동하면 된다.
Widget의 위치를 변경하는 인자로 x라는 인자가 존재한다. Default 값은 0이다,
x에 옮길 만큼의 x값을 대입하면 된다. 이때, 필요한 x값을 구하는 방법은 다음과 같다.



x = -(전체 가로mm – 서버 핸드폰의 가로mm)*가로 dpi/25.4

여기까지 서버 핸드폰의 동영상 플레이어가 재생할 화면을 설정하는 부분이다.
클라이언트 핸드폰에서 Video View의 크기와 위치를 맞추는 것은 Server Activity에서
전송 받았던 deviceOrder 변수에 따라 위치만 맞춰주면 완성이다.

4. 클라이언트의 Video View 위치 설정
앞서 Server 핸드폰이 전체 가로 길이에서 앞선 두 기기의 길이만큼 위치를 옮겼듯이,
클라이언트도 마찬가지로 앞선 기기의 길이만큼 위치를 옮기면 된다.
Server Activity에서 연결된 순서대로 deviceOrder변수를 설정하고 클라이언트에게 그 변수 값을 전송했다. 또한, 앞선 기기의 가로 길이 역시 전송했는데, 클라이언트에서 Player로 넘어가며 이 값들을 넘겨준다. 값을 정확히 받았다면 클라이언트 Player는 현재 기기의
순서와, 앞선 기기의 가로 길이를 변수에 저장해 둔다. 이제 Server와 마찬가지로 x의 값을 계산하여 위치 인자에 대입해야 한다. 먼저 해당 기기의 배치 순서가 가장 첫 번째라면 앞선 기기는 존재하지 않을 것이므로 앞선 기기의 가로 길이는 0이 된다. 따라서 첫 번째 기기는 생성된 동영상 플레이어를 그대로 재생하면 되는 것이다.



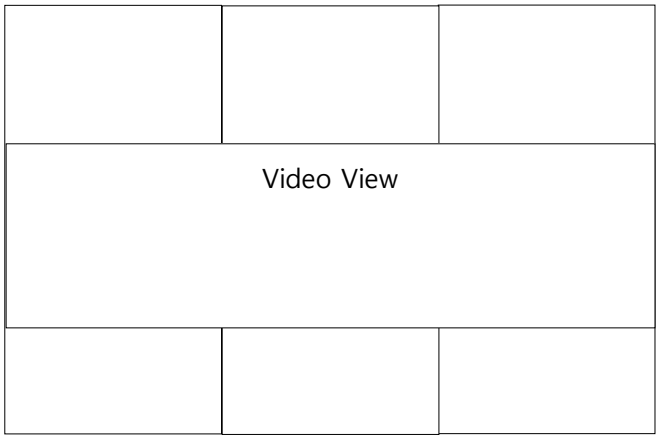
다음은 해당 기기의 배치 순서가 두 번째일 때이다. 두 번째 기기는 앞선 기기의 가로 길이가 0이 아닌 상태이다. 그렇다면 해당 기기는 앞선 기기의 가로 길이만큼 Video View를 x축으로 이동해야 할 것이다. 이 때 이동할 x거리는 다음과 같다.

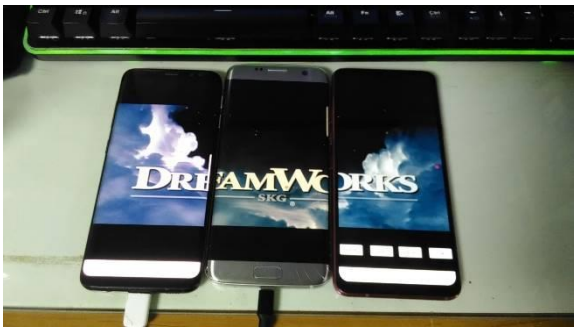
x = -(앞선 핸드폰의 가로mm)*가로 dpi/25.4

위치 인자 x에 위의 값을 대입하면 Video View가 다음과 같이 배치될 것이다.

위 과정들을 모두 마쳤을 경우 기기들은 앞선 기기에 따라 알맞게 화면을 이동했을 것이고 다음과 같이 영상이 모두 분할된다.

화면 분할 스트리밍





플레이어에서 일어난 과정을 정리하면 다음과 같다.

Player Activity 입장
↓
동영상 파일 헤더 확인
↓
헤더 다운로드가 안된 경우 다시 시도
↓
동영상 해상도 확인
↓
받아온 정보에 맞게 화면 분할
↓
순서에 맞게 핸드폰 배치
↓
영상 제어 전용 소켓 연결

이렇게 하여 영상을 재생하기 위한 준비가 완료된 상태이다. 이제 동영상을 재생하면 된다.

-Playing Video
화면분할까지 준비가 완료된 상태라면 Video View에 재생할 동영상의 경로를 삽입한다.
경로를 읽으면 Video View는 현재까지 다운로드 된 미디어 데이터를 재생할 수 있게 된다.
예를 들어, 100Mb의 영상 파일이 있을 때, 10Mb의 파일을 다운로드한 상태이다. 그리고 앞선 과정을 모두 수행하고 동영상의 경로를 지정해주었을 때, Video View가 재생할 수 있는 동영상의 길이는 10Mb만큼이라는 의미이다.
비록 10Mb 길이를 재생했을 때 다운로드 된 파일 크기가 20Mb라 하더라도, Video View는 10Mb 길이의 미디어 데이터만을 읽었기 때문에 그 이상을 재생할 수 없다는 것이다. 때문에, 서버 핸드폰에는 온전한 파일을 재생하여 문제가 되지 않지만, 클라이언트 핸드폰에서는 이때마다 비디오 경로를 재설정해야 한다.

-Synchronize
파일을 다운로드하며 재생을 하는 스트리밍 방식을 선택함으로써, 화면 동기화를 위해 고려해야 할 부분이 몇 가지 있다.

1.Video View가 읽은 미디어 데이터의 길이에 도달했을 때
2. 10초 앞이 다운로드 된 파일 시간 크기보다 큰 경우

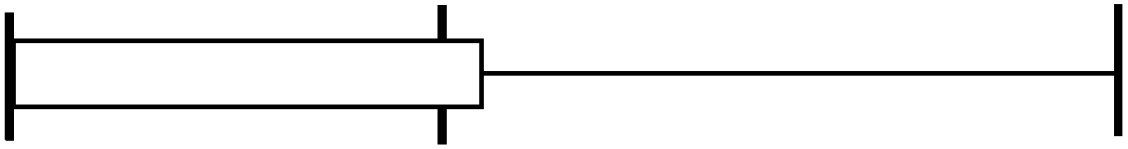
위와 같은 상황들이 나타났을 때, 서버 핸드폰 영상을 계속 재생시키고 클라이언트 자체적으로 해결하려고 할 경우, 재생 시간이 길어질수록 분할된 화면 사이의 괴리감은 심해졌다.
따라서 클라이언트에서 위와 같은 상황이 나타나면 서버와 네트워크 통신으로 화면 간의 괴리를 줄여야 했다.

하지만 안드로이드의 와이파이 다이렉트로 네트워크 통신을 하면, 대부분의 경우에서 네트워크 속도가 8~10Mbps 이상이다. 따라서 일반적인 상황에서는 2번의 경우를 테스트를 할 수가 없었다. 이 상황을 테스트하기 위해 서버에서 전송하는 패킷 사이에 sleep() 함수를 삽입했다.
다음은 위의 상황들을 해결하기 위한 솔루션이다.

1.Video View가 읽은 미디어 데이터의 길이에 도달했을 때

Playing Video에서 언급했듯, Video View는 파일 경로를 통해 마지막으로 읽은 미디어 데이터의 길이까지의 영상을 재생한다. 때문에, 다운로드가 완료되지 않은 채로 영상을 실행하면 감상 중 다운로드가 완료되어도 비디오 경로를 다시 설정해줘야 한다. 마지막으로 파일 경로를 재설정했을 때, 파일 다운로드가 끝났다면 더 이상 경로를 재설정할 필요가 없으며, 자유롭게 10초 전후로 영상을 옮길 수도 있다.

ex)



(마지막으로 읽은 미디어 데이터의 위치)

이 위치까지 영상을 재생했을 때, 헤더에서 읽은 정보는 아직 영상의 재생 시간이 더 남아있지만, 남은 미디어 데이터가 없으므로 에러를 발생하고 영상을 종료한다.
따라서, 우리는 클라이언트 핸드폰에서 이런 상황이 발생할 경우, 서버 핸드폰에 알리고 파일 경로를 재설정하며 플레이 중이던 시간으로 비디오를 돌려놓아야 한다.
순서는 다음과 같다.

1) 읽을 미디어 데이터가 없는 클라이언트는 Error Listener를 실행한다.
-클라이언트에서 최초로 동기화 작업이 일어나는 부분이다. Video View가 파악한 미디어 데이터의 위치까지 영상이 재생됐다면 Video View의 Error Listener가 작동한다.
현재 파일의 크기에 따라 Listener에 작성해둔 메소드를 실행함으로써 동기화 작업을 이어 나간다.

2) On Error 시에 클라이언트 핸드폰은 서버로 메시지를 보내기 위한 Thread를 실행한다.
Thread를 통해 클라이언트 핸드폰은 현재 미디어 데이터가 끝이 났으니 동기화 작업을 위해 영상 정지를 요청하고 파일 경로를 재설정하기 위한 준비를 한다.

3) 메시지를 받은 서버는 다른 한쪽 클라이언트에 영상 정지 신호를 보낸다.
-Error가 발생한 클라이언트 핸드폰의 메시지를 서버에서 수신한다. 서버는 메시지 수신 후 Error가 발생하지 않은 다른 클라이언트에 영상 정지를 요청한다.

4) 다른 클라이언트에서는 신호가 올 때까지 영상을 정지한다.
-Error가 발생하지 않은 다른 클라이언트는 서버의 영상 정지 요청으로 영상을 정지한다.
그리고 서버의 재생 요청 메시지를 받기 전까지 영상을 정지 상태로 유지한다.

5) Error 상황인 클라이언트에서 현재까지 다운로드 된 파일로 경로를 재설정한다.
-파일 경로를 재설정함으로 Error가 난 시점 이후까지의 영상을 재생할 수 있게 된다.
하지만 현재 파일이 다운로드가 완료된 파일이 아니라면 이후 Error 상황이 다시 나타나게 된다.
현재까지 다운로드 된 미디어 데이터의 크기만큼 영상을 재생한 후 다시 Error Listener가 작동한다.
6) 준비가 완료되었음을 서버 핸드폰에 알린다.
-파일 경로를 재설정했다면 서버 핸드폰에 재생할 준비가 됨을 알린다. Error 메시지를 보낼 때와 마찬가지로 Thread를 이용해 준비 메시지를 보낸다.

7) 신호를 수신한 서버는 클라이언트 핸드폰 모두에 현재 영상 위치를 전송한다.
-준비 메시지를 받은 서버는 현재 영상의 위치를 변수에 저장한다. 저장된 변수를 불러와 양 클라이언트 핸드폰에 프로토콜과 함께 영상의 위치를 전송한다.

8) 영상 위치를 수신한 위치로 변경한다.
-클라이언트 핸드폰은 영상의 위치를 수신한 값으로 변경한다. 후 서버에 재생 준비 신호를 보내고 서버의 재생 신호를 기다린다.

9) 영상을 재생한다.
-클라이언트에서 전송한 재생 준비 신호를 서버가 수신한다. 이때, 양쪽 모두에서 신호를 수신한 뒤 재생 신호를 보내야 하기 때문에 Boolean 변수를 통해 조건문을 작성한다.
false일 때 준비 신호를 받으면 true로 변경한 뒤 다시 수신을 기다린다. true일 때 신호를 받으면 두 클라이언트 모두 준비 신호를 보낸 상태이므로 클라이언트에 재생 신호를 전송하고 다시 false로 변경한다. 3대의 핸드폰 모두 신호를 수신하고 영상을 재생한다.

대부분의 경우 위의 방법으로 해결이 가능하다. 다음은 다운로드 속도가 느린 경우에 싱크를 맞추는 솔루션이다.

2-1. 10초 앞이 다운로드 된 파일 시간 크기보다 큰 경우
이 경우는 앞서 말한 것과 같이 네트워크 환경이 좋지 않아 다운로드 속도가 재생속도보다 느린 경우에 해당한다. 예를 들어, 현재 읽어 들인 미디어 데이터가 10초 분량이고 7초를 재생 중, 현재 다운로드 된 파일 크기가 11초 분량이라고 가정하겠다. 그리고 사용자가 10초 빨리 감기 버튼을 클릭해 18초를 재생하려고 한다. 이 경우 현재 다운로드 된 파일 크기가 18초 분량이 되지 않으며 당연하게도 읽어 들인 미디어 데이터의 크기 역시 18초 분량이 되지 않는다. 이 경우 18초 이상의 분량이 다운로드가 되어야 동영상이 재생된다. 물론 정상적인 와이파이 다이렉트로 연결이 되었다면 위와 같은 상황은 발생하지 않을 것이다. 그래서 이 부분을 테스트하기 위해 앞서 말했듯 sleep() 함수를 이용해 의도적으로 파일 다운로드 속도를 늦췄다.
순서는 다음과 같다.

1) 서버 핸드폰에서 10초 빨리 감기 버튼을 클릭한다.
버튼을 클릭하여 클라이언트 핸드폰에 10초 앞의 영상 재생을 요청한다.

2) 위치 변경을 요청 받은 클라이언트는 10초 앞의 파일이 존재하지 않으므로 Error Listener가 작동한다.
이 때 작동하는 Error Listener의 메소드는 1번의 경우와는 다르다. 1번의 경우에는 파일 경로를 재설정했을 때 재생할 수 있는 파일이 충분히 있는 상황이다. 하지만 이 경우는 파일 경로를 재설정 하더라도 10초 앞의 영상을 재생할 수 있는 파일이 존재하지 않기 때문에 다운로드를

기다려야한다. 따라서 Error Listener를 통해 서버에 파일이 없음을 알리고 다운로드를 대기한다.

2, 3, 4) 1번과 동일

5) Error 상황인 핸드폰에서 현재 다운로드 된 파일 크기를 계산한다.
처음 파일 전송을 시작할 때, 서버에서 파일의 크기와 헤더의 크기를 계산하여 전송해줬다.
이 정보를 이용해 현재 파일의 크기대비 재생가능한 시간을 계산한다. 이것을 계산하는 방법은 다음과 같다.
전체 파일의 크기에서 헤더의 크기를 제하면 mdat의 크기가 남게 된다. 헤더에서 영상의 총 재생시간을 추출하여(ms) 초 단위로 변환한 뒤 mdat의 크기를 재생시간(sec)으로 나눠준다.
이렇게 될 경우 1초 분량의 mdat크기가 남게 된다. 이것을 이용해 현재 다운로드 된 파일의 재생 가능 시간을 계산한 뒤, 그것에 따라 파일 경로를 재설정하고 실행하면 된다.

6) 10초 뒤의 영상을 재생할 수 있는 파일이 다운로드 되는 것을 기다린다.
위에서 계산한 mdat크기를 이용해 10초 뒤의 영상이 재생가능 할 때까지 파일 다운로드를 기다린다.

7) 파일 경로를 재설정한다.
원하는 위치 분량만큼 파일이 다운로드 되었다면 파일 경로를 재설정해준다.

이후로는 1번의 방법과 같다.

2번 상황이 나타났을 때의 테스트 결과를 살펴보면 다음과 같다.

```
D/####: bufferStart
D/###filesize: 934776832
D/###filesize: 934776832
D/###filesize: 934776832
D/###filesize: 934776832
D/###filesize: 934776832
D/###filesize: 934780928
D/###filesize: 934780928
D/###filesize: 934785024
D/###filesize: 934793216
```

현재 클라이언트 하나에서 10초 앞의 파일 패킷이 부족해 2번 방법대로 파일 다운로드를 대기중이다. 10초 분량의 mdat파일이 다운로드가 되면 서버에 메시지를 보내며 영상 재생을 준비한다.

```
D/###filesize: 937492232
D/###filesize: 937496328
D/###filesize: 937504520
D/###filesize: 937508616
D/###filesize: 937516808
D/###filesize: 937520904
D/###filesize: 937520904
D/###filesize: 937529096
D/###filesize: 937533192
D/###filesize: 937541384
D/####: bufferOver
```

이렇게 두 가지 상황에 대하여 핸드폰끼리 화면 싱크를 맞추는 작업을 하는 부분이었다.
앞서 말한 것과 같이 대부분의 상황에서는 1번의 작업을 하는 것만으로 작업을 마칠 수 있다.
와이파이 다이렉트의 경우 네트워크 속도가 일정하게 빠르기 때문이다. 하지만 네트워크에
문제가 생겨 느려 지는 상황을 고려하여 2번과 같은 솔루션을 준비했다.
하지만 위의 솔루션은 쓰레드를 이용하여 핸드폰 내부적으로 여러가지 작업을 수행한다. 때문에
동영상을 재생할 때 하드웨어 상태에 따라 실행시간이 조금씩 달라진다. 매우 큰 차이는
아니지만 프레임 단위의 미세한 싱크를 맞추는 것은 성공하지 못한 상태이다.

-Exit Player
동영상을 원하는 만큼 재생했고 종료하고 싶다면 서버 핸드폰에서 뒤로 가기 버튼을 클릭하면
된다. 그럼 소켓 통신으로 클라이언트 핸드폰에 종료 메시지를 전송하게 되며 3개의 핸드폰
모두 동영상 플레이어가 종료되고, 스트리밍 파일은 삭제한다.

여기까지 모든 기능에 대한 구현방법을 순차적으로 나열해봤다. 구현한 기능에 대한 결과를
요약하자면 다음과 같다.

헤더 확인 기능에서 헤더 파일이 앞에 존재하는지 뒤에 존재하는지 정확히 파악하는데 성공했다.
또 파일 변환에 있어서 기존 파일의 손실 없이 헤더를 앞으로 옮기며 파일을 재생성 하는데
성공했다. 하지만 moov 박스가 아닌 cmov 박스(압축 헤더)에 대해서는 변환을 할 수 없다.

화면 분할의 경우 각 기기의 화면 길이 및 픽셀, dpi값을 이용하여 모든 핸드폰 기종에 동기화를
성공했다. 영상을 재생할 경우 영상의 해상도에 맞춰 화면을 분할할 수 있다.

스트리밍의 경우 네트워크 상황, 미디어 데이터의 크기에 따라 싱크를 맞추는데 성공했다.
하지만 핸드폰 하드웨어에 따라 플레이어 재생 시간이 달라지는 부분에 대한 프레임 단위의
싱크를 맞추는 것은 구현하지 못했다.

위 나열한 3가지는 이번 프로젝트의 주요 요구사항들이다.

6. SUCCESS CRITERIA

1. 파일이 손실 없이 재생성 되어야 한다.

- 동영상 파일 선택 시 파일 구조를 정확히 파악한다.
- 헤더 박스가 앞에 없는 경우 미디어 데이터의 크기와 헤더의 크기를 파악한다.
- 박스 단위의 파일 구조를 파악하여 moov박스와 mdat박스의 위치를 변경한다.
- 파일을 생성한다.

위 기능들이 정상적으로 동작하며 헤더를 앞으로 옮긴 파일이 재생성 된다.

PASS

2. 영상이 핸드폰 기종에 맞게 분할되어야 한다.

- 핸드폰 기종 별로 화면 해상도, dpi, 이를 통한 실제 길이를 받아온다.
- 영상 재생 시 플레이어의 크기를 핸드폰 길이의 총합에 맞춘다.
- 플레이어의 세로 길이를 영상의 해상도에 맞춰 늘린다.
- 앞선 핸드폰의 크기에 맞춰 픽셀 단위로 플레이어를 옮긴다.

위 기능들이 정상적으로 동작하며 영상이 3개의 핸드폰에 걸쳐 분할된다.

PASS

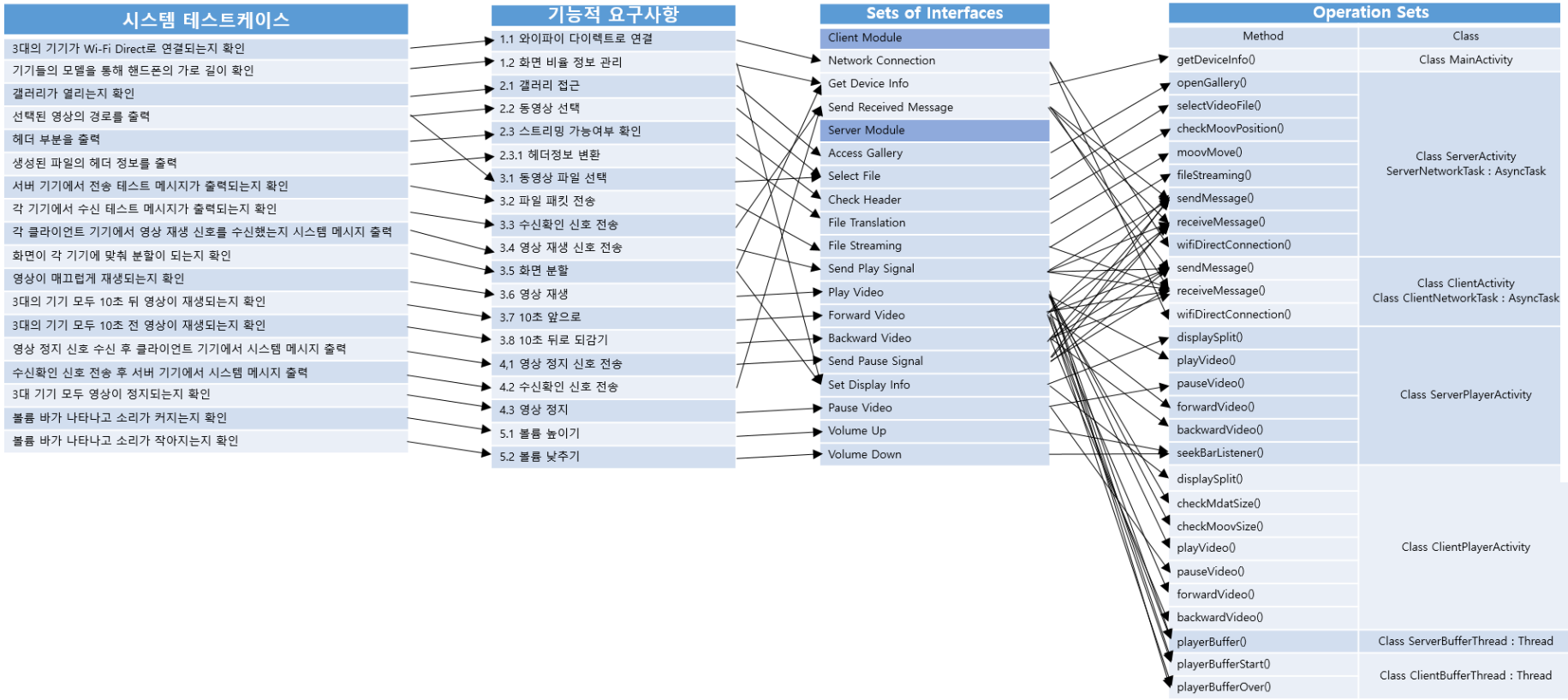
3. 모든 핸드폰의 영상 재생 싱크가 맞아야 한다.

- 소켓을 통해 영상을 제어한다.
- 읽어낸 미디어 데이터의 크기 이상으로 영상을 재생하면 파일 경로를 재설정한 뒤 동기화 작업을 시작한다.
- 네트워크 상황에 따라 다운로드 속도가 느릴 경우 읽어낸 파일 크기에 따라 동기화 작업을 시작한다.

위 기능들이 정상적으로 동작하며 핸드폰 3대의 네트워크 싱크를 맞췄다.
하지만 하드웨어에 따라 영상이 재생되는데 걸리는 시간을 제어하지 못하여 프레임 단위의 정확한 싱크를 맞추는 것은 성공하지 못했다.

PARTIAL PASS

7. TRACABILITY MATRIX



-끝-