Journal of The Korea Society of Computer and Information
Vol. 20 No. 9, pp. 97-104, September 2015
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2015.20.9.097

Study on Web Services Middleware for Real-Time Monitoring in the IoT Environment

Seung-Hyeok Shin *

Abstract

Recently, the need for real-time systems which are providing various types of information that occur in large quantities in IoT environment is increasing. In this paper, we propose a middleware system which can monitor in real time on a web environment. The proposed system is designed to be integrated by using communication functions provided by a network operating system and external sensors. The proposed system is compared with an existing system and analysed by the server performance testing tool.

▸ Keyword : IoT, Middleware, Web Socket, Big Data

I. Introduction

현대의 인터넷 서비스는 이용자로 부터 생성되는 신규 서비스에 대한 요구를 수용하며 발전하고 있다. 이러한 서비스의 발전은 지역적으로 국한 되었던 기존의 사용자 중심의 서비스에서 인터넷으로 확장된 사물과 서비스, 그리고 사용자가 융합하는 사물인터넷(Internet of Thigns) 서비스로 확산되고 있다[1]. 사물인터넷 서비스의 확산은 웨어러블 스마트 디바이스(Wearable Smart Device)로 대표되는 사용자 중심의 서비스 분야에서부터 제품을 생산하는 제조 산업의 생산 자동화 시스템 분야, 설비 산업의 설비 운영 자동화 시스템 분야와 같이 다양한 형태의 산업으로 확산되고 있는 추세이다[2]. 자동화 시스템 분야로의 사물인터넷 서비스 확산은 신규 센서 출현과 이로 인한 신규 서비스의 창출을 의미한다. 시장조사 기관 IC Insights에서는 그림 1과 같이 사물인터넷 관련 산업용 센서가 지속적으로 증가할 것으로 예측하고 있다[3].

• First Author: Seung-Hyeok Shin, Corresponding Author: Seung-Hyeok Shin
*Seung-Hyeok Shin (shinbaad@gmail.com), Dept. of Cyber Security, Gumi University
• Received: 2015. 08. 18, Revised: 2015. 09. 07, Accepted: 2015. 09. 21.

Fig. 1. IoT sensor growth

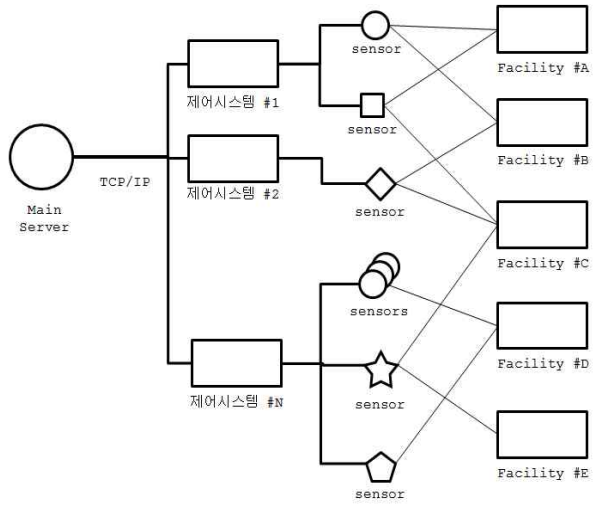

Fig. 2. Configuration for IoT facilities controlling systems

자동화 시스템을 구성하는 각 요소는 그림 2와 같이 설비(Facility), 센서(Sensor), 제어시스템(Control System) 그리고 메인 서버(Main Server)로 구성한다. 설비를 제어하기 위한 센서는 유/무선으로 구성된 설비를 제어하기 위하여 다양한 기능으로 구성이 된다. 센서의 다양한 구성은 각 설비의 규격과 제어를 위한 통신 방식에 의하여 결정된다. 제어 시스템은 다양한 제어 규격에 의하여 생성되는 정보를 센서로 부터 수신하여 중앙의 메인 서버로 전송하기 위한 파사드(Facade) 구조의 미들웨어 서비스 시스템으로 구축한다.

요청(Request)과 응답(Response)의 구조인 기존의 웹 방식으로는 실시간 표현에 한계가 있다. 요청/응답 구조의 웹 방식은 클라이언트에서 서버 측으로 정보를 담고 있는 URL에 대하여 요청한다. 클라이언트의 요청을 수신한 서버는 해당 URL에 대한 응답을 송신하고, 응답을 수신한 클라이언트는 웹 화면에 응답 정보를 도시한다. 이러한 요청/응답 과정은 프로토콜의 세부 구성 상 연결-요청-응답-연결해제 4단계의 네트워크 트래픽이 발생하게 된다. 이러한 구성으로 실시간 정보를 갱신하기 위해서는 전체 URL를 다시 요청해야 하는 네트워크 부담감이 존재한다. 또한 정보의 요청은 반드시 클라이언트에 의하여 먼저 요청이 되어야 하는 단방향 통신의 문제점도 존재한다. 이러한 웹 환경에서의 단방향 통신 문제를 해소하기 위한 기존의 방법들이 존재한다. 웹기반의 양방향 통신을 위한 대표적인 방법은 Adobe사의 플래시(Flash), 마이크로소프트의 ActiveX, 실버라이트, 그리고 오라클의 자바 애플릿 등이다. 이러한 방법들은 웹 환경에서 화려한 동작과 뛰어난 상호작용을 보장하지만 순수 웹 환경이 아니라 별도의 런타임(run time) 플러그인 형태로 브라우저에 설치해야 가능하다. IETF RFC6455의 웹 소켓은 기존의 웹 방식을 개선하여 표준화된 웹 응용 환경하에서 클라이언트와 서버간의 연결을 설정하고 그 연결을 이용하여 양방향 통신이 가능하도록 규정하고 있으며, 웹 표준인 HTML5, CSS3와 jQuery를 이용하여 웹 소켓 서버와 웹 소켓 클라이언트로 구현이 가능하다. 그림 3은 기존의 웹 방식과 웹 소켓 방식의 요청/응답 방식을 도시한다.

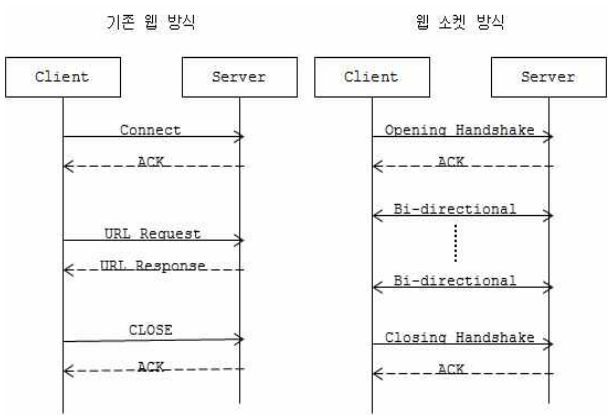

Fig. 3. Method of request/response on the web

웹 환경에서 실시간으로 정보를 도시할 수 있는 웹 소켓은 웹 소켓 서버(Web Socket Server)와 웹 소켓 클라이언트(Web Socket Client)로 구성한다. 웹 소켓 클라이언트는 웹 표준 브라우저를 이용하여 실시간 정보의 도시가 가능하다. 이러한 웹 소켓 클라이언트에 정보를 송신하기 위한 웹 소켓 서버가 구성되어야 한다. Node.js는 자바스크립트(JavaScript)로 구현이 가능한 웹 소켓 서버 플랫폼이다. 웹 표준 브라우저에 탑재되어 운영되는 웹 소켓 클라이언트와 통신이 가능하고, 실시간 정보를 생성하여 전송 할 수 있다.

그림 4는 자동화 시스템을 구성하는 제어 시스템의 구성도를 도시한다. 제어 시스템은 센서에서 발생하는 정보를 수집하고 인터넷에 연결된 웹 클라이언트와 저장서버(Storage Server)로 정보를 전송하는 기능을 담당한다. 자동화 시스템을 구성하는 센서는 고속의 빅 데이터를 발생시킨다. 이를 효율적으로 처리하기 위해서는 스크립트 언어인 자바스크립트 보다는 시스템 자원을 효율적으로 제어 할 수 있는 C/C++로 시스템을 구성하는 것이 효율적이다. 따라서 자동화 시스템의 특성상 Node.js를 제어 시스템에 활용하기에 부적합한 경향이 있다.

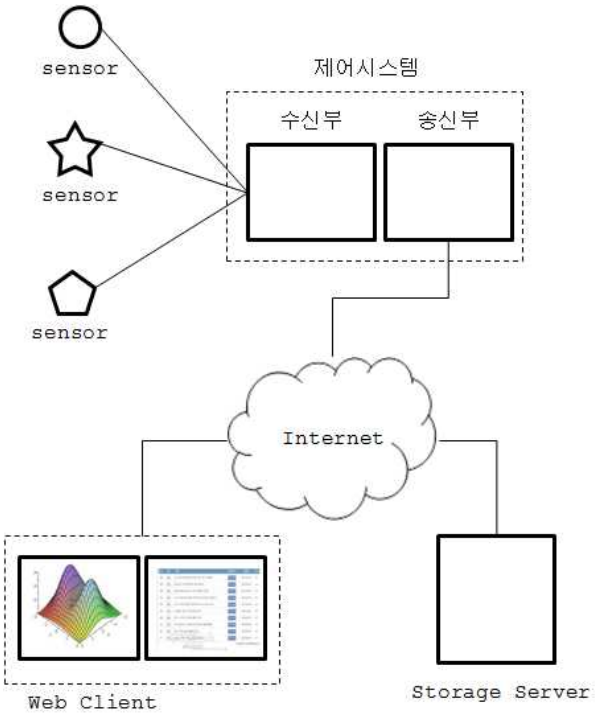

Fig. 4. Control system architectures

본 연구에서는 자동화 시스템에서 생성되는 고속의 빅 데이터를 웹 환경에서 실시간으로 도시하기 위한 시스템을 제안한다. 이를 위하여 제어시스템 구성에 적합한 웹 소켓 서버를 탑재한 웹 서비스 미들웨어를 설계한다. 논문의 전체 구성은 다음과 같다. 2장에서는 실시간 웹 서비스를 위한 기술인 웹 소켓과 Node.js 기술을 연구한다. 3장에서는 제안하는 시스템에 대한 요구사항을 정리하고, 시스템 설계 및 구현을 진행한다. 4장에서는 제안된 시스템에 대하여 평가 한 후, 향후 연구 계획에 대하여 논의한다.

II. Preliminaries

본 장의 1절에서는 IETF RFC6455의 웹 소켓에 대하여 고찰하고 2절에서는 대표적인 웹 소켓 오픈 소스인 Node.js에 대하여 기술한다.

1. WebSocket

기존의 웹 방식은 웹 클라이언트인 웹 브라우저에서 웹 서버로 URL를 요청하고, 웹 서버는 해당 웹 클라이언트로 응답을 송신하는 구조이다. 웹 클라이언트에서는 수신된 응답을 이용하여 정보를 갱신한다. 웹 클라이언트에서는 수신된 정보를 화면에 도시하기 위하여 웹 브라우저 전체를 갱신하여야 한다. 이때 웹 브라우저의 깜빡임이 발생하게 되며, 사용자의 시각적인 피로감을 증가시키는 원인이 된다. 이를 해소하기 위하여 iFrame을 이용하여 숨겨진 프레임을 이용하거나 Long Polling, Stream 등의 방식을 사용하기도 한다. 그러나 이러한 방식은 네트워크 트래픽을 발생시키며, 전체적인 네트워크 부하를 증가시키는 단점이 있다[4][5]. IETF RFC6455에 정의된 웹 소켓은 웹 브라우저상에서 웹 서버와 양방향 통신이 가능한 TCP 기반의 웹 소켓 표준 프로토콜을 기술한다[6].

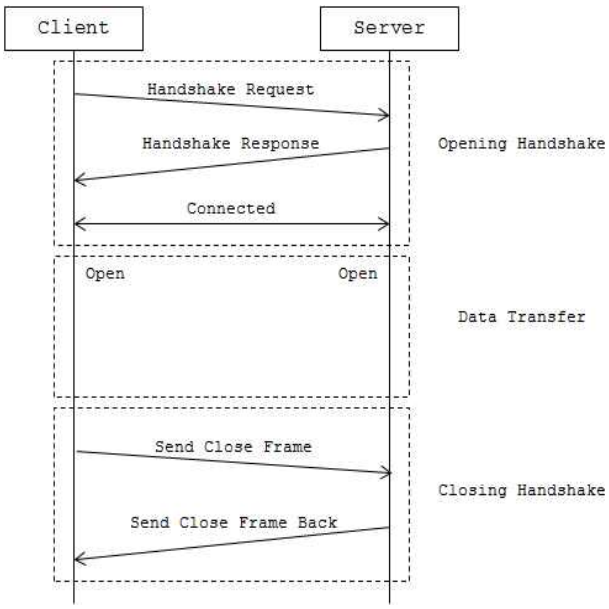

Fig. 5. WebSocket protocol

그림 5는 웹 소켓 프로토콜을 도시한다. 웹 소켓은 웹 서버와 웹 클라이언트간의 양방향 통신을 지원하기 위하여 연결(Connection)을 설정하고 해제하는 과정으로 기술된다. 웹 서버와 웹 클라이언트가 양방향 통신을 수행하기 전에 먼저 Opening Handshake를 수행하여 연결을 설정한다. 연결이 설정된 웹 서버와 웹 클라이언트는 상시적으로 정보를 송수신 할 수 있다. 이러한 방법은 푸시 서버(Push Server)의 기본으로 네트워크의 트래픽을 줄일 수 있는 장점이 있다. 그림 6은 기존 웹 방식에서의 Polling 과 웹 소켓의 통신 방식 비교하여 도시한다.

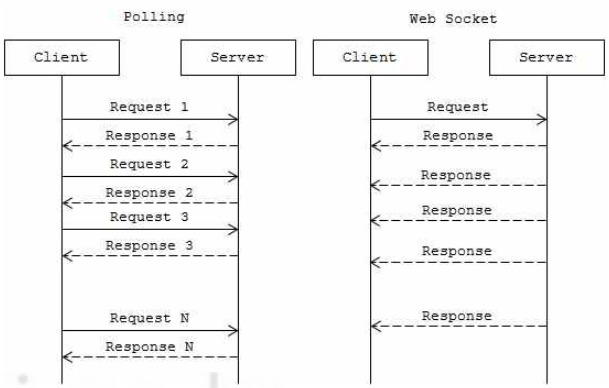

Fig. 6. Comparison of polling and web socket

2. Node.js

Node.js는 Google의 크롬 V8 자바 스크립트 엔진을 기반으로 제작된 고성능의 네트워크 서버이다. 단일 쓰래드(Thread)로 동작되는 Node.js는 이벤트 방식의 비동기식 입출력을 처리하는 미들웨어 서버이다. 기존의 네트워크 서버는 다중 클라이언트에 대한 접속 및 해제, 그리고 요청을 처리하기 위한 개별의 쓰래드 혹은 쓰래트 풀(Pool)로 구성되는 것이 일반적인 방법이다[7-9]. 이러한 네트워크 서버 방식은 클라이언트의 새로운 요청을 추가하기 위하여 메모리를 지속적으로 할당하도록 구성한다. 이 경우 메모리 소모량이 폭주하여 네트워크 서버의 성능이 저하되는 단점이 있다. Node.js는 단일 쓰래드와 비동기 입출력으로 구성되어 다중 클라이언트의 연결 시 메모리 사용량을 효율적으로 운영할 수 있는 장점이 있다. 또한 비동기식 입출력 방식을 사용하기 때문에 CPU의 사용률이 높은 장점이 있다. Node.js의 비동기 입출력은 파일, 네트워크 그리고 데이터베이스 입출력 등의 요청이 완료 된 후 결과를 이벤트 방식으로 수신하여 처리하는 구조이다. 그림 7은 동기식 입출력 방식과 비동기 입출력 방식을 비교하여 도시한다.

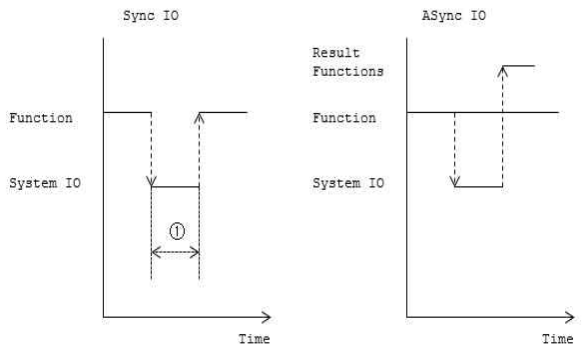

Fig. 7. Comparison of Synchronous/ASynchronous IO

동기식 입출력의 경우 주 기능인 Function에서 System IO가 발생 할 경우, System IO가 완료될 때까지 시간 ① 만큼의 지연이 발생한다. 그러나 비동기식 입출력의 경우 System IO를 호출한 후 지속적인 작업을 진행하며, System IO의 완료된 결과는 별도의 결과 기능에서 처리하도록 한다. 이러한 기능에 의하여 구성된 Node.js는 웹 소켓 클라이언트를 위한 웹 소켓 서버 개발 시간을 효율적으로 단축할 수 있는 장점이 있다. 그러나 단일 쓰래드 모델의 특성 상, 시간이 오래 걸리는 작업에 의하여 전체 시스템의 성능이 급격히 저하되는 단점이 있다[10][11].

III. The Proposed System

본 장의 1절에서는 웹 서비스 미들웨어에 대한 기본적인 요구사항을 기술하고, 2절에서는 1절에서 제시된 요구사항을 기반으로 시스템을 설계한다. 마지막 3절에서는 설계된 시스템을 이용하여 웹 서비스 미들웨어를 구현한다.

1. 시스템 요구사항

제어 시스템은 센서에 의하여 제어되는 설비를 효율적으로 운영하기 위하여 다음과 같은 기능이 만족해야 한다. 첫째, 각종 센서와 양방향 통신이 가능하도록 구성되어야 한다. 둘째, 센서에서 수집된 정보를 메인 서버로 전송이 가능하도록 구성이 되어야 한다. 셋째, 제어 설비의 증설 및 이동, 그리고 신규 센서의 추가로 인한 자동화 소프트웨어의 업그레이드 및 배포가 용이해야 한다[12-16].

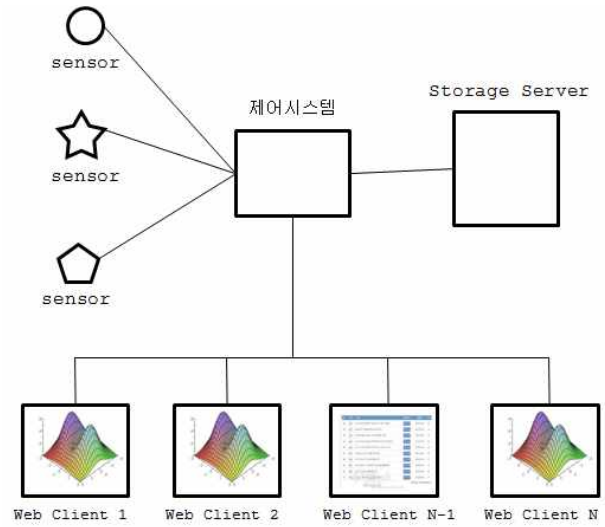

Fig. 8. System configurations

그림 8은 시스템의 요구사항에 적합한 구성도를 도시한다. 센서와 양방향 통신을 위하여 1대 1 연결 구조에 의하여 구성하며, 요청/응답 방식으로 센서의 정보를 수집한다. 수집된 정보를 메인 서버로 전송하기 위하여 다양한 형태의 센서와 메인 서버를 연결하는 파사드 구조의 미들웨어 시스템으로 구성한다. 설비의 증설, 교체, 유지 보수등 일시적인 센서 통신 문제와 더불어 다양한 클라이언트에 정보를 송신하고 별도의 소프트웨어 설치나 변경 없이 지속적인 서비스를 보장하도록 웹 기반으로 시스템을 구축한다.

2. 시스템 설계

자동화 시스템을 구성하는 제어 시스템은 센서와 메인 서버 사이에 위치한 전형적인 미들웨어 서비스 시스템 형태로 구성한다. 센서와 통신하기 위한 제어시스템을 유연하게 구현하기 위하여 공통된 기능을 제공하는 미들웨어를 기반으로 자동화 소프트웨어를 설계한다. 그림 9는 제어 시스템을 구성하는 자동화 소프트웨어의 시스템 계층 구조를 도시한다.

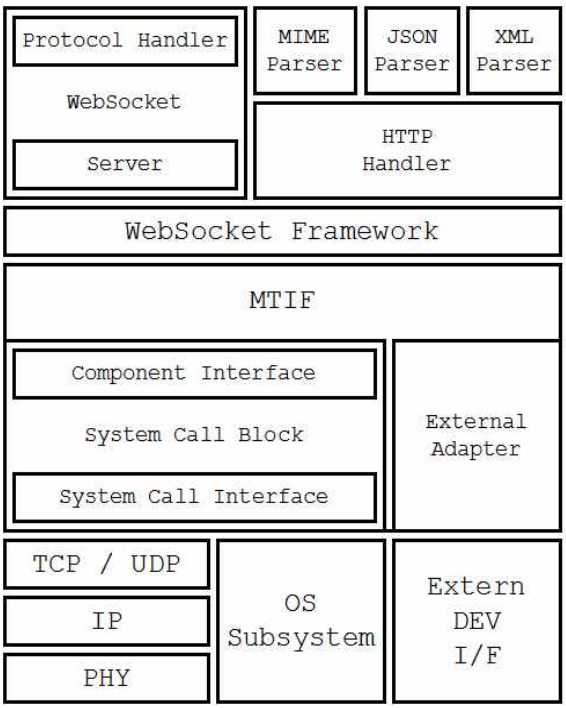

Fig. 9. System layered architecture

제안하는 시스템은 네트워크 운영체제에서 제공하는 통신 기능과 외부 센서별 통신 기능을 공통적으로 사용하기 위한 미들웨어인 MTIF(Middle-Tier Interface)를 설계한다. 또한 Application을 Interface하기 WebSocket Framework를 설계한다. MTIF는 System Call Block과 External Adapter로 구성된다. System Call Block은 TCP/IP 통신 기능과 운영체제에서 제공하는 Thread, Signal, Process, Event 및 File 등의 System call을 처리하기 위한 System Call Interface, Application인 WebSocket Framework가 연결하기 위한 Component Interface로 구성된다. External Adapter는 IoT 환경에서 서로 다른 기능을 제공하는 센서를 연결하기 위한 기능을 제공한다.

3. 시스템 구현

MTIF에서는 TCP/IP 통신을 위한 socket 기능을 제공한다. 제공되는 socket은 TCP, UDP 프로토콜에 대하여 클라이언트와 서버 기능을 각각 설정하도록 구성한다. 또한 시스템의 통신 방식 제어를 위하여 Unicast, Broadcast, Multicast도 입력 파라미터에 의하여 결정하도록 한다. 웹 소켓 서버는 MTIF의 TCP 서버 방식으로 운용된다. MTIF의 TCP 서버에서는 접속되는 클라이언트에 대한 정보를 유지하고 효율적인 통신을 지원하기 위하여 그림 10과 같은 체이닝 기반의 해시 테이블을 이용한다. 해시 테이블은 Top Node를 기준으로 클라이언트와 통신하기 위한 쓰래드를 생성한다. Sub Node의 클라이언트 요청은 Top Node에 의하여 생성된 쓰래드에서 처리하도록 구성한다.

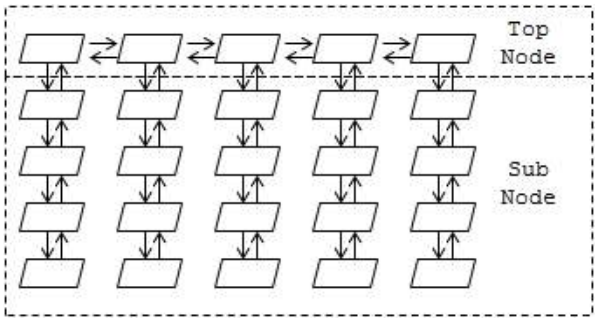

Fig. 10. Structure of the client's hash table

```
int add_to_next(xdlist* _this, void* arg, int size){
  int e = 0;
  if ( _this->_curr == 0 ) {
    e = _this->vtbl->create(_this, &_this->_curr);
    if ( arg ) {
      _this->_curr->item = (void *)calloc(1, size);
      memcpy(_this->_curr->item, arg, size);
      _this->_curr->size = size;
    }
    return e;
  }
  if ( _this->_curr->next == 0 )
    e = _this->vtbl->append(_this, arg, size);
  else {
    e = _this->vtbl->create(_this, &_this->_temp);
    _this->_temp->next = _this->_curr->next;
    _this->_curr->next->prev = _this->_temp;
    _this->_temp->prev = _this->_curr;
    _this->_curr->next = _this->_temp;
    _this->_curr = _this->_temp;
    if ( arg ){
      _this->_curr->item = (void *)calloc(1, size);
      memcpy(_this->_curr->item, arg, size);
      _this->_curr->size = size;
    }
  }
  return e;
}
```

Fig. 11. Add algorithms of hash table with channing

```
int remove(xdlist* _this, void** arg, int* size){
  int e = 0;

  e = _this->vtbl->get_last(_this, 0, 0);
  if ( e<0 )
    return e;

  if ( arg ) {
    memcpy(*arg,
           _this->_curr->item, _this->_curr->size);
    *size = _this->_curr->size;
  }

  free(_this->_curr->item);
  _this->_curr->item = 0;
  _this->_curr->size = 0;

  if ( _this->_curr->prev ) {
    _this->_curr = _this->_curr->prev;
    _this->_curr->next->prev = 0;
    _this->vtbl->destroy(_this, &_this->_curr->next);
    _this->_curr->next = 0;
  }
  else
    _this->vtbl->destroy(_this, &_this->_curr);

  return e;
}
```

Fig. 12. Remove algorithms of hash table with channing

```
int find(xdlist* _this, void* arg, int size) {
  int e = 0;
  xdlist_t * _temp;

  if(_this->_curr==0)
    return eLIST_EMPTY;

  if(memcmp(_this->_curr->item,arg,size)==0)
    return 0;

  _temp = _this->_curr;

  e = _this->vtbl->find_prev(_this, arg, size);
  if ( e < 0 ) {
    _this->_curr = _temp;
    e = _this->vtbl->find_next(_this, arg, size);
  }

  return e;
}
```

Fig. 13. Find algorithms of hash table with channing

그림 11, 그림 12, 그림 13은 체이닝 기반 해시 테이블에서의 추가, 삭제, 검색 알고리즘을 나타낸다. 그림 11은 신규 클라이언트 접속 시 추가 알고리즘을, 그림 12는 접속 해제 시 제거 알고리즘을 나타낸다. 그림 13은 접속한 클라이언트의 검색 알고리즘을 나타낸다. 그림 14는 웹 소켓 서버의 Opening Handshake 알고리즘을 나타낸다.

```
int WebSocketHandShake(char* serverkey, int len) {
  char buf[1024];
  char temp[1024];
  int i=0;
  SHA1Context sha;

  SHA1Reset(&sha);
  SHA1Input(&sha, serverkey, len);
  SHA1Result(&sha);

  for (i = 0 i < 5 i++)  {
    sha.Message_Digest[i]
     = htonl(sha.Message_Digest[i]);
  }

  base64_encode((char*)sha.Message_Digest,20,temp);

  sprintf(buf,"HTTP/1.1 101 Switching Protocols₩r₩n"
              "Upgrade: websocket₩r₩n"
              "Connection: Upgrade₩r₩n"
              "Sec-WebSocket-Accept: %s₩r₩n₩r₩n",
              temp);

  return  WebSocketSend(buf, strlen(buf));
}
```

Fig. 14. Web Socket Server Opening Handshake

IV. Validation

제안한 미들웨어인 MTIF는 통신을 주요 기능으로 하는 MOM (Message Oriented Middleware)으로 MTIF의 성능을 측정하기 위하여 대표적인 웹 서버 Open Source 프로젝트인 Apache Group에서 진행하는 JMeter를 이용하였다. 테스트 방식은 가상의 Client user를 100, 500, 1000으로 생성하여 Server의 응답시간(Response Time)을 측정하였다.

Table 1. Response time of node.js

	100 user	500 user	1000 user
MIN (msec)	1	2	1
MAX (msec)	76	244	846
AVG (msec)	6	20	15

Table 2. Response time of MTIF

	100 user	500 user	1000 user
MIN (msec)	1	1	1
MAX (msec)	19	32	54
AVG (msec)	5	12	16

표 1과 표 2는 Node.js와 MTIF를 이용하여 구성한 웹 소켓 서버의 응답시간을 측정한 결과이다. 단일 쓰레드 비동기 입출력 처리 방식의 Node.js 와 MTIF 기반 웹 소켓 서버는 각 100 user의 처리율에 있어 크지 않은 차이를 나타낸다. 그러나 500

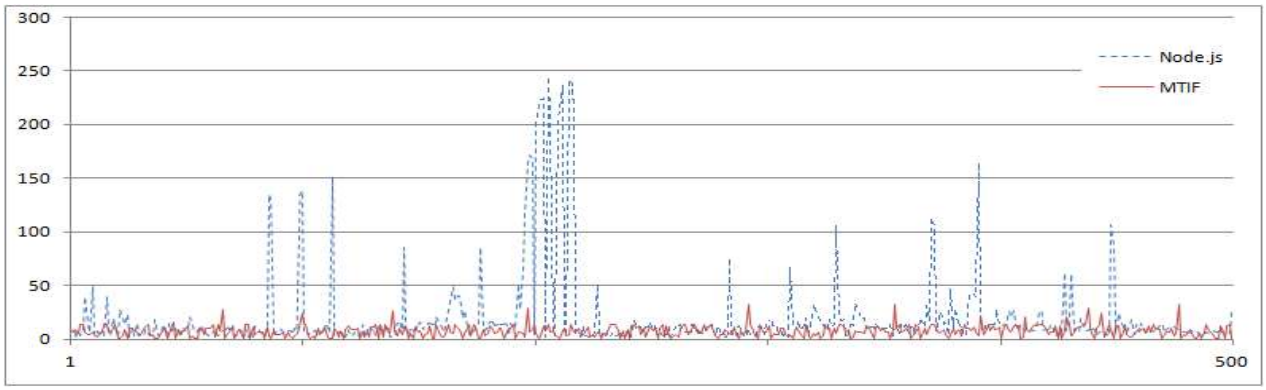

Fig. 15. Comparison of response time about 500 user

user와 1000 user 등으로 load를 발생 시키면서 테스트 한 결과 제안한 시스템이 기존 Node.js 방식보다 2배 이상의 효율적인 처리율을 나타내는 것을 확인 할 수 있었다. 그림 15는 Node.js와 MTIF의 500 user 응답시간을 비교한 그래프이다. Node.js의 경우 응답 시간 지연이 발생하는 구간이 빈번히 나타나는 것을 확인 할 수 있다. MTIF의 경우 전체 구간에서 균일한 응답시간을 나타내는 것을 확인 할 수 있다. 이는 단일 Thread에 의하여 동작하는 시스템의 특성상 일부 응답 지연에 의하여 전반적인 처리율이 낮아지는 현상으로 나타난다.

V. Conclusion

본 논문에서는 IoT 환경에서 발생하는 센서의 데이터를 웹을 이용하여 실시간으로 모니터링 할 수 있는 웹 기반의 미들웨어에 대하여 제안하였다. 통신 기능을 담당하는 미들웨어를 구현하고 웹 환경에서 실시간 모니터링이 가능한 웹 소켓 서버를 구현하였다.

Node.js는 다중 사용자 혹은 서버 측의 부하가 발생하는 대규모 시스템에서 성능 저하 현상이 나타남을 실험을 통해 확인할 수 있었다. 또한 동일 실험에 의하여 Node.js의 단점을 보완한 MTIF 기반의 웹 소켓 서버는 대규모 시스템에 적합함을 보였다. 제안된 MTIF는 MOM으로 웹 소켓 서버 외에 실시간으로 정보를 전송하기 위한 시스템에 적용이 가능할 것으로 판단된다.

현대의 정보는 기존의 정량화 된 정보 이외 멀티미디어 형태의 정보 등 다양한 형태의 정보로 생성되고 있으며, 폭발적으로 증가하고 있다. 대량의 정보를 효율적으로 처리하기 위하여 실시간 및 대량 정보를 처리 할 수 있는 시스템이 필요하다. 제안된 MTIF는 대량의 정보를 실시간으로 처리하기 위하여 제안되었다. 향후 통신 기능 외에 사용자 환경에 적합한 GUI 등을 공통적으로 제공할 수 있는 Framework 등으로 개발이 진행 된다면, 네트워크 환경에서 보다 효율적으로 시스템을 구축할 수 있는 방안이 마련될 것으로 판단된다.

REFERENCE

[1] H. J. La, C. W. Park and S. D. Kim, "A Framework for Effectively Managing Dynamism of IoT Devices," Journal of Korean Institute of Information Scientists and Engineers: Software and Applications, Vol. 41, No. 8, pp.545-556, Aug. 2014.

[2] S. H. Lee "Various Meanings for Internet of Things," Proceedings of Korean Institute of Information Technology Summer Conference, pp. 131-133, May,2014.

[3] IC Insights, http://www.icinsights.com/

[4] Y. H. Kang, "Design of a Seamless Messaging Application using WebSocket in IoT Environment," Proceedings of Korean Institute of Information Technology Summer Conference, pp. 245-247, May. 2014.

[5] J. T. Park, H. S. Hwang and I. Y. Moon, "Implementation of Smart TV Application using HTML5 and Health Bicycle," Journal of Advanced Navigation Technology, Vol. 18, No. 1, pp. 101-106, Feb. 2014.

[6] RFC6455, http://www.websocket.org/

[7] Y. Zhangling and D. Mai, "A Real-Time Group Communication Architecture Based on WebSocket," Internation Journal of Computer and Communication Engineering, Vol. 1, No. 4, pp. 408-411, Nov. 2014

[8] S. Guoqing, Z. Wei, L. Huajun, and Z. Peng, "Survey on Server-Push Technology of Web Applications," Computer Systems and Applications, Vol. 18, 2008.

[9] P. Victoria and G. N. Bradford, "Communicating and Diaplaying Real-time Data with WebSocket," Internet Computing, Vol. 16, No. 4, pp. 44-53, Jul. 2012.

[10] A. Wessels, M. Purvis, J. Jackson. and S. Rahman, "Remote Data Visualization through WebSockets,"

Proceedings 8th. Annu. International Conference Information Technology: New Generations, pp. 1050-1051, 2011.

[11] P. Lubbers and F. Greco, "HTML5 Web Sockets: A Quantum Leap in Scalabilty for the Web," SOA World Magazine, Jun. 2011.

[12] K. S. Choi, H. G. Jun and H. J. Yoo, "A Development of Software for monitoring Data Logger," Journal of Korean Institute of Information Technology, Vol. 2, No. 1, pp. 103-108, Mar. 2004.

[13] S. Y. Joo, H. S. Lim and D. S. Kang, "Effective Memory Structure for Middleware for Collecting Data in M2M Network," Proceedings of Korean Institute of Information Technology Summer Conference, pp. 167-170, May. 2014.

[14] Y. U. Jo, H. T. Chung and K. R. Kwon, "Development of Service Information System with Real Time Using RFID Middleware Based on 3-Tier Structure," Journal of Korean Institute of Information Technology, Vol. 5, No. 1, pp. 171-180, Mar. 2007.

[15] H. J. Yoo, "Design and Implementation of a Web-based Framework for Creating Flash based on Web," Journal of Korean Institute of Information Technology, Vol. 6, No. 1, pp. 96-102, Feb. 2008.

[16] H. S. Yoon, "Design Methodology of Network-on-Chip (NoC) for Digital Neural Network," Journal of Advanced Information Technology and Convergence, Vol. 2, No. 1, pp. 15-21, Jul. 2008.

Authors

Seung Hyeok Shin received the B.S. degrees in Applied Mathematics and received the M.S. degrees and Ph.D student in Department of Computer Engineering from Kumoh National Institute of Technology, Korea, in 1998, 2000 and 2013 respectively. He joined the faculty of the Department of Cyber Security at Gumi University, Gumi, Korea, in 2014. He is currently a Professor in the Department of Cyber Security, Gumi University. He is interested in Network Protocol, Embedded System. Software Engineering.