KaaIoT 플랫폼을 활용한 크라우드센싱 네트워크 구축

윤준혁, 김미희*
한경대학교 컴퓨터공학과
e-mail:{junhyeok2723, mhkim}@hknu.ac.kr

Building Crowdsensing Network Using KaaIoT Platform

Joonhyuk Yoon, Mihui Kim*
Dept. of Computer Science and Engineering, Hankyong National University

요 약

크라우드센싱은 센서를 설치하는 대신 일반 대중들의 모바일 기기의 센서 정보를 이용하는 시스템이다. 본 논문에서는 오픈 소스 IoT 네트워크 플랫폼인 KaaIoT 플랫폼을 활용해 크라우드센싱 네트워크를 구축하는 방법을 제안한다. 제안된 시스템의 프로토타입을 구현하여 그 실현가능성과 성능을 보인다.

1. 서론

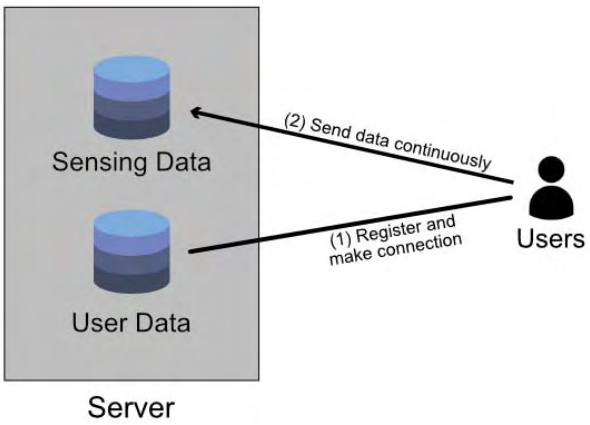
스마트 시티는 데이터 수집 센서를 기반으로 자산과 자원을 효율적으로 관리하는 도시 지역이다[1]. 스마트 시티가 효율적으로 운용되기 위해서는 다수의 데이터 수집 센서를 포함하는 센서 네트워크가 필수적이다. 데이터 수집 센서의 수가 많을수록 스마트 시티가 활용할 수 있는 데이터는 많아지고 자산과 자원 관리의 효율성은 높아진다. 그러나 필요로 하는 데이터 수집 센서가 많을수록 센서 네트워크 구축에 있어 높은 비용이 요구된다.

크라우드센싱은 별도의 데이터 수집 센서를 설치하지 않고 일반 대중들이 사용하고 있는 스마트폰이나 웨어러블 장비 등 모바일 기기에 내장되어 있는 센서를 활용하는 시스템이다[2]. 이러한 크라우드센싱 시스템을 활용하면 데이터 수집을 위해 새로운 센서를 설치하지 않아도 많은 양의 데이터를 수집하고 활용할 수 있다.

2. 관련 연구

2. 1. 크라우드센싱

센서 정보를 사용해야 하는 시스템에서 센서를 설치하기 어렵거나 불가능한 경우, 크라우드센싱 즉 이미 널리 보급되어 있는 모바일 기기의 센서 정보를 이용할 수 있다. 크라우드센싱은 크게 참여형 크라우드센싱과 지속형 크라우드센싱 두 가지 형태로 구분한다. 참여형 크라우드센싱은 필요한 정보가 있을 때 서비스 제공자가 이용자에게 요청을 보낸다. 이 때 정보를 가진 이용자는 서버로 정보를 전달하고 보상을 얻는다. 지속형 크라우드센싱은 서버가 이용자로부터 지속적으로 정보를 수집한다.



(그림 1) 지속형 크라우드센싱 네트워크 흐름도

(그림 1)은 지속형 크라우드센싱 네트워크에서 서버와 사용자 사이의 데이터 처리 흐름을 도식화한 것이다. (1) 이용자는 서버의 이용자 관리 시스템에 등록하고 서버-서비스 이용자 간 연결을 생성한다. (2)서비스 이용자는 센싱 데이터를 지속적으로 서버에게 전송하고 서버는 센싱 데이터를 데이터베이스에 저장하여 관리한다.

본 논문에서는 KaaIoT 플랫폼을 활용하여 지속형 크라우드센싱 네트워크를 구축하는 방법을 제안한다.

2. 2. IoT플랫폼과 KaaIoT

<표 1>과 같이 크라우드센싱 네트워크 구축을 위한 IoT플랫폼을 조사 비교하였다[3]. 본 논문에서는 오픈소스 플랫폼인 KaaIoT를 사용하여 구축한다.

<표 1> IoT플랫폼

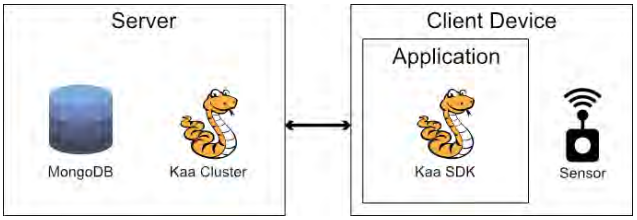
Classification	Platform	Features
Publicly traded	AWS IOT	- Supports HTTP, WebSockets, and MQTT - Create a persistent, virtual version, or "shadow," of each device that includes the device's latest state
	Google Cloud IoT	- Take advantage of Google's heritage of web-scale processing, analytics, and machine intelligence. - Utilizes Google's global fiber network for ultra-low latency
	Microsoft Azure IoT Suite	- Easily integrate Azure IoT Suite with your systems and applications, including Salesforce, SAP, Oracle Database, and Microsoft Dynamics - Supports HTTP, Advanced Message Queuing Protocol (AMQP), and MQ Telemetry Transport (MQTT).
	IBM Watson IoT	- Machine Learning, Automate data processing and rank data based on learned priorities. - Raspberry Pi Support, Real-Time Insights
Open source	KaaIoT	- Pre-integrated with popular data processing systems - Minimum hardware requirements for the C SDK are about 10 kB RAM and 40 kB ROM
	Macchina.io	- Implements a web-enabled, modular and extensible JavaScript and C++ runtime environment - Includes HTTP(S) and MQTT clients and SQLite as its embedded database

KaaIoT 플랫폼은 IoT 센서 네트워크 구축을 위한 오픈소스 플랫폼이다[4]. KaaIoT 플랫폼은 서버-엔드포인트 구조로 이루어진 IoT 센서 네트워크를 제공한다. 엔드포인트는 데이터 수집 센서를 포함하며 이를 사용해 수집한 센싱 데이터를 서버에 제공하는 컴퓨팅 노드이다. 서버는 각 엔드포인트와의 연결을 관리하고 엔드포인트가 제공한 데이터를 데이터베이스에 저장한다. KaaIoT 플랫폼에서 센싱 데이터를 관리하기 위한 데이터베이스로 문서 데이터베이스인 MongoDB를 사용한다[5][6]. 문서형 데이터베이스는 여러 서비스 이용자가 제공한 대량의 정형, 비정형 데이터를 저장하고 관리하는데 적합하다[7]. 또한 각 엔드포인트와 서버 간 연결의 보안성을 유지하기 위해 2048비트 키 기반 RSA(Rivest Shamir Adleman) 암호화를 제공한다[8].

본 논문에서는 KaaIoT 플랫폼이 제공하는 서버-엔드포인트 구조와 MongoDB, RSA 암호화를 활용하여 높은 보안성과 데이터 관리 능력을 가진 크라우드센싱 네트워크를 구축하는 방법을 제안한다.

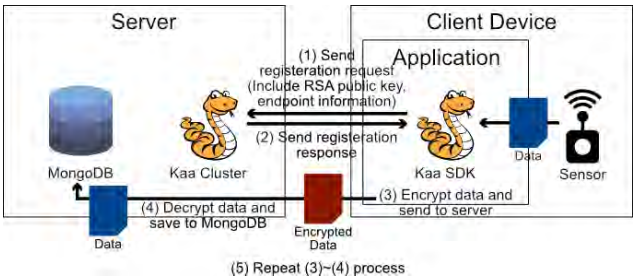
3. 제안하는 시스템

3. 1. 제안 시스템 구조



(그림 2) 제안 시스템 구조도

제안 시스템 구조도는 (그림 2)와 같다. 본 논문에서 제안하는 시스템은 서버와 서비스 이용자(Client) 기기로 구성된다. 서버는 서비스 이용자와 서비스 이용자로부터 제공받은 센싱 데이터를 관리하는 주체로 Kaa Cluster와 MongoDB로 구성된다. Kaa Cluster는 서비스 이용자 및 서비스 이용자와의 연결을 관리한다. 서비스 이용자로부터 전송받은 센싱 데이터는 Kaa Cluster를 거쳐 MongoDB에 저장하고 관리한다. 서비스 이용자는 센싱 데이터를 제공하고 사용하는 주체로 서비스 이용자 기기는 서비스 이용자 어플리케이션과 기기에 포함된 센서로 구성된다. 서비스 이용자 어플리케이션은 Kaa SDK를 포함한다. 서비스 이용자 어플리케이션은 Kaa SDK를 통해 서버와의 연결을 관리하고 센서를 통해 수집한 센싱 데이터를 서버로 전송한다. 서버로 전송되는 서비스 이용자 기기의 정보와 센싱 데이터는 KaaIoT 플랫폼이 제공하는 2048비트 키 기반 RSA 알고리즘으로 암호화된다.

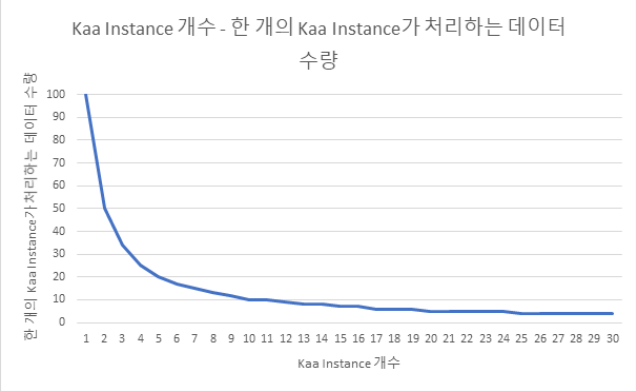


(그림 3) 제안 시스템 흐름도

서버와 서비스 이용자간의 처리 흐름은 (그림 3)과 같다. (1)서비스 이용자는 서비스 이용자 어플리케이션을 통해 서버와 통신에 사용할 RSA 암호키를 생성하고 RSA 공용키와 엔드포인트 정보를 포함하는 서비스 이용자 등록 요청을 서버로 전송한다. (2)Kaa Cluster는 서비스 이용자 등록 요청에 대한 응답을 전송한다. (3)서비스 이용자는 센서를 사용해 수집한 센싱 데이터를 서버로 전송한다. 이 때 센싱 데이터는 2048비트 키 기반 RSA 알고리즘으로 암호화한다. (4)서비스 제공자가 제공한 센싱 데이터를 복호화하여 MongoDB에 저장한다. (5)프로세스 (3),(4)를 반복한다.

3. 2. Kaa Cluster

Kaa Cluster는 여러 개의 Kaa Instance로 구성되며, 각 Kaa Instance는 서비스 이용자 및 서비스 이용자와의 연결을 관리하는 동일한 역할을 수행한다[9]. 여러 개의 Kaa Instance를 사용함으로써 다수의 사용자가 전송하는 센싱 데이터로 인한 네트워크 트래픽을 분산시킬 수 있다. (그림 4)는 총 100개의 센싱 데이터를 처리 할 때, Kaa Cluster를 구성하는 Kaa Instance의 개수와 각 Kaa Instance가 처리해야하는 데이터 수량의 관계를 나타낸 그래프이다. Kaa Cluster를 구성하는 Kaa Instance의 개수가 증가함에 따라 한 개의 Kaa Instance가 처리해야하는 센싱 데이터의 수량은 감소한다. 센싱 데이터를 여러 개의 Kaa Instance가 분산 처리함으로써 각 Kaa Instance의 부하를 낮추고 전체 센싱 데이터 처리 속도를 높일 수 있다.



(그림 4) Kaa Instance 개수 - 한 개의 Kaa Instance가 처리하는 데이터 수량

또한 서버 관리자가 2명 이상이며 각 관리자마다 접근할 수 있는 센싱 데이터의 종류가 달라야 하는 경우, 센싱 데이터의 종류에 따라 연결하는 Kaa Instance를 다르게 설정하여 관리자의 접근 권한을 설정할 수 있다.

3. 3. Kaa SDK를 이용한 응용

(코드 1)과 (코드 2)는 ARM 프로세서를 기반으로 하는 소형 PC인 라즈베리파이[10]에 연결된 센서를 통해 데이터를 수집하고 서버로 전송하는 자바 기반 클라이언트 어플리케이션 코드이다. (코드 1)은 서버와의 연결을 관리하고 센싱 데이터를 서버로 전송하는 메인 모듈이다. (코드 2)는 센서로부터 센싱 데이터를 읽어 메인 모듈에 주기적으로 전송하는 모듈이다.

(코드1) RPiGPIODataCollection.java

```
import org.kaaproject.kaa.client.DesktopKaaPlatformContext;
import org.kaaproject.kaa.client.Kaa;
import org.kaaproject.kaa.client.KaaClient;
import
org.kaaproject.kaa.client.SimpleKaaClientStateListener;
import
org.kaaproject.kaa.client.logging.strategies.RecordCountLogU
ploadStrategy;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;

public class RPiGPIODataCollection {
    private static Logger LOG =
LoggerFactory.getLogger(RPiGPIODataCollection.class);
    private static ScheduledExecutorService executor;
    private static final int LOG_THRESHOLD = 1;
    private static final int SAMPLE_PERIOD = 1;

    public static void main(String[] args) {
        KaaClient kaaClient = Kaa.newClient(new
DesktopKaaPlatformContext(), new
SimpleKaaClientStateListener() {
            @Override
            public void onStarted() {
                LOG.info("Kaa client started.");
            }

            @Override
            public void onStopped() {
                LOG.info("Kaa client stopped.");
            }
        }, true);

        kaaClient.setLogUploadStrategy(new
RecordCountLogUploadStrategy(LOG_THRESHOLD));

        kaaClient.start();
        startMeasurement(kaaClient);
    }

    private static void startMeasurement(KaaClient
kaaClient) {
        executor =
Executors.newSingleThreadScheduledExecutor();
        executor.scheduleAtFixedRate(new
MeasureSender(kaaClient, LOG), 0, SAMPLE_PERIOD,
TimeUnit.SECONDS);
    }
}
```

(코드 2) MeasureSender.java

```
import com.maxnit.kaa.scheme.TempData;
import org.kaaproject.kaa.client.KaaClient;
import org.kaaproject.kaa.client.logging.BucketInfo;
import org.kaaproject.kaa.client.logging.RecordInfo;
import org.kaaproject.kaa.client.logging.future.RecordFuture;
import org.slf4j.Logger;

import java.io.*;

public class MeasureSender extends Thread {
    private KaaClient kaaClient;
    private Logger LOG;

    private int temperature;

    MeasureSender(KaaClient kaaClient, Logger LOG) {
        this.kaaClient = kaaClient;
        this.LOG = LOG;
    }

    @Override
    public void run() {
        try {
            BufferedReader tempReader = new
BufferedReader(new FileReader("/tmp/temp_data"));
            temperature =
Integer.valueOf(tempReader.readLine());
            tempReader.close();
        } catch (IOException e) {
            LOG.error("Fail to read temp_data.");
        }

        TempData tempData = new
TempData(temperature, System.currentTimeMillis());
        RecordFuture recordFuture =
kaaClient.addLogRecord(tempData);
        LOG.info("Log record {} submitted.", tempData);

        try {
            RecordInfo recordInfo = recordFuture.get();
            BucketInfo bucketInfo =
recordInfo.getBucketInfo();
            LOG.info("Received log record delivered.
Bucket ID [{}], Record delivery time [{} ms]", bucketInfo,
recordInfo.getRecordDeliveryTimeMs());
        } catch (Exception e) {
            LOG.error("Exception occurred.");
        }
    }
}
```

4. 결론

본 논문에서는 KaaIoT 플랫폼을 활용하여 높은 보안성을 지닌 지속형 크라우드센싱 네트워크를 구축하는 방법을 제안했다. KaaIoT 플랫폼을 활용하면 MongoDB 기반의 고효율 센싱 데이터 관리 시스템을 포함하고 RSA 암호화 기반의 높은 보안성을 만족하는 크라우드센싱 네트워크를 구축할 수 있다. 또한 Kaa Cluster를 활용하여 네트워크의 부담을 최소화하고 관리자의 센싱 데이터 접근 권한을 제한하여 보안성을 높일 수 있다.

향후 해당 시스템을 스마트폰을 활용한 도시 환경 정보 수집 시스템으로서의 응용을 목적으로 구현하여 실제 시스템의 성능과 운영가능성을 보이고자 한다.

6. Acknowledgement

이 논문은 2018년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. 2018R1A2B6009620), 교신저자 김미희.

참고문헌

[1] J. Burke, D. Estrin, M. Hansen, A.Parker, N.Ramanathan, S. Reddy, M. B. Srivastava, “Participatory Sensing”, Center for Embedded Networked Sensing, May. 2006.

[2] Crowdsensing, Wikipedia, https://en.wikipedia.org/wiki/Crowdsensing

[3] https://www.postscapes.com/internet-of-things-platforms/

[4] KaaIoT Project, https://www.kaaproject.org/

[5] N. D. Karande, A Survey Paper on NoSQL Databases: Key-Value Data Stores and Document Stores, IJRAT, Vol. 6, No. 2, pp. 153-157, Feb. 2018

[6] MongoDB, https://www.mongodb.com/

[7] J. Bhogal and I. Choksi, "Handling Big Data Using NoSQL," 2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops, Gwangiu, 2015, pp. 393-398.

[8] Endpoint registration, Kaa documentation, https://docs.kaaproject.org/display/KAA/Endpoint+registration

[9] Architecture overview, Kaa documentation, https://kaaproject.github.io/kaa/docs/v0.10.0/Architecture-overview/

[10] Raspberry Pi, https://www.raspberrypi.org/