

오픈 소스 (Open Source)를 활용한 교내 (PNU) LoRaWAN 네트워크 망 구축 및 운용



201624481 박윤형

201624517 오세영

지도교수 김종덕

목 차

1. 서론	1
1.1. 과제 배경	1
1.2. 기존 문제점	1
1.3. 과제 목표	2
2. 과제 배경	3
2.1. LoRa, LoRaWAN	3
2.2. Docker	3
2.3. ChirpStack	4
2.4. 임베디드 시스템	4
2.5. Node.js	4
2.6. MySQL	4
2.7. 안드로이드 SDK	4
3. 과제 내용	5
3.1. Docker를 이용한 서버 구축	5
3.1.1. ChirpStack	7
3.1.2. Node.js	9
3.1.3. mysql	11
3.2. 임베디드 시스템	12
3.2.1. End node	13
3.2.2. ChirpStack 서버 연동	15
3.3. 안드로이드 어플리케이션	19
4. 결과 분석 및 평가	22
5. 역할분담 및 개발일정	28
6. 참고 문헌	29

1. 서론

1.1. 과제 배경

자동차는 도로에서 운행되는 시간보다 주차장에 머물러 있는 시간이 많아 대부분의 주차공간은 이미 차량이 점유하고 있는 상태이다. 주차장은 고정되어 있지만, 차량은 이동으로 인해 시간에 따라 위치가 바뀌기 때문에 도시내에 평균적으로 차량 1대당 주차장 1면이 확보되어 있다고 해서 주차장 부족 문제가 해결되지 않는다. 이런 상황에서 주차장의 물리적인 공급량을 조절하기엔 비용적인 문제가 크기 때문에, 본 프로젝트에서는 이러한 근본적 한계를 극복하기 위해 IoT 기술을 접목하여 해결하고자 한다. 부산대 내에 비어 있는 옥외 주차장을 알려주는 서비스를 개발하여 교내 주차장 문제를 해결해 보는 것이 주차장 부족 문제에 많은 도움이 될 것으로 판단되어 주제를 이와 같이 선정하였다.

1.2. 기존 문제점

배경에서 설명한 주차장 부족 문제를 해결하기 위해 주차장 자원 활용을 최대화하는 시도들이 기존에 있었다. 그 중 International Journal of Scientific & Technology Research에서 스마트 시티를 위한 주차장 시스템을 구축하려는 시도를 확인할 수 있었다[1]. 이 프로젝트에서는 아두이노 UNO보드와 초음파 센서 그리고 NodeMCU ESP-32 모듈을 사용하여 주차장 내 차량을 감지하여 Wi-Fi를 통해 통신하는 시스템을 만들었다. 또한 Iscte - University Institute of Lisbon에서도 같은 목적을 위한 시도가 있었다[2]. 이 프로젝트에서는 아두이노 보드와 적외선 센서 그리고 ZigBee를 사용하였다.

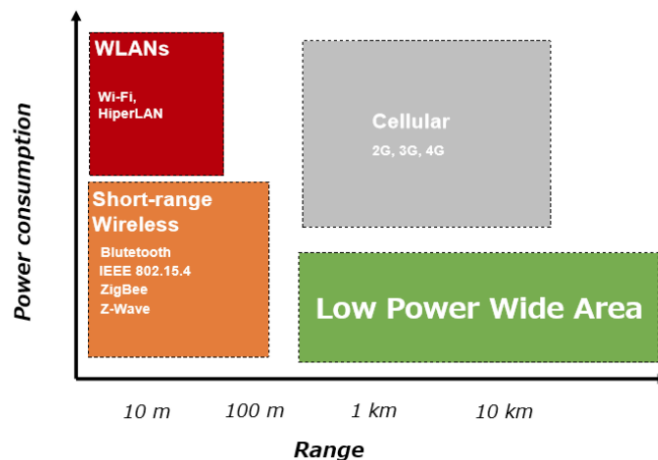


그림1. 무선 통신기술들의 비교

그러나 이런 시스템들은 우리가 목표로 하는 부산대학교의 옥외 주차장에 적용하기엔 적합하지 않다. 그림1에서 알 수 있듯, 실외 주차장에서 차량 감지를 하기 때문에 전원 공급이 용이하지 않아 Wi-Fi처럼 전력 소비가 큰 무선통신기술은 배터리로 작동하기엔 무리가 있고, 일반적인 아두이노 보드를 사용하면 오랜 시간동안 안정적으로 센서를 구동할 수 없다. 그리고 Wi-Fi나 ZigBee같은 실외 최대 전송거리가 100M이하인 통신 기술을 사용하려면 옥외 주차장이 넓게 분포된 부산대학교의 특성상 수많은 게이트웨이를 필요로 해 비용적인 문제가 발생할 것이다.

1.3. 과제 목표



그림2. 과제 목표

그림2는 본 프로젝트의 대략적인 목표를 나타낸다. 먼저 옥외 주차장의 차량을 감지하여 해당 데이터를 장거리 무선통신망을 통해 수집한다. 최종적으로 수집된 데이터를 시각화한 서비스를 안드로이드 어플리케이션으로 제공하는 것이다.

앞서 말한 문제점 들에서 보았듯 본 과제에서 요구하는 가장 중요한 사항은 저전력으로 장거리 통신을 수행해야 한다는 것이다. 이를 위해 초저전력 stm32 보드와 초음파 센서를 이용해 주차장 내 차량을 감지한다. 또한 통신기술로 LoRa를 사용함으로써 상용전력이 공급되지 않는 환경에서 배터리만으로 동작하는 원거리 무선통신망을 구축하는 것이 중요한 목표가 된다.

2. 과제 배경

2.1. LoRa, LoRaWAN

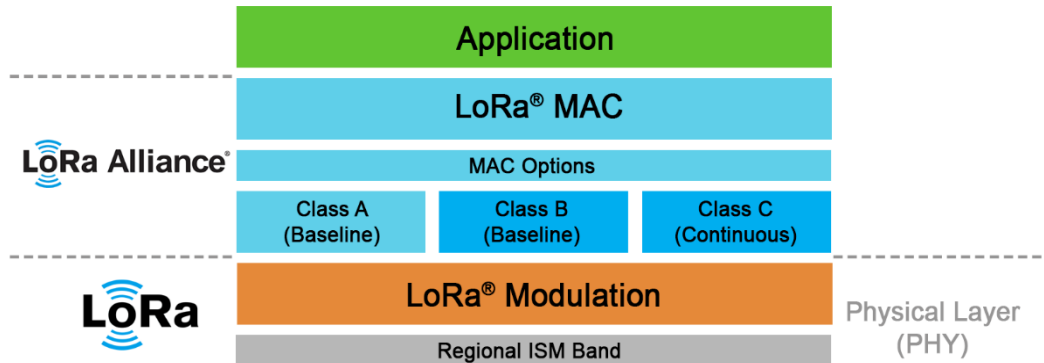


그림3. Physical and Communication layers of LoRaWAN Network

LoRa는 css기술에서 파생된 무선 변조 기술로써 그림3에서 알 수 있듯 물리적 계층에 해당한다. LoRa는 주파수의 변화를 감지해 신호를 구분하기 때문에 장거리통신에 유리하다. 또한 LoRa는 전력 소비가 낮기 때문에 상용전원이 없는 배터리를 이용한 환경에 설치하는 것이 적절하다.

LoRaWAN은 LoRa 하드웨어를 관리하기 위한 방식을 정의하는 소프트웨어 계층이다. 각각의 LoRa디바이스들과의 통신을 관리하고 보안기능을 제공하며 디바이스로부터 보내진 데이터를 저장하기 위한 기능을 제공한다.

2.2. Docker

Docker는 프로그램을 컨테이너 단위로 관리하는 오픈소스 프로젝트로서 여러 개의 서버를 효율적으로 관리할 수 있다. 각각의 컨테이너는 별개의 가상머신으로 취급되며 개발자가 원하는 환경을 이미지파일의 형태로 저장할 수 있다. 이미지 파일은 Docker를 사용할 수 있다면 다른 환경의 머신에서도 동일하게 작동하기 때문에 서버의 버전관리와 서버이전에 용이하다.

2.3. ChirpStack

ChirpStack은 LoRaWAN을 구축하기 위한 오픈소스 소프트웨어다. 무료로 사용할 수 있으며 Docker 환경에서 사용하기 위한 이미지파일과 프로젝트를 배포하고 있기 때문에 바로 구동 가능한 ChirpStack서버를 어렵지 않게 구축할 수 있다. LoRaWAN으로서 요구되는 기능들을 기본적으로 제공하고 있으며 디바이스가 보낸 데이터를 외부 서비스로 전송하기 위해 Http의 Post방식을 선택할 수 있다.

2.4. 임베디드 시스템

LoRa는 상용전력을 사용하지 않고 배터리를 사용하기 때문에 전력소모를 줄이는 것이 중요하다. 이를 위해 정해진 기능만을 사용하는 임베디드 보드를 사용해 전력소모를 최적화했다. 본 과제에서는 LoRa 엔드 노드와 게이트웨이로 임베디드 보드를 사용하며 초음파센서를 LoRa 엔드 노드에 연결해 데이터를 수집한다.

2.5. Node.js

Node.js는 브라우저 없이 javascript를 실행할 수 있는 환경을 제공함으로써 javascript만으로 서버를 간단히 구축할 수 있다. Node.js는 단일 스레드로 동작하지만 비동기식 처리를 지원하기 때문에 여러 개의 동작을 효율적으로 처리할 수 있다. 따라서, Node.js는 간단한 작업을 여러 개 처리해야 하는 본 과제에 적절하다.

2.6. MySQL

MySQL은 SQL기반의 오픈소스 데이터베이스로써 외부에서 접속할 수 있는 데이터베이스 서버를 구축할 수 있다. 다른 데이터베이스와 비교했을 때 간단한 쿼리, 작은 범위의 데이터 조회에 강점을 두고 있기 때문에, 작은 데이터를 자주 처리하는 LoRaWAN의 데이터를 저장하는데 적절하다.

2.7. 안드로이드 SDK

안드로이드 SDK는 안드로이드 기기에서 사용가능한 어플리케이션을 만들기 위한 포괄적인 개발도구를 포함하는 소프트웨어 개발 키트이다. 본 과제에서는 사람들이 운전 중에도 쉽게 서비스에 접근 가능하도록 안드로이드 어플리케이션을 통해 서비스를 제공한다. 어플리케이션에서 사용자가 원하는 건물위치에 주차자리가 있는지 시각적으로 알 수 있다.

3. 과제 내용

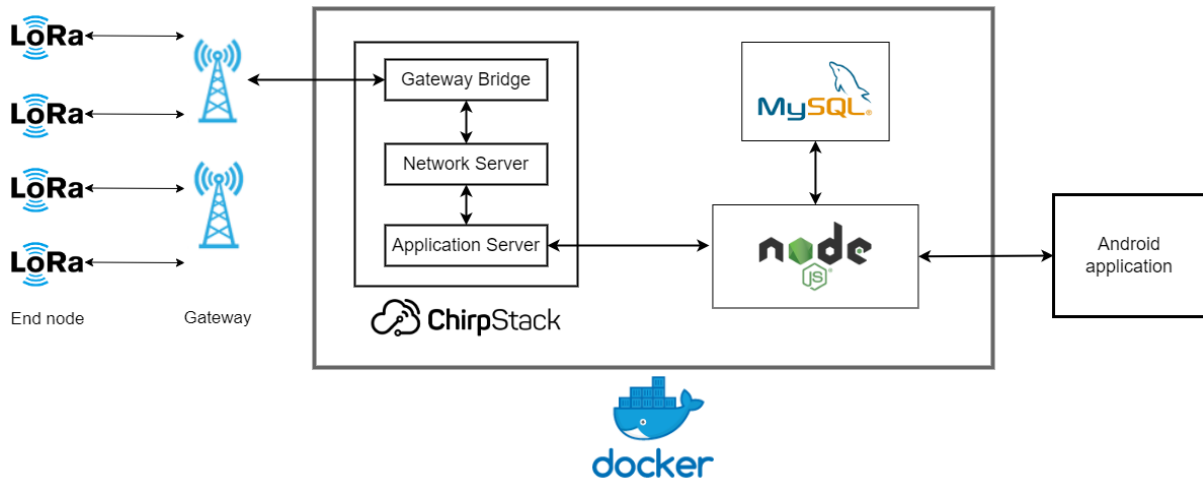


그림4. 프로젝트 구성도

그림4는 프로젝트의 전체적인 개요를 보여준다. 데이터의 전체 흐름은 다음과 같다.

1. End node에서 데이터를 측정해서 Gateway를 통해 ChirpStack서버로 전송
2. ChirpStack로 보내진 데이터를 Node.js서버로 전송
3. Node.js서버에서 받은 데이터를 MySQL서버에 저장
4. 안드로이드 어플리케이션에서 Node.js 서버로 데이터를 요청해서 서비스 제공

3.1. Docker를 이용한 서버 구축

아래쪽의 그림5는 Docker 각각의 컨테이너가 프로젝트의 어느 부분과 통신하는지를 나타낸다. 컨테이너들은 ChirpStack, node.js 그리고 MySQL의 기능을 구현한다. docker외부에 있는 Gateway는 ChirpStack 내부컨테이너인 gateway bridge와 통신하고 gateway bridge가 MQTT broker를 통해 network server로 데이터를 전송하면 application server에서 데이터를 Node.js 서버로 보낸다. Node.js서버가 데이터를 수신하면 MySQL에 데이터를 저장하고 저장된 데이터를 외부의 어플리케이션에 제공한다.

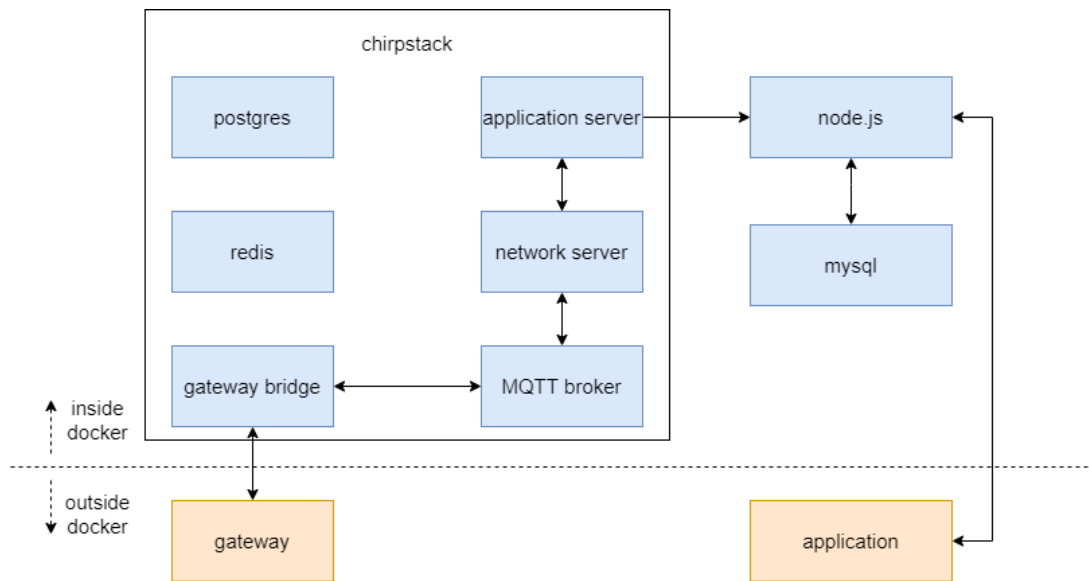


그림5. Docker 컨테이너 간의 관계

```

docker-compose.yml X Dockerfile JS main.js
docker-compose.yml
58
59   mosquito:
60     image: eclipse-mosquitto:1.6
61     ports:
62       - 1883:1883
63
64   database:
65     image: the4456/database_img
66     ports:
67       - "3306:3306"
68
69   node_app:
70     image: the4456/node_app_img
71     volumes:
72       - ./frontend/node_app:/home/node/app
73     ports:
74       - "3030:3000"
75     links:
76       - database
77

```

코드1. 수정된 docker-compose.yml

Docker 컨테이너를 만들기 위해 ChirpStack 프로젝트의 docker-compose.yml 파일에 웹서버 컨테이너와 데이터베이스 컨테이너를 추가했다. 코드1에서 확인할 수 있듯 데이터베이스는 database라는 이름으로, 웹서버는 node_app이라는 이름으로 만들도록 했다. 각각의 컨테이너에 사용된 이미지는 공식 이미지를 프로젝트에 맞게 환경을 맞춰 놓고 dockerhub에 올려놓은 것이다.

3.1.1. ChirpStack

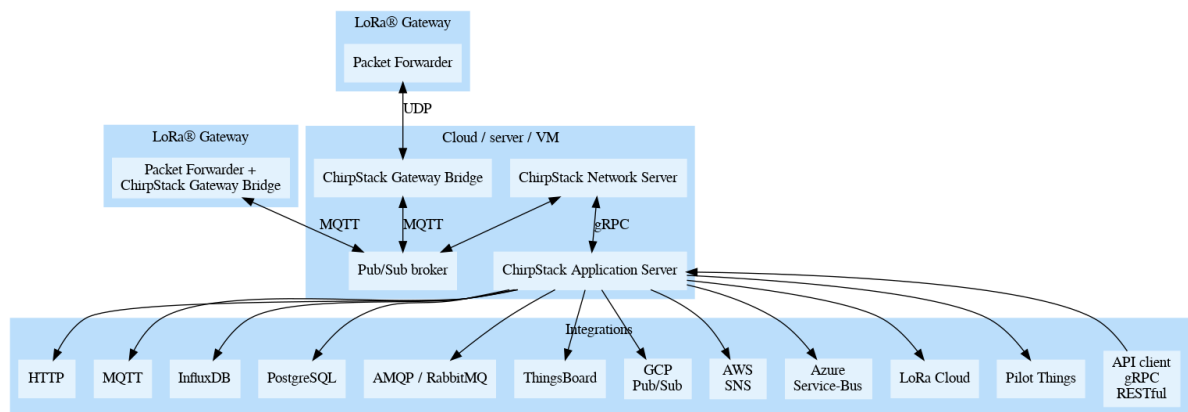


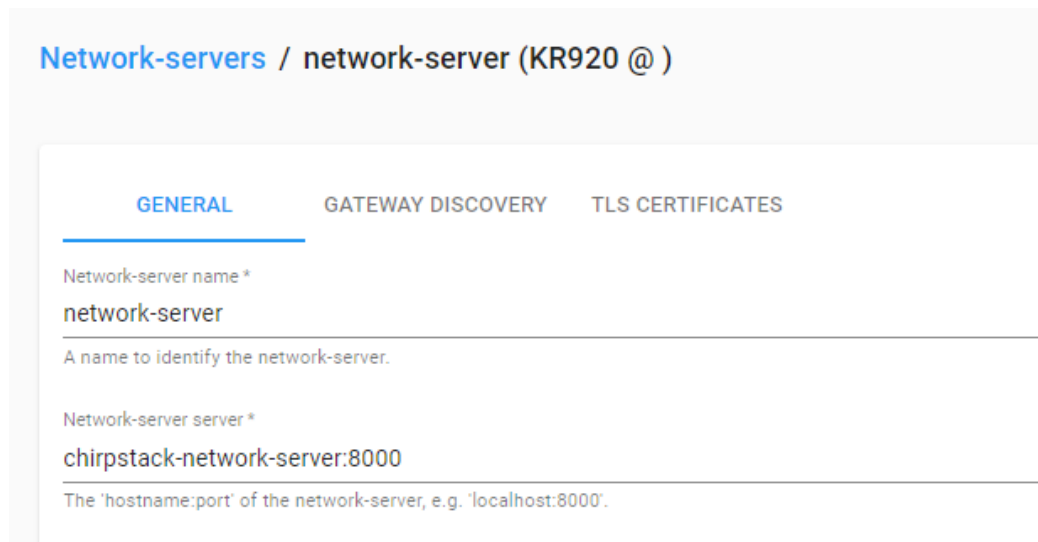
그림6. ChirpStack architecture

그림6는 ChirpStack의 아키텍처이다. 그림에서 확인할 수 있듯 본 과제에서 구현해야 할 것은 LoRa디바이스와 ChirpStack의 연결 그리고 ChirpStack으로부터 외부서비스로의 연동이다. 그 외의 LoRaWAN으로서의 기능들은 모두 기본적으로 구현되어 있다.

```
docker-compose.yml x  chirpstack-network-server.toml  JS main.js
docker-compose.yml
1  version: "3"
2
3  services:
4    chirpstack-network-server:
5      image: chirpstack/chirpstack-network-server:3
6      ports:
7        - 8000:8000
8      environment:
9        - POSTGRES_URL=postgres://chirpstack_ns:chirpstack_ns@postgresql/chirpstack_ns?sslmode=disable
10       - REDIS_URL=redis://redis:6379
11       - NETWORK_SERVER_BAND_NAME=KR920
12       - NETWORK_SERVER_GATEWAY_BACKEND_MQTT_SERVER=tcp://mosquitto:1883
13       - JOIN_SERVER_DEFAULT_SERVER=http://chirpstack-application-server:8003
14      depends_on:
15        - postgresql
16        - mosquitto
```

코드2. Docker 지역설정

lora-alliance는 국가나 지역마다 lora통신에 사용할 수 있는 주파수나 채널 등의 스펙을 정해 놓고 있다. 이에 따라 LoRaWAN의 국가 설정을 할 필요가 있다. ChirpStack에서는 국가에 따라 기본 설정 값을 가지고 있다. 코드2의 11번째줄에서 서버의 국가를 KR920으로 설정해 놓은 것을 확인할 수 있다. KR920은 lora-alliance에서 정해 놓은 한국의 lora스펙이다.



Network-servers / network-server (KR920 @)

GENERAL GATEWAY DISCOVERY TLS CERTIFICATES

Network-server name *

network-server

A name to identify the network-server.

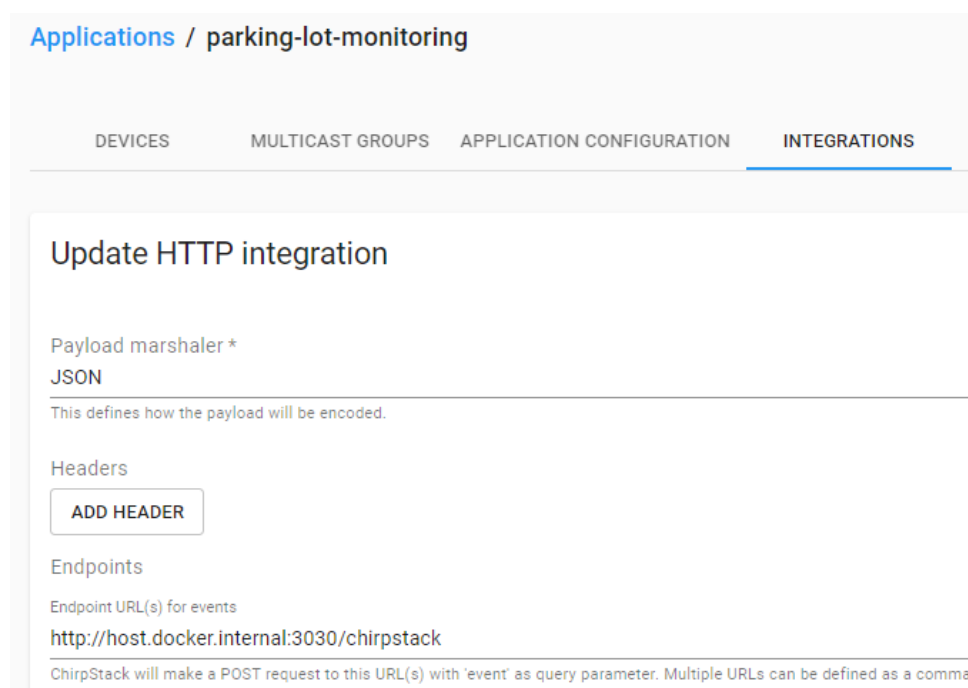
Network-server server *

chirpstack-network-server:8000

The 'hostname:port' of the network-server, e.g. 'localhost:8000'.

그림7. ChirpStack Network server 등록

다음으로 ChirpStack의 application server에 network server를 등록해야 한다. 그림7처럼 Network server의 이름과 주소로 등록을 할 수 있다. 주소로는 network server의 url을 입력해야 하지만 본 프로젝트의 모든 서버는 docker로 구현되어 있기 때문에 docker의 컨테이너이름과 포트명을 사용해 서버의 주소를 입력할 수 있다.



Applications / parking-lot-monitoring

DEVICES MULTICAST GROUPS APPLICATION CONFIGURATION INTEGRATIONS

Update HTTP integration

Payload marshaler *

JSON

This defines how the payload will be encoded.

Headers

ADD HEADER

Endpoints

Endpoint URL(s) for events

http://host.docker.internal:3030/chirpstack

ChirpStack will make a POST request to this URL(s) with 'event' as query parameter. Multiple URLs can be defined as a comma separated list.

그림8. ChirpStack 외부서비스 연동

그림8은 ChirpStack을 외부의 서비스에 연동한 모습이다. 연동해야 할 외부서비스로 Node.js의 웹서버를 선택했기 때문에 http방식으로 연동을 했다. 연동을 위해 웹서버의 주소를 입력해야 하는데 network server를 입력하는 것과 달리 docker 컨테이너이름으로 는 post 요청을 보내지 못하기 때문에 서버를 실행하고 있는 host 머신의 주소인 host.docker.internal과 웹서버의 포트번호 3030을 url로 입력했다.

3.1.2. Node.js

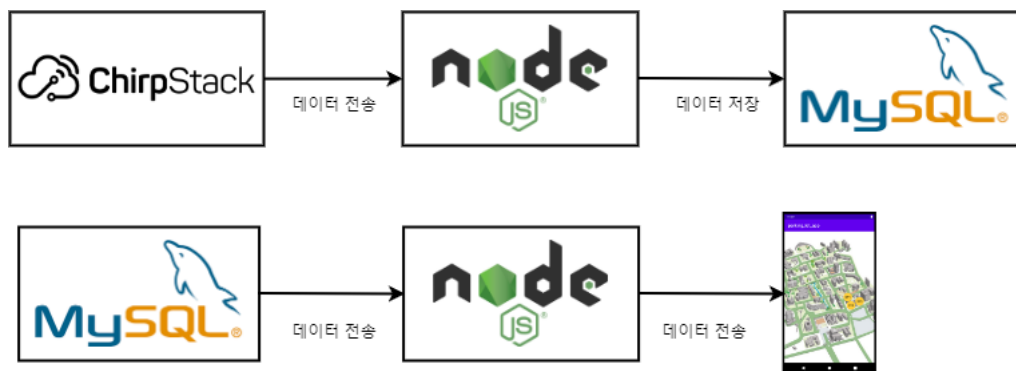


그림9. 웹서버의 기능

웹서버의 기능은 크게 두개로 나뉘는데 위의 그림과 같다. 첫번째 기능은 ChirpStack으로부터 수신한 데이터를 MySQL서버에 저장하는 것이다. ChirpStack에서 보낸 데이터 중 디바이스 주소에 따라 측정 데이터를 데이터베이스의 어디에 저장할지를 정한다. 디바이스 주소는 '(건물번호) + (주차자리번호)'의 형태로 되어 있으며 측정 데이터는 주차자리에 물체가 있다면 1, 물체가 없다면 0을 보내도록 되어있다.

```

46  ▾ else if(resource == '/chirpstack'){
47      let connection = mysql.createConnection(conn); // DB 커넥션 생성
48      connection.connect(); // DB 접속
49
50      console.log(date.format());
51      req.setEncoding('utf-8');
52
53      var postdata = '';
54  ▾   req.on('data',function(data){
55       |   postdata = postdata + data;
56       |   });
57

```

코드3. 웹서버 기능1 – post 요청 처리

코드3은 웹서버의 코드 중 일부로서, ChirpStack으로부터 받은 post요청을 처리하는 코드이다. 웹서버의 '/chirpstack'라는 path로 요청이 들어오면 Post 데이터를 비동기처리로 받도록 되어있다.

```

58  ▾ req.on('end', function () {
59      datajson = JSON.parse(postdata).objectJSON;
60      data = JSON.parse(datajson).mydata*1;
61      deviceAddress = JSON.parse(postdata).devAddr;
62      console.log('data : '+data);
63
64      buffer = Buffer.from(deviceAddress, 'base64');
65      bufString = buffer.toString('hex');
66
67      building_number = bufString.substr(0,4)*1;
68      parking_lot_number = bufString.substr(4)*1;
69
70      console.log('building number : '+ building_number);
71      console.log('parking_lot_number : '+ parking_lot_number);
72
73      let sql = "UPDATE lorawan.parking_lot SET occupied = "+ data +" WHERE building_number = "
74      + building_number + " AND parking_lot_number = " + parking_lot_number +"";
75
76  ▾   connection.query(sql, function (err, results, fields) {
77      ▾       if (err) {
78          |           console.log(err);
79          |       }
80      |   });
81
82      connection.end();
83  });

```

코드4. 웹서버 기능1 – post 데이터 파싱 및 저장

Post요청을 처리가 끝나면 받은 데이터를 JSON형태로 파싱해 device address와 mydata를 뽑아낸다. device address는 건물번호와 주차자리 번호의 연결로 이루어져 있기 때문에 각각의 번호로 다시 나눠주고 database의 해당하는 record를 특정할 수 있다. 특정된 record의 occupied를 mydata의 값으로 update해주면 데이터의 저장이 완료된다.

```

23  if(resource == '/app'){
24      res.writeHead(200, {'Content-Type' : 'JSON'});
25      console.log(date.format());
26      let connection = mysql.createConnection(conn); // DB 커넥션 생성
27      connection.connect(); // DB 접속
28
29      if(qry == 0){
30          sql = "SELECT * FROM lorawan.parking_lot";
31      }
32      else{
33          sql = "SELECT * FROM lorawan.parking_lot where building_number="+qry;
34      }
35
36      connection.query(sql, function (err, results, fields) {
37          if (err) {
38              console.log(err);
39              setTimeout(handleDisconnect, 2000);
40          }
41          console.log("app request : " + qry);
42          connection.end();
43          res.end(JSON.stringify(results, null, 2));
44      });
45  }

```

코드5. 웹서버의 기능2 – application에 데이터 전달

두번째 기능은 데이터베이스의 데이터를 어플리케이션으로 보내는 기능이다. 어플리케이션에서 웹서버로 원하는 건물번호의 데이터를 요청하면 웹서버는 데이터베이스에 있는 해당 건물번호의 모든 데이터를 출력한다. 코드5는 웹서버의 두번째 기능을 담당한다. http요청의 url에 path가 'app'일 때 query의 값에 해당하는 건물번호의 데이터를 출력하는데 query가 0일때는 데이터베이스의 모든 데이터를 출력한다.

3.1.3. mysql

	building_number	parking_lot_number	occupied
	312	1	0
	313	1	1
	313	2	0
	313	8	0
	781	1	0
	781	2	0
	781	24	1

그림10. 데이터베이스 테이블

데이터베이스는 각 주차자리에 차량이나 물체가 있는지 없는지를 저장한다. 이를 위해 부산대의 건물번호와 임의로 지정한 주차자리 번호로 각 주차자리를 구분하고 주차자리에 물체가 있는지 없는지 1또는 0으로 표시한다. 그림10의 테이블에서 각 컬럼의 의미는 building_number가 건물번호, parking_lot_number가 주차자리번호, occupied가 물체 존재 여부를 의미한다.

3.2. 임베디드 시스템



사진1. 임베디드 보드(좌: end node, 우: gateway)

LoRa디바이스로는 임베디드 보드를 사용한다. 데이터를 측정할 end node는 B-L072Z-LRWAN1, gateway는 p-nucleo-lpwan2을 사용한다.



사진2. 초음파 센서(HC-SR04)

데이터 측정을 위한 센서로는 초음파 센서(HC-SR04)를 사용한다. 초음파 센서를 선택한 이유는 측정거리가 3m이상을 보장하기 때문에 주차공간에 소형차가 있을 경우에도 측정이 가능하기 때문이다. 또한 소비전력이 15mA로 배터리수명이 중요한 본 과제에 가장 적절하다고 볼 수 있다.

3.2.1. End node

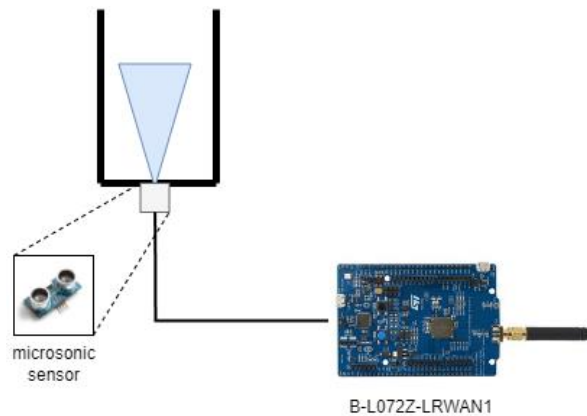


그림11. End node 구성

그림11는 end node를 어떻게 구성하고 주차장에 설치했는지를 나타낸다. 초음파센서를 B-L072Z-LRWAN1보드에 연결하고 장치를 주차장의 한쪽면에 설치한다. 센서로부터 측정되는 물체까지의 거리가 2미터를 넘으면 물체가 없다고 판단을 한다. 데이터는 end node가 데이터를 보내기 3초전에 측정해서 다음 데이터를 보낼 때 딜레이가 생기지 않도록 했다.

```
389 static void SendTxData(void)
390 {
391     UTIL_TIMER_Time_t nextTxIn = 0;
392     uint32_t i = 0;
393     AppData.Port = LORAWAN_USER_APP_PORT;
394
395     AppData.Buffer[i++] = (uint8_t)spacecheck;
396     AppData.BufferSize = i;
397
398
399     APP_LOG(TS_ON, VLEVEL_L, "FOR TEST\r\n");
400     if (LORAMAC_HANDLER_SUCCESS == LmHandlerSend(&AppData, LORAWAN_DEFAULT_CONFIRMED_MSG_STATE, &nextTxIn, false))
401     {
402         APP_LOG(TS_ON, VLEVEL_L, "SEND REQUEST\r\n");
403     }
404     else if (nextTxIn > 0)
405     {
406         APP_LOG(TS_ON, VLEVEL_L, "Next Tx in : ~%d second(s)\r\n", (nextTxIn / 1000));
407     }
408
409     HAL_Delay(17000);
410     spacecheck = ReadUit();
411 }
```

코드6. 데이터 전송 함수

코드6은 B-L072Z-LRWAN1의 펌웨어에 있던 데이터 전송 함수를 수정한 것이다. 보내는 데이터는 AppData로서 AppData의 Buffer에 전역변수인 측정 데이터를 담고 LmHandlerSend 함수로 AppData를 Gateway에 보내게 된다. 그 이후에 ReadUit함수를

사용해 초음파센서의 측정 데이터를 전역변수에 저장한다. 이때 데이터를 보내기 3초전에 데이터측정을 하기 위해 ReadUlt함수 실행 전에 17초의 딜레이를 주었다.

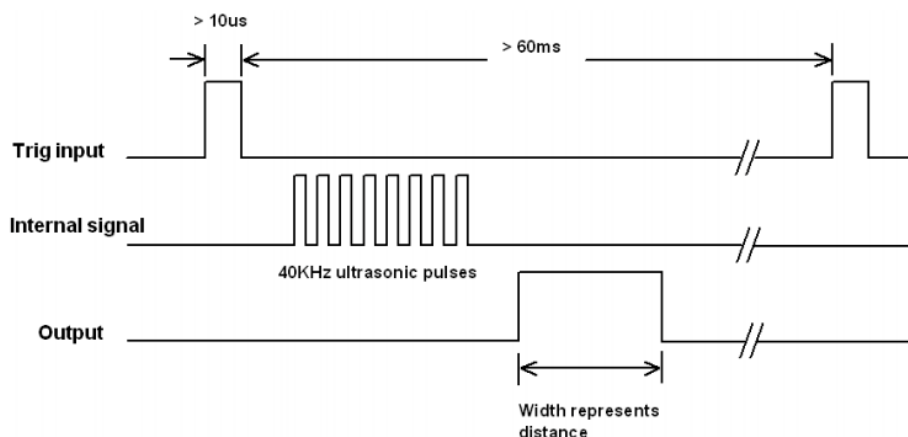


그림12. HC-SR04 Timing Diagram

End node에서 초음파센서를 사용하기 위해서는 동작 함수를 만들 필요가 있었다. 동작 함수는 위의 타이밍 다이어그램(그림12)을 참고해 만들었다. 초음파센서에 10μs의 신호를 입력시키면 센서와 물체와의 거리에 따라 출력신호를 발생시킨다. 이때의 출력신호의 시간을 측정함으로써 End node와 차량까지의 거리를 측정할 수 있다.

```

517 uint32_t ReadUlt(){
518     uint32_t echo = 0, distance = 0;
519     BSP_ULT_TRIG_Off();
520     HAL_MSDelay(2);
521     BSP_ULT_TRIG_On();
522     HAL_MSDelay(10);
523     BSP_ULT_TRIG_Off();
524     HAL_MSDelay(100);
525
526     while(!echo){
527         echo = BSP_ULT_ECHO_Read();
528     }
529     while(BSP_ULT_ECHO_Read() && distance < 20000){
530         distance = distance + 1;
531     }
532     if (distance >= 20000) distance = 0;
533     distance = (distance * 24)/1000;
534     APP_LOG(TS_OFF, VLEVEL_M, "\r\n##### distance= %d\r\n", distance);
535
536     if (distance <= 200) return 1;
537     else return 0;
538 }
539

```

코드7. 초음파센서 사용 코드

코드7은 초음파센서를 사용하기 위한 코드이다. 위의 설명에서 알 수 있듯 초음파센서를 사용하기 위해서는 보드 측에서 $10\mu\text{s}$ 의 출력신호를 주고 입력되는 신호의 시간의 측정해야 한다. 그러나 기존의 펌웨어는 μs 단위로 딜레이를 줄 수 있는 함수가 없고 ms단위의 함수가 있었다. 우리는 ms단위의 딜레이 함수에서 1ms의 기준이 되는 클럭 수를 1000으로 나눔으로 μs 단위의 딜레이 함수를 만들 수 있었다. 코드7의 521줄부터 523줄까지가 $10\mu\text{s}$ 의 신호를 주는 부분이며, 526줄부터 531줄까지가 초음파센서가 발생한 신호의 시간을 재는 부분이다.

3.2.2. ChirpStack 서버 연동

```

)/*!
 * Application session key
 */
#define LORAWAN_APP_S_KEY          97,0F,8A,A4,6A,AC,3D,85,08,53,79,13,F8,C0,8E,ED

)/*!
 * Forwarding Network session key
 */
#define LORAWAN_NWK_S_KEY          79,C3,CF,FF,29,00,91,2A,FB,82,DC,F7,5A,FC,FF,AC

```

그림13. End node 세션키 설정

End node가 서버와 통신하기 위해서는 서버에서 생성한 세션키를 펌웨어에 입력해줘야 한다. 그림13은 장치의 펌웨어 중 일부로써 각 세션키를 입력하는 부분이다. 세션키로는 network session key, app session key 두 개가 있다. Network session key는 End node와 네트워크 서버 간의 통신에 사용되고 app session key는 End node에서 보내진 데이터의 암호화/복호화에 사용된다

ChirpStack

Search organization, application, gateway or device

Applications / parking-lot-monitoring / Devices / end-node1

DETAILS CONFIGURATION KEYS (OTAA) **ACTIVATION** DEVICE DATA LORAWAN FRAMES

Device address *

03 13 00 01

While any device address can be entered, please note that a LoRaWAN compliant device address consists of an AddrPrefix (derived from the NetID) + NwkAddr.

Network session key (LoRaWAN 1.0) *

79 c3 cf ff 29 00 91 2a fb 82 dc f7 5a fc ff ac

Application session key (LoRaWAN 1.0) *

97 0f 8a a4 6a ac 3d 85 08 53 79 13 f8 c0 8e ed

Uplink frame-counter *

63

그림14. ChirpStack 서버에 End node 등록

그 다음으로 ChirpStack 서버에 End node를 등록해야 한다. 그림14는 ChirpStack서버 내에서 End node를 등록하는 페이지이다. 이때 입력해줘야 하는 정보는 end node의 device address, network session key, app session key 세 개다. Network session key, app session key는 end node에 펌웨어를 올릴 때 입력했던 것을 사용한다. Device address는 데이터 베이스에서 각 주차자리를 특정하는데 사용되어야 하기 때문에 address의 앞의 2byte는 건물번호, 뒤의 2byte는 임의로 정한 주차자리 번호의 형태로 되어있다.

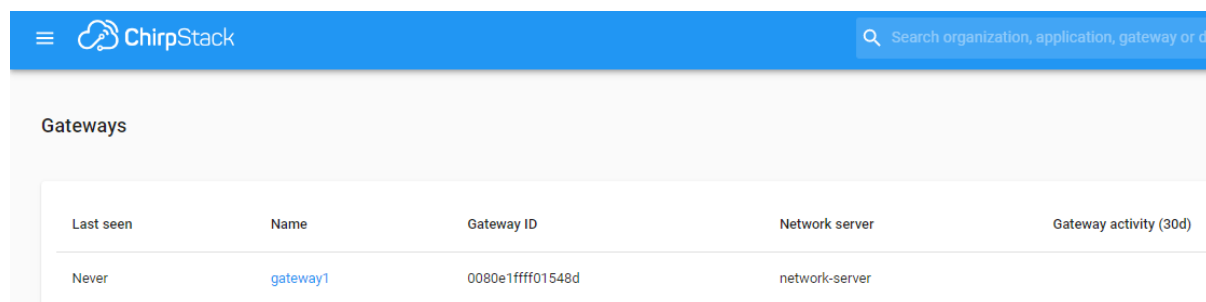
```
AT+MAC=0080E1015480
+MAC: 00:80:E1:01:54:80

AT+CH=KR920
+CH: KR920
+CH: 0, 922100000, A, SF7/SF12, 84125KHz (LORA_MULTI_SF)
+CH: 1, 922300000, A, SF7/SF12, 84125KHz (LORA_MULTI_SF)
+CH: 2, 922500000, A, SF7/SF12, 84125KHz (LORA_MULTI_SF)
+CH: 3, 922700000, A, SF7/SF12, 84125KHz (LORA_MULTI_SF)

AT+PKTFWD=nan1.cloud.thethings.network,1700,1700
+PKTFWD: nan1.cloud.thethings.network, 1700, 1700
AT+RESET
+RESET: OK
Restarting...
```

그림15. 시리얼통신으로 게이트웨이 설정

End node가 ChirpStack으로 데이터를 보내기 전에 Gateway와의 lora통신이 이루어진다. 따라서 Gateway또한 ChirpStack에 연결되어야 한다. 이를 위해 게이트웨이가 ChirpStack의 네트워크서버와 통신할 수 있어야 한다. 다만 Gateway가 네트워크 서버와 직접 통신할 수 없고 네트워크 서버와 연결되어 있는 게이트웨이 브릿지 서버로 연결해야 한다. 그림15는 gateway에 시리얼통신으로 게이트웨이 브릿지 서버의 주소를 입력해 주는 모습이다.



The image shows the ChirpStack web interface for managing gateways. It features a blue header with the ChirpStack logo and a search bar. Below the header, there's a section titled 'Gateways' which contains a table listing gateway information.

Last seen	Name	Gateway ID	Network server	Gateway activity (30d)
Never	gateway1	0080e1ffff01548d	network-server	

그림16. ChirpStack서버에 Gateway 등록

End node와 마찬가지로 Gateway 또한 ChirpStack에 등록해줘야 한다. 이를 위해 Gateway를 특정할 수 있는 Gateway ID를 입력한다. 이로서 ChirpStack서버가 gateway를 특정하여 통신할 수 있다. 그림16에서 Gateway를 ChirpStack에 등록한 것을 확인할 수 있다.



The image shows the 'Device-profiles / device1' page in the ChirpStack interface, specifically the 'CODEC' tab. It displays a section for 'Payload codec' with 'Custom JavaScript codec functions'. Below this, there is a text area containing JavaScript code for encoding and decoding payloads. A note at the bottom states: 'The function must have the signature function Decode(fPort, bytes) and must return an object. ChirpStack Application Server will convert th'.

```

15
16
17 function toHexString(bytes) {
18     return bytes.map(function(byte) {
19         return ("00" + (byte & 0xFF).toString(16)).slice(-2)
20     }).join('')
21 }
22
23 function Decode(fPort, bytes) {
24
25     var tohex = toHexString(bytes);
26
27     return {"mydata": tohex};
28 }
29

```

그림17. Javascript로 작성된 Base64디코더

Gateway와 End node를 모두 서버에 연결하면 End node에서 ChirpStack으로 데이터를 보낼 수 있다. 다만 연결한 것만으로는 보내진 데이터를 그대로 확인할 수 없다. End node에서 ChirpStack으로 데이터를 보낼 때, 데이터 손실을 막기 위해 Base64로 인코딩해 데이터를 전송한다. 따라서 서버 측에서 수신한 데이터를 다시 Base64에서 16진수로 디코딩 하는 작업이 필요하다. 그림17는 이를 위한 디코더를 작성한 것이다. 위의 디코더를 ChirpStack에 등록하면 End node로부터 데이터를 받을 때마다 자동으로 디코딩 작업을 해준다. 그 결과를 아래의 그림18에서 mydata라는 이름으로 확인할 수 있다.

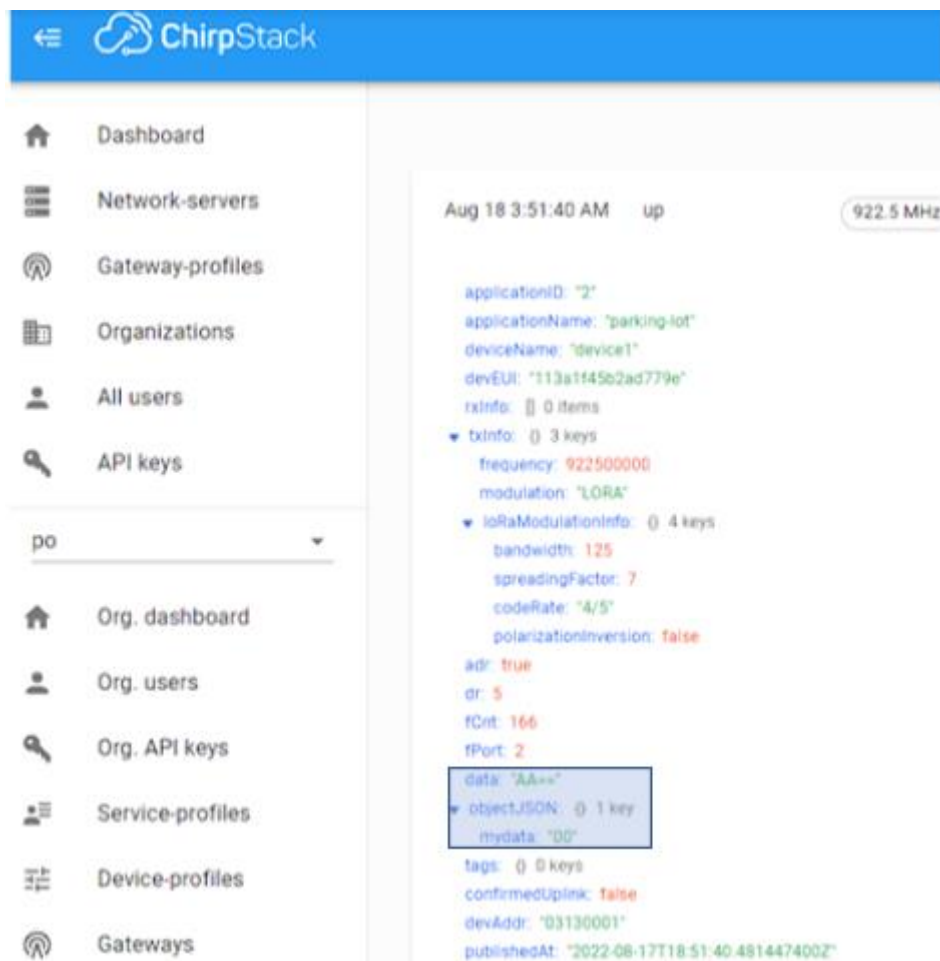


그림18. Base64에서 16진수로 디코딩 된 data

3.3. 안드로이드 어플리케이션

어플리케이션에서는 센서 측정 값을 저장한 mysql DB의 데이터를 웹서버를 통해 JSON형태로 받아와 이를 시각화 하였다. 이 과정에서 OkHttp 라이브러리를 사용해 웹서버에 값을 요청하여 받아온 후 시각화를 위해 값을 처리해주었다. 앱 동작 화면은 아래의 그림19에서 확인 가능하다.

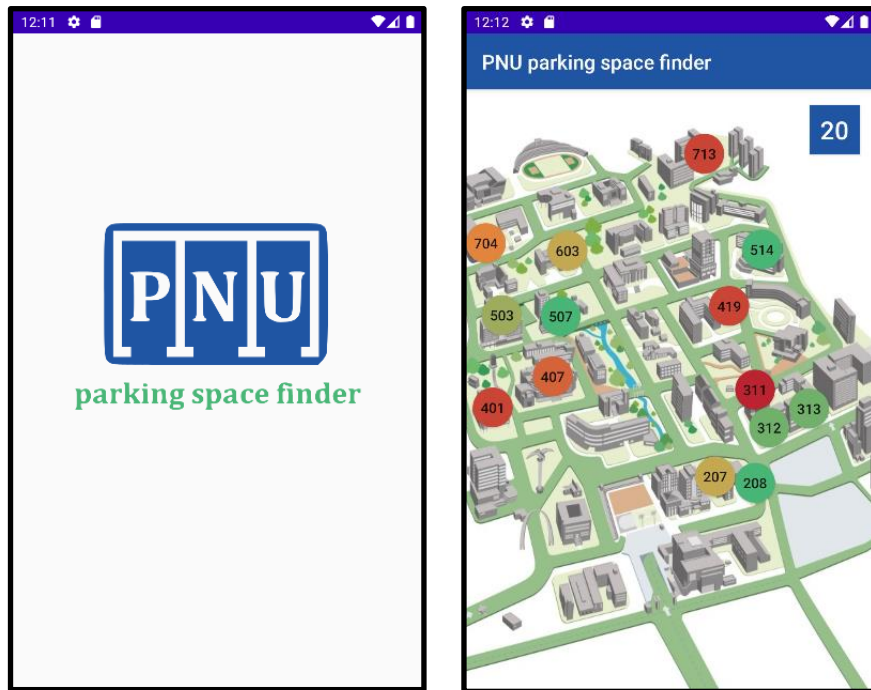


그림19. Splash 화면/ 메인 화면

메인 화면에는 부산대 지도가 보이며 그 위 건물 번호 버튼이 있다. 오른쪽 상단에는 DB로부터 정보를 받아올 때까지 남은 시간을 타이머로 나타내고 있고, 시간이 다 되었을 시 정보를 업데이트하고 타이머를 초기화 한다. 타이머를 클릭시에 즉시 정보를 업데이트 하고 타이머를 초기화 한다. 기본적으로 업데이트 주기는 센서 데이터 전송 주기와 같은 20초로 설정 되어있다. 건물 아이콘의 색깔은 히트맵으로 표현되어 해당 건물의 주차장에 빈자리가 있는지 직관적으로 알 수 있다. 그림20은 히트맵에 사용된 색상들을 보여준다. 주차장이 모두 빈자리일 경우 건물 아이콘이 가장 왼쪽의 초록색을 띄게 되고 주차장의 자리가 채워질수록 9단계에 걸쳐 오른쪽의 빨간색에 가까운 색깔이 된다.



그림20. 히트맵 색상

```

156 public void colorbynum(int building_usage, int building_capacity, Button building_button){
157     int color = 0;
158     if (building_usage <= building_capacity/9) color = Color.parseColor("#49B577");
159     else if (building_usage <= (building_capacity * 2)/9) color = Color.parseColor("#71B16B");
160     else if (building_usage <= (building_capacity * 3)/9) color = Color.parseColor("#9DAC5E");
161     else if (building_usage <= (building_capacity * 4)/9) color = Color.parseColor("#C5A852");
162     else if (building_usage <= (building_capacity * 5)/9) color = Color.parseColor("#F1A344");
163     else if (building_usage <= (building_capacity * 6)/9) color = Color.parseColor("#E4853F");
164     else if (building_usage <= (building_capacity * 7)/9) color = Color.parseColor("#D6643A");
165     else if (building_usage <= (building_capacity * 8)/9) color = Color.parseColor("#CA4536");
166     else if (building_usage <= building_capacity) color = Color.parseColor("#BC2431");
167     @SuppressWarnings("UseCompatLoadingForDrawables") Drawable circle_button
168     = getResources().getDrawable(R.drawable.circle_button);
169     circle_button.setColorFilter(color, PorterDuff.Mode.SRC_ATOP);
170     building_button.setBackground(circle_button);
171 }
172 }

```

코드8. 건물 아이콘 히트맵 구현

코드8은 메인 화면의 건물 아이콘 히트맵 기능을 구현한 코드이다. 주차자리의 1/9이 채워질 때마다 Drawable객체의 색상을 빨간색에 가까운 색상으로 바꾸어 기능을 구현했다.

```

50 client = new OkHttpClient();
51 request = new Request.Builder().url(apiurl).build();
52

```

```

113 public void GetDataFromSQL(){
114     client.newCall(request).enqueue(new Callback() {
115         @Override
116         public void onFailure(Request request, IOException e) {
117             e.printStackTrace();
118         }
119
120         @Override
121         public void onResponse(Response response) throws IOException {
122             if (response.isSuccessful()){
123                 strjson = response.body().string();
124                 StringBuffer buffer = new StringBuffer();
125                 buffer.append(strjson);
126
127                 MainActivity.this.runOnUiThread(new Runnable() {

```

코드9. OkHttpClient를 이용한 post 요청

코드8의 위쪽의 두줄은 OkHttpClient라이브러리를 사용해서 웹서버에 post요청을 보내는 코드이다. 변수 apiurl에는 웹서버의 주소가 담겨있다. 아래쪽의 함수 GetDataFromSQL에서는 post요청으로 받은 데이터를 처리한다. 함수 내의 onResponse는 post요청이 성공적으로 처리되었을 경우 수신한 데이터를 json객체로 변환한다. 이후에 json객체는 마지막 줄의 runnable함수에서 각 주차자리를 화면에 띄우는데 사용된다.

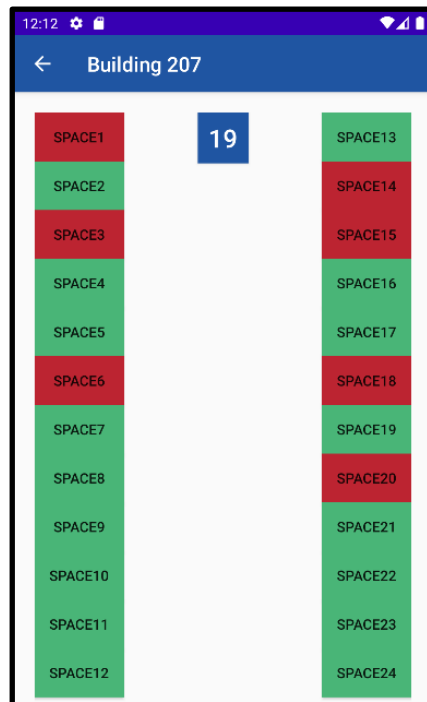


그림21. 주차장 화면

메인 화면에서 건물 아이콘을 클릭 시 그림21과 같은 화면을 표시한다. 각 네모 칸은 주차자리를 의미하며 네모 칸이 초록색이면 해당 주차자리에 차량이 없음을 의미하고 빨간색이면 해당 주차자리를 현재 어떤 물체가 차지하고 있음을 의미한다.

주차자리의 색상을 결정하기 위해 사용되는 데이터는 메인 화면과 같은 방식으로 웹 서버에 post요청을 해서 받게 된다. 다만 데이터 요청시에 어떤 건물번호의 데이터를 요청하는지 url의 쿼리부분에 담는다.

```
building207.this.runOnUiThread(new Runnable() {
    @Override
    public void run() {
        try {
            JSONArray jsonArray = new JSONArray(buffer.toString());
            int jsonArray_size = jsonArray.length();
            JSONObject jsonObj[] = new JSONObject[jsonArray_size];
            for (int i = 0; i < jsonArray_size; i++){
                jsonObj[i] = jsonArray.getJSONObject(i);
                String oc = jsonObj[i].getString("occupied");
                int space_num = jsonObj[i].getInt("parking_lot_number");
                if (oc.equals("1")) ps[space_num-1].setBackgroundColor(Color.parseColor("#BC2431"));
                else ps[space_num-1].setBackgroundColor(Color.parseColor("#49B577"));
            }
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }
});
```

코드10. 주차 차리 출력

코드10은 웹서버에서 수신한 데이터에 따라 주차장 화면의 각 주차자리의 색상을 정하는 코드이다. 웹서버에서 수신한 데이터는 json형태로 되어있고 이를 사용하기 위해 코드상에서 json배열로 파싱한다. 각 json객체는 주차자리의 번호와 주차자리 사용여부를 담고 있다. 주차자리의 사용여부가 1일경우 화면에 빨간색인 색상코드 #BC2431을 사용하고 0일경우 초록색인 #49B577을 사용해서 화면에 표시한다.

4. 결과 분석 및 평가

본 과제에서의 주요 목표는 아래의 두 가지로 나뉜다.

- 초음파센서의 측정결과를 어플리케이션에서 띄우기
- 저전력 상태에서의 원거리 통신

이를 분석 및 평가하기 위해 두 가지의 실험을 수행했다. 첫 번째 실험은 초음파 센서가 실제 차량을 감지해서 데이터를 서버로 보내 해당 데이터가 어플리케이션에 반영이 되는가에 대한 실험이다. 이 실험에서는 야외에서 실제 차량 뒤쪽에 엔드 노드를 설치해서 차량이 일정거리 이내에 있을 때와 충분히 먼 거리에 있을 때 어플리케이션에서의 변화를 기록했다. 두 번째 실험은 엔드 노드가 게이트웨이로부터 얼마나 멀리 통신 가능한지에 대한 실험이다. 이 실험은 게이트웨이가 설치된 자연대연구실험동에서부터 점차 거리를 벌려가며 통신이 되는가를 기록했다.

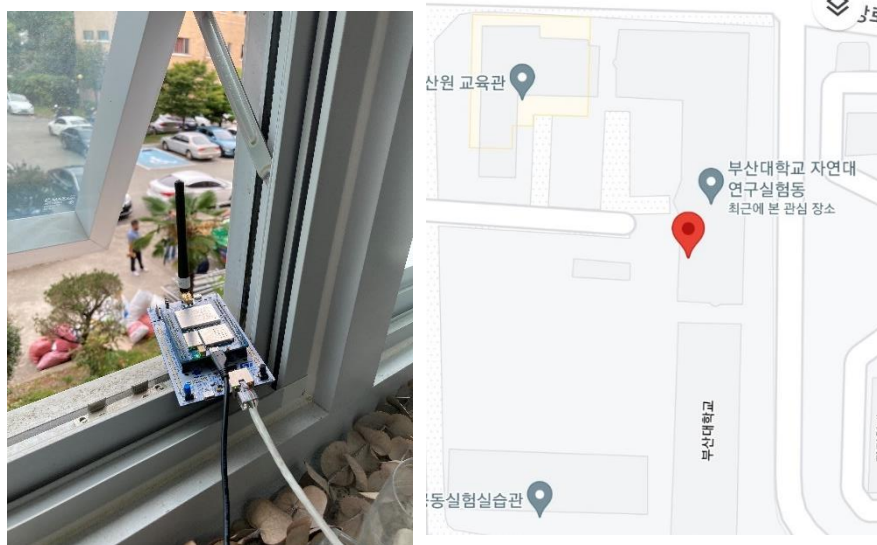


그림22. (좌)게이트웨이 설치사진, (우)게이트웨이 설치 장소

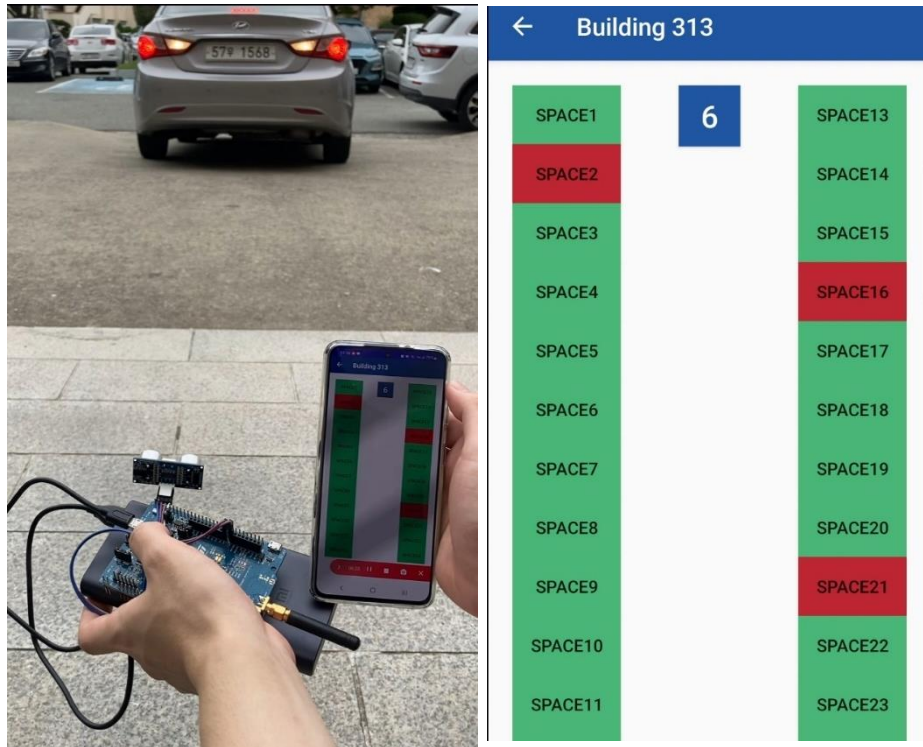


그림23. 첫번째 실험 - 차량이 감지되지 않은 상태

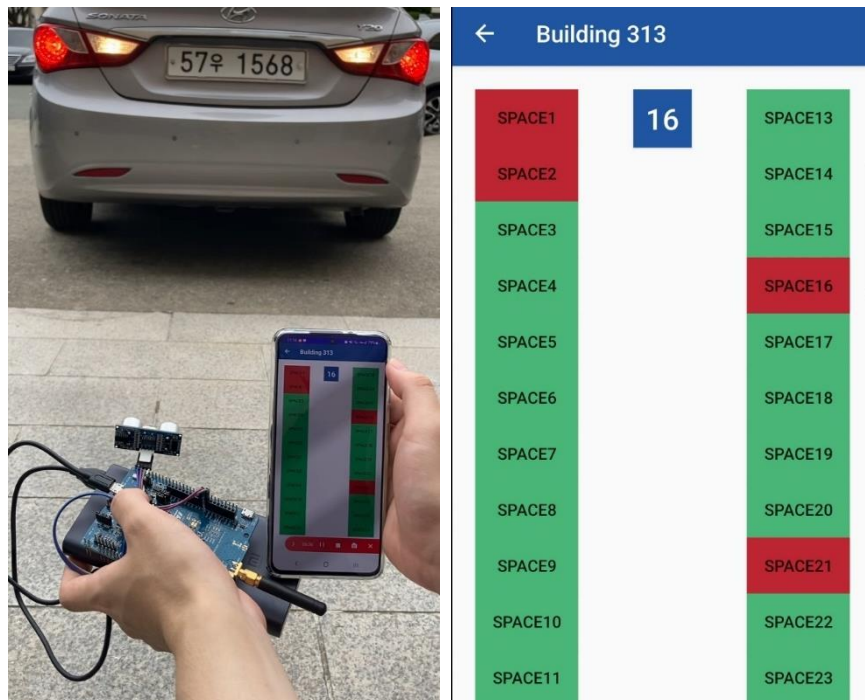


그림24. 첫번째 실험 - 차량이 감지된 상태

그림22는 실험을 위해 게이트웨이를 설치한 모습이다. LoRa특성상 옥상과 같은 환경이 가장 통신이 잘 되지만 실험환경을 옥상을 사용하는 것이 어려워 그림22의 위치에 있는 창문에 게이트웨이를 설치해 실험을 진행했다.

첫번째 실험의 진행방법과 결과는 그림23, 그림24에서 확인 가능하다. 사진에서 왼손에 들고 있는 장치가 엔드노드이며 보조배터리를 사용하는 것으로 저전력 상태를 재현하고 있다. 첫번째 실험의 내용은 초음파센서가 감지하는 거리에 차량이 있을 때와 감지하는 거리 내에 차량이 없을 때의 어플리케이션 화면을 확인하는 것이다. 초음파센서에 차량이 감지되는 범위는 실험의 원활한 진행을 위해 1m로 설정했으며, 엔드노드의 데이터는 어플리케이션의 space1에서 확인할 수 있도록 설정해 놓았다. 그림23은 차량이 센서로부터 1m이상 떨어져 있는 상태로 어플리케이션의 space1이 차량이 없음을 의미하는 초록색임을 확인할 수 있다. 반대로 그림23은 차량과 센서의 거리가 1m이내인 상태로서 어플리케이션의 space1부분이 빨간색으로 되어 있음을 확인할 수 있다. 이로서 초음파센서의 데이터가 정상적으로 서버에 수집되고 수집된 데이터를 어플리케이션에서 확인이 가능하다는 것을 알 수 있다.

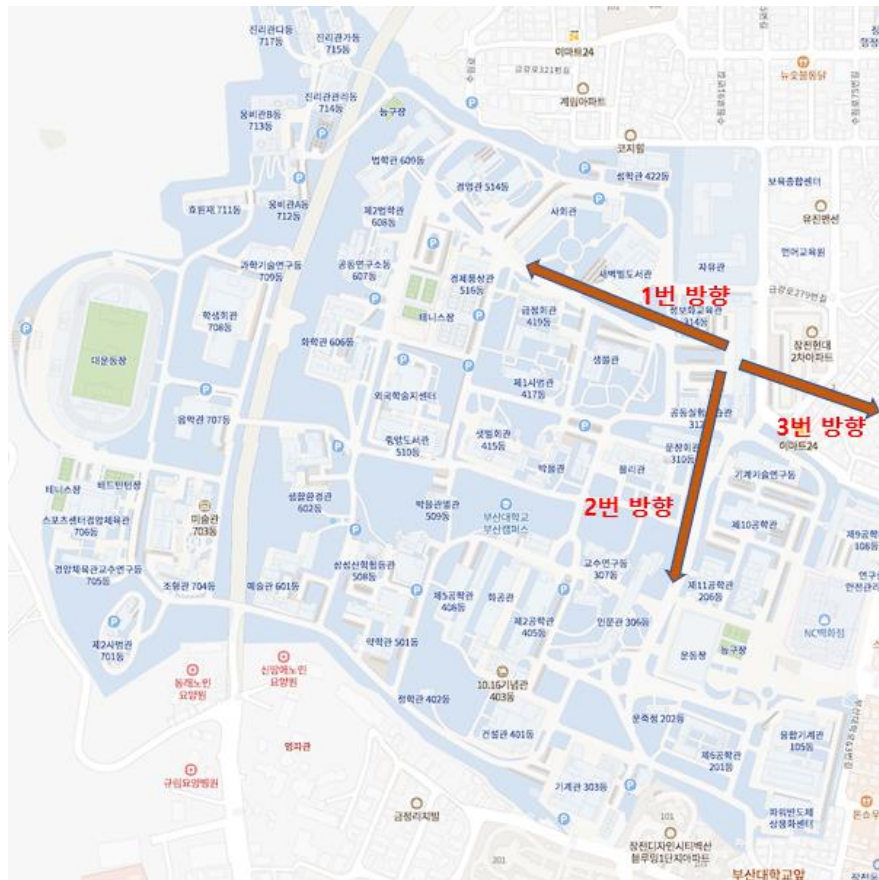


그림25. 두 번째 실험 진행 방향

두 번째 실험은 교내환경에서 게이트웨이로부터 얼마나 멀리 통신 가능한지 알아보는 실험이다. 이를 위해 게이트웨이가 설치된 자연대연구실험동에서부터 세 방향으로 실험을 진행했으며 각각의 방향은 그림25에서 확인할 수 있다. 세 개의 방향을 설정한 근거는 각 방향마다 이유가 있다. 1번방향은 부산대학교의 지형 특성상 엔드노드와 게이트웨이의 고도가 크게 차이가 나게 될 때 통신의 원활함을 알아보기 위함이다. 2번방향은 1번방향과는 달리 엔드노드와 게이트웨이의 고도차이가 별로 나지 않을 때 통신이 얼마나 원활한지 알아볼 수 있다. 3번방향은 조성된 실험환경에서 게이트웨이가 설치되어 있는 창문과 반대방향에서 통신했을 때 건물이 얼마나 방해되는지 알아보기 위함이다.

실험의 결과는 그림26에서 확인 가능하다. 실험은 각각의 방향으로 엔드노드를 들고 이동하면서 통신이 되는지 확인하면서 진행했다. 이동하면서 통신이 가능한 지역은 그림에서 노란색 선으로 표시하고 통신이 불가능한 지역은 빨간색 선으로 표시했다. 결과 2번방향으로는 운죽정까지, 3번방향으로는 게이트웨이로부터 동쪽으로 50m정도 통신이 가능함을 알 수 있었다. 1번 방향으로는 진리관까지 통신이 원활하게 되었지만 그 이후로 길이 없어 더 진행하지 못했다.



그림26. 두 번째 실험



사진3. 각 실험 방향 통신 최대거리에서 게이트웨이방향 사진 (좌: 1번/중앙: 2번/우: 3번)

사진3은 실험2에서 각 방향에서 최대 통신가능거리에서 게이트웨이 방향으로의 사진을 찍은 모습이다. 2번방향인 중앙의 사진으로 볼 때 나무가 빽빽하게 심어져 있고 그 뒤로도 몇 개의 건물이 있기 때문에 통신이 방해받는 것으로 보인다. 3번 방향에서 찍은 오른쪽 사진에서는 사진의 가운데쪽에 자연대연구실험동이 육안으로 확인 가능하지만 게이트웨이가 건물의 반대편에 있기 때문에 통신이 불가능한 것으로 보인다. 1번방향에서 찍은 왼쪽의 사진에서는 자연대연구실험동이 건물들에 가려져 전혀 보이지 않지만 고도가 높기 때문에 원활하게 통신이 가능한 것으로 보인다.

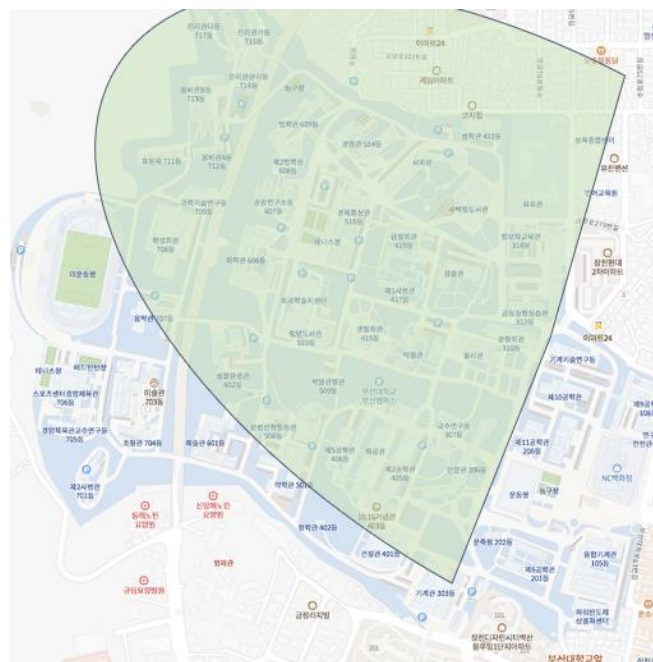


그림27. 통신가능범위

그림27은 두번째 실험의 결과인 그림26을 바탕으로 통신 가능 범위를 지도에 표시한 것이다. 범위를 부채꼴로 그린 이유는 게이트웨이가 창문에 설치되어 있어 해당 창문의 방향으로만 먼 거리까지 통신이 가능하지만 반대방향으로는 대부분 통신이 이루어지지 않았기 때문이다.

만약 게이트웨이가 창문이 아닌 옥상에 설치되어 있다면 통신가능범위는 부채꼴모양이 아닌 원형 또는 타원형의 모습을 띠는 것이다. 아래의 그림28은 게이트웨이가 옥상에 설치되었다고 가정했을 때의 통신가능범위를 예상한 것이다. 예상의 근거는 두번째 실험의 3번 방향으로도 1번 방향과 비슷한 수준의 내리막길이 있기 때문에 게이트웨이가 옥상에 설치되어 있다면 그림27만큼의 통신범위가 반대방향으로도 가능할 것이기 때문이다.

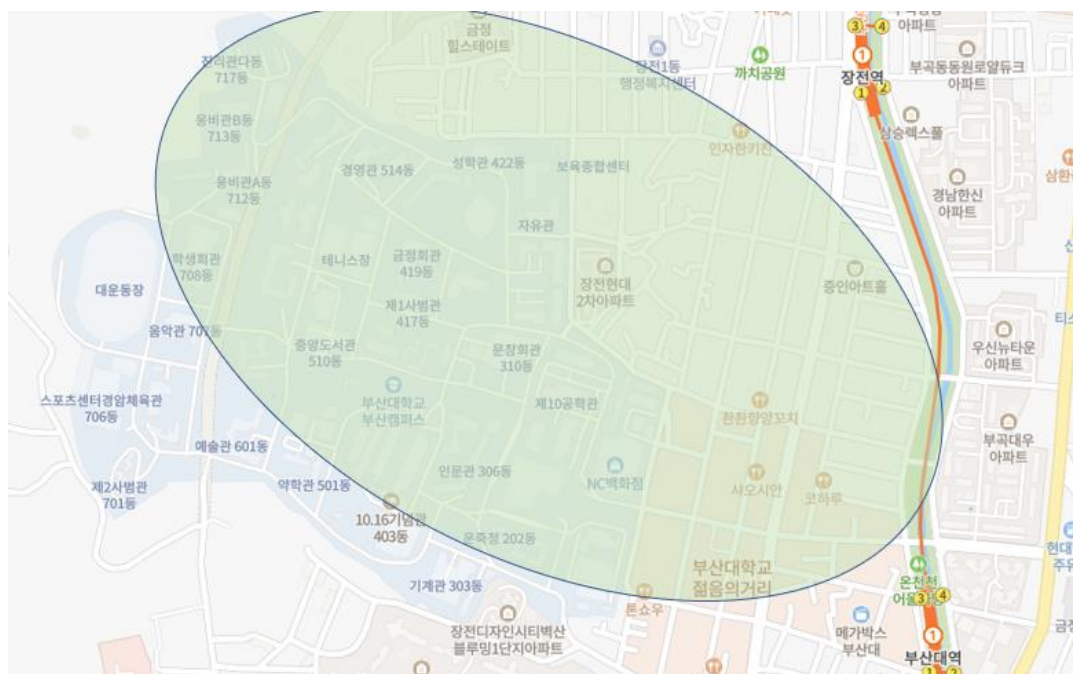


그림28. 게이트웨이를 옥상에 설치했을 경우의 예상 통신범위

그림28의 예상 범위를 볼 때 하나의 게이트웨이의 통신범위가 교내 면적의 대부분을 차지할 수 있다. 따라서 교내에서 LoRaWAN을 이용해 야외주차장의 데이터를 수집한다고 할 때 한 개의 게이트웨이로 대부분의 주차장의 데이터를 수집 가능할 것으로 보인다. 그러나 실험2의 결과에서 알 수 있듯 LoRa는 나무나 건물에 대해 크게 영향을 받는다. 따라서 주차장과 게이트웨이 사이에 건물이 있는 경우에는 추가적인 게이트웨이의 설치가 필요할 것으로 예상된다.

5. 역할분담 및 개발일정

개발일정

5월		6월					7월				8월				9월				
3주	4주	1주	2주	3주	4주	5주	1주	2주	3주	4주	1주	2주	3주	4주	1주	2주	3주	4주	5주
LoraWAN 및 chirpstack 관련 지식 스터디																			
				Chirpstack 서버 구축 및 lora gateway 연결															
						Web 서버, mysql 서버 구축 및 연동													
							어플리케이션 개발												
								초음파 센서 연결											
									중간보고										
											안정성 및 성능 평가								
												오류 수정 및 문제 점 파악							
															최종 보고서 작성 및 발표 준비				

역할분담

박윤형	Docker를 이용한 서버 구축 및 각 서버 연동 <ul style="list-style-type: none"> - Chirpstack 서버 구축 - node.js 서버 구축 - mysql 서버 구축
오세영	디바이스 펌웨어 작성 <ul style="list-style-type: none"> - gateway 설정 - lora 통신 구현 - end node와 초음파센서 연결 안드로이드 어플리케이션 개발

6. 참고 문헌

- [1] Ayaskanta Mishra, Abhijit Karmakar, Abhirup Ghatak, Subhranil Ghosh, Aayush Ojha, Kaustav Patra, “Low Cost Parking System For Smart Cities: A vehicle Occupancy Sensing And Resource Optimization Technique Using IoT And Cloud Paas”, International Journal of Scientific & Technology Research, Vol. 8, No. 9, pp. 115-122, Sep. 2019.
- [2] João Fernandes, Carlos Serrão, Nuno Miguel de Figueiredo Garrido, “A Low-Cost Smart Parking solution for Smart Cities based on open software and hardware,” Conference on Intelligent Transport Systems, 2018
- [3] Jae-wook Ko, Hye-Jeong Kim, Bo-Kyung Lee, “Factory environmental management system based on MQTT using LoRa,” The Journal of The Institute of Internet, Broadcasting and Communication Vol. 18, No. 6, pp.83-90, Dec. 31, 2018
- [4] T.Perković, P.Šolić, H.Zargarias^b, D.Čoko^b, Joel J.P.C.Rodrigues “Smart Parking Sensors: State of the Art and Performance Evaluation,” Journal of Cleaner Production Volume 262, 20 July 2020
- [5] RP002-1.0.3 LoRaWAN® Regional 41 Parameters, Available: <https://lora-alliance.org/wp-content/uploads/2021/05/RP002-1.0.3-FINAL-1.pdf>
- [6] ChirpStack Documentation, Available: <https://www.chirpstack.io/docs/>
- [7] Node.js v16.17.1 documentation, Available: <https://nodejs.org/dist/latest-v16.x/docs/api/>
- [8] Docker Desktop Documentation, Available: <https://docs.docker.com/desktop/>