

〈황금오리〉

MQTT기반 메시지 서버의 성능 모니터링 시스템

-최종 보고서-

팀명	황금오리	
담당 교수님	황 기 태 교수님	
팀원	이름	학번
	손지혜(팀장)	1692037
	변민정	1592057
	김시훈	1591002
	이도연	1692052

- 목차 -

1. 프로젝트 수행 목적

- 1.1 MQTT 개념과 구성
- 1.2 프로젝트 정의
- 1.3 프로젝트 배경
- 1.4 프로젝트 목표

2. 프로젝트 결과물의 개요

- 2.1 프로젝트 구조
 - 2.1.1 전체 시스템 구조
 - 2.1.2 테스트 로드 시스템
 - 2.1.3 메시지 서버의 성능 모니터링 파트
 - 2.1.4 대시보드 웹 시스템 파트
 - 2.1.5 실시간 성능 데이터 출력 웹 서비스 파트
- 2.2 프로젝트 결과물
 - 2.2.1 프로젝트 UI
 - 2.2.2 파일 구성
 - 2.2.2.1 테스트 로드 시스템 Master PC (Windows)
 - 2.2.2.2 테스트 로드 시스템 Client PCs (Raspbian Linux)
 - 2.2.2.3 메시지 서버 모니터링 PC (Ubuntu Linux)
 - 2.2.2.4 웹 서버 PC (Ubuntu Linux)
 - 2.2.3 함수별 기능
 - 2.2.3.1 테스트 로드 시스템 Master PC (Windows)
 - 2.2.3.2 테스트 로드 시스템 Client PCs (Raspbian Linux)
 - 2.2.3.3 메시지 서버 모니터링 PC (Ubuntu Linux)
 - 2.2.3.4 웹 서버 PC (Ubuntu Linux)
- 2.3 프로젝트 개발 환경

3. 개발 중 장애요인과 해결 방안

- 3.1 프로젝트 중 장애요인과 해결 방안

4. 기대 효과 및 활용 분야

- 4.1 기대 효과
- 4.2 활용 분야

5. 프로젝트 수행 추진 체계 및 일정

- 5.1 각 조원의 조직도
- 5.2 역할 분담
- 5.3 주 단위의 프로젝트 수행 일정

6. 참고 자료

- 본문 -

1. 프로젝트 수행 목적

1.1 MQTT 개념과 구성

MQTT 프로토콜을 활용하는 시스템은 그림 1와 같이 MQTT broker, subscriber(메시지 구독자), publisher(메시지 발행자)로 구성된다.

MQTT는 publish/subscribe로 불리는 메시지 푸시 모델을 실현하는 표준 프로토콜이다. 메시지 구독자와 메시지 발행자를 연결하는 것이 토픽(topic)으로, 메시지 구독자는 특별한 토픽에 관한 메시지를 받겠다고 MQTT 브로커에 가입한다(subscribe). MQTT 브로커는 가입자들을 리스트에 저장하고 관리한다. 그리고 클라이언트가 MQTT 브로커에 토픽과 함께 메시지를 발행하면, MQTT 브로커는 가입자 리스트를 참고하여 토픽을 기다리는 가입자들 모두에게 메시지를 보낸다. 메시지를 보내고 받는 중계는 MQTT 브로커를 통해 일어나지만, 구독자를 결정하는 것은 토픽이다.

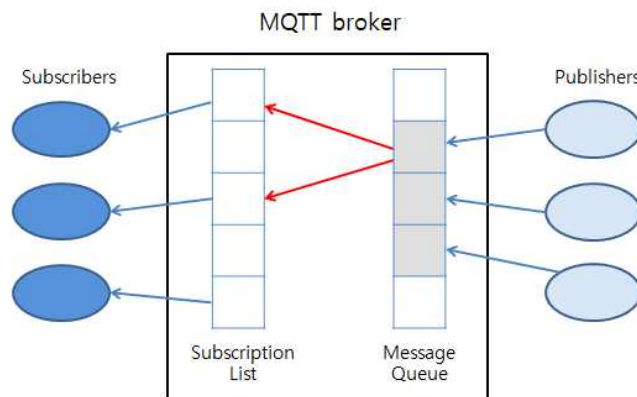


그림 1. MQTT 프로토콜 구성

1.2 프로젝트 정의

FaceBook이나 KakaoTalk 등 메시지 서버를 사용하는 곳이 점점 많아지는 추세이다. 이들은 대부분 대용량의 메시지를 처리하는 메시지 브로커를 가지고 있으며 그 중 대표적인 브로커가 MQTT 브로커이다. 따라서 우리는 MQTT 기반 메시지 서버의 성능을 모니터링하는 응용 시스템을 설계했다.

메시지 서버에 작업 부하를 걸기 위한 테스트 로드 시스템을 함께 개발하며, 메시지 서버의 성능으로는 서버 컴퓨터의 CPU 사용률, 메모리 사용률 등 메시지 서버의 성능과 함께, 메시지 서버에 접속된 구독자(subscriber)와 메시지 발행자(publisher)의 개수나 메시지 트래픽 등을 함께 측정한다. 측정된 성능은 웹 서비스를 통해 주기적으로 운영자의 웹 브라우저에 실시간으로 출력된다. 웹 서비스는 다양한 차트를 이용하여 GUI 방식으로 출력한다.

1.3 프로젝트 배경

4차 산업혁명으로 인해 미래 사회는 센서 등의 소형 기기에서부터 대형 컴퓨터에 이르기까지 수백 억 개의 디바이스들이 인터넷에 연결되는 초연결사회가 될 것으로 예측된다. 이러한 초연결사회를 위해 수백만의 동시접속과 초당 수천만의 메시지를 처리하는 그림 2와 같은 메시지 서비스가 필요하며 이 메시지 서버의 성능을 모니터링 하는 기술 또한 반드시 필요하게 될 것이다.

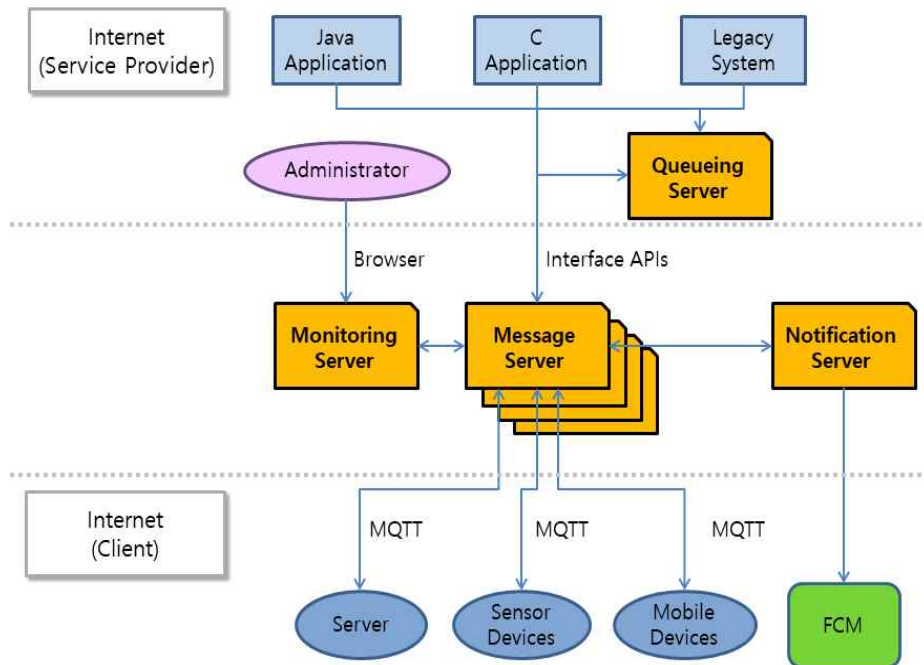


그림 2. 초연결사회 대비 메시지 서비스

1.4 프로젝트 목표

4차 산업혁명 시대의 초연결을 위해 실시간 통신 처리 기술이 필요하며, 2020년 500억 개의 디바이스를 연결하는 초연결사회로 진화하기 위해 수백만의 동시접속과 초당 수천만의 메시지를 처리하는 메시지 서비스가 필요하다. 본 프로젝트는 위에서 설명한 메시징 서비스의 기반이 될 것이다. 또한 대시보드, 차트를 통해 웹 페이지에서 한 눈에 모니터링이 가능하게 된다.

2. 프로젝트 결과물의 개요

2.1 프로젝트 구조

2.1.1 전체 시스템 구성

본 팀이 개발하고자 하는 모니터링 시스템과 테스트 시스템은 아래 그림 3과 같이 구성된다. 이 시스템은 크게 4 부분으로 나뉘어진다.

- 테스트 로드 시스템(①)
- 메시지 서버의 성능 모니터링 파트(②)
- 실시간 성능 데이터 출력 웹 서비스 파트(③)
- 대시보드 웹 시스템 파트(④)

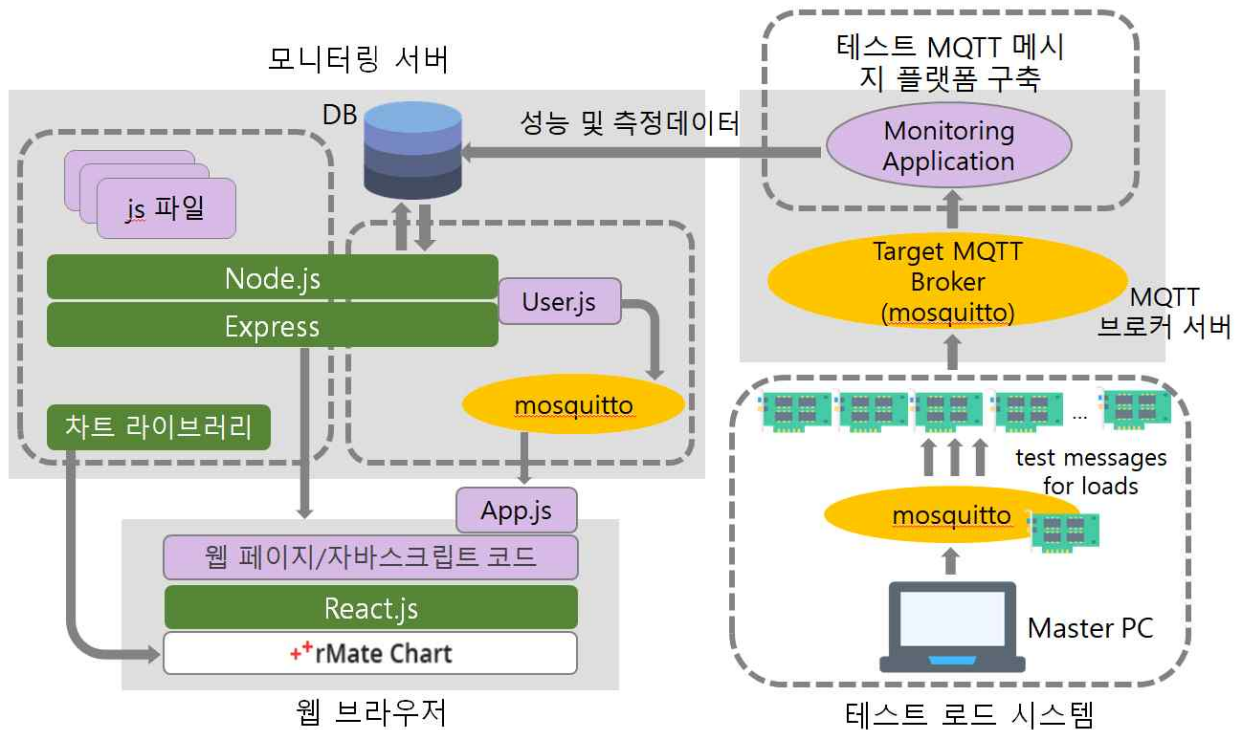


그림 3. 프로그램 전체 구조도

그림 3에 있는 타겟 메시지 시스템은 성능을 모니터링 하고자 하는 대상 메시지 시스템이다. 이 시스템을 개발하기 전에 앞서, 본 프로젝트에서는 이 시스템을 모니터링하고 테스트하는 시스템을 구축한다. 타겟 메시지 시스템은 현재 이클립스에 만들어 공개한 MQTT 기반의 Mosquitto 브로커를 이용한다.

2.1.2 테스트 로드 시스템

테스트 로드 시스템은 타겟 메시지 서버에 MQTT 메시지를 주고받는 응용을 시뮬레이션 하는 시스템이다. 여러 대에서 수십 대의 컴퓨터를 이용하여 MQTT 기반 메시지 서버에 MQTT 메시지를 발행(publish) 하고 구독(subscribe) 하는 애플리케이션을 통해 메시지를 타겟 메시지 서버에 유발 시킨다.

테스트 로드 시스템은 실제 MQTT 메시지를 유발시키는 여러 대의 client PC들과, 이들에게 메시지 유발을 지시하는 master PC 그리고 master PC가 client PC들에게 명령을 내리기 위해 사

용하는 Test Load Broker (Mosquitto Broker)와 Target Message Server (Mosquitto Broker)로 구성된다. client PC가 부하를 유발(Publish)하고 구독(Subscribe)하는 작업이 대단한 성능을 요구하지 않고, 임베디드 환경을 구축하기 위해서 본 프로젝트에서 client PC들을 Raspberry PI3 10대로 구성하였다. 전체적인 구성도는 그림 4 와 같다.

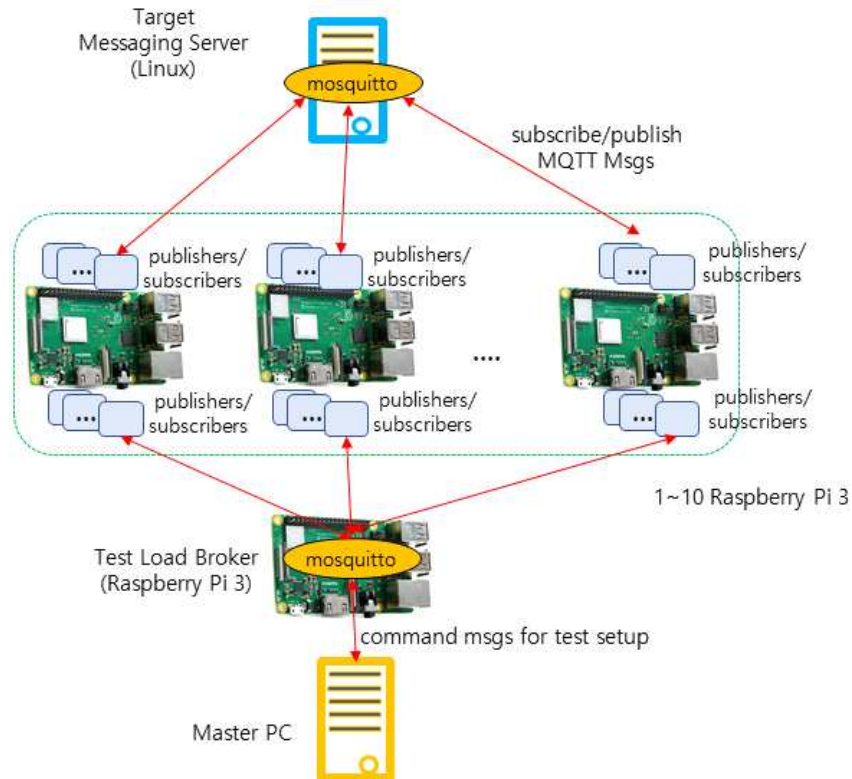


그림 4. 로드 제너레이터 구성도

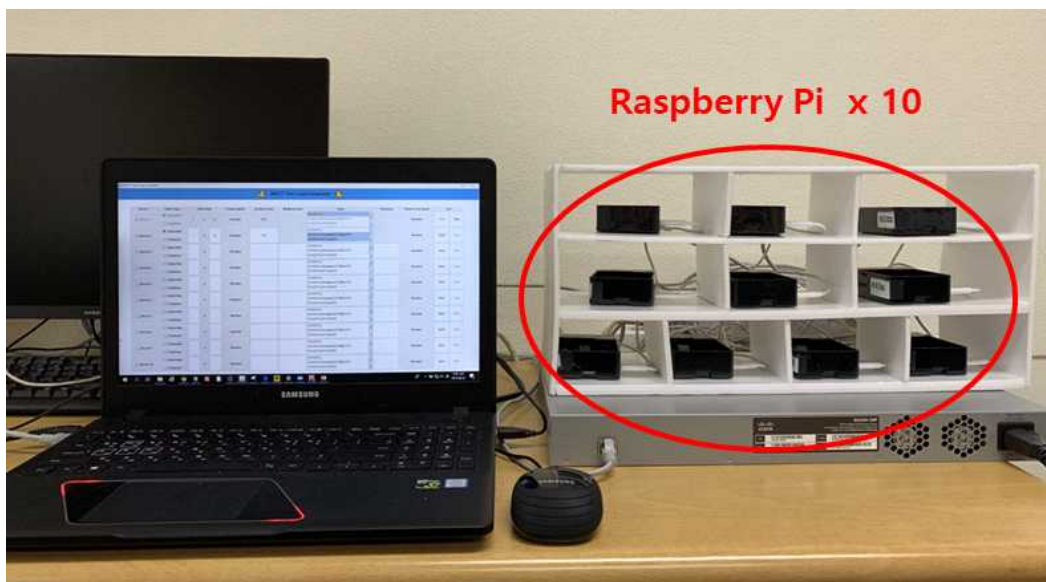


사진 1. MQTT 테스트 로드 시스템 실제 환경

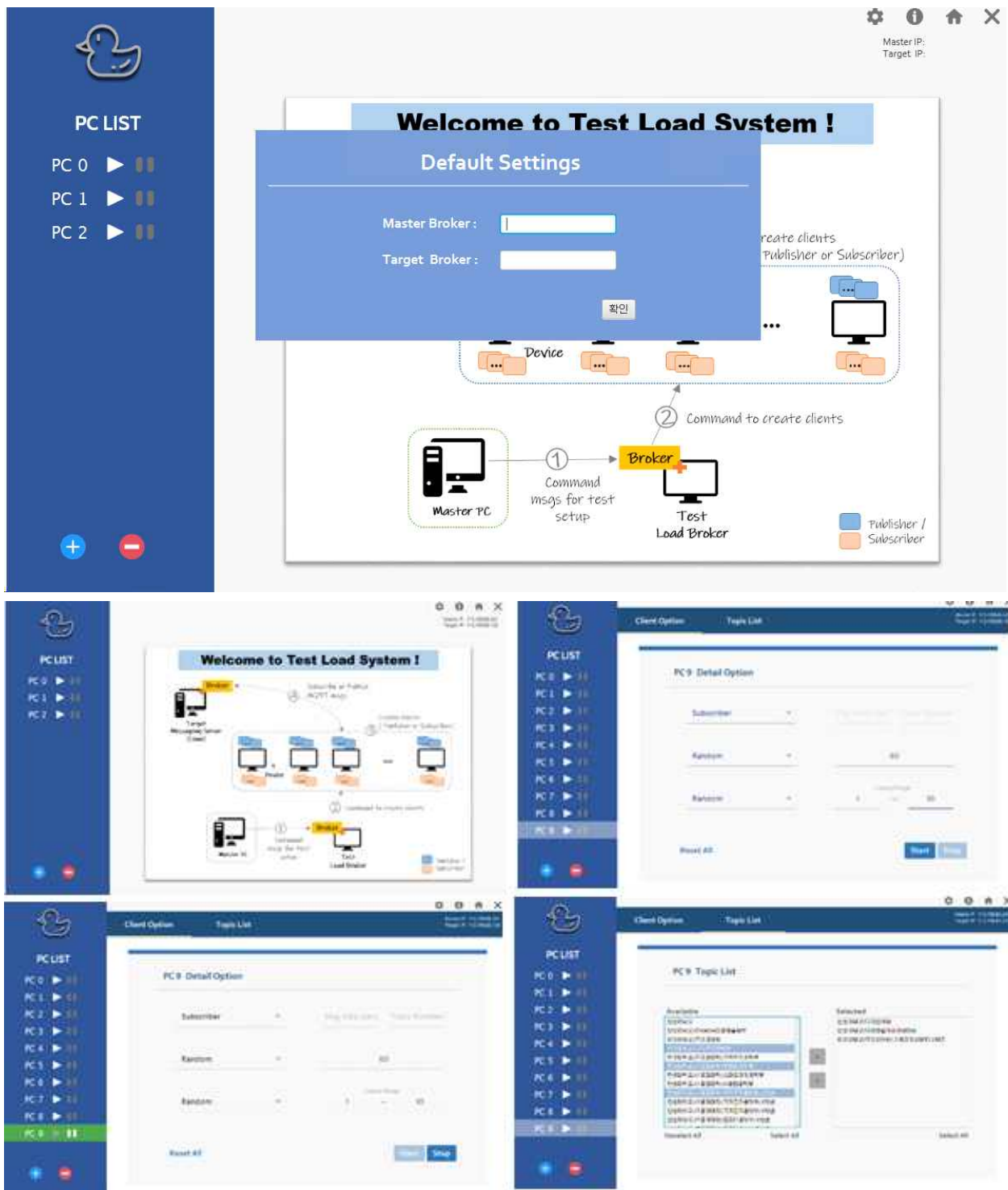


그림 5. MQTT 테스트 로드 시스템 UI

본 프로젝트에서는 master PC의 관리자가 client PC에게 테스트 로드를 발생시키도록 자바 기반 애플리케이션을 개발하였다.

- 애플리케이션이 시작됨과 동시에 master broker와 target broker의 IP 주소를 입력한다. 입력한 IP주소는 애플리케이션 상단에 기록되며, 설정페이지에서 바꿀 수 있다.
- 홈버튼과 상단 음각된 애플리케이션 로고를 눌렀을 때, 사용법을 알 수 있다.
- master PC의 관리자는 이 애플리케이션을 이용하여 client type을 지정할 수 있고, 각 client PC에서 생성할 client 객체의 수를 지정한다.
- 또 client들이 Target Message Server에 connect를 유지하는 시간을 지정할 수 있으며, 각 client들의 고유한 id의 생성 범위를 지정한다.
- client type이 Publisher인 경우 1초에 보낼 메시지의 개수를 입력받고, 메시지를 보낼 topic을 선택한다. 이 때 topic은 다중 선택이 가능하며 선택한 topic의 하위 topic 번호의 개수를 관리자로부터 입력 받는다.
- client type이 Subscriber인 경우에는 topic을 원하는 개수만큼 지정한다.
- 관리자는 각 옵션의 선택과 지정이 끝마친 후, Start 버튼과 Stop 버튼을 통해 개시와 종료를 지시할 수 있다.
- 시작과 중지 버튼을 리스트에서 관리할 수 있다.
- 동작중인 PC는 리스트에서 구분되며, 툴팁을 통해서 각 옵션들을 한 눈에 볼 수 있다.
- PC 리스트는 총 10대로 제한하며 PC0 부터 PC 9까지로 이루어진다. 리스트에서 PC 수를 추가 및 삭제가 가능하다.

2.1.3 메시지 서버의 성능 모니터링 파트

메시지 서버 성능 모니터링 파트에서는 타겟 메시지 서버로부터 모니터링 하고자 하는 성능 데이터를 수집하여, 웹 서버의 데이터베이스에 저장하는 역할을 한다. 내부 구조는 아래 그림 6과 같다.

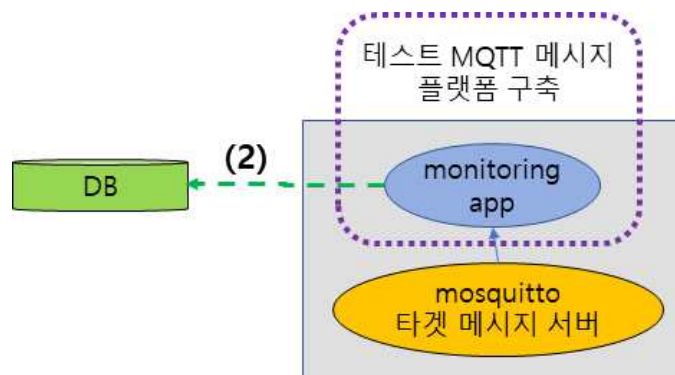


그림 6. 메시지 서버 성능 모니터링 파트 내부 구조

타겟 메시지 서버로부터 모니터링 하고자 하는 정보는 크게 타겟 메시지 서버 컴퓨터의 성능 정보와, 서버에서 처리되는 트래픽에 대한 정보의 2가지로 나눌 수 있다.

① 타겟 메시지 서버 컴퓨터의 성능 정보

- CPU 사용량
- 메모리 사용량
- Network I/O
- Process Information(Status, CPU, Memory)

타겟 메시지 서버 컴퓨터는 리눅스 OS에서 작동한다. 위와 같은 시스템 정보를 알아내기 위해서 리눅스의 가상 파일 시스템인 proc 파일 시스템을 이용하였다. proc 파일 시스템은 리눅스 커널이 가지고 있는 여러 데이터 구조체를 시스템 사용자가 쉽게 사용하도록 만들어진 파일 시스템이다. 또한 proc 파일 시스템은 커널의 내부 데이터 구조에 대한 인터페이스 역할로 프로세스 정보나 기타 시스템 정보 등 커널 데이터에 액세스하기 위한 편리하고 표준화 된 방법을 제공한다.

성능 정보를 알아내기 위해 '/proc/stat', '/proc/meminfo', '/proc/net/dev' 파일을 읽어 각각 cpu 정보, 메모리 정보, network I/O 정보를 주기적으로 수집한다. 또한 실행 중인 프로세스의 각종 정보에 대해서는, ps 명령어를 이용하여 현재 실행 중인 프로세스 id를 알아내고 이들 데이터로 '/proc/[pid]/stat' 파일을 읽어 각 프로세스의 cpu 사용률을 알아낸다. 사용하는 각 파일에 대한 설명은 아래와 같다.

- ▶ `/proc/stat`

[illegible]

그림 7. /proc/stat 파일 내용

cpu 사용률을 알아내기 위해 그림 8의 붉은 면적에 표시된 cpu, cpu(번호)인 라인을 이용한다. 첫 번째 라인은 모든 cpu를 통틀어 나타내고, 다음의 'cpu(번호)' 라인은 코어 별 cpu에 대한 정보를 나타낸다. 이 라인들의 각 수치는 지금까지 소모된 jiffies의 크기를 나타낸다. jiffies는 리눅스 커널에서 가장 기본이 되는 시간 값으로서 특정 클럭 단위로 값이 1 증가한다. 이 라인의 첫 번째 필드는 CPU 번호, 두 번째 필드는 user모드, 세 번째 필드는 low priority(nice 상태)의 user모드를 나타낸다. 네 번째 필드는 system모드, 다섯 번째 필드는 idle 상태의 jiffies 소모 값을 나타낸다.

전체 cpu 사용 jiffies에 대한 idle jiffies를 알아내어 백분율로 표시하는 것으로 cpu 사용률을 구한다.

▶ /proc/meminfo

MemTotal:	8121780 kB	KernelStack:	15056 kB
MemFree:	2376180 kB	PageTables:	71416 kB
MemAvailable:	4517728 kB	NFS_Unstable:	0 kB
Buffers:	89092 kB	Bounce:	0 kB
Cached:	2341540 kB	WritebackTmp:	0 kB
SwapCached:	0 kB	CommitLimit:	6158036 kB
Active:	3718588 kB	Committed_AS:	10605304 kB
Inactive:	1484324 kB	VmallocTotal:	34359738367 kB
Active(anon):	2773552 kB	VmallocUsed:	0 kB
Inactive(anon):	192960 kB	VmallocChunk:	0 kB
Active(file):	945036 kB	HardwareCorrupted:	0 kB
Inactive(file):	1291364 kB	AnonHugePages:	0 kB
Unevictable:	48 kB	ShmemHugePages:	0 kB
Mlocked:	48 kB	ShmemPmdMapped:	0 kB
SwapTotal:	2097148 kB	CmaTotal:	0 kB
SwapFree:	2097148 kB	CmaFree:	0 kB
Dirty:	44 kB	HugePages_Total:	0
Writeback:	0 kB	HugePages_Free:	0
AnonPages:	2772460 kB	HugePages_Rsvd:	0
Mapped:	655548 kB	HugePages_Surp:	0
Shmem:	194236 kB	Hugepagesize:	2048 kB
Slab:	270344 kB	DirectMap4k:	289144 kB
SReclaimable:	208900 kB	DirectMap2M:	7004160 kB
SUnreclaim:	61444 kB	DirectMap1G:	2097152 kB

그림 8. /proc/meminfo 파일 내용

리눅스에서 시스템에 할당된 메모리에 대한 정보가 기록된 파일이다. 모니터링에 필요한 메모리 정보를 얻기 위해 사용하는 항목은 다음과 같다.

- 'MemTotal' : 전체 사용 가능한 물리적인 메모리의 크기이다.
- 'MemFree' : 시스템에서 사용하지 않은 물리적인 메모리 크기이다.
- 'Buffers' : 버퍼 캐시의 메모리로, 블록 디바이스가 가지고 있는 블록 자체에 대한 캐시이다.
- 'Cached' : 페이지 캐시의 메모리로, 파일 데이터 캐시에 사용되는 메모리 크기이다.
- 'SReclaimable' : 커널이 사용하는 메모리이다. 파일 시스템 캐시에 사용하는 것 외에도 다른 요구사항이 있을 경우에는 다른 용도로 사용할 수 있다.

총 메모리 크기는 MemTotal의 수치를 얻어 사용하고, 현재 메모리 사용량은 MemFree, Buffers,

Cached, SReclaimable 의 수치의 합으로 구한다.

▶ /proc/net/dev

Inter- Receive										Transmit									
face	bytes	packets	errs	drop	fifo	frame	compressed	multicast		bytes	packets	errs	drop	fifo	colls	carrier	compressed		
lo: 50384903	83824	0	0	0	0	0	0	0	0	50384903	83824	0	0	0	0	0	0	0	0
enp4s0:176014839	1063136	0	43242	0	0	0	0	86337	130751165	463645	0	0	0	0	0	0	0	0	0

그림 9. /proc/net/dev 파일 내용

lo, enp4s0과 같은 네트워크 인터페이스에 대하여 Receive, Transmit의 여러 항목들을 나타내는 파일이다. 여기서는 Network I/O를 구하기 위해 각 영역의 bytes 항목만 사용한다. 이는 인터페이스가 전송하거나 수신한 총 바이트 크기를 나타낸다.

Network I/O를 구하기 위해서는 주기적으로 파일을 읽어 현재 bytes 항목의 값과 바로 이전에 저장한 bytes 값의 차를 1024로 나누는 것으로 구한다.

▶ /proc/[pid]/stat

```
32064 (mosquitto) S 1641 32064 1641 34816 32064 4194304 314 0 0 0 15 30 0 0 20 0 1 0 8281116
39907328 1157 18446744073709551615 94681297149952 94681297337776 140732798072656 0 0 0 18947
4096 18947 1 0 0 17 1 0 0 0 0 94681299437208 94681299439800 94681311707136 140732798079568
140732798079581 140732798079581 140732798083044 0
```

그림 10. /proc/32064/stat 파일 내용

실행 중인 프로세스 중 프로세스 번호가 [pid]인 해당 프로세스에 대한 정보가 기록되는 파일이다. 공백을 기준으로 나누어 14번째 항목과 15번째 항목을 이용한다.

- 14번째 항목 'utime' : 사용자 모드에서 프로세스가 스케줄 된 시간, clock ticks 단위로 측정된다.
 - 15번째 항목 'stime' : 커널 모드에서 프로세스가 스케줄 된 시간, clock ticks 단위로 측정된다.
- 주기적으로 파일을 읽어 /proc/stat의 전체 cpu 수치의 합, utime과 stime 합을 알아내어 프로세스 별 cpu 사용률을 구하도록 한다.

② 타겟 메시지 서버에서 처리되는 트래픽에 관한 정보

- 현재 연결된 클라이언트 수
- 최근 연결된 클라이언트
- 가장 오래 연결된 클라이언트
- 메시지 수를 기반으로 최대 처리량/최소 처리량을 갖는 클라이언트

- 토픽 별 가입자 리스트와 메시지 송/수신 횟수, 처리량
- 클라이언트 별 처리량

본 프로젝트에서는 독자적인 메시지 서버를 구축하기 위해 mosquitto 브로커를 이용하며, 메시지 서버의 트래픽에 관한 정보를 얻기 위해 타겟 메시지 서버로부터 날아온 mosquitto 로그를 분석한다.

```
1557167277: New connection from 127.0.0.1 on port 1883.
1557167277: New client connected from 127.0.0.1 as Person1 (c1, k60).
1557167277: Sending CONNACK to Per①1 (0, 0)
1557167277: Received SUBSCRIBE from Person1
1557167277: ②AAA (QoS 0)
1557167277: Person1 0 AAA
1557167277: Sending SUBACK to Person1
```

① Subscriber id
② Topic

예를 들어 위 그림처럼 'SUBSCRIBE' 단어를 읽으면 subscriber id와 구독한 topic 이름을 알아낼 수 있게 된다. 이처럼 mosquitto 로그를 읽으면서 'CONNACK', 'DISCONNECT', 'disconnecting', 'SUBSCRIBE', 'Receiving PUBLISH', 'Sending PUBLISH'의 특정 키워드가 읽히면 그에 맞는 트래픽 정보를 업데이트하도록 프로그램을 구현하였다.

다음은 수집한 데이터들을 콘솔에 출력한 결과이다. 이렇게 측정된 성능 정보들은 주기적으로 웹 서버의 데이터베이스에 기록한다.

```
current time : 2019-05-24 04:33:48
total cpu percentage : 5.37% cores : 3.75/4.39/5.15/8.16
total memory : 8121764KB used memory : 1630940KB
network in : 4.17kb/s network out : 0.73kb/s
1      0.1      0.0      systemd
2      0.0      0.0      kthreadd
4      0.0      0.0      kworker/0:0H
6      0.0      0.0      mm_percpu_wq
7      0.0      0.0      ksoftirqd/0
8      0.0      0.0      rcu_sched
9      0.0      0.0      rcu_bh
10     0.0      0.0      migration/0
11     0.0      0.0      watchdog/0
12     0.0      0.0      cpuhp/0
```

그림 11. 타겟 메시지 서버 컴퓨터 성능 데이터 예시

```

current time : 2019-05-24 04:33:20
number of clients : 2
recent client id : client1
old client id : client2
min throughput client id : client1 (5 bytes)
max throughput client id : client2 (13 bytes)

topic list
korea/busan/3 -> sending count (1) receiving count (0) total throughput (6 bytes)
korea/seoul/ -> sending count (0) receiving count (0) total throughput (0 bytes)
korea/seoul/2 -> sending count (1) receiving count (0) total throughput (6 bytes)
korea/seoul/3 -> sending count (3) receiving count (3) total throughput (26 bytes)
korea/seoul/1 -> sending count (1) receiving count (0) total throughput (6 bytes)
korea/seoul/6 -> sending count (2) receiving count (0) total throughput (11 bytes)
korea/seoul/4 -> sending count (3) receiving count (1) total throughput (21 bytes)

subscriber list
subscriber id (client2) topic (korea/seoul/3)subscriber id (client1) topic (korea/seoul/)
subscriber id (client1) topic (korea/seoul/1)subscriber id (client1) topic (korea/seoul/2)
subscriber id (client1) topic (korea/seoul/3)subscriber id (client1) topic (korea/seoul/4)

```

그림 12. 트래픽 데이터 예시

웹 서버의 데이터베이스 구성은 다음 표에서 확인할 수 있다.

- 컴퓨터 성능 정보를 저장하는 테이블

이름	내용
basic_info	총 메모리 크기(KB), cpu 코어 개수 정보를 가지는 테이블
performance	cpu 사용률(%), 메모리 사용량(KB), Network I/O(KB/sec), 시간 정보를 가지는 테이블
process	프로세스 당 cpu 사용률(%), 메모리 사용률(%) 정보를 가지는 테이블

total_memory (KB)	number_of_cores
8121780	4

basic_info 테이블

cpu_util (%)	cores_util (%)	memory_usage (KB)	network_in (KB/sec)	network_out (KB/sec)	date
3.48	0.66/0.44/0.21/0.11	954624	6.11	0	2019-05-15 15:34:10
4.67	0.36/0.11/0.23/0.55	755028	6.18	3.12	2019-05-15 15:34:20
4.88	0.91/0.12/0.12/0.23	746072	4.12	0	2019-05-15 15:34:30
12.34	4.27/3.81/2.34/2.11	741128	0	0	2019-05-15 15:34:40

performance 테이블

pid (번호)	cpu_util (%)	memory_util (%)	command
877	1.34	2.30	mysqld
29495	1.68	1.00	java
29494	1.78	0.91	mosquitto
319	0.00	0.00	system-udev
28379	0.00	0.00	bash

process 테이블

· 트래픽 정보를 저장하는 테이블

이름	설명
connection_info	현재 연결된 클라이언트 수, 최근 연결된 클라이언트 등의 연결에 관한 정보를 가지는 테이블
topic	토픽 이름과 송수신 횟수, 누적 메시지 크기(byte)을 가지는 테이블
client	클라이언트 이름과 누적 메시지 개수 정보가 있는 테이블
subscription	클라이언트 이름과 클라이언트가 구독하는 토픽 이름을 가지는 테이블
publication	클라이언트 이름과 클라이언트가 발행하는 토픽 이름을 가지는 테이블

number_of_ current_ connections (개수)	recent_ client_id (문자열)	old_ client_id (문자열)	client_id_of_ minimum_ msg (누적)	client_id_of_ maximum_ msg (누적)	date
6134	Client552	Client82	Client552	Client82	2019-06-22 13:55:01
7436	Client620	Client102	Client555	Client100	2019-06-22 13:55:04
7923	Client620	Client106	Client230	Client106	2019-06-22 13:55:07
8125	Client1200	Client202	Client245	Client202	2019-06-22 13:55:10

connection_info 테이블

topic (문자열)	message_receiving_ count	message_sending_ count	accumulated_ msg_size (바이트)
/Hansung/Creative	345	164	23426
/Hansung/Design	243	326	24321
/Hansung/IT	146	243	8652
/Hansung/IT/ComputerEngineering	374	854	25474

topic 테이블

client_id	number_of_messages (개수)
Client1	1237
Client2	257
Client23	386
Client19	859
Client320	286

client 테이블

client_id (문자열)	subscription_topic (문자열)
Client23	/Hansung/Design
Client135	/Hansung/Imagination
Client203	/Hansung/IT/ComputerEngineering
Client203	/Hansung/IT/ConvergenceEngineering
Client302	/Hansung/Creative

subscription 테이블

client_id (문자열)	publication_topic (문자열)
Client25	/Hansung/Creative
Client198	/Hansung/Imagination
Client206	/Hansung/IT
Client134	/Hansung/IT/ConvergenceEngineering
Client299	/Hansung/IT/ComputerEngineering/1grade

publication 테이블

2.1.4 대시보드 웹 시스템 파트

모니터링한 성능 정보를 운영자에게 출력하는 대시보드는 웹 서비스로 만든다.

이를 위해서 본 연구에서는 Node.js를 서비스 엔진(웹서버 엔진) 및 플랫폼으로 사용하고, 운영자에게 출력하고 제어하는 것은 React기반의 자바스크립트를 사용하여 구현한다. 그리고 Node.js 상에서 동작하는 Express.js 웹 프레임워크를 사용하여 서버 프로그램을 개발한다. 성능 데이터들은 데이터베이스로부터 읽어오고, 웹 브라우저에서는 React를 사용하여 다양한 차트 형식으로 운영자에게 보여준다.

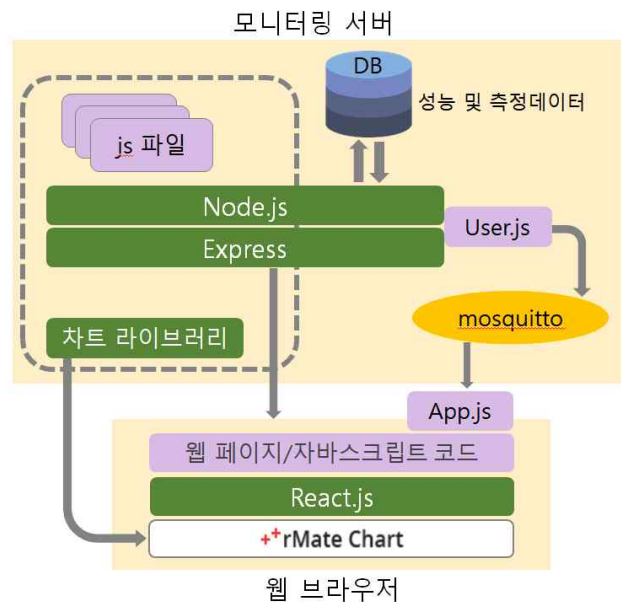


그림 13. 대시보드 웹 시스템 파트 구조

- React 웹 브라우저에서는 탭(Tab)을 클릭할 때 마다 페이지 컴포넌트가 생성되며 페이지 컴포넌트들은 여러 개의 차트 컴포넌트, 트리와 테이블 컴포넌트들을 담고 있다.
- 페이지 컴포넌트 생성이 완료되었을 시에 호출되는 이벤트 메소드인 `componentDidMount`에서 MQTT Broker를 통해 컴포넌트가 Mount되었다는 메시지를 Publish하며, 페이지 컴포넌트 별로 다른 Topic/#으로 Subscribe를 시작한다.
- Node 서버에서는 Subscribe된 메시지를 처리하는 Callback 메소드에서 컴포넌트가 Mount되었다는 메시지를 받으면 `setTimeout` 메소드를 사용하여 주기적으로 차트 데이터를 Publish한다.
- 페이지 컴포넌트에서는 하위 Topic별로 메시지들을 구분하여 자식 컴포넌트에게 props로 넘겨준다. 자식 컴포넌트에서는 props가 업데이트 될 때마다 호출되는 이벤트 메소드인 `componentWillReceiveProps`에서 페이지 컴포넌트로부터 받은 메시지 서버의 성능 데이터들과 트래픽 정보들로 차트 데이터를 업데이트하고, 실시간으로 바뀐 차트를 렌더링한다.
- 데이터베이스에 계속 누적되어 저장되는 테이블 데이터를 차트 데이터로 둘 시에는 `react-client/src/withSplitting.js`에서 x축의 시간 간격과, 표시되는 데이터 개수를 지정하며, 차트 가장 오른쪽에 가장 최근 데이터가 보여 지게 된다. 이 때 가장 최근 데이터의 x축 값으로 해당 데이터가 데이터베이스에 저장된 절대시간을 표시한다.
- 모니터링한 메시지 서버의 성능 정보 및 MQTT Message의 트래픽 정보들은 <http://riamore.net>의 rMate Chart를 사용하여 대시보드에 출력한다.
- `react-client/public/index.html` 파일의 <head> 섹션 내에 rMate 라이선스 파일, 라이브러리 파일, CSS 파일을 include 한다. `react-client/src/js/MakeChart.js`에서 차트의 종류, 모양, 기능 등을 설정하는 차트 레이아웃(Layout)을 정의하고 차트 생성을 실행하는 함수가 구현되어 있으며, 각각의 컴포넌트에서 차트를 생성할 때 참조한다.
- 차트 컴포넌트에서는 차트가 생성될 <div>를 정의한다. 컴포넌트가 마운트 될 때 불리는 `componentDidMount` 이벤트 메소드에서 해당 <div>에 차트를 생성하고, 차트 데이터를 초기화한다. 이 후 부모 컴포넌트에게 props로 전달받은 메시지로 생성된 데이터 배열로 차트 데이터셋(Dataset)을 갱신한다.
- 다른 탭이 클릭되면 컴포넌트가 소멸되게 되는데, 컴포넌트가 소멸되는 시점에 호출되는

componentWillUnmount 이벤트 메소드에서 컴포넌트가 Unmount된다는 메시지를 Publish 하며, Node 서버에서 이 메시지를 받으면 clearTimeout 메소드를 호출하여 해당 탭의 Topic 으로 주기적으로 보내던 Publish를 멈춘다.

```
25;0.76;16:59:39;0.59;16:59:36;0.51;16:59:33;0.68;16:59:30;0.6 CPU.js:33
8;16:59:27;0.34;16:59:24;0.84;16:59:21;0.51;16:59:18;0.51;16:59:15;0.59;
16:59:12;0.67;16:59:09;0.76;16:59:06;0.51;16:59:03;0.42;16:59:00;0.84;16
:58:57;0.67;16:58:54;0.59;16:58:51;0.42;16:58:48;0.67;16:58:45;0.59;16:5
8:42;0.42;16:58:39;0.76;16:58:36;0.51;16:58:33;0.67;16:58:30;0.76;16:58:
27
```

그림 14. Node 서버로부터 전달된 메시지 예시

```
▶ 0: {xValue: 120, date: "16:58:27", totalCpu: 0.76}
▶ 1: {xValue: 115, date: "16:58:30", totalCpu: 0.67}
▶ 2: {xValue: 110, date: "16:58:33", totalCpu: 0.51}
▶ 3: {xValue: 105, date: "16:58:36", totalCpu: 0.76}
▶ 4: {xValue: 100, date: "16:58:39", totalCpu: 0.42}
▶ 5: {xValue: 95, date: "16:58:42", totalCpu: 0.59}
▶ 6: {xValue: 90, date: "16:58:45", totalCpu: 0.67}
▶ 7: {xValue: 85, date: "16:58:48", totalCpu: 0.42}
▶ 8: {xValue: 80, date: "16:58:51", totalCpu: 0.59}
▶ 9: {xValue: 75, date: "16:58:54", totalCpu: 0.67}
▶ 10: {xValue: 70, date: "16:58:57", totalCpu: 0.84}
▶ 11: {xValue: 65, date: "16:59:00", totalCpu: 0.42}
▶ 12: {xValue: 60, date: "16:59:03", totalCpu: 0.51}
▶ 13: {xValue: 55, date: "16:59:06", totalCpu: 0.76}
▶ 14: {xValue: 50, date: "16:59:09", totalCpu: 0.67}
▶ 15: {xValue: 45, date: "16:59:12", totalCpu: 0.59}
▶ 16: {xValue: 40, date: "16:59:15", totalCpu: 0.51}
▶ 17: {xValue: 35, date: "16:59:18", totalCpu: 0.51}
▶ 18: {xValue: 30, date: "16:59:21", totalCpu: 0.84}
▶ 19: {xValue: 25, date: "16:59:24", totalCpu: 0.34}
▶ 20: {xValue: 20, date: "16:59:27", totalCpu: 0.68}
▶ 21: {xValue: 15, date: "16:59:30", totalCpu: 0.68}
▶ 22: {xValue: 10, date: "16:59:33", totalCpu: 0.51}
▶ 23: {xValue: 5, date: "16:59:36", totalCpu: 0.59}
▶ 24: {xValue: "16:59:39", date: "16:59:39", totalCpu: 0.76}
```

그림 15. 서버 메시지를 JSON 형식의 차트 데이터로 변환한 예시

탭 이름	컴포넌트	url	구독 토픽	하위토픽	컴포넌트 메시지	서버 메시지
Performance	CPU (차트)	/	performance/ #	performance/cpu	"performance did mount"	"데이터개수;cpu_util;date; ..."
	Memory (차트)			performance/memory		"totalMemory; 데이터개수; memory_usage;date; ..."
	Network I/O (차트)			performance/network		"데이터개수; network_in;network_out; date; ..."
	Process List (표)			performance/process		"전체process개수 ;pid;cpu_util;memory_util; command; ... "
Traffic	Topic Chart (차트)	/traffic	traffic/ #	traffic/msgtopic	"traffic did mount"	"전체토픽개수; topic;receiving;sending; ..."
	Accumulation Chart (차트)			traffic/accumulate		"전체토픽개수; topic; accumulated_msg_size; ..."
	Connection Info (차트)			subscription		"데이터개수; connections;recent;old;min imum;maximum;date; ..."
	Topic Pie (차트)			traffic/receivingPie, traffic/sendingPie, traffic/accumulationPie		"데이터개수;topic; (송신횟수, 수신횟수, 누적크기); ..."
Topic Tree & Subscription	Subscription (트리, 표)	/subscription	subscription/#	subscription	"subscription did mount"	"전체토픽개수;topic; ..."
				/tree, subscription /table		"client_id; ..."

웹 페이지 구성은 다음과 같다.

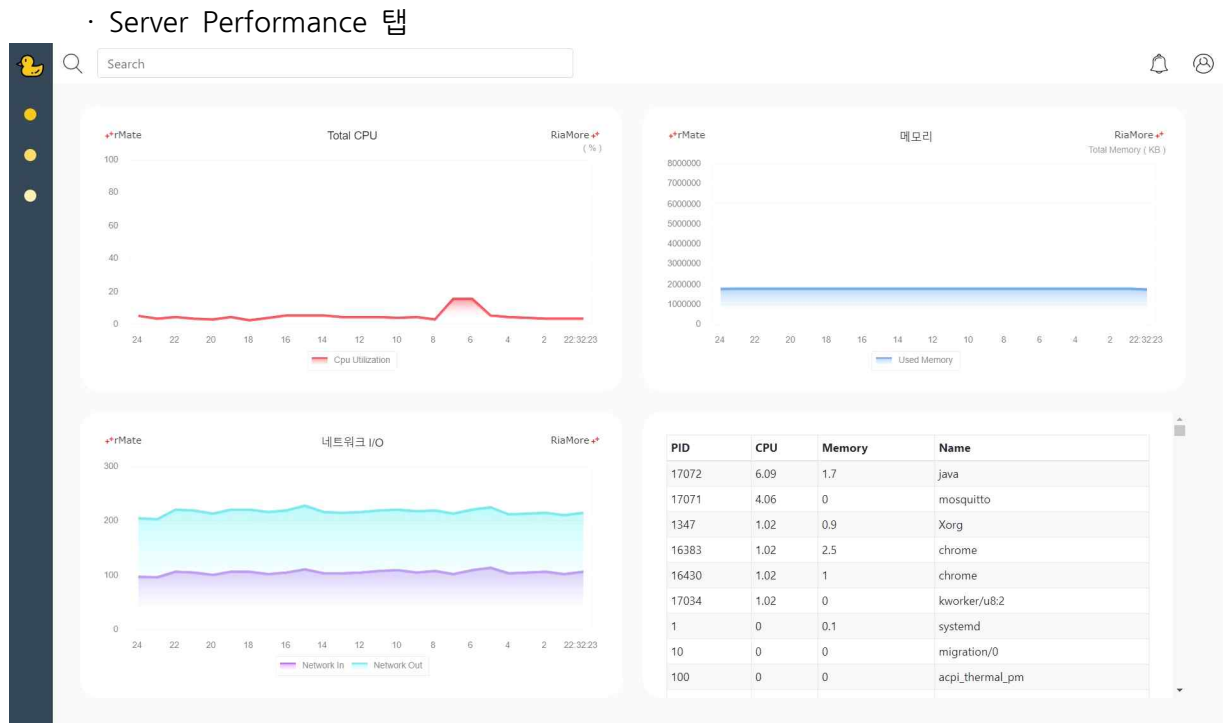


그림 16. server performance 대시보드

React 웹 페이지가 처음 로딩 되면 나오는 메시지 서버 성능 모니터링 탭의 모습이다.

데이터베이스에 저장된 메시지 서버의 성능 데이터 정보를 Node 서버로부터 MQTT Message로 받아 차트 데이터를 갱신한다. CPU, Memory, Network I/O, Process List 각각이 컴포넌트로 구성되어 있으며, 4개의 컴포넌트를 관리하는 부모 컴포넌트 Perform이 존재한다.

Perform 컴포넌트에서 performance/#라는 Topic으로 Subscribe하고 있으면, Node 서버에서 performance/cpu, performance/memory 등의 하위 토픽으로 성능 데이터를 담은 메시지를 publish한다. Perform 컴포넌트에서는 메시지가 도착하면 호출되는 MQTT Callback 메소드를 통해 하위 토픽 별 메시지를 각각의 컴포넌트에게 props로 전달한다. 4개의 자식 컴포넌트에서는 props로 전달된 메시지를 통해 차트 데이터를 갱신한다.

· Traffic 탭



그림 17. message traffic 대시보드

트래픽 데이터 모니터링 탭의 모습이다.

테스트 로드 시스템에서 걸리는 로드에 대한 메시지 서버의 트래픽 데이터를 차트로 나타낸다. Publish, Subscribe되는 Topic별 송신 횟수, 수신 횟수, 누적 크기와 현재 Connect된 Client들의 정보를 담고 있으며, 송신 횟수, 수신 횟수, 누적 크기가 가장 큰 3개의 Topic들을 파이 차트로 나타낸다.

첫 번째 탭과 마찬가지로 Topic별 송수신 횟수, Topic별 누적 메시지 크기, Client Connection Info, Topic관련 파이 차트들을 관리하는 컴포넌트 4개가 존재하며, 이 4개의 컴포넌트를 관리하는 Traffic 컴포넌트가 존재한다.

current connections(개수)	28
recent client	2.publisher19
old client	0.subscriber7
minimum msg	2.publisher19
maximum msg	0.subscriber6

그림 18. connection_info 차트의 툴팁

Client Connection Info 차트의 툴팁이다. 툴팁에는 현재 Connection수를 포함하여 가장 최근 연결된, 가장 오래전에 연결된, 메시지를 가장 많이 주고받은, 메시지를 가장 적게 주고받은 Client들의 정보가 보여 진다.

· Topic Tree & Subscription 탭

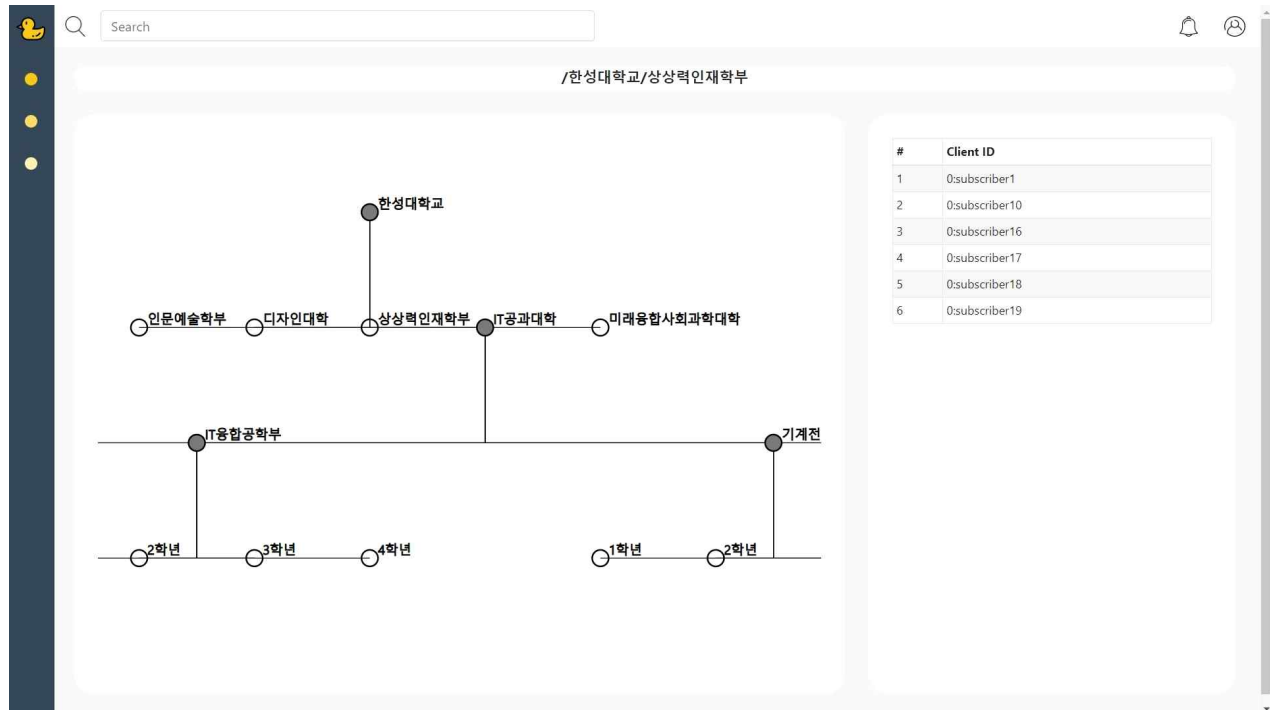


그림 19. topic tree&subscription 대시보드

마지막으로 메시지 서버에서 처리하는 MQTT Message들의 Topic구조를 'react-d3-tree' 라이브러리를 사용하여 트리구조로 나타낸다. 데이터베이스에서 전체 Topic들을 Node 서버로부터 받아 트리 형식의 JSON 데이터를 만든 뒤 트리를 갱신한다. 트리에서 Topic을 나타내는 노드를 클릭하면 해당 Topic을 구독중인 Subscriber List를 담은 표를 동적으로 생성한다.

트리와 표를 관리하는 2개의 컴포넌트로 구성되어 있으며, Subscription 컴포넌트에서 관리한다.

2.1.5 실시간 성능 데이터 출력 웹 서비스 파트

본 프로젝트의 모니터링 시스템은 주기적으로 웹 페이지에 실시간으로 성능 정보를 출력한다. 이를 위해 서버 측에서 개발하는 Node.js 애플리케이션은 MySQL 데이터베이스에 저장된 데이터를 주기적으로 읽어와 Mosquitto broker를 통해 웹 브라우저의 자바스크립트 코드와 연계하여 웹 페이지에 출력한다. Node.js의 Express Framework를 사용했으며, 이 Express Framework는 어떤 DB에도 의존적이지 않기 때문에 다른 DB를 사용한다 해도 충분히 대체 가능하다.

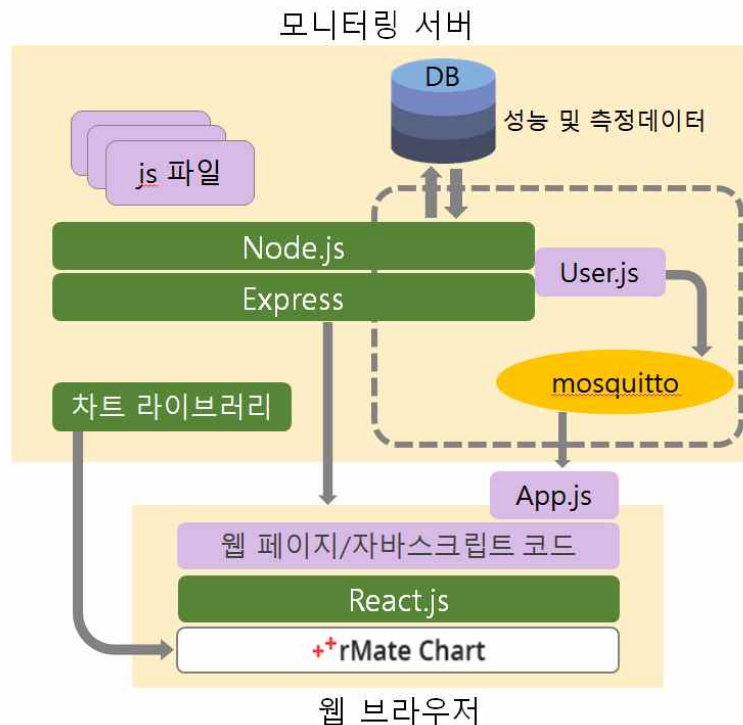


그림 20. 성능 데이터 출력 파트 구조

- Node.js Express 서버가 실행되는 데스크탑을 서버 컴퓨터로 한다.
- 서버 컴퓨터에 Mosquitto MQTT Broker를 설치한다.
- 서버 컴퓨터에 MySQL Database를 설치하고 사용자와 테이블을 생성한다.
- Node.js 서버가 존재하는 디렉토리에서 필요한 모듈들을 install 한다.
- Node.js의 mqtt 모듈을 install 하고 코드상에서 require 하여 mqtt 통신을 사용할 준비를 완료한다.
- Mosquitto MQTT Broker의 IP와 client 정보 등을 지정하여 Node.js 서버와 mosquitto broker와 연결한다.
- Node.js 의 mysql 모듈을 install 하고 코드상에서 require 하여 MySQL DB를 사용할 준비

를 완료한다.

- MySQL 서버의 IP와 username, password 등의 config정보를 코드로 구현하여 Node.js서버와 MySQL 서버를 연동한다.
- MySQL 서버에서는 외부 접속을 허용하여 물리적으로 다른 위치에 있는 서버에서의 접근을 가능하게 한다.
- Node.js 와 연동 된 MySQL 데이터베이스에서 테이블 별로 저장 된 레코드들을 JSON 형식으로 가져온다.
- 가져온 레코드들을 String 형식으로 변환 후, 웹 페이지의 차트마다 출력하기 쉬운 형태로 가공한다.
- React 기반 웹 페이지에서 탭(Tab)을 클릭할 때 마다 서버 컴퓨터에 있는 Mosquitto로 메시지가 publish 되는데, Node.js 서버에서 구독하고 있는 토픽과 일치하는 메시지가 subscribe될 때, 해당 메시지와 토픽에 따라서 다른 테이블과 형식으로 데이터를 가져와 가공한다.
- 가공된 문자열 데이터를 Node.js 와 연동 되어 있는 Mosquitto MQTT broker로 publish 한다.

```
[{"cpu_util":3.48, "cores_util":"0.66/0.44/0.21/0.11", "memory_usage":954624, "network_in":6.11, "network_out":0, "date":"2019-05-15 15:34:10"}, {"cpu_util":4.67, "cores_util":"0.36/0.11/0.23/0.55", "memory_usage":755028, "network_in":6.18, "network_out":3.12, "date":"2019-05-15 15:34:20"}, {"cpu_util":4.88, "cores_util":"0.91/0.12/0.12/0.23", "memory_usage":755028, "network_in":6.20, "network_out":0, "date":"2019-05-15 15:34:30"}, {"cpu_util":12.34, "cores_util":"4.27/3.81/2.34/2.11", "memory_usage":746072, "network_in":6.46, "network_out":0, "date":"2019-05-15 15:34:40"}, {"cpu_util":14.51, "cores_util":"5.88/2.46/3.31/1.94", "memory_usage":741128, "network_in":9.24, "network_out":6.11, "date":"2019-05-15 15:34:50"}, {"cpu_util":11.22, "cores_util":"4.12/3.57/3.11/0.17", "memory_usage":780513, "network_in":10.51, "network_out":9.48, "date":"2019-05-15 15:35:00"}, {"cpu_util":12.01, "cores_util":"4.99/3.64/3.44/1.23", "memory_usage":865147, "network_in":11.87, "network_out":9.61, "date":"2019-05-15 15:35:10"}, {"cpu_util":13.69, "cores_util":"5.47/4.12/2.54/1.59", "memory_usage":840361, "network_in":10.43, "network_out":10.82, "date":"2019-05-15 15:35:20"}, {"cpu_util":14.28, "cores_util":"5.82/4.54/3.61/0.24", "memory_usage":919325, "network_in":9.88, "network_out":5.22, "date":"2019-05-15 15:35:30"}, {"cpu_util":16.98, "cores_util":"6.05/5.49/5.30/2.41", "memory_usage":922450, "network_in":9.13, "network_out":0, "date":"2019-05-15 15:35:40"}]
```

그림 21. DB의 topic 테이블에서 가져 온 JSON 형태의 레코드 형식 예시- performance 테이블


```
[{"Topic": "A", "receiving": 2000, "sending": 700, "throughput": 20},
{"Topic": "A/B", "receiving": 3000, "sending": 1200, "throughput": 45},
{"Topic": "A/C", "receiving": 4200, "sending": 1700, "throughput": 83},
{"Topic": "A/D", "receiving": 5500, "sending": 2000, "throughput": 45},
{"Topic": "A/B/E", "receiving": 6200, "sending": 2200, "throughput": 20},
{"Topic": "A/B/F", "receiving": 7200, "sending": 2700, "throughput": 10}]
```

그림 22. DB에서 가져 온 JSON 형태의 레코드 형식 예시- topic 테이블

```
25;0.76;16:59:39;0.59;16:59:36;0.51;16:59:33;0.68;16:59:30;0.6 CPU.js:33
8;16:59:27;0.34;16:59:24;0.84;16:59:21;0.51;16:59:18;0.51;16:59:15;0.59;
16:59:12;0.67;16:59:09;0.76;16:59:06;0.51;16:59:03;0.42;16:59:00;0.84;16
:58:57;0.67;16:58:54;0.59;16:58:51;0.42;16:58:48;0.67;16:58:45;0.59;16:5
8:42;0.42;16:58:39;0.76;16:58:36;0.51;16:58:33;0.67;16:58:30;0.76;16:58:
27
```

그림 23. JSON 형태의 레코드를 리액트의 차트에 따라 String형식으로 가공한 예시 - performance 테이블

```
6;A;2000;700;20;A/B;3000;1200;45;A/C;4200;1700;83;A/D;5500;
2000;45;A/B/E;6200;2200;20;A/B/F;7200;2700;10
```

그림 24. JSON 형태의 레코드를 리액트의 차트에 따라 String형식으로 가공한 예시 - topic 테이블

2.2 프로젝트 결과물

2.2.1 프로젝트 UI

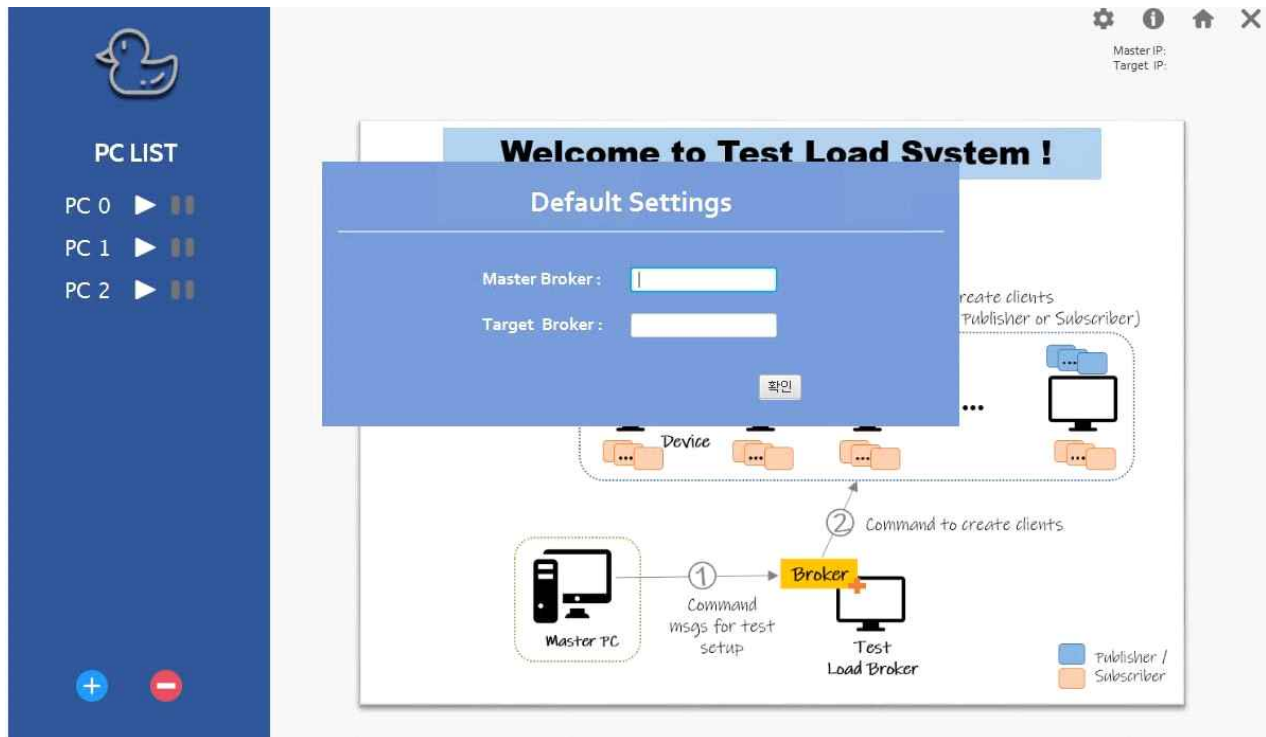


그림 25. 로드 제너레이터 start

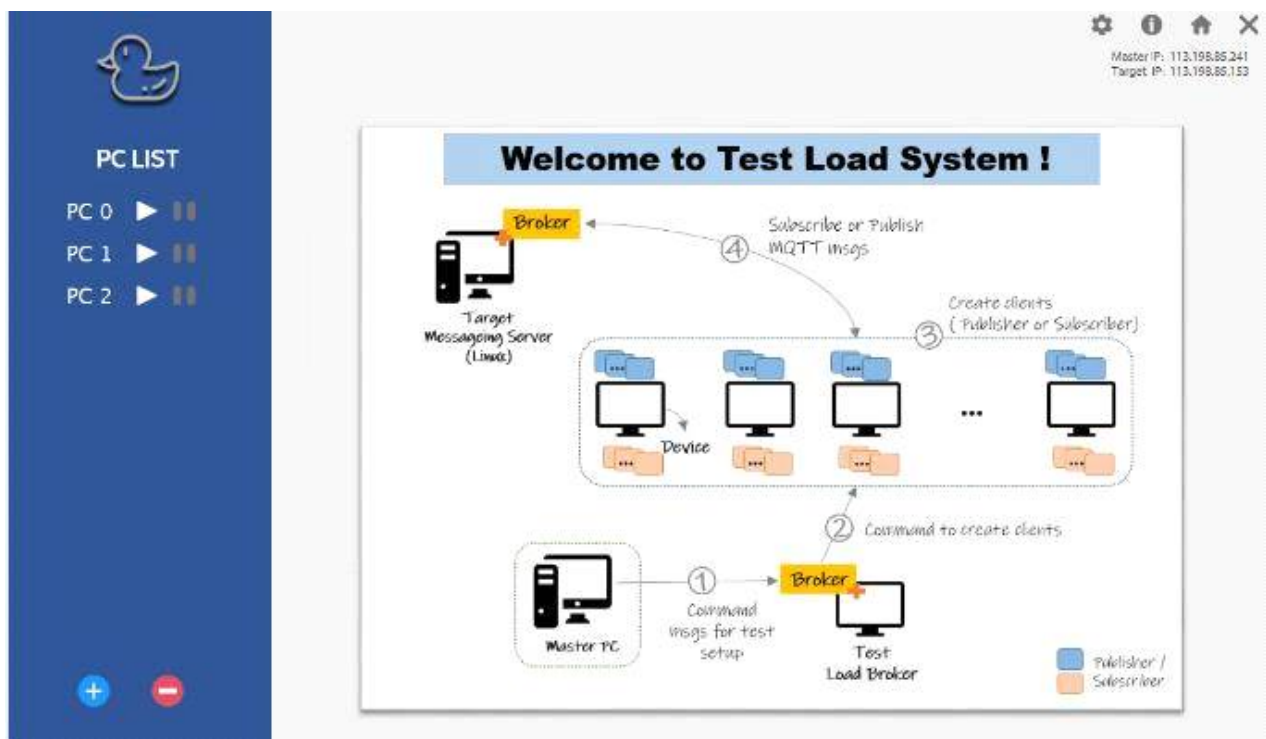


그림 26. 로드 제너레이터 home

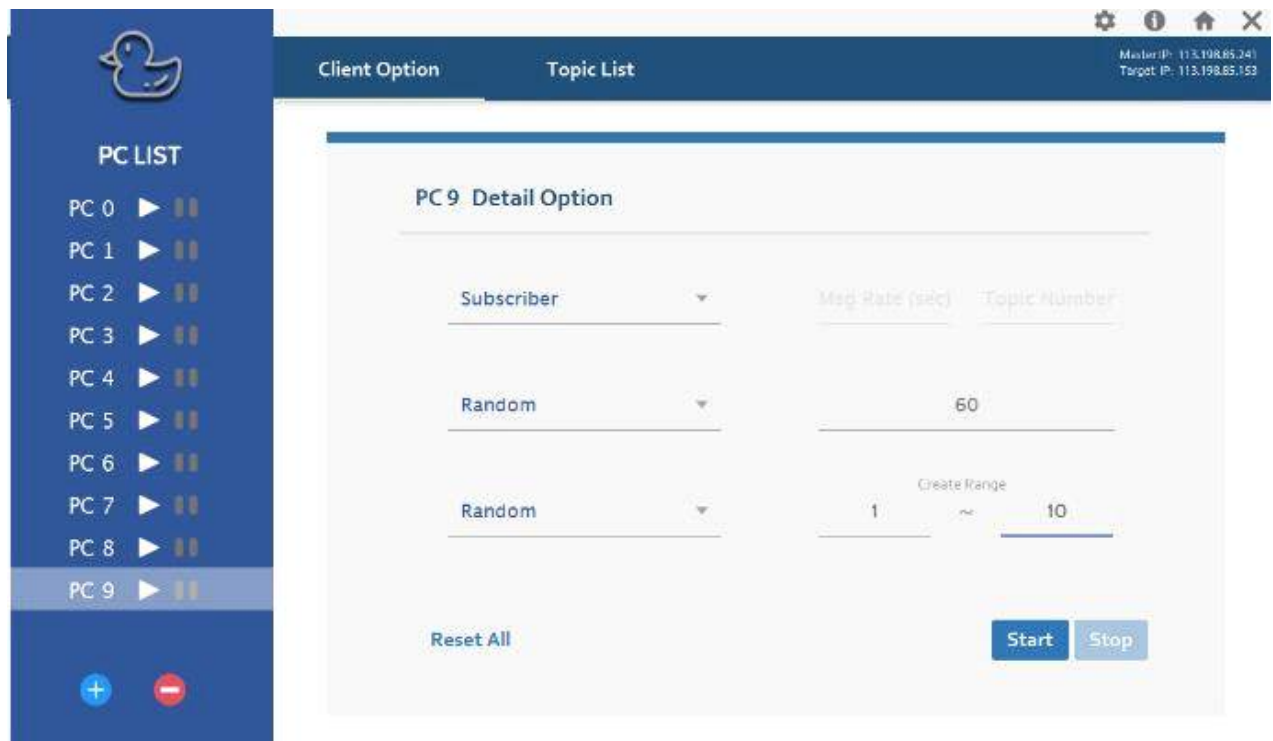


그림 27. 로드 제너레이터 options

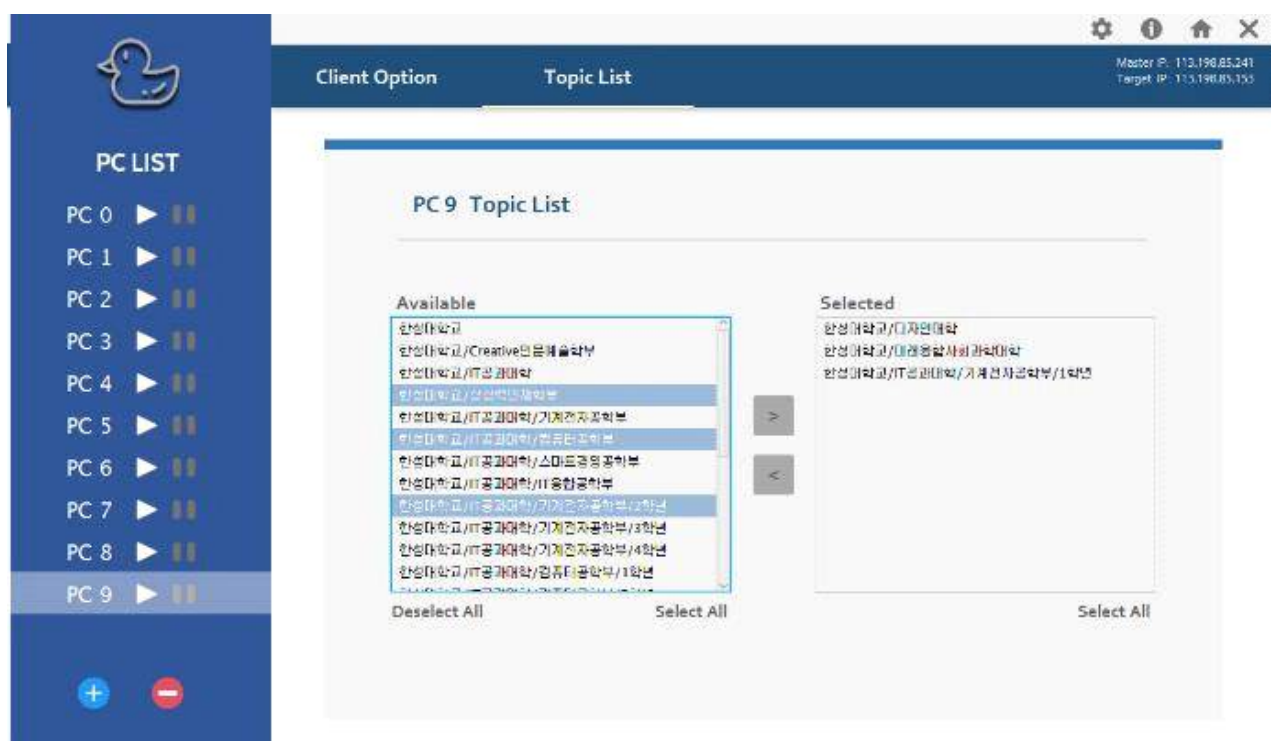


그림 28. 로드 제너레이터 topic list

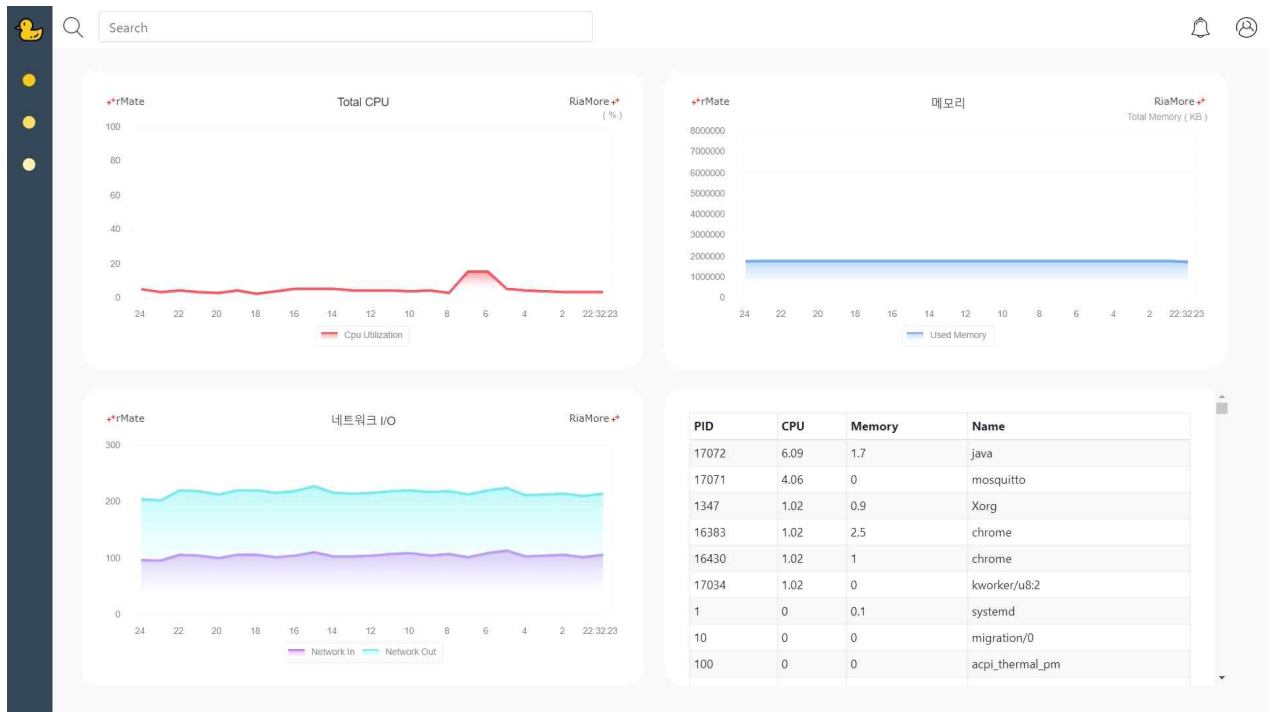


그림 29. server performance 대시보드

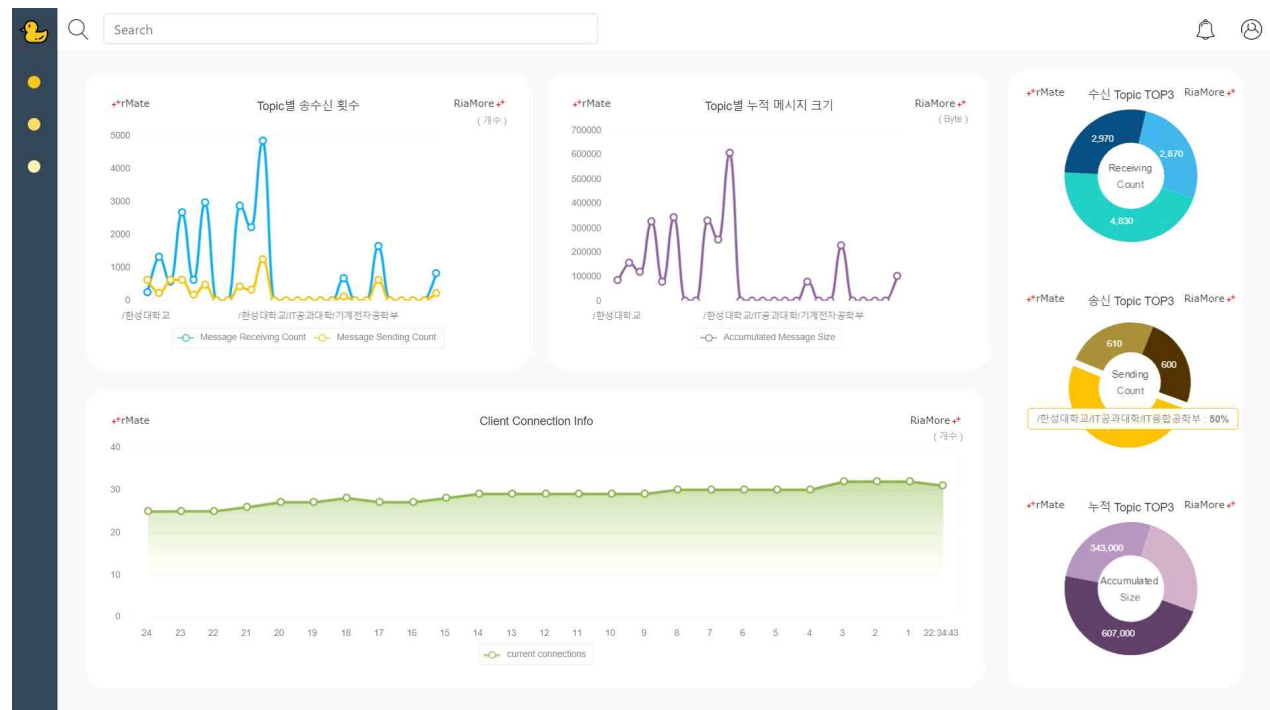


그림 30. message traffic 대시보드

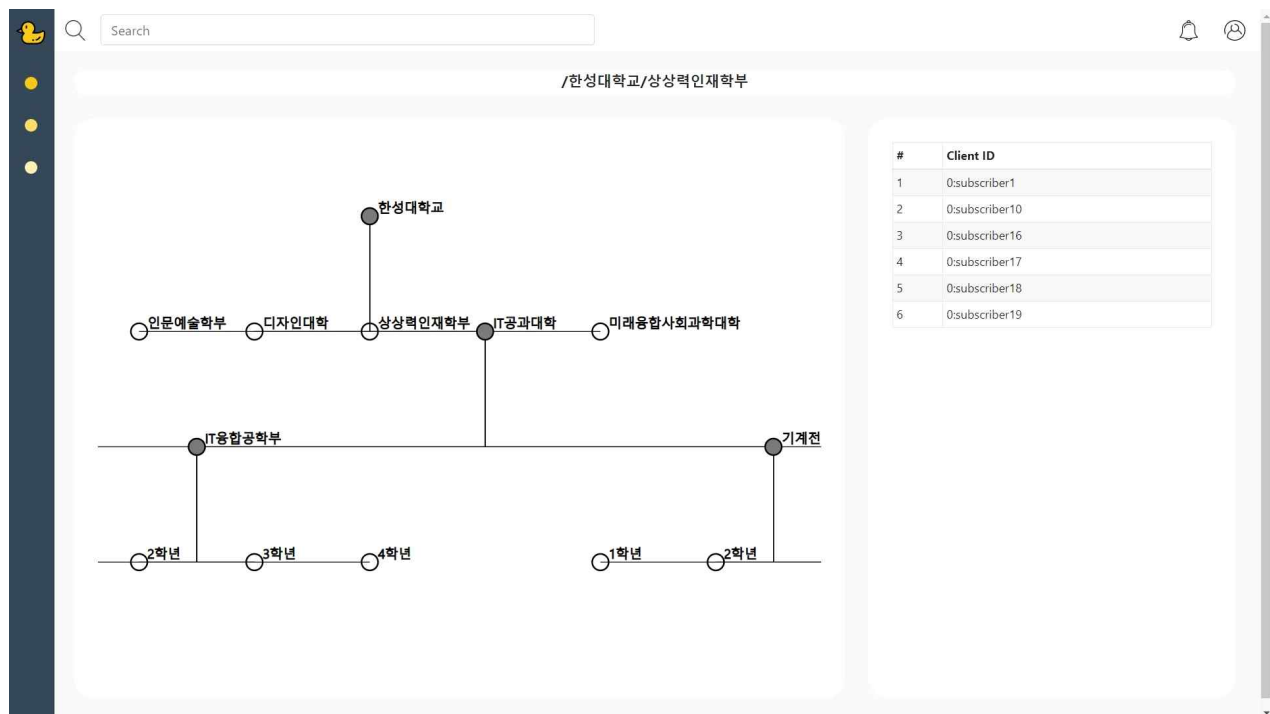


그림 31. topic tree&subscription 대시보드

2.2.2 파일 구성

2.2.2.1 테스트 로드 시스템 Master PC (Windows)

Test Load Generator - Master TestFx.controller 파일	
이름	내용
DetailViewController.java	DetailView.fxml 컨트롤러, 옵션 선택 및 동작을 담당하는 파일
EffectUtilities.java	선택된 노드(컴포넌트) 이동 관련 파일
HomeController.java	homeView.fxml 컨트롤러
InputDialogController.java	inputDialogView.fxml 컨트롤러, IP입력 관련 파일
IntroController.java	introView.fxml 컨트롤러, 애니메이션 효과
MainApp.java	메인 애플리케이션 파일
MainApplicationController.java	rootLayout.fxml 컨트롤러, 메인 비즈니스 로직 파일
SettingController.java	settingView.fxml 컨트롤러, IP 입력 설정 파일
StageStore.java	Stage를 보관하는 파일
Test Load Generator - Master TestFx.model 파일	
Device.java	각 옵션과 각 pc의 동작 유무를 판단하여 저장하는 파일
Test Load Generator - Master TestFx.mqtt 파일	
App.java	test load broker에 개시와 종료, 각 pc마다 topic list와 선택한 pc 번호 또한 publish 하는 파일
Setting.java	토픽 파일을 읽어와 리스트에 저장하는 파일
topicList(english).txt	구독 가능한 전체 토픽 리스트를 영어로 기록한 파일
topicList(korean).txt	구독 가능한 전체 토픽 리스트를 한글로 기록한 파일
Test Load Generator - Master TestFx.view 파일	
detailStyle.css	detailView.fxml 관련 css 파일
myStyle.css	rootLayout.fxml 관련 css 파일
topicStyle.css	detailView.fxml 의 topic 창 관련 css 파일
detailView.fxml	옵션 선택 레이아웃 파일
homeView.fxml	홈 레이아웃 파일
InputDialogView.fxml	애플리케이션 시작시, IP 입력 레이아웃 파일
introView.fxml	애플리케이션 시작시, 애니메이션 관련 파일
itemView.fxml	리스트의 아이템 커스텀 레이아웃 파일
rootLayout.fxml	애플리케이션 메인 레이아웃 파일
settingView.fxml	IP 설정 레이아웃 파일

2.2.2.2 테스트 로드 시스템 Client PC (Raspbian Linux)

Test Load Generator client 파일 - main package	
이름	내용
Main.java	pcNum.txt를 읽어 pcNum을 담아 Connect 클래스의 생성자를 호출
Connect.java	test load broker에 연결하여 각 토픽으로 온 메시지들을 받아 client type에 따라 client 생성
pcNum.txt	각 pc의 고유번호를 기록한 파일
Test Load Generator client 파일 - publisher package	
Instance.java	client type이 publisher일 경우 publisher들을 선택한 개수와 범위내로 생성
Publisher.java	Timer 객체를 이용하여 지정된 시간동안 publisher 들을 연결 및 해제하고 1초에 지정된 메시지의 개수를 보내는 파일
Test Load Generator client 파일 - subscriber package	

Instance.java	client type이 subscriber일 경우 subscriber들을 선택한 개수와 범위내로 생성하는 파일
Subscriber.java	Timer 객체를 이용하여 지정된 시간동안 subscriber 들을 연결 및 해제하고 선택된 토픽 내에서 랜덤으로 1개의 토픽을 선택하여 구독하는 파일

2.2.2.3 메시지 서버 성능 모니터링 PC (Ubuntu Linux)

컴퓨터 성능 데이터 관련 파일 - performance package	
이름	내용
CpuInfo.java	총 cpu 사용량과 코어 별 cpu 사용량을 반환하는 파일
MemoryInfo.java	총 메모리 크기와 메모리 사용량을 반환하는 파일
NetworkInfo.java	네트워크 input, output을 반환하는 파일
ProcessInfo.java	프로세스 정보를 저장하는 파일
Performance.java	성능 데이터를 데이터베이스에 저장하는 파일
트래픽 데이터 관련 파일 - log_reader package	
Client.java	클라이언트 이름과 처리량을 저장하는 파일
Topic.java	토픽 이름과 토픽에 대한 메시지 송수신량, 처리량을 저장하는 파일
MessageType.java	mosquitto log 메시지를 구분하는 문자열을 가지는 파일
Publication.java	publisher가 발행한 토픽의 이름과 그에 관한 처리량을 저장하는 파일
Publisher.java	publisher의 이름과 발행(Publication) 정보를 저장하는 파일
Subscription.java	subscriber가 구독한 토픽의 이름과 그에 관한 처리량을 저장하는 파일
Subscriber.java	subscriber의 이름과 구독(Subscription) 정보를 저장하는 파일
LogReader.java	mosquitto log를 읽고 트래픽 데이터를 수집하여 데이터베이스에 저장

2.2.2.4 웹 서버 PC (Ubuntu Linux)

Node.js - MQTTDashboard/node-server	
이름	내용
app.js	express 설정이 담겨 있는 핵심 파일
package.json	node 프로그램 및 설치된 모듈들의 버전 등 정보 저장
www	서버 구동을 위한 코드 존재. app.js파일을 가져와 HTTP 객체와 연동
users.js	url 요청이 '/users' 로 올 때 실행되는 파일
index.js	url 요청이 '/'로 올 때 실행되는 파일
databaseConf.js	연결할 database 정보 저장한 파일

React.js - MQTTDashboard/react-client	
이름	내용
public	
index.html	웹 브라우저 로드 시 처음 불러오는 페이지
index.css	기본 css 파일
chartSetting.js	차트의 툴팁 설정 함수, 차트 테마 지정 함수가 구현되어있는 파일
src	
index.js	초기에 렌더링할 컴포넌트 지정 파일
src/client	
Root.js	index에서 렌더링 시켜주는 초기 컴포넌트

	BrowserRouter로 감싸서 App.js 컴포넌트를 생성
src/shared	
App.js	메인 컴포넌트
src/pages	
index.js	페이지 컴포넌트들을 export 하는 파일
Perform.js	첫 번째 탭 클릭 시에 나타나는 성능 모니터링 페이지 컴포넌트
Traffic.js	두 번째 탭 클릭 시에 나타나는 트래픽 정보 모니터링 페이지 컴포넌트
Subscription.js	세 번째 탭 클릭 시에 나타나는 Topic 트리와 Subscription 모니터링 페이지 컴포넌트
src/performanceComponents	
index.js	성능 컴포넌트들을 export 하는 파일
CPU.js	CPU 성능 차트 컴포넌트
Memory.js	Memory 성능 차트 컴포넌트
Network.js	Network I/O 차트 컴포넌트
ProcessTable.js	Process List 표 컴포넌트
src/trafficComponents	
index.js	트래픽 컴포넌트들을 export 하는 파일
TopicChart.js	Topic별 송수신 횟수 차트 컴포넌트
AccumulationChart.js	Topic별 누적 메시지 크기 차트 컴포넌트
ConnectionInfoChart.js	Client Connection Info 차트 컴포넌트
TopicPie.js	송신 횟수, 수신 횟수, 누적 메시지 크기가 가장 큰 Topic 3개씩 보여주는 파이 차트 컴포넌트
src/components	
Menu.js	상단 메뉴바 컴포넌트
Sidebar.js	좌측 사이드바 컴포넌트
src/js	
withSplitting.js	컴포넌트들을 코드 스플리팅하는 함수 파일
MakeChart.js	차트 생성 함수와 차트 생성 시에 필요한 초기 Layout을 지정하는 파일

2.2.3 함수별 기능

2.2.3.1 테스트 로드 시스템 Master PC (Windows)

Test Load Generator Master TestFx.controller	
함수명	설명
DetailViewController.java	
Initialize()	멤버 변수 초기값 처리
startAction()	Start 버튼 눌렀을 때 이벤트 처리
stopAction()	Stop 버튼 눌렀을 때 이벤트 처리
resetAction()	Reset 버튼 눌렀을 때 이벤트 처리
leftToRightAction()	토픽 선택창 이벤트 처리 (왼쪽->오른쪽)
rightToLeftAction()	토픽 선택창 이벤트 처리 (오른쪽->왼쪽)
deselectedAllAction()	토픽 선택창 이벤트 처리 (전체 해제)
selectedAllAction()	토픽 선택창 이벤트 처리 (전체 선택)
isTfInputValid()	Text field validation
isCbxInputValid()	Check box validation
TopicListValid()	Topic list validation
publisherIndexValid()	Publisher index validation
ShowDetailData()	해당 PC의 전체 옵션 데이터 보여주기
ClearData()	해당 PC의 옵션 데이터 초기화
SaveData()	해당 PC의 옵션 데이터 저장
MainApplicationController.java	
showInputDialog()	애플리케이션 시작 시 IP 입력 창을 띄움
changeListIndex()	PC 리스트의 index를 통해 pcName에 맞게 detailView 처리
addAction()	PC 리스트에서 PC를 추가할 때 이벤트 처리
deleteAction()	PC 리스트에서 PC를 삭제할 때 이벤트 처리
settingAction()	Setting 버튼 눌렀을 때 이벤트 처리
homeAction()	Home 버튼 눌렀을 때 이벤트 처리
closeAction()	Close 버튼 눌렀을 때 이벤트 처리
setToolTip()	Tooltip 세팅 처리
MyEventHandler 클래스	PC 리스트 클릭시 이벤트 처리 클래스
XCell 클래스	PC 리스트 안에 들어갈 List Cell 관련 클래스
Test Load Generator Master TestFx.mqtt	
함수명	설명
App.java	
App()	멤버변수 초기화 및 클라이언트 타입에 맞게 동작 지시
connect()	Test broker에 연결
pubTopicCommand()	선택한 토픽을 담아 subTopic으로 토픽종류들을 전송
subTopicCommand()	선택한 토픽을 담아 subTopic으로 토픽종류들을 전송
subCommand()	subscriber 생성을 지시하라는 message를 전송
pubCommand()	publisher 생성을 지시하라는 message를 전송
pubStopCommand()	publisher stop 명령 지시
subStopCommand()	Subscriber stop 명령 지시
Setting.java	
readTopicList()	Topic 파일을 읽어 멤버변수 리스트에 저장

2.2.3.2 테스트 로드 시스템 Client PCs (Raspbian Linux)

Test Load Generator client program	
함수명	설명
main package - Connect.java	
Connect()	test load broker에 연결한 후 토픽리스트, 개시, 종료 토픽을 전부 다 구독하는 메서드
messageArrived()	구독한 토픽마다 메시지를 받아 토픽 리스트에 저장, 개시, 종료 명령
publisher package - Instance.java	
run()	지정한 범위 내에 지정한 개수의 publisher 생성
selectTopic()	리스트에 담긴 토픽들 중 하나의 토픽을 선택
publisher package - Publisher.java	
connect()	Timer 객체를 이용해 지정한 duration동안 publisher가 살아있도록 하며 지속 시간 이후에는 time_disconnect() 호출
send()	makePayload()로 리턴 받은 배열에 메시지를 담고 Timer 객체를 이용해 1초에 지정한 수만큼 메시지를 publish
time_disconnect()	정해진 duration 이후에 Publisher 객체 리스트에서 해당 publisher를 disconnect 후 삭제
stop_disconnect()	Stop 버튼을 통해 master로부터 종료 지시가 내려졌을 경우 해당 publisher들을 disconnect 한 후 publisher 리스트 전부 삭제
selectPayloadSize()	원하는 범위 내에서 1개의 정수형 숫자를 랜덤으로 뽑아 리턴
makePayload()	selectPayloadSize()의 리턴 값의 크기를 가진 byte 배열을 만들고 리턴
subscriber package - Instance.java	
run()	지정한 범위 내에 지정한 개수의 subscriber 생성
subscriber package - Subscriber.java	
topic_subscribe()	Timer 객체를 이용해 지정한 duration동안 subscriber가 살아있도록 하며, 선택한 토픽을 구독, 지속시간 이후에는 time_disconnect() 호출
selectTopic()	리스트에 담긴 토픽들 중 하나의 토픽을 선택
time_disconnect()	정해진 duration 이후에 subscriber 객체 리스트에서 해당 subscriber를 disconnect 후 삭제
stop_disconnect()	Stop 버튼을 통해 master로부터 종료 지시가 내려졌을 경우 해당 subscriber들을 disconnect 한 후 subscriber 리스트 전부 삭제

2.2.2.3 메시지 서버 성능 모니터링 PC (Ubuntu Linux)

컴퓨터 성능 데이터 관련 - performance package	
함수명	설명
CpuInfo.java	
findNumberOfCores()	cpu 코어 개수를 구하는 메서드
update()	interval 동안의 총 cpu 사용량, core 별 cpu 사용량을 업데이트 하는 메서드
MemoryInfo.java	
findTotalMemory()	총 메모리 크기를 구하는 메서드
update()	interval 동안의 메모리 사용량을 업데이트하는 메서드
NetworkInfo.java	
findInterfaceName()	사용하는 network interface 이름을 구하는 메서드
update()	interval 동안의 네트워크 input, output을 업데이트하는 메서드


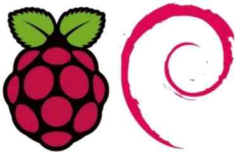



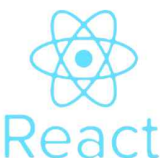






Performance.java	
start()	timer와 timertask를 실행하여 주기적으로 데이터베이스에 성능 정보를 저장하도록 하는 메서드
setBasicInfoTable()	총 메모리 크기와 코어 개수를 데이터베이스에 저장
update()	모든 성능 정보를 업데이트
executePs()	ps 명령어를 실행하여 실행중인 프로세스 리스트를 불러오는 메서드
트래픽 데이터 관련 - log_reader package	
Publisher.java	
publish()	토픽 이름과 처리량 정보를 가지는 publication을 생성하여 vector에 저장하는 메서드
Subscriber.java	
subscribe()	토픽 이름과 처리량 정보를 가지는 subscription을 생성하여 vector에 저장하는 메서드
receivedMessage()	구독 중인 토픽으로 메시지가 수신되면 처리량을 갱신하는 메서드
LogReader.java	
startTimer()	timer와 timertask를 실행하여 주기적으로 데이터베이스에 트래픽 정보를 저장하도록 하는 메서드
read()	mosquitto log를 한 줄씩 읽어 필요한 트래픽 정보를 업데이트

2.2.2.4 웹 서버 PC (Ubuntu Linux)

Node.js - MQTTDashboard/node-server	
이름	내용
mysql.createConnection()	node 서버와 mysql 서버를 연동하는 메서드
mqtt.connect()	node 서버와 mqtt broker를 연결하는 메서드
mysqlConnection.query()	mysql 데이터베이스에 연결 후 쿼리문 사용하여 데이터에 접근하는 메서드
setTimeout()	데이터 접근을 반복적으로 실행하기 위해 사용
clearAllTimeout()	모든 setTimeout()을 멈추는 메서드
client.on("message",...)	mqtt message가 도착했을 때 실행되는 메서드
client.on("connect",...)	mqtt connect가 성공했을 때 실행되는 메서드
client.on("error",...)	mqtt 관련 error가 발생했을 때 실행되는 메서드
msgTopic()	topic 테이블에 접근하여 데이터 가져오는 메서드
subscription()	subscription 테이블에 접근하여 데이터 가져오고 publish하는 메서드
processTable()	process 테이블에 접근하여 데이터 가져오고 publish하는 메서드
memory()	performance 테이블의 memory 관련 데이터 가져오고 publish하는 메서드
network()	performance 테이블의 network 관련 데이터 가져오고 publish하는 메서드
cpu()	performance 테이블의 cpu 관련 데이터 가져오고 publish하는 메서드
accumulate()	connection_info 테이블의 client 관련 데이터 가져오고 publish하는 메서드
accumulationPie()	accumulated 파이차트에 띄울 topic 테이블의 accumulated_msdg_size 필드 데이터 publish하는 메서드
receivingPie()	수신 Topic TOP3 파이차트에 띄울 데이터 publish하는 메서드
sendingPie()	송신 Topic TOP3 파이차트에 띄울 데이터 publish하는 메서드
connentions()	connection_info 테이블의 client 연결 정보 데이터 publish 하는 메서드
getAlltopics()	topic 테이블의 모든 topic 리스트를 가져오는 메서드

React.js - MQTTDashboard/react-client	
이름	내용
componentDidMount()	컴포넌트 Mount시 호출되는 이벤트 메소드
componentWillUnmount()	컴포넌트 Unmount시 호출되는 이벤트 메소드
componentWillReceiveProps(nextProps)	컴포넌트의 props가 갱신되면 호출되는 이벤트 메소드
messageArrived(data)	props로 전달받은 메시지를 차트 데이터로 바꿔서 차트를 갱신하는 메소드
mqtt.on('message', ...)	MQTT Message 도착 시 호출되는 Callback 메소드

2.3 프로젝트 개발 환경

			
Windows10	Raspbian Linux	Ubuntu Linux	Eclipse
			
Node.js	React.js	MySQL	Mosquitto MQTT broker
			
Chrome Debugger	Atom Editor	Java Fx	VS Code

3. 개발 중 장애요인과 해결방안

3.1 프로젝트 중 장애요인과 해결방안

•데이터베이스 선택의 어려움

MongoDB, Altibase, MySQL 등 여러 데이터베이스 중 어떤 데이터베이스를 사용할지 고민했었다. 결국 팀원 모두가 사용 해 본 적이 있어 익숙하고, node와 연동하기 쉽다는 장점이 있는 MySQL을 선택하게 되었다.

•데이터 가공의 어려움

데이터베이스로부터 가져온 데이터를 어떠한 형식으로 publish 해야 할지 회의 한 결과, “”을 사용하여 String문자열을 나눌 수 있도록 하는 브라우저에 띄울 때 가장 간단하고 어려움이 없는 적당한 방법을 결정했다.

•가공 된 데이터를 브라우저로 보내는 방법의 어려움

가장 처음 프로젝트를 구성할 때 웹 브라우저로 data push 방법으로 AJAX, WebSocket,

http2의 web-push 등 여러 방법을 고민하고 실제로 구현까지 해 보았지만 각각 node와의 연동이나 데이터 푸시 형식의 충돌 등 사소한 문제들이 생겨 최종적으로 서버 컴퓨터에 mosquitto broker를 설치하여 데이터를 publish하는 방식을 선택하게 되었다. 길지 않은 코드로 구현이 가능하며 본 프로젝트에서 원하는 String형식의 데이터를 보내는 것이 가능하다는 장점이 있다.

4. 기대 효과 및 활용분야

4.1 기대 효과

1. 대용량 메시지 서버의 모니터링 기술을 확보한다.
2. 웹 기반의 HTML5 대시보드를 확보한다.
3. 로드 제너레이션 기술을 확보한다.
4. MQTT 기반의 모든 메시지 서버에 적용 가능한 플랫폼 독립적인 시스템이다.

4.2 활용 분야

- MQTT 통신 프로토콜을 사용하는 서버라면 서버의 역할이 무엇이고 topic 설계가 어떻게 이루어지든 상관없이 대시보드 모니터링 시스템을 적용하여 관리할 수 있다.
- 카카오톡, 라인 등의 채팅 서버라면 채팅방 하나하나가 topic이 될 수 있고, 구독자와 발행자가 자연스럽게 생성된다.
- 재난 관리 서버라면 어느 지역에, 어떠한 재난/범죄가 많이 발생하는지 등을 모니터링 가능하게 한다. 지역에 따라 또는 재난/범죄의 종류에 따라 topic 설계가 다양하게 가능하며, 그 topic 설계에 따라 대시보드를 통해 한 눈에 재난/범죄의 통계치를 모니터링할 수 있다.
- 또 다른 예시로 스트리밍 서비스가 있다.

예시) 스트리밍 서비스

최근 가장 핫한 분야 중 하나가 스트리밍 서비스이다. 동영상 스트리밍, 음악 스트리밍 등 여러 종류의 스트리밍이 인기를 끌고 있으며 이러한 스트리밍 서비스를 제공하는 서버도 MQTT 통신을 사용한다면 우리의 시스템 활용이 충분히 가능하다.

MQTT 기반 topic 설계는 다음과 같이 된다고 가정한다.

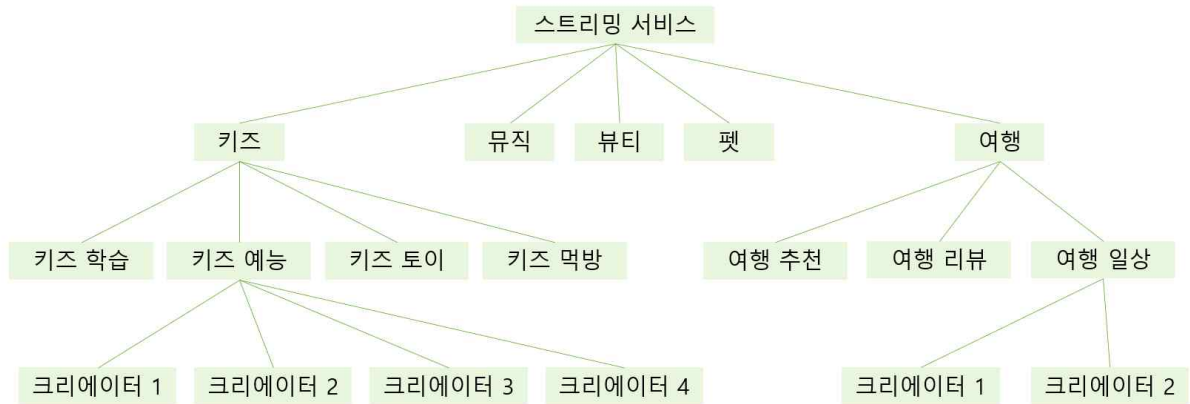


그림 32. 동영상 스트리밍 서버의 topic 설계

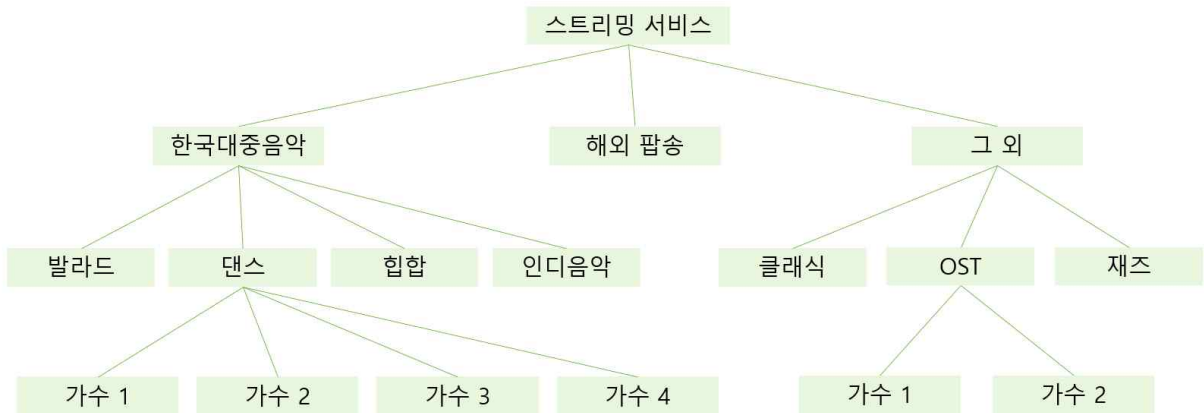


그림 33. 음악 스트리밍 서버의 topic 설계

위 topic 중 동영상 스트리밍 서비스를 예로 들어 본다.

- 동영상 크리에이터 A가 채널 개설을 할 때, 카테고리를 선택하면 해당 카테고리/서브 카테고리/A의 닉네임으로 <스트리밍/메인 카테고리/서브 카테고리/A의 닉네임>이라는 topic이 생성된다.
- 사용자들은 크리에이터의 채널을 구독하게 되는데, 이 행위가 MQTT 서버에서 해당 크리에이터 채널의 topic을 구독하는 것과 일치한다.
- 이후, 크리에이터가 영상을 업로드 하거나 알림을 띄울 때 해당 채널(topic)을 구독하고 있는 모든 사용자들에게 알림이 가도록 서버에서 MQTT 메시지를 전송한다.
- 이 모든 과정을 웹 페이지 대시보드로 모니터링 가능하다. topic 별 메시지 송수신 횟수는 카테고리 별 구독과 알림 횟수로, topic 트리와 구독자 리스트는 카테고리 구성과 해당 채널 구독자 리스트로, client 정보 차트는 스트리밍 서버에 가장 오래, 가장 최근에 접속한 구독자 아이디 등으로 나타내어질 것이다.



그림 34. 동영상 스트리밍 서비스 예시

5. 프로젝트 수행 추진 체계 및 일정

5.1 각 조원의 조직도



5.2 역할 분담

- 이 도 연 : 메시지 서버의 성능 모니터링 파트
- 김 시 훈 : 테스트 로드 시스템 파트
- 변 민 정 : 대시보드 웹 시스템 파트
- 손 지 혜 : 실시간 성능 데이터 출력 웹 서비스 파트

5.3 주 단위의 프로젝트 수행 일정

업무		담당자	4월			5월			6월			7월			8월			9월		
주제 선정		팀																		
자료 수집		팀																		
관련기술 학습		팀																		
구조도 작성		팀																		
개발환경 구축		팀																		
설 계	웹 서비스	손지혜																		
		변민정																		
	성능 모니터링	이도연																		
		테스트 로드	김시훈																	
구 현	웹 서비스	손지혜																		
		변민정																		
	성능 모니터링	이도연																		
		테스트 로드	김시훈																	
파트 병합		팀																		
테스트		팀																		
보고서 작성		팀																		

6. 참고 자료

-ubuntu에 mosquito broker with websockets

<http://www.yasith.me/2017/08/enable-websocket-support-in-mosquitto.html>

-React.js 강의

<https://velopert.com/reactjs-tutorials>

-Node.js 강의

<https://velopert.com/node-js-tutorials>

-JAVA 서적

명품 JAVA Programming