웹소켓 기반의 사물 인터넷 동향

허원진 Angeliot 아키텍트

1. 머리말

사물인터넷은 먼저 시작된 M2M(기기 간 연결)을 일부 포함하고 있으며, 인터넷 연결성이 강조된 개념이다. 따라서 인터넷 연결을 통한 통신 방법이 매우 중요하다. 사물인터넷의 프로토콜은 HTTP 1.1, MQTT(Message Queuing Telemetry Transport), 그리고 본 고에서 중심적으로 다룰 웹소켓(WebSocket)이 있다.

HTTP 1.1의 경우 너무나 오래전(20년 전)에 개발된 표준으로 현재의 웹 유스케이스를 만족하기 어렵다. MQTT의 경우에는 W3C 표준이 아니기 때문에 웹 에이전트에서 사용하기 위해서는 별도의 플러그인이나 중간에서 MQTT를 웹소켓으로 변환해주는 과정이 필요하다. 웹서비스의 기반이 되는 웹서버의 경우에도 최신 웹서버의 경우 웹소켓 표준을 지원하나 MQTT를 사용하기 위해서는 경량 서버를 추가로 설치하거나 모듈을 웹서버에 설치하고 설정해줘야 한다. 데이터의 전송 측면만으로 보자면 저전력으로 설계된 MQTT가 더 우수하지만,

<표 1> 인터넷 표준 프로토콜 비교

이름	표준화 기관	표준화 시기	특징
HTTP 1.1	W3C	1990	레거시 웹에서 가장 폭넓게 사용된 방법으로 단방향 통신과 요청/응답 방식의 통신 방식을 지원함
MQTT	OASIS(Organization for the Advancement of Structured Information Standards)	2013	IBM 개발 후 OASIS에 기증, 기술 위원회 구성, 3.1.1 버전 후보 논의 진행 중, 페이스북 메신저에서 채택함
WebSocket	W3C	2011	웹에이전트와 웹서버 간의 양방향 통신을 위한 프로토콜, 인터넷 연결 포트를 공유(80포트)하여 레거시 웹 친화성이 높음

연결성과 활용성 면에서는 웹소켓이 더 우수하다. 필자는 연결성과 활용성에 좀더 점수를 주어 웹소켓을 사물인터넷 프로토콜로 선정했다.

2. 인터넷 사물이 되기 위한 조건

매일 다양한 아이디어의 제품이 IoT 제품으로 포장되어 시장에 나오고 있다. 사실 대부분의 제품은 IoT 제품이 아닌 앱세사리(Appcessorry)나 스마트 가전에 가깝다. 인터넷 사물이 되기 위해서는 크게 식별과 통신 기능을 제공해야 한다.

2.1 식별

IoT 제품과 서비스는 URI(Uniform Resource Identifier)로 구분 제공된다. 인터넷 사물들은 사람의 개입 없이 하나 이상의 사물에 의하여 서비스를 제공하고 각 기기는 URI를 통해 구분한다.

2.2 통신

제어/모니터링에 필요한 인터페이스를 인터넷 프로토콜로 제공해야 한다. 다수의 인터넷 사물이 연결되기 위해서는 최종적인 서비스가 관심 있는 데이터를 습득할 수 있어야 한다. 이 서비스를 위한 데이터는 특별한 방식이 아닌 인터넷 프로토콜을 통해 제공해야 다수의 기기를 통한 서비스가 가능하다

2.3 현재 시장 IoT 제품의 상황

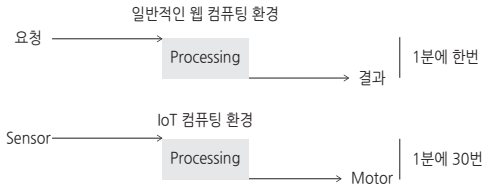
현재 시장에 나오고 있는 제품들이 이 두 기능을 모두 제공하지 못하는 이유는 동일 인터넷 사물기기를 통한 네트워크 서비스를 벗어나지 못한 서비스 제공환경이 부족하기 때문이다. 이 제공 환경은 구체적으로 IoT 플랫폼과 소비자의 서비스 기대이다. 요즘 IoT 제품을 출시하는 대부분의 규모가 영세한 스타트업은 단순 센서 수집과 이를 이용한 서비스 넘어 플랫폼으로 제공하기에는 매우 벅찬 일이다. 뿐만 아니라 이종 기기를 연결해서 추가 투자를 감수할 수 있을 정도로 매력적인 서비스의 경험이 없기때문에 IoT 서비스에 대한 사용자의 기대도 낮은 편이다. 따라서 비용이 많이 드는 플랫폼 대신 앱세사리 형태로 제공한다.

3. 웹 프로토콜 기반의 인터넷 사물의 장벽과 웹소켓

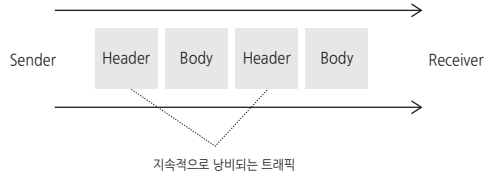
인터넷 사물이 되기 위한 식별은 과거의 인터넷부터 데이터와 서비스를 구분하기 위해서 사용했기 때문에 매우 익숙하고 쉽게 제공할 수 있다. 문제는 통신이다. 사용자가 폼(Form)을 완성하여 제출(Submit) 버튼을 눌러서 클라이언트의 데이터가 서버로 수동으로 전달되는 데이터 전달 방식에 비해 인터넷 사물은 사용자의 개입 없이 지속적으로 서버나 플랫폼으로 데이터가 전달된다.

빈도와 양에서 IoT 컴퓨팅 환경은 전통적인 웹 컴퓨팅 환경을 압도한다. 이러한 컴퓨팅 환경에서 기존 HTTP 전송방식을 사용하는 것은 부하가 크다. 왜냐하면, HTTP 1.1로 대표되는 전통적인 웹 통신 방식은 실 데이터에 비해 헤더의 양의 크기 때문이다. 네트워크로 전송하는 데이터는 헤더와 보디 영역으로 나뉜다. 헤더는 데이터의 성격을 설명하는 메타정보를 포함하고 있고 보디 영역은 이름, 주소와 같은 실제 데이터를 포함하고 있다.

HTML5.0 스펙에 포함된 웹소켓은 이러한 환경에서 효과적으로 통신할 수 있는 방법을 제공하고 있다. 통신을 시작하기 위해 핸드쉐이킹 이후에 실제 데이터만 주고 받아서 트래픽을 최소화하고 있다. 물론 이 방법은 굉장히 단순한 방법이지만 별도의 방법



[그림 1] 전송 방식 비교



[그림 2] 통신 패킷 구조

```
GET /PollingStock//PollingStock HTTP/1.1
  Host: localhost:8080
  User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.1.5)
  Gecko/20091102 Firefox/3.5.5
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
  Accept-Language: en-us
  Accept-Encoding: gzip,deflate
  Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
  Keep-Alive: 300
  Connection: keep-alive
  Referer: http://www.example.com/PollingStock/
  Cookie: showInheritedConstant=false; showInheritedProtectedConstant=false;
  showInheritedProperty=false; showInheritedProtectedProperty=false;
  showInheritedMethod=false; showInheritedProtectedMethod=false;
  showInheritedEvent=false; showInheritedStyle=false; showInheritedEffect=false
```

[그림 3] 실제 HTTP 1.1 요청 해더 예

없이 웹 기술만으로 서비스 제공이 가능한 것은 인터넷 사물이 되기 위한 조건 중 통신 부분을 매우 훌륭히 만족시킨다.

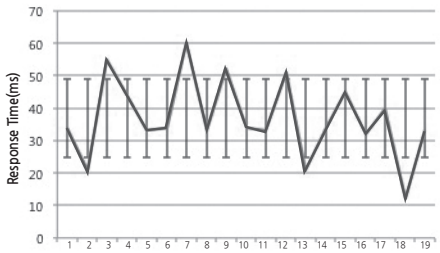
4. 마이크로컨트롤러에서의 웹소켓

한정된 자원 환경(메모리와 연산)에서 웹소켓의 효율성을 시험하기 위해 시험을 진행했다. 아두이노와 와이파이를 연결하여 네트워크를 구성하였다. 80바이트 정도의 센서 데이터를 전송하였다. 실험의 결과는 [그림 4], [그림 5]와 같다.

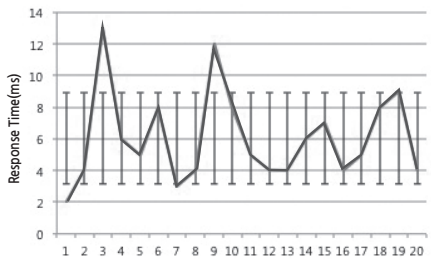
HTTP 1.1 방식이 평균 응답속도가 35ms인 것에 비해 웹소켓 방식은 6ms로 6배 정도의 응답속도를 보였다. 별도의 프로토콜없이 표준기술만으로 매우 빠르게 센서데이터를 전송할 수 있다. 데이터 전송량을 비교해보자. 데이터 전송량이 많으면 더 많은 에너지를 사용하게 된다. HTTP 1.1 방식에서는 요청과 응답 헤더의 크기가 800바이트 정도된다. 웹소켓에서 하프듀플랙스로 통신하였을 경우에는 헤더의 크기는 6바이트이다. 이러한 장점에도 불구하고 웹소켓은 프론트앤드(화면)영역 기술이라는 인식이 강하다. 웹소켓 기술 공개됐을 때의 데모들이 좀더 부드럽게 웹으로 연결되는 내용의 데모였기 때문이다.

5. 웹소켓으로 연결되는 인터넷 사물 RC카

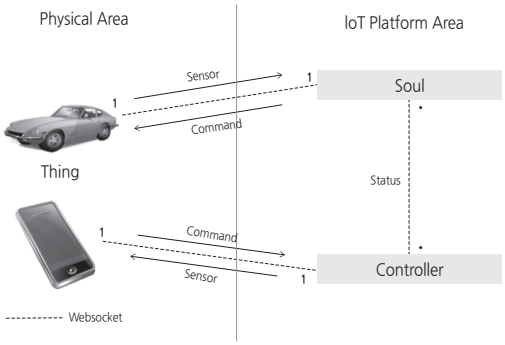
마이크로컨트롤러를 통한 네트워크 테스트보다 더 실제 IoT 서비스 상황과 비슷한 환경을 테스트한다. 원격 조정 자동차를 웹소켓으로 연결하는 실험이다. 이 테스트를 선택한 이유는 많은 센서 데이터를 실시간으로 전송해야 하고, 이를 확인한 사용자의 명령을 실시간으로 움직이는 자동차에 전송해야



[그림 4] HTTP 1.1 방식의 응답속도



[그림 5] 웹소켓 방식의 응답속도



[그림 6] IoT 네트워크 연결도

```
var wSocket = new WebSocket(
"wss://192.168.0.6:8090/controller/car/258f3b51-a292-42c0-9525-668c79d73007");

wSocket.onmessage = function(e) {
        if(e.data != null) {
            CarController.sensorData = JSON.parse(e.data);
        }
}
```

[그림 7] 웹소켓 초기화 코드

하기 때문에 기술 요건이 어려운 시나리오를 선택하여 IoT와 웹소켓을 결합한 결과에 대한 증명으로 적합하다고 판단했기 때문이다.

센서가 수집되는 기기로부터 플랫폼까지, 이 센서 데이터를 받아 사용자에게 보여주고 사용자의 지시를 다시 기기로 보내는 전구간을 웹소켓으로 통신한다. 10ms마다 전송된다. RC카가 주고 받는 데이터는 다음과 같다.

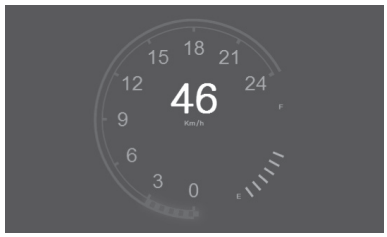
1. Led
2. Light Sensor
3. Network Signal Strength
4. Power Source Level
5. Motor Temperature
6. Electronic Speed Controller Temperature
7. Motor RPM
8. Electronic Speed Controller Volt
9. Electronic Speed Controller Ampere
10. Throttle
11. Steer
12. Avoid Collision
13. Anti-Brake System

IoT Car를 모니터링 하거나 제어할 때는 [그림 7]의 웹소켓 초기화 코드만으로 프로그래밍이 가능하다. 간단한 초기화와 개념만으로 프로그래밍이 가능하다.

실험 결과를 글로 표현하면 이전 마이크로컨트롤러에서의 결과형식이기 때문에 결과를 확인하기에는 부적합하다고 생각한다. 실험 결과는 유튜브 링크(http://www.youtube.com/watch?v=CVt-B5T-2Gk)를 통해 확인할 수 있다.

본 실험에서는 기기에서 발생하는 데이터를 웹소켓으로 플랫폼에서 프로세싱하는 동일한 데이터 프로세싱 아키텍처를 사용하여 좀 더 효과적으로 모니터링/제어 환경도 구성하였다. 이를 통해 자동화 소프트웨어 엔지니어뿐만 아니라 프론트엔드 엔지니어까지 동일한 방법으로 데이터 처리가 가능하다. 이 부분은 시스템을 간결하게 해주고 서비스 개발의 난이도를 낮춰준다. 앞서 나열한 웹소켓의 이점과

[그림 8] 웹소켓으로 연결되어 센서를 전송하고 명령을 받는 IoT Car(인터넷 사물 자동차)



[그림 9] 웹소켓으로 연결되어 IoT Car를 모니터링하고 제어할 수 있는 HTML5 화면

결합하여 IoT에 최고의 네트워크 환경을 제공한다.

6. 맺음말

웹소켓이 사물인터넷에 매우 중요한 이점을 주고 있다. 웹소켓은 이미 완성된 표준 통신 기술을 제공하여 2년 정도 안에 배포된 거의 모든 웹서버와 웹 에이전트에서 쉽게 쓸 수 있다. 높은 레이턴시의 데이터를 낮은 트래픽으로 전송할 수 있기 때문에 한정된 자원 환경에서도 고효율의 통신을 할 수 있다. 인터넷 사물은 사물에서 플랫폼으로 센서 데이터뿐만 아니라 플랫폼에서 사물로 전송되는 명령어를 양방향으로 전송할 수 있어야 하기 때문에 양방향 통신이 가능해야 한다. 사물인터넷은 기기 간의 통신뿐만 아니라 여전히 사용자의 스크린에도 정보를 전달해야 한다. 웹소켓은 모니터링과 제어도 센서 전송과 동일한 방식으로 통신을 가능하게 해준다. 이미 구축된 웹환경(방화벽 등 네트워크 환경)에도 이질감 없이 친화적으로 웹소켓을 쓸 수 있다. 이런 이점들을 활용하여 가치 있는 사물인터넷 서비스를 제공할 수 있기를 바란다.

[참고문헌]

[1] Internet Engineering Task Force, RFC 6455

[2] WebSocket.org, A Quantum Leap in Scalability for the Web

[3] OASIS, oasis-open.org

[4] IoT 사물의 스크린 UX 설계와 구현, http://angeliot.blogspot.kr/2014/08/iot-ux.html