# Gateway Application Architecture

## 현 게이트웨이 애플리케이션

- ARTIK SDK Serialport 기반 C++ 애플리케이션
- 정해진 시간 `-p 600`과 `env` 환경파일을 옵션으로 설정파일 `apros_gw.cfg`를 읽어들여 주기적으로 데이터 송수신 명령을 보내고 받음
- 프로세스는 시스템 서비스에 등록하여 재부팅후에도 자동으로 수행
- 설정파일 `apros_gw.cfg`에 정해진 TCP, UDS(unix data socket)등의 설정을 통해 IPC(Inter-Process Communication)을 수행 : TCP socket 설정은 외부서버를 지정하여 데이터를 수신 받을 수 있고, UDS socket 의 경우 socket file(server)형태로 지정함에 따라 다른 애플리케이션을 수행하여 데이터를 수신 받을 수 있음
- 내부 저장 데이터는 설정파일에 설정된 폴더에 저장되며, 데이터클린 작업을 주기적으로 cron을 실행하여 제거함
- 프로세스의 로그 또한 설정파일에 지정된 폴더에 저장되며, 엔지니어가 직업 로그파일을 열어보며 프로세스의 상태를 모니터링 함

### 장점

- 예외 상황 거의 없이 상당히 안정적으로 프로세스가 동작함 : 특이한 상황이 없으면 프로세스가 죽지 않음

### 단점

- 패시브로 센서 데이터를 얻는 방법이 없음 : 설정파일을 통해 바이너리를 구동 시켜야 함
- 여러 프로젝트에서 사용하다보니 여러 버전의 바이너리가 존재하고, 설정파일 형식도 다름
- 센서에 명령을 보내고 받는 대기시간과, 재시도 횟수에 따라 전체 인터벌 시간에 영향을 받음, 시간이 부족할 경우 프로세스가 에러
- 가끔 프로세스가 동작을 멈춘 상태가 있는데 이때 로그파일을 열어서 확인해보아야 함 : 대부분 전체 프로세스 인터벌 시간안에 프로세싱이 모두 완료되지 않았을 때 오류를 배출하고 멈추어 있음
- 설치된 네트워크와 동일한 네트워크가 아니면 로그파일의 확인이 불가하여, 현장에가서 확인해야 함
- 모던 IoT 통신 프로토콜을 지원하지 않음 : 별도의 프로그램을 개발하여 동작시켜야 함 (MQTT, CoAP, Modbus등)
- Heartbeat이나 동작상태를 외부로 주기적으로 알려줄 수 있는 아키텍쳐가 없음

## 새로운 게이트웨이 애플리케이션

### Motivation

- 앞선 게이트웨이 애플리케이션의 단점과 타사의 게이트웨이(AP)등을 사용해본 결과 업그레이드의 필요성이 있음
- 센서 데이터를 모두 한번에 보내는 방식이 아니고, 16개의 데이터를 한 패킷으로 256개의 패킷을 전송함 -> 센서 단위로 하나의 전체 패킷을 보내는 방법이 필요함
- IPC 작업을 nodejs로 수행했을 때 TCP, UDS모두 스트림 개체를 사용 : 전통적인 방식인 buffer를 정하고 데이터를 받아 전송하는 방식이 아니고 `buffer=0`인 상태에서 Stream을 사용하여 읽을 수 있을 때(readable) 데이터를 보내는 방식은 패킷이 나뉘어 짐 -> 이 방식은 Web에서 병목현상을 줄이는 방법으로 효율적일 수 있으나, 쪼개진 패킷을 받은 상황에서는 패킷의 순서가 뒤바뀌는 경우가 있어 `Index`를 기반으로 Rearrange를 수행해야함
- 사용자(클라이언트)입장에서 shell 로 접속하여 설정파일을 수정하는 방식의 게이트웨이 활용은 보안상의 이유 등으로 부적절함
- 게이트웨이 및 센서의 상태 모니터링이 필요함
- 패시브하게 센서의 데이터 수집을 하고, 이를 직접 차트로 읽어 데이터를 확인 할 수 있는 방식이 필요함 ex) 김건우 차장이 개발한 PC프로그램

## 기능 정의

### Device Scanner : PC 프로그램

- 게이트웨이를 네트워크에 연결하고, 동일 네트워크상에서 프로그램을 실행하여 동작중인 게이트웨이를 찾는 애플리케이션
- IP주소와, Mac 주소를 송출하여 웹애플리케이션으로 접속

#### 기능

- 네트워크 특정 포트(4567) Broadcasting
- 게이트웨이의 응답 수신
- IP주소 혹은 Gateway 고유 이름을 수신
- 목록에서 게이트웨이 더블 클릭 시 브라우저를 열어 설정 웹페이지에 연결 함

#### 개발 플랫폼

- ElectonJS : Chromium, nodeJS기반의 크로스 플랫폼 데스크탑 애플리케이션 개발 (Windows, Linux, macOS)
- C#

### Service Worker 개념의 static 웹애플리케이션

- 최상위 층에서 모든 애플리케이션을 관장하며, 어떤 에러에도 프로세스가 Robust하게 실행되고 있어야 함
- 여러 Child Process들을 관리, 웹은 nginx와 같은 시스템 서비스로 실행 됨

#### 기능

- 특정 포트(4567)를 정해놓고 요청에 대한 응답을 보냄
- 정적 웹페이지
- 서비스 모니터링
- 버전 관리
- 시리얼 포트 관리 (serial포트 점유 해제, 연결 등)
- Child process 관리 (start/stop/restart 기능)
- 구버전 게이트웨이 애플리케이션을 child process로 구동 : 단, 구버전 애플리케이션을 구동 시 플랫폼으로 데이터를 전송하지 않음(호환성 문제)
- 로그인 기능

#### 개발 플랫폼

- 프론트 엔진 : nginx
- 프론트 앱 : HTML, CSS, Javascripts
- API 앱 : nodejs
- 미들웨어 : express, pug

### 게이트웨이 애플리케이션

- 센서노드 설정, 게이트웨이 설정, 액티브 데이터 수집, 패시트 데이터 수집등 기존 애플리케이션의 기능에 업그레이드된 애플리케이션
- MQTT, CoAP등 데이터 브로커와 메시지 큐와 인터페이스 가능한 어댑터 개발
- Edge 분석을 위한 미들웨어 개발 (알람, Trending, 특징추출, Anomaly Detection)

#### 기능

- 센서노드 통신 프로토콜 탑재
- 바이너리 파싱, 데이터 저장 기능
- Active Mode 데이터 수집 : 기존 게이트웨이 애플리케이션과 동일, Passive 데이터 수집을 기반으로 자동 루틴 구현
- Passive Mode 데이터 수집 : PC프로그램과 유사 (Register, Ready, Fetch 등) 버튼 형태로 구현(active, inactive)
- Chart : 웹브라우저에서 데이터를 모니터링 할 수 있도록 차트로 데이터 조회
- 데이터 분석 : FFT, PSD, Order 분석 결과 리포트 -> 차트 및 테이블
- IoT 프로토콜 어댑터 : 브로커 Host, 브로커 Port 설정 페이지와 X.509인증서 관리 기능 등 현재 플랫폼 MQTT기준으로 개발. 필요에 따라 추가 개발
- Heartbeat 및 로그 전송 : 설정된 브로커로 게이트웨이 상태와 로그 데이터를 전송함

#### 개발 플랫폼

- 프론트 엔진 : nginx
- 프론트 앱 : HTML, CSS, Javascripts
- API 앱: nodejs
- 미들웨어 : express, graphql, socketio, python or C++

#### 개발 스코프

- nodejs 기반 센서 통신 프로토콜 API 개발 완료
- MQTT 어댑터 개발 완료, but 플랫폼과 게이트웨이 및 센서데이터의 기준이 되는 MQTT 메시지 형식 및 연동 규격 작업이 필요
- 새로운 애플리케이션 개발 이후 플랫폼 안정화 작업이 필요 : 현재 패킷 데이터 Rearrange 및 파싱을 플랫폼에서 수행 중