



SPN v3.0(a.k.a IoTSec) - LoRa Security Project *(Part II)*

08.21.2019 ~

Doc. Revision: 1.5

We Secure the Internet of Things with 2STON SPN.

Chunghan.Yi(michael@2ipco.com)
R&D Center
2ip Inc.

Table of Contents

Part I.

1. LoRa 개요
2. RAKWireless RAK831 LoRaWAN Kit
3. Dragino IoT Kit v2
4. Dragino LG308 LoRa Gateway
5. RAKWireless **RAK7258 LoRa Gateway**
6. RAKWireless **RAK7249 Outdoor LoRa Gateway**
7. MatchX **MX1702 LoRa Gateway(LBT 지원 모델)**

Part II.

8. MultiTech **MultiConnect Conduit IP67 LoRa Gateway**
9. **LoRa Gateway에 SPN S/W Porting하기**
10. **LoRaServer Project 1 - 설치 & 운용**
11. **LoRaServer Project 2 - External Interface**
12. **ThingsBoard Integration - 대박 :)**
13. Our LoRa Viewer: **OLoRa(= ThingsBoard)**

Part III.

14. **Outdoor LoRa Node - Libelium**
15. **URSALink LoRa Products**
16. **LoRaWAN Stack**

Part II. Chapter 8 ~ Chapter 13.

8. MultiTech MultiConnect Conduit IP67 LoRa Gateway

이 장에서는 KR920/LBT를 지원하는 또 다른 모델 즉, **MultiConnect Conduit**를 소개해 보고자 한다. 기필코 국내에서 LoRa 제품을 만들어 판매하고야 말 것이다^^. 너무 비장한가 흥 흥.

<2STON SPN의 방향>

1. IP Camera 보안 Solution
2. Texaking/NF model 보안 Solution - 유사한 형태의 Project 진행
3. LoRa Security & Management Solution - 국내/국외(아시아권) LoRa solution 제공/망 구축

1) MultiTech MultiConnect Conduit 제품 소개



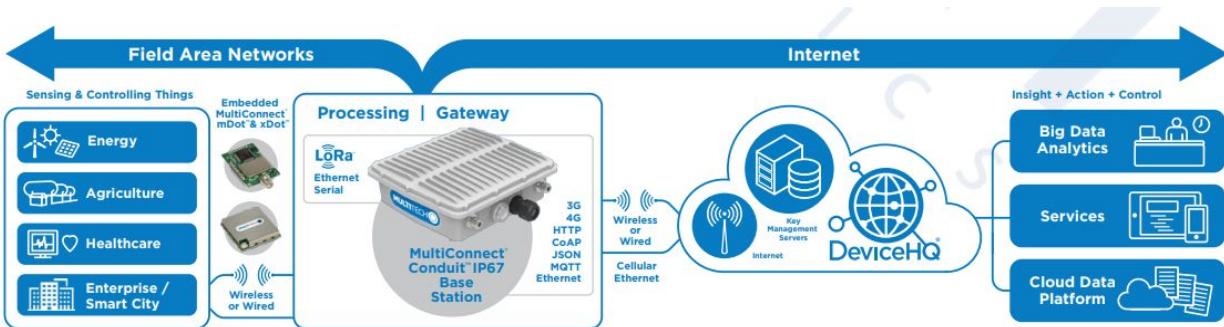
[그림 8.1] MultiTech MultiConnect Conduit 제품(실내형 및 옥외형)



[그림 8.2] MultiTech MultiConnect Conduit 제품(실내형)



[그림 8.3] MultiTech MultiConnect Conduit IP67(옥외형)



[그림 8.4] MultiTech MultiConnect Conduit IP67 기반 LoRa Network 구성

2) MultiTech MultiConnect Conduit 제품 상세 분석

<TBD> 제품을 하나 구매해야 한다 ~

[http://www.multitech.net/developer/wp-content/uploads/downloads/2018/04/AEP-1.4.16-U
pguideGuide.pdf](http://www.multitech.net/developer/wp-content/uploads/downloads/2018/04/AEP-1.4.16-UpgradeGuide.pdf)

9. LoRa Gateway에 SPN S/W Porting 하기

이번 장에서는 RAK7258 LoRa Gateway에 SPN kernel module을 porting 하는 절차를 소개해 보고자 한다. RAK7258은 OpenWrt를 기반으로 하고 있으나, RAKWireless 사에서는 code를 open하지 않고 있다. 그렇다고 이대로 포기할 수는 없는 일이다^^.

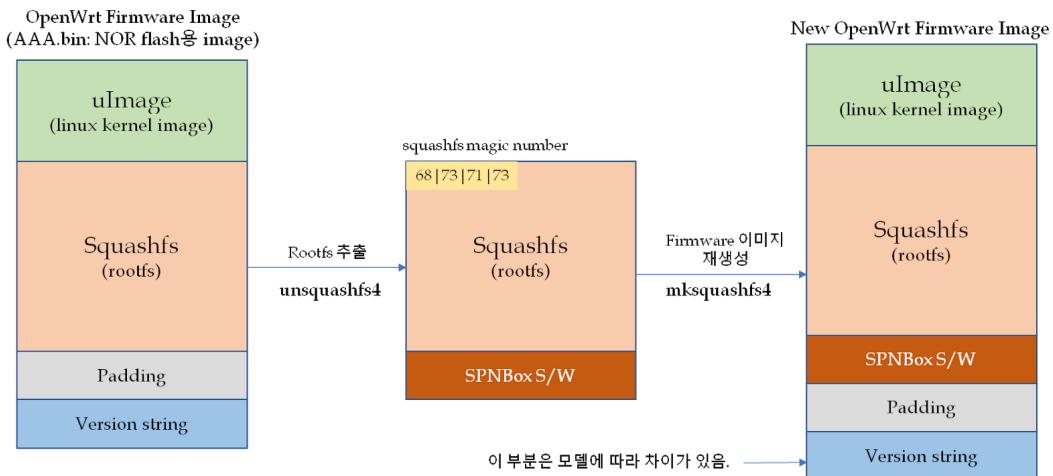
<SPN Kernel Module Porting 절차>

- 1) RAK7258 firmware 이미지 해부
→ SPN S/W 탑재 가능성 탐진
- 2) OpenWrt code 확보 - RAK7258용은 아니지만, linux 3.18.45 kernel을 사용하는 RAKWireless 제공 openwrt를 찾아 보아야 한다.
→ 아래, 하나를 찾았다.
→ <https://github.com/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628>

3) SPN kernel module porting

1) RAK7258 firmware image 해부하기

다행이도, RAK7258은 Gl.iNet SPNBox-200Y(망고 박스)와 CPU(MediaTek MT7628AN)가 동일하다. 따라서 아래와 같은 firmware 분해 및 재생성 과정을 적용해 볼 수 있다. 어디서 많이 본 그림이다. ㅎㅎ



[그림 9.1] RAK7258 firmware image 해부 절차도

```
$ sudo ./openwrt-repack.sh ./LoRaGateway_1.1.0049_Release_r182.bin rak7249
```

- 기존에 만들어 두었던 openwrt-repack.sh를 약간 수정하여 시도!
- OK, firmware image가 kernel과 rootfs로 정상 분해 된다.

```
>>> Analysing source firmware: ./LoRaGateway_1.1.0049_Release_r182.bin ...
Found SquashFS at 1195073.
>>> Extracting kernel, rootfs partitions ...
>>> Extracting SquashFS into directory squashfs-root/ ...
Parallel unsquashfs: Using 8 processors
2221 inodes (2238 blocks) to write

[=====]
] 2238/2238 100%
created 1776 files
```



```
created 131 directories
created 444 symlinks
created 1 devices
created 0 fifos
>>> Patching the firmware with spnbox s/w...
Checking rc.d links for changed services ...
>>> Repackaging the modified firmware ...
Parallel mksquashfs: Using 1 processor
Creating 4.0 filesystem on root.squashfs, block size 262144.
Pseudo file "/dev/console" exists in source filesystem "squashfs-root/dev/console".
Ignoring, exclude it (-e/-ef) to override.
Pseudo file "/dev" exists in source filesystem "squashfs-root/dev".
Ignoring, exclude it (-e/-ef) to override.
[=====]/
1793/1793 100%
Exportable Squashfs 4.0 filesystem, xz compressed, data block size 262144
    compressed data, compressed metadata, compressed fragments, no xattrs
    duplicates are removed
Filesystem size 7272.22 Kbytes (7.10 Mbytes)
    30.66% of uncompressed filesystem size (23721.53 Kbytes)
Inode table size 20217 bytes (19.74 Kbytes)
    24.65% of uncompressed inode table size (82029 bytes)
Directory table size 24366 bytes (23.79 Kbytes)
    43.60% of uncompressed directory table size (55883 bytes)
Number of duplicate files found 73
Number of inodes 2352
Number of files 1776
Number of fragments 66
Number of symbolic links 444
Number of device nodes 1
Number of fifo nodes 0
Number of socket nodes 0
Number of directories 131
Number of ids (unique uids + gids) 1
```

```

Number of uids 1
root (0)

Number of gids 1
root (0)

padding image to 00840000

>>> Done. New firmware: ./LoRaGateway_1.1.0049_Release_r182.bin.out

```

```

chyi@mars:~/workspace/new_boards/RAKWireless/RAK7258$ ls -la
합계 4112
drwxr-xr-x 4 chyi chyi 4096 9월  2 10:55 .
drwxr-xr-x 8 chyi chyi 4096 9월  4 15:05 ..
-rw-r--r-- 1 chyi chyi 8650756 8월 22 17:39 LoRaGateway_1.1.0049_Release_r182.bin
-rw-r--r-- 1 chyi chyi 8650756 9월  2 10:55 LoRaGateway_1.1.0049_Release_r182.bin.out
-rw-rw-r-- 1 chyi chyi 8650976 9월  2 09:49 LoRaGateway_1.1.0049_Release_r182.rar
-rw xr-xr-x 1 chyi chyi 8640 9월  2 09:58 openwrt-repack.sh
drwxr-xr-x 2 chyi chyi 4096 4월 17 17:17 openwrt_tools
-rw-r--r-- 1 chyi chyi 7446871 9월  2 10:55 root.squashfs
-rw-r--r-- 1 chyi chyi 7455683 9월  2 10:55 root.squashfs.orig
drwxr-xr-x 16 chyi chyi 4096 8월 21 19:39 squashfs-root
-rw-r--r-- 1 chyi chyi 1195073 9월  2 10:55 uImage.bin
chyi@mars:~/workspace/new_boards/RAKWireless/RAK7258$ cd squashfs-root/
chyi@mars:~/workspace/new_boards/RAKWireless/RAK7258/squashfs-root$ ls -la
합계 64
drwxr-xr-x 16 chyi chyi 4096 8월  21 19:39 .
drwxr-xr-x 4 chyi chyi 4096 9월  2 10:55 ..
drwxr-xr-x 2 chyi chyi 4096 8월 21 19:39 bin
drwxr-xr-x 2 chyi chyi 4096 8월 21 19:37 dev
drwxr-xr-x 24 chyi chyi 4096 8월 21 19:39 etc
drwxr-xr-x 12 chyi chyi 4096 8월 21 19:38 lib
drwxr-xr-x 2 chyi chyi 4096 8월 21 19:37 mnt
drwxr-xr-x 2 chyi chyi 4096 8월 21 19:37 overlay
drwxr-xr-x 2 chyi chyi 4096 8월 21 19:37 proc
drwxr-xr-x 2 chyi chyi 4096 8월 21 19:39 rom
drwxr-xr-x 2 chyi chyi 4096 8월 21 19:37 root
drwxr-xr-x 2 chyi chyi 4096 8월 21 19:39 sbin
drwxr-xr-x 2 chyi chyi 4096 8월 21 19:37 sys
drwxr-xr-t 2 chyi chyi 4096 8월 21 19:39 tmp
drwxr-xr-x 6 chyi chyi 4096 8월 21 19:37 usr
lrwxrwxrwx 1 chyi chyi 4 9월  2 10:55 var -> /tmp
drwxr-xr-x 4 chyi chyi 4096 8월 21 19:34 www
chyi@mars:~/workspace/new_boards/RAKWireless/RAK7258/squashfs-root$ █

```

[그림 9.2] RAK7258 firmware image 해부 결과

RAK7258 firmware가 kernel(uImage.bin)과 rootfs(squashfs-root)로 정상 분리되었으니, 여기에 SPN S/W를 탑재하여 새로운 firmware image를 생성하는 절차만 남았다.

2) RAK831 용 OpenWRT 검토하기

RAK7258용 공식 openwrt code는 open 안되어 있으나, linux kernel version을 가지고, 역으로 아래 site(RAK831용 openwrt source site)를 찾았다. 어차피 SPN kernel module을 build하기 위해서는 kernel source와 toolchain만 있으면 되므로, 아래 site 내용이면 작업하는데 충분할 것으로 보인다. 우선 코드를 내려 받아 정상 build가 되는지를 확인해 보도록 하자.

```
$ git clone https://git.archive.openwrt.org/15.05/openwrt.git
```

```
$ cd RAK831-LoRaGateway-OpenWRT-MT7628/
```

```
$ ./build/envsetup.sh
```

→ Ubuntu 18.04 환경에서 build 시도

```
...
...
Checking 'svn'... ok.
Checking 'git'... failed.
Checking 'file'... ok.
Checking 'openssl'... ok.
Checking 'ldconfig-stub'... ok.

Build dependency: Please install Git (git-core) >= 1.6.5

/home/chyi/workspace/new_boards/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628/wiswrt/15.05-rak-rc2/include/prereq.mk:12: recipe for target 'prereq' failed
Prerequisite check failed. Use FORCE=1 to override.
/home/chyi/workspace/new_boards/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628/wiswrt/15.05-rak-rc2/include/toplevel.mk:140: recipe for target 'staging_dir/host/.prereq-build' failed
make[8]: *** [staging_dir/host/.prereq-build] Error 1
make[8]: Leaving directory '/home/chyi/workspace/new_boards/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628/wiswrt/15.05-rak-rc2'
/home/chyi/workspace/new_boards/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628/wiswrt/15.05-rak-rc2/include/toplevel.mk:69: recipe for target 'prepare-tmpinfo' failed
make[7]: *** [prepare-tmpinfo] Error 2
make[7]: Leaving directory '/home/chyi/workspace/new_boards/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628/wiswrt/15.05-rak-rc2'
/home/chyi/workspace/new_boards/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628/wiswrt/15.05-rak-rc2/include/toplevel.mk:181: recipe for target 'package/utils/packet_forwarder/clean' failed
make[6]: *** [package/utils/packet_forwarder/clean] Error 2
make[6]: 디렉터리 '/home/chyi/workspace/new_boards/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628/wiswrt/15.05-rak-rc2' 나감
Makefile:9: recipe for target 'compile' failed
```



```

make[5]: *** [compile] Error 2
make[5]: 디렉터리
'/home/chyi/workspace/new_boards/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628/products/wisLora/package/utis' 나감
Makefile:13: recipe for target 'compile' failed
make[4]: *** [compile] Error 2
make[4]: 디렉터리
'/home/chyi/workspace/new_boards/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628/products/wisLora/package' 나감
Makefile:11: recipe for target 'compile' failed
make[3]: *** [compile] Error 2
make[3]: 디렉터리
'/home/chyi/workspace/new_boards/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628/products/wisLora' 나감
Makefile:7: recipe for target 'compile' failed
make[2]: *** [compile] Error 2
make[2]: 디렉터리 '/home/chyi/workspace/new_boards/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628/products' 나감
Makefile:20: recipe for target 'compile' failed
make[1]: *** [compile] Error 2
make[1]: 디렉터리
'/home/chyi/workspace/new_boards/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628/build/target' 나감
Makefile:67: recipe for target 'compile' failed
make: *** [compile] Error 2
/home/chyi/workspace/new_boards/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628//wiswrt/15.05-rak-rc2

```

(어라)에러가 발생한다?! ... 아무래도 예전 버전이라 Ubuntu 18.04에서 문제가 되는 것 같다.
그렇다면, **Ubuntu 16.04(202 server)** 환경에서 build해 보면 어떨까?

<Ubuntu 16.04 환경에서 build>

```
$ sudo apt-get install build-essential subversion git-core libncurses5-dev zlib1g-dev
gawk flex quilt libssl-dev xsltproc libxml-parser-perl mercurial bzr ecj cvs unzip
```

→ Build에 필요한 몇가지 package 설치는 기본(18.04에서도 동일한 패키지 설치해 주었다).

```
$ git clone
https://github.com/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628.git
→ source 를 내려 받자.
$ ./build/envsetup.sh
```

- Build를 위한 환경 설정 절차를 진행한다.
- OK, 문제가 되었던 부분이 해결되었다.

\$ make

- build가 정상적으로 진행된다.

```
...
...
Generating index for package ./spi-tools_1-cc6a41fdcec60610703ba6db488c621c64952898_ramips_24kec.ipk
Generating index for package ./luci-app-firewall_git-17.136.58961-13aa5ff-1_all.ipk
Generating index for package ./luci-app-mjpg-streamer_git-17.136.58961-13aa5ff-1_all.ipk
Generating index for package ./luci-app-samba_git-17.136.58961-13aa5ff-1_all.ipk
Generating index for package ./luci-base_git-17.136.58961-13aa5ff-1_ramips_24kec.ipk
Generating index for package ./luci-lib-ip_git-17.136.58961-13aa5ff-1_ramips_24kec.ipk
Generating index for package ./luci-lib-json_git-17.136.58961-13aa5ff-1_all.ipk
Generating index for package ./luci-lib-nxio_git-17.136.58961-13aa5ff-1_ramips_24kec.ipk
Generating index for package ./luci-mod-admin-full_git-17.136.58961-13aa5ff-1_ramips_24kec.ipk
Generating index for package ./luci-proto-ipv6_git-17.136.58961-13aa5ff-1_all.ipk
Generating index for package ./luci-proto-ppp_git-17.136.58961-13aa5ff-1_all.ipk
Generating index for package ./luci-theme-bootstrap_git-17.136.58961-13aa5ff-1_all.ipk
Generating index for package ./luci-theme-openwrt_git-17.136.58961-13aa5ff-1_all.ipk
Generating index for package ./luci_git-17.136.58961-13aa5ff-1_all.ipk
Signing package index...
make[3]: Leaving directory
'/home/chyi/workspace/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628/wiswrt/15.05-rak-rc2'
make[2]: Leaving directory
'/home/chyi/workspace/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628/wiswrt/15.05-rak-rc2'
make[1]: Leaving directory
'/home/chyi/workspace/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628/wiswrt/15.05-rak-rc2'
cp
/home/chyi/workspace/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628/build/..//wiswrt/15.05-rak-rc2/
bin/ramips/openwrt-ramips-mt7628-mt7628-squashfs-sysupgrade.bin out/target/bin
Building openwrt finished.
kernel image into out/target/bin
```

3) WireGuard Kernel Porting하기

자, 그렇다면 이제부터는 wireguard kernel module을 porting할 차례이다.

3-1) wireguard porting 시도[1차]

먼저 wireguard를 kernel 외부에서 직접 build해 보도록 하자. 이를 위해서는 먼저 몇가지 환경 변수를 export해 주어야 한다.

<toolchain path>

```
$ export
PATH=~/workspace/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628/wiswrt/15.
05-rak-rc2/staging_dir/toolchain-mipsel_24kec+dsp_gcc-4.8-linaro_uClibc-0.9.33.2/bin:
$PATH
```

<staging dir>

```
$ export
STAGING_DIR=~/workspace/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628/wi
swrt/15.05-rak-rc2/staging_dir
```

```
$ export ARCH=mips
$ export CROSS_COMPILE=mipsel-openwrt-linux-
```

다음으로 적당한 위치에 wireguard code를 내려 받도록 한다.

```
$ git clone https://git.zx2c4.com/WireGuard
```

→ 이거 대신 아래 stable version으로 해 보자.

```
$ wget https://git.zx2c4.com/WireGuard/snapshot/WireGuard-0.0.20190905.tar.xz
```

<wireguard source compile>

→ Build를 시도해 본다.

```
$ cd src
$ KERNELDIR=~/workspace/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628/wis
```

wrt/15.05-rak-rc2/build_dir/target-mipsel_24kec+dsp_uClibc-0.9.33.2/linux-ramips_mt7628/linux-3.18.45

SRCDIR=~/workspace/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628/wiswrt/15.05-rak-rc2/build_dir/target-mipsel_24kec+dsp_uClibc-0.9.33.2/linux-ramips_mt7628/linux-3.18.45 CROSS_COMPILE=mipsel-openwrt-linux- ARCH=mips make

ERROR: Kernel configuration is invalid.

include/generated/autoconf.h or include/config/auto.conf are missing.

Run 'make oldconfig && make prepare' on kernel src to fix it.

WARNING: Symbol version dump ./Module.symvers

is missing; modules will have no dependencies and modversions.

CC [M]

/home/chyi/workspace/RAKWireless/20190905/WireGuard-0.0.20190905/src/main.o

In file included from <command-line>:0:0:

./include/linux/kconfig.h:4:32: fatal error: generated/autoconf.h: No such file or directory

#include <generated/autoconf.h>

 ^

compilation terminated.

scripts/Makefile.build:257: recipe for target

'/home/chyi/workspace/RAKWireless/20190905/WireGuard-0.0.20190905/src/main.o'

failed

make[2]: ***

[/home/chyi/workspace/RAKWireless/20190905/WireGuard-0.0.20190905/src/main.o]

Error 1

Makefile:1384: recipe for target

'_module_/home/chyi/workspace/RAKWireless/20190905/WireGuard-0.0.20190905/src'

failed

make[1]: ***

module/home/chyi/workspace/RAKWireless/20190905/WireGuard-0.0.20190905/src]

Error 2

Makefile:38: recipe for target 'module' failed

make: *** [module] Error 2

어라, 에러가 발생한다. 뭐가 문제일까? 이후 몇가지 시도해 본 내용은 생략 :(아무래도 이 방법 보다는 다른 방법을 찾는 것이 맞을 듯 싶다.

3-2) wireguard porting 시도[2차]

이번에는 다른 방법으로 문제를 풀어 보자. 즉, linux kernel에 wireguard code를 patch한 후, openwrt를 build하는 과정 중에 wireguard가 build 되도록 해보자. Wireguard가 동작하기 위해서는 아래 feature가 모두 enable되어야 한다는 점을 고려하면, 결국은 이 방법이 맞을 듯 싶다. 다시말해 앞선 방법이 성공하더라도, 이 방법을 다시 반복해 주어야만 한다 뜻이다~

<wireguard가 정상 동작하기 위해 enable되어야 하는 kernel features>

[*] Networking support -->	
Networking options -->	
[*] TCP/IP networking	
[*] IP: Foo (IP protocols) over UDP	
[*] Cryptographic API -->	
[*] Cryptographic algorithm manager	<= cryptomgr.ko
	<= CONFIG_CRYPTO_MANAGER2

참고: wireguard kernel build를 위한 방법은 아래 site 하단을 참조하기 바란다.

<https://www.wireguard.com/install/>

<Kernel directory로 이동 후 아래 명령 수행>

→ dl/folder에 있는 linux-3.18.45.tar.xz file의 압축을 푼 후, wireguard patch 적용. 이후 다시 압축하여 원본 파일에 복사하자.

```
$ cd RAK831-LoRaGateway-OpenWRT-MT7628/wiswrt/15.05-rak-rc2/dl
```

```
$ ls -l linux-3.18.45.tar.xz
```

```
$ mkdir tmp; cp ./linux-3.18.45.tar.xz tmp
```

```
$ tar xvJf ./linux-3.18.45.tar.xz
```

→ xvJf의 J는 대문자임(주의).

```
$ cd linux-3.18.45
```

```
$ ~/wireguard/contrib/kernel-tree/create-patch.sh | patch -p1
```



```
patching file net/wireguard/allowedips.c
patching file net/wireguard/compat/dst_cache/dst_cache.c
patching file net/wireguard/compat/memneq/memneq.c
patching file net/wireguard/compat/siphash/siphash.c
patching file net/wireguard/compat/udp_tunnel/udp_tunnel.c
patching file net/wireguard/cookie.c
patching file net/wireguard/crypto/zinc/blake2s/blake2s-x86_64-glue.c
patching file net/wireguard/crypto/zinc/blake2s/blake2s.c
patching file net/wireguard/crypto/zinc/chacha20/chacha20-arm-glue.c
patching file net/wireguard/crypto/zinc/chacha20/chacha20-mips-glue.c
patching file net/wireguard/crypto/zinc/chacha20/chacha20-x86_64-glue.c
patching file net/wireguard/crypto/zinc/chacha20/chacha20.c
patching file net/wireguard/crypto/zinc/chacha20poly1305.c
patching file net/wireguard/crypto/zinc/curve25519/curve25519-arm-glue.c
patching file net/wireguard/crypto/zinc/curve25519/curve25519-fiat32.c
patching file net/wireguard/crypto/zinc/curve25519/curve25519-hacl64.c
patching file net/wireguard/crypto/zinc/curve25519/curve25519-x86_64-glue.c
patching file net/wireguard/crypto/zinc/curve25519/curve25519-x86_64.c
patching file net/wireguard/crypto/zinc/curve25519/curve25519.c
patching file net/wireguard/crypto/zinc/poly1305/poly1305-arm-glue.c
patching file net/wireguard/crypto/zinc/poly1305/poly1305-donna32.c
patching file net/wireguard/crypto/zinc/poly1305/poly1305-donna64.c
patching file net/wireguard/crypto/zinc/poly1305/poly1305-mips-glue.c
patching file net/wireguard/crypto/zinc/poly1305/poly1305-x86_64-glue.c
patching file net/wireguard/crypto/zinc/poly1305/poly1305.c
patching file net/wireguard/crypto/zinc/selftest/blake2s.c
patching file net/wireguard/crypto/zinc/selftest/chacha20.c
patching file net/wireguard/crypto/zinc/selftest/chacha20poly1305.c
patching file net/wireguard/crypto/zinc/selftest/curve25519.c
patching file net/wireguard/crypto/zinc/selftest/poly1305.c
patching file net/wireguard/device.c
patching file net/wireguard/main.c
```



```
patching file net/wireguard/netlink.c
patching file net/wireguard/noise.c
patching file net/wireguard/peer.c
patching file net/wireguard/peerlookup.c
patching file net/wireguard/queueing.c
patching file net/wireguard/ratelimiter.c
patching file net/wireguard/receive.c
patching file net/wireguard/selftest/allowedips.c
patching file net/wireguard/selftest/counter.c
patching file net/wireguard/selftest/ratelimiter.c
patching file net/wireguard/send.c
patching file net/wireguard/socket.c
patching file net/wireguard/timers.c
patching file net/wireguard/allowedips.h
patching file net/wireguard/compat/checksum/checksum_partial_compat.h
patching file net/wireguard/compat/compat-compat-asm.h
patching file net/wireguard/compat/compat.h
patching file net/wireguard/compat/dst_cache/include/net/dst_cache.h
patching file net/wireguard/compat/fpu-x86/include/asm/fpu/api.h
patching file net/wireguard/compat/intel-family-x86/include/asm/intel-family.h
patching file net/wireguard/compat/memneq/include.h
patching file net/wireguard/compat/neon-arm/include/asm/neon.h
patching file net/wireguard/compat/ptr_ring/include/linux/ptr_ring.h
patching file net/wireguard/compat/simd-asm/include/asm/simd.h
patching file net/wireguard/compat/simd/include/linux/simd.h
patching file net/wireguard/compat/siphash/include/linux/siphash.h
patching file net/wireguard/compat/udp_tunnel/include/net/udp_tunnel.h
patching file net/wireguard/compat/udp_tunnel/udp_tunnel_partial_compat.h
patching file net/wireguard/cookie.h
patching file net/wireguard/crypto/include/zinc/blake2s.h
patching file net/wireguard/crypto/include/zinc/chacha20.h
patching file net/wireguard/crypto/include/zinc/chacha20poly1305.h
patching file net/wireguard/crypto/include/zinc/curve25519.h
patching file net/wireguard/crypto/include/zinc/poly1305.h
```



```
patching file net/wireguard/crypto/zinc.h
patching file net/wireguard/crypto/zinc/selftest/run.h
patching file net/wireguard/device.h
patching file net/wireguard/messages.h
patching file net/wireguard/netlink.h
patching file net/wireguard/noise.h
patching file net/wireguard/peer.h
patching file net/wireguard/peerlookup.h
patching file net/wireguard/queueing.h
patching file net/wireguard/ratelimiter.h
patching file net/wireguard/socket.h
patching file net/wireguard/timers.h
patching file net/wireguard/uapi/wireguard.h
patching file net/wireguard/version.h
patching file net/wireguard/crypto/zinc/blake2s/blake2s-x86_64.S
patching file net/wireguard/crypto/zinc/chacha20/chacha20-mips.S
patching file net/wireguard/crypto/zinc/chacha20/chacha20-unrolled-arm.S
patching file net/wireguard/crypto/zinc/curve25519/curve25519-arm.S
patching file net/wireguard/crypto/zinc/poly1305/poly1305-mips.S
patching file net/wireguard/crypto/zinc/chacha20/chacha20-arm.pl
patching file net/wireguard/crypto/zinc/chacha20/chacha20-arm64.pl
patching file net/wireguard/crypto/zinc/chacha20/chacha20-x86_64.pl
patching file net/wireguard/crypto/zinc/poly1305/poly1305-arm.pl
patching file net/wireguard/crypto/zinc/poly1305/poly1305-arm64.pl
patching file net/wireguard/crypto/zinc/poly1305/poly1305-mips64.pl
patching file net/wireguard/crypto/zinc/poly1305/poly1305-x86_64.pl
patching file net/wireguard/compat/Makefile.include
patching file net/wireguard/crypto/Makefile.include
patching file net/wireguard/Makefile
patching file net/wireguard/Kconfig
patching file net/Kconfig
patching file net/Makefile
Hunk #1 succeeded at 18 (offset 2 lines).
```

```
$ cd ..  
$ tar cvjf linux-3.18.45.tar.xz ./linux-3.18.45  
$ cp ./linux-3.18.45.tar.xz ..
```

→ 상위 folder로 복사한다.

<kernel config file 수정>

→ 다음으로 아래와 같이 몇가지 kernel config를 추가해 주자.

```
$ vi  
RAK831-LoRaGateway-OpenWRT-MT7628/products/wisLora/target/linux/ramips/mt7628/config-3.18
```

→ 파일 끝에 아래 내용 추가

```
CONFIG_WIREGUARD=m  
CONFIG_WIREGUARD_DEBUG=y  
CONFIG_NET_UDP_TUNNEL=m  
CONFIG_NET_FOU=m  
CONFIG_CRYPTO_BLKCIPHER=m  
=> [다시 시도]  
CONFIG_WIREGUARD=m  
#CONFIG_WIREGUARD_DEBUG=y  
CONFIG_NET_UDP_TUNNEL=m  
CONFIG_NET_FOU=m  
CONFIG_CRYPTO_BLKCIPHER=m
```

<build/envsetup.sh 파일 수정>

→ 위에서 추가한 kernel config 내용이 정상적으로 수정되었는지를 확인하기 위해, build/envsetup.sh를 실행하는 중에 kernel menuconfig 화면을 띄울 수 있도록 아래와 같이 코드 수정을 한다.

```
target() {  
    cd $TOPDIR/build  
    if [ "$option" = "null" ]; then  
        echo "Compile target....."  
        make $MAKE_COMMAND BUILD_TARGET=$DEF_BUILD_TARGET  
        mkdir -p $OPENWRT_TREE  
        cd $OPENWRT_TREE  
        echo $OPENWRT_TREE
```

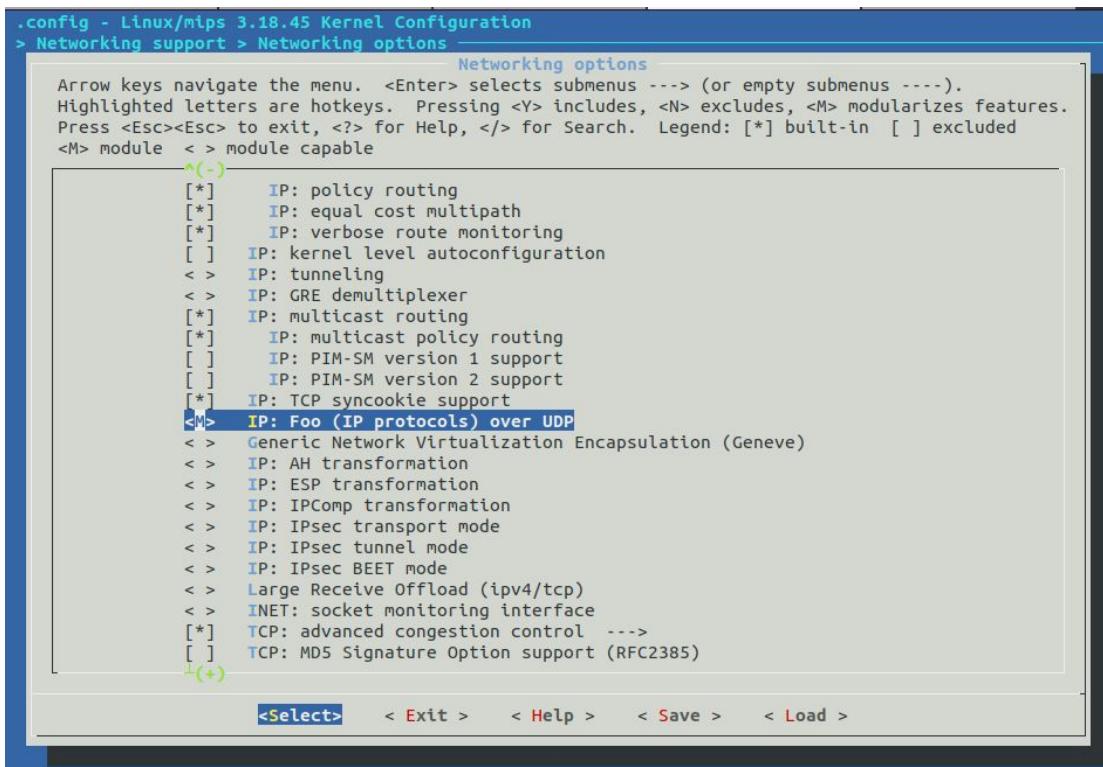
```
#added by michael@2019.09.06 --
make kernel_menuconfig # 여기서 kernel config(wireguard Module) 확인

exit 0
make $MAKE_COMMAND V=99 -j1
fi
if [ "$option" = "clean" ]; then
    echo "Clean target....."
    make $MAKE_COMMAND BUILD_TARGET=target clean
    for i in $CLEAN_DIRS
    do
        echo "Clean temp files and dir.....$i"
        ./clean-tmp.sh $i $BOARD
    done
fi
cd $TOPDIR
```

<kernel config 확인 후, openwrt 전체 build하기>

\$ build/envsetup.sh

- 아래 부분이 M으로 지정(CONFIG_NET_FOU=m)되어 있는지 확인
- 물론 당연히 WireGuard도 M으로 지정되어 있는지 확인해야 함.



[그림 9.3] openwrt kernel menuconfig - for wireguard

\$ make

- 정상 build가 진행된다.
- 전체 build하는데는 30분 가량이 소요된다. 하지만, 우리는 kernel module이 build되기만 하면 이후 테스트를 진행할 수 있으니, 끝까지 기다릴 필요는 없다.

\$ cd

RAK831-LoRaGateway-OpenWRT-MT7628/wiswrt/15.05-rak-rc2/build_dir/target-mipsel_24k
ec+dsp_uClibc-0.9.33.2/linux-ramips_mt7628/linux-3.18.45

- 여기서 wireguard.ko를 포함한 몇가지 kernel module을 찾아 RAK7258 장비로 복사하자.

<RAK7258로 복사해야 할 kernel modules>

```
-rw-r--r-- 1 chyi chyi 7397 9월 6 18:12 fou.ko
-rw-r--r-- 1 chyi chyi 3460 9월 6 18:14 ip6_udp_tunnel.ko
-rw-r--r-- 1 chyi chyi 3810 9월 6 18:13 udp_tunnel.ko
-rw-r--r-- 1 chyi chyi 309954 9월 6 19:11 wireguard.ko
```

자, 이제부터는 RAK7258 target board 상에서 앞서 build한 kernel module을 구동시켜 보자.

<Target board - RAK7258>

- Insmod 순서가 중요하다.

```
# insmod ./udp_tunnel.ko
# insmod ./ip6_udp_tunnel.ko
# insmod ./fou.ko
# insmod ./wireguard.ko
```

failed to insert ./wireguard.ko

dmesg

```
...
[ 5357.250000] wireguard: Unknown symbol ip6_dst_hoplimit (err 0)
[ 5357.260000] wireguard: Unknown symbol ipv6_chk_addr (err 0)
```

```
[ 5357.270000] wireguard: Unknown symbol icmpv6_send (err 0)
```

위의 에러 부분을 wireguard code에서 찾아 보니, 모두 CONFIG_IPV6 코드와 연관되어 있다. Kernel config에서 CONFIG_IPV6를 disable하는 방법을 찾는게 맞겠지만, (몇번 시도해 본 바) 쉽게 해결되지가 않는다. 그렇다면, 일단 wireguard 코드를 직접 수정하여 해당 부분을 막아 보도록 하자(당분간 ipv6용 SPN 터널을 사용할 일은 없을 테니 말이다).

이후 다시 시도해 보니, 앞선 에러는 사라졌지만, 이번에는 다른 에러가 보인다.

```
# insmod ./wireguard.ko
```

```
failed to insert ./wireguard.ko
```

```
# dmesg
```

```
...
[ 1891.160000] wireguard: chacha20 self-tests: pass
[ 1891.190000] wireguard: poly1305 self-tests: pass
[ 1891.210000] wireguard: chacha20poly1305 self-tests: pass
[ 1891.220000] wireguard: blake2s self-tests: pass
[ 1891.910000] wireguard: curve25519 self-tests: pass
[ 1891.920000] wireguard: allowedips self-tests: pass
[ 1891.940000] wireguard: nonce counter self-tests: pass
[ 1891.950000] wireguard: ratelimiter self-test 14: FAIL
```

- 이 문제는 CONFIG_WIREGUARD_DEBUG=y와 연관이 있다. 따라서 이 flag를 끈 상태에서 테스트해 보자.

참고: dmesg로 보는 것도 한 방법이지만, console 상에서 실행하게 되면, 에러 메시지가 바로 화면에 출력된다. RAK7258는 RJ45 console port를 제공하고 있는데, 57600, 8N1 설정(예: **sudo picocom 57600 /dev/ttyUSB0**)을 해 주어야만 한다.

이번에는 wireguard self test routine에서 문제가 생겼다. 역시 (어떻게 될지 모르니) 일단 막아 보자. 코드는 main.c에서 호출되고 있다.

```
# insmod ./wireguard.ko
```

dmesg

→ OK, 드디어 올라간다.

[5268.060000] wireguard: WireGuard 0.0.20190905 loaded. See www.wireguard.com for information.

[5268.080000] wireguard: Copyright (C) 2015-2019 Jason A. Donenfeld <jason@zx2c4.com>. All Rights Reserved.

혹시, 앞서 강제로 수정한 것 때문에 정상 동작에 문제가 없는지는 추후에 다시 봄 일이다.

참고: 본 문서에서는 wireguard code 수정 부분은 별도로 명기하지 않는다. 어려운 내용이 아니니, 위의 예러를 보고 직접 수정해 보기 바란다.

4) WireGuard wg(application program) Porting 하기

자, wireguard.ko가 target board에 올라 갔으니, 이제부터는 wg program을 porting하는 일만 남았다.

<cross-compile 환경 설정하기>

<toolchain path>

```
$ export
PATH=/home/chyi/workspace/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628/
wiswrt/15.05-rak-rc2/staging_dir/toolchain-mipsel_24kec+dsp_gcc-4.8-linaro_uClibc-0.
9.33.2/bin:$PATH
```

<staging dir>

```
$ export
STAGING_DIR=/home/chyi/workspace/RAKWireless/RAK831-LoRaGateway-OpenWRT-
MT7628/wiswrt/15.05-rak-rc2/staging_dir
```

<arch & compiler>

```
$ export ARCH=mips
$ export CROSS_COMPILE=mipsel-openwrt-linux-
```

<libmnl cross compile 하기>

wg binary를 생성 및 실행하기 위해서는 libmnl library가 필요하다. 따라서 애를 먼저 build한 후, wg binary(src/tools/)를 생성해 보도록 하자.

```
$ git clone git://git.netfilter.org/libmnl
$ cd libmnl/
$ ./autogen.sh
$ ./configure --host=mipsel-openwrt-linux --enable-shared
$ make
```

\$ cd src/.libs

```
chyi@twoip-desktop:~/workspace/RAKWireless/libmnl/src/.libs$ ls -la
total 144
drwxrwxr-x 2 chyi chyi 4096 9월  9 10:38 .
drwxrwxr-x 4 chyi chyi 4096 9월  9 10:38 ..
-rw-rw-r-- 1 chyi chyi 25580 9월  9 10:38 attr.o
-rw-rw-r-- 1 chyi chyi 10760 9월  9 10:38 callback.o
lrwxrwxrwx 1 chyi chyi    12 9월  9 10:38 libmnl.la -> ../libmnl.la
-rw-rw-r-- 1 chyi chyi   918 9월  9 10:38 libmnl.lai
lrwxrwxrwx 1 chyi chyi    15 9월  9 10:38 libmnl.so -> libmnl.so.0.2.0
lrwxrwxrwx 1 chyi chyi    15 9월  9 10:38 libmnl.so.0 -> libmnl.so.0.2.0
-rwxrwxr-x 1 chyi chyi 56260 9월  9 10:38 libmnl.so.0.2.0
-rw-rw-r-- 1 chyi chyi 17784 9월  9 10:38 nlmsg.o
-rw-rw-r-- 1 chyi chyi 12584 9월  9 10:38 socket.o
```

[그림 9.4] libmnl cross compile 결과물

Libmnl이 준비되었으니, wg binary를 build해 보기로 하자.

```
$ cd WireGuard/src/tools
$ vi Makefile
→ Makefile을 아래와 같이 수정하도록 하자.
```

```

endif
endif

PLATFORM ?= $(shell uname -s | tr '[[:upper:]]' '[[:lower:]]')

CFLAGS ?= -O3
CFLAGS += -std=gnu99 -D_GNU_SOURCE
CFLAGS += -Wall -Wextra
CFLAGS += -MMD -MP
CFLAGS += -DRUNSTATEDIR="\$(RUNSTATEDIR)\"
ifeq ($(DEBUG_TOOLS),y)
CFLAGS += -g
endif
ifeq ($(PLATFORM),linux)
LIBMNL_CFLAGS := $(shell $(PKG_CONFIG) --cflags libmnl 2>/dev/null)
LIBMNL_LDLIBS := $(shell $(PKG_CONFIG) --libs libmnl 2>/dev/null || echo -lmnl)
CFLAGS += $(LIBMNL_CFLAGS)
#nichael@2018.09.10 --
#LDLIBS += $(LIBMNL_LDLIBS)
CFLAGS += -I/home/chyi/workspace/RAKWireless/libmnl/include
LDLIBS += -L/home/chyi/workspace/RAKWireless/libmnl/src/.libs -lmnl

endif
ifeq ($(PLATFORM),haiku)
LDLIBS += -lnetwork -lbsd
endif
ifeq ($(PLATFORM),windows)
CC := x86_64-w64-mingw32-gcc
CFLAGS += -Iwincompat/include -include wincompat/compat.h
LDLIBS += -lwsl_32
wg: wincompat/libc.o wincompat/init.o
endif

ifneq ($(V),1)
BUILT_IN_LINK.o := $(LINK.o)

```

[그림 9.5] wg Makefile 수정 내용

\$ CC=mipsel-openwrt-linux-gcc LD=mipsel-openwrt-linux-ld make

→ Cross-compile을 해 보자.

```

drwxrwxr-x 2 chyi chyi 4096 9월 6 11:03 man
-rw-rw-r-- 1 chyi chyi 7558 9월 6 11:03 mnlg.c
-rw-rw-r-- 1 chyi chyi 428 9월 9 10:56 mnlg.d
-rw-rw-r-- 1 chyi chyi 755 9월 6 11:03 mnlg.h
-rw-rw-r-- 1 chyi chyi 6960 9월 9 10:56 mnlg.o
-rw-rw-r-- 1 chyi chyi 1230 9월 6 11:03 pubkey.c
-rw-rw-r-- 1 chyi chyi 174 9월 9 10:56 pubkey.d
-rw-rw-r-- 1 chyi chyi 2576 9월 9 10:56 pubkey.o
-rw-rw-r-- 1 chyi chyi 1026 9월 6 11:03 set.c
-rw-rw-r-- 1 chyi chyi 150 9월 9 10:56 set.d
-rw-rw-r-- 1 chyi chyi 2268 9월 9 10:56 set.o
-rw-rw-r-- 1 chyi chyi 1441 9월 6 11:03 setconf.c
-rw-rw-r-- 1 chyi chyi 158 9월 9 10:56 setconf.d
-rw-rw-r-- 1 chyi chyi 2924 9월 9 10:56 setconf.o
-rw-rw-r-- 1 chyi chyi 14957 9월 6 11:03 show.c
-rw-rw-r-- 1 chyi chyi 180 9월 9 10:56 show.d
-rw-rw-r-- 1 chyi chyi 20496 9월 9 10:56 show.o
-rw-rw-r-- 1 chyi chyi 2851 9월 6 11:03 showconf.c
-rw-rw-r-- 1 chyi chyi 164 9월 9 10:56 showconf.d
-rw-rw-r-- 1 chyi chyi 3776 9월 9 10:56 showconf.o
-rw-rw-r-- 1 chyi chyi 460 9월 6 11:03 subcommands.h
drwxrwxr-x 2 chyi chyi 4096 9월 6 11:03 systemd
-rw-rw-r-- 1 chyi chyi 1507 9월 6 11:03 terminal.c
-rw-rw-r-- 1 chyi chyi 23 9월 9 10:56 terminal.d
-rw-rw-r-- 1 chyi chyi 1705 9월 6 11:03 terminal.h
-rw-rw-r-- 1 chyi chyi 2820 9월 9 10:56 terminal.o
-rw-rw-r-- 1 chyi chyi 83502 9월 9 10:56 wg
drwxrwxr-x 2 chyi chyi 4096 9월 6 11:03 wg-quick
-rw-rw-r-- 1 chyi chyi 2078 9월 6 11:03 wg.c
-rw-rw-r-- 1 chyi chyi 41 9월 9 10:56 wg.d
-rw-rw-r-- 1 chyi chyi 4612 9월 9 10:56 wg.o
drwxrwxr-x 3 chyi chyi 4096 9월 6 11:03 wincompat
chyi@twotp-desktop:~/workspace/RAKWireless/Wireguard/src/tools$ file wg
wg: ELF 32-bit LSB executable, MIPS, MIPS32 rel2 version 1, dynamically linked, interpreter /lib/ld-, not stripped

```

[그림 9.6] wg cross-compile 결과물

Compile이 완료되었으니, 이제 target board에 올려, 동작 시험을 해 보자.

<target board - RAK7258>

아래 내용은 target board에 올린 파일을 보여준다.

```

root@RAK7258:~/workspace# ls -la
drwxr-xr-x 2 root      root          0 Sep  9 02:08 .
drwxr-xr-x 1 root      root          0 Sep  6 08:08 ..
-rw-r--r-- 1 root      root        7397 Sep  6 09:12 fou.ko
-rw-r--r-- 1 root      root       3460 Sep  6 09:15 ip6_udp_tunnel.ko
lrwxrwxrwx 1 root      root         15 Sep  9 02:05 libmnl.so -> libmnl.so.0.2.0
lrwxrwxrwx 1 root      root         15 Sep  9 02:05 libmnl.so.0 -> libmnl.so.0.2.0
-rwxrwxr-x 1 1001     1001      56260 Sep  9 01:38 libmnl.so.0.2.0
-rwxr-xr-x 1 root      root         99 Sep  9 02:03 start_wg.sh
-rw-r--r-- 1 root      root       3810 Sep  6 09:14 udp_tunnel.ko
-rwxr-xr-x 1 root      root     83502 Sep  9 02:05 wg
-rw-r--r-- 1 root      root    309954 Sep  6 10:11 wireguard.ko
root@RAK7258:~/workspace#

```

[그림 9.7] target board에 올려둔 binary 파일

```

root@RAK7258:~/workspace# export
LD_LIBRARY_PATH=/root/workspace:$LD_LIBRARY_PATH

```

→ Wg를 실행하기 위해 shared library path를 잡아주자.

```
root@RAK7258:~/workspace# ./wg -h
```

→ wg가 정상 실행되는지 확인해 본다.

```
Usage: ./wg <cmd> [<args>]
```

Available subcommands:

show: Shows the current configuration and device information

showconf: Shows the current configuration of a given WireGuard interface, for use with `setconf`

set: Change the current configuration, add peers, remove peers, or change peers

setconf: Applies a configuration file to a WireGuard interface

addconf: Appends a configuration file to a WireGuard interface

genkey: Generates a new private key and writes it to stdout

genpsk: Generates a new preshared key and writes it to stdout

pubkey: Reads a private key from stdin and writes a public key to stdout

You may pass `--help` to any of these subcommands to view usage.

```
# ip link add wg0 type wireguard
```

→ wg0 interface를 추가한다.

```

        collisions:0 txqueuelen:0
        RX bytes:11984439 (11.4 MiB) TX bytes:1345162 (1.2 MiB)

lo      Link encap:Local Loopback
        inet addr:127.0.0.1 Mask:255.0.0.0
        UP LOOPBACK RUNNING MTU:65536 Metric:1
        RX packets:95 errors:0 dropped:0 overruns:0 frame:0
        TX packets:95 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:14531 (14.1 KiB) TX bytes:14531 (14.1 KiB)

mon0    Link encap:Ethernet HWaddr 60:C5:A8:76:13:0C
        BROADCAST MULTICAST MTU:1500 Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

ra0      Link encap:Ethernet HWaddr 60:C5:A8:76:13:0C
        UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
        RX packets:209773 errors:0 dropped:0 overruns:0 frame:0
        TX packets:11145 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:49345354 (47.0 MiB) TX bytes:11650817 (11.1 MiB)
        Interrupt:6

wg0      Link encap:UNSPEC HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
        POINTOPOINT NOARP MTU:1420 Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

```

[그림 9.8] 생성된 wg0 interface

```
# ifconfig wg0 10.1.1.30 netmask 255.255.255.0 up
```

Segmentation fault

```
# ifconfig -a
```

어라 명령이 안먹힌다.... 다른 terminal로 로긴해서 보니, kernel panic이 발생한 것이 보인다.
위에서 했던 **wireguard.ko porting**에 문제가 있는 것 같다.

```
[ 5227.760000] wireguard: WireGuard 0.0.20190905 loaded. See www.wireguard.com for
information.
[ 5227.780000] wireguard: Copyright (C) 2015-2019 Jason A. Donenfeld
<Jason@zx2c4.com>. All Rights Reserved.
[ 5612.150000] wireguard: wg0: Interface created
[ 5952.500000] CPU 0 Unable to handle kernel paging request at virtual address
00000018, epc == 871e313c, ra == 80201628
```

```
[ 5952.520000] Oops[#1]:
[ 5952.520000] CPU: 0 PID: 3913 Comm: ifconfig Not tainted 3.18.45 #1
[ 5952.520000] task: 86045d10 ti: 86170000 task.ti: 86170000
[ 5952.520000] $ 0 : 00000000 00000000 00000000 870cdæ0
[ 5952.520000] $ 4 : 8615f420 00000000 ffffffff 00000001
[ 5952.520000] $ 8 : 8032a36c 00000001 00000000 00000000
[ 5952.520000] $12 : 000000d1 776de3a0 00000000 1e01010a
[ 5952.520000] $16 : 8615f000 8615f000 871fc988 00000000
[ 5952.520000] $20 : 00000000 870cda00 ffffffff 1e01010a
[ 5952.520000] $24 : 00000000 776bf260
[ 5952.520000] $28 : 86170000 86171d90 00000100 80201628
[ 5952.520000] Hi : 000002f4
[ 5952.520000] Lo : 0001cae9
[ 5952.520000] epc : 871e313c wg_noise_handshake_begin_session+0xffc/0x1094
[wireguard]
[ 5952.520000]   Not tainted
[ 5952.520000] ra : 80201628 __dev_open+0xe0/0x148
[ 5952.520000] Status: 1100e403 KERNEL EXL IE
[ 5952.520000] Cause : 0080000c
[ 5952.520000] BadVA : 000000018
[ 5952.520000] PrId : 00019655 (MIPS 24KEc)
[ 5952.520000] Modules linked in: wireguard fou ip6_udp_tunnel udp_tunnel pppoe
ppp_async option iptable_nat usb_wwan qmi_wwan pppox ppp_synctty ppp_mppe
ppp_generic nf_nat_ipv4 nf_conntrack_ipv4 ipt_REJECT ipt_MASQUERADE ftdi_sio cp210x
ch341 xt_time xt_tcpudp xt_tcmrss xt_string xt_statistic xt_state xt_recent xt_quota
xt_pktype xt_owner xt_nat xt_multiport xt_mark xt_mac xt_limit xt_length xt_id xt_hl
xt_helper xt_ecn xt_dscp xt_conntrack xt_connmark xt_connlimit xt_connbytes
xt_comment xt_addrtype xt_TCPMSS xt_REDIRECT xt_LOG xt_HL xt_DSCP xt_CT
xt_CLASSIFY usbserial usbnet ums_usbat ums_sddr55 ums_sddr09 ums_karma
ums_jumpshot ums_isd200 ums_freecom ums_datafab ums_cypress ums_alauda ts_kmp
ts_fsm ts_bm spidev slhc nf_reject_ipv4 nf_nat_masquerade_ipv4 nf_nat_ftp nf_nat
nf_log_ipv4 nf_log_common nf_defrag_ipv4 nf_conntrack_rtcache nf_conntrack_ftp
nf_conntrack lz4_decompress lz4_compress iptable_raw iptable_mangle iptable_filter
ipt_ECN ip_tables crc_ccitt cdc_wdm i2c_ralink mt_wifi i2c_dev ledtrig_usbdev xt_set
x_tables ip_set_list_set ip_set_hash_netiface ip_set_hash_netport ip_set_hash_netnet
ip_set_hash_net ip_set_hash_netportnet ip_set_hash_mac ip_set_hash_ipportnet
ip_set_hash_ipportip ip_set_hash_ipport ip_set_hash_ipmark ip_set_hash_ip
ip_set_bitmap_port ip_set_bitmap_ipmac ip_set_bitmap_ip ip_set_nfnetlink ntfs
sha1_generic ecb arc4 crypto_blkcipher edev mmc_block usb_storage sdhci_pltfm sdhci
mtk_sd mmc_core leds_gpio ohci_pci ohci_platform ohci_hcd ehci_pci ehci_platform
ehci_hcd sd_mod scsi_mod gpio_button_hotplug vfat fat ext4 jbd2 mbcache nls_utf8
nls_iso8859_1 nls_cp437 usbcore nls_base usb_common crc16 mii aead crypto_hash
[ 5952.520000] Process ifconfig (pid: 3913, threadinfo=86170000, task=86045d10,
tls=77765440)
[ 5952.520000] Stack : 8033dbe0 ffffffff 872e5200 00000001 8615f000 8615f02c 871fc988
80201628
[ 5952.520000]          000000d1 80201530 00000000 8025c978 8615f000 000000d1
```

```

00000000 80201918
[ 5952.520000] 00000000 8003adac 872e5200 80340000 00000000 8615f000
7f85a6e0 00000090
[ 5952.520000] 00000000 802019f8 872e5200 86171e30 7f85a6e0 80214384
00000000 872e5200
[ 5952.520000] 7f85a6e0 870cda0c 8615f000 8025f1bc 000000a8 86171e58
7f85a6e0 80214384
[ 5952.520000] ...
[ 5952.520000] Call Trace:
[ 5952.520000] [<871e313c>] wg_noise_handshake_begin_session+0xffc/0x1094
[wireguard] <----- 여기서 죽었는데 ... 왜 일까 ?
[ 5952.520000] [<8003adac>] blocking_notifier_call_chain+0x14/0x20
[ 5952.520000] [<802019f8>] dev_change_flags+0x28/0x70
[ 5952.520000] [<8025f1bc>] devinet_ioctl+0x650/0x688
[ 5952.520000] [<801e990c>] sock_ioctl+0x2b0/0x30c
[ 5952.520000] [<800b0338>] do_vfs_ioctl+0x4f0/0x5fc
[ 5952.520000] [<800b0494>] SyS_ioctl+0x50/0x94
[ 5952.520000] [<80006b5c>] handle_sys+0x11c/0x140
[ 5952.520000]
[ 5952.520000]
[ 5952.520000] Code: ac400084 3c028034 8c42bbdc <ac400018> 10a00002 24020001
a0a20175 0dc7a11f 962505b4
[ 5953.180000] ---[ end trace 441ec365961b2602 ]---
root@RAK7258:~#

```

문제가 뭘까 ? 원인을 파악해 보도록 하자.

ip address add 10.1.1.30 dev wg0

ip link set wg0 up ← interface up 시에 문제 발생함.

Segmentation fault

```

[ 717.790000] Call Trace:
[ 717.790000] [<862630fc>] wg_noise_handshake_begin_session+0x1210/0x12bc
[wireguard] <----- 역시 똑 같은 point에서 죽었다.
[ 717.790000] [<8021007c>] do_setlink+0x290/0x6e4
[ 717.790000] [<8021159c>] rtnl_newlink+0x384/0x5ec

```

Wireguard code를 kernel에 integration 시킨 상태에서 build한 후, 해당 kernel로 부팅해 보는 것이 답인데 일단 rakwireless.com에 메일이나 보내 두자.

Dear Ken Yu,

*My name is Michael working at 2ip, Inc in South Korea.
We are developing 2STON SPN product that means Secure Private Network and is composed of security gateways and some clients.*

== == ==

*Recently we've got your RAK7258 LoRa Gateways(3 sets) and tested them.
We are planning to use your H/W and would like to customize your S/W(OpenWrt) to our purpose.*

*Can I get your OpenWrt source codes ?
To be honest, I can't guarantee anything about the business at this time.
Please tell me the way to agreement process if possible.*

*Best regards,
Michael.*

Hello, Michael

*Sorry, we don't provide the source code of our openWRT. We only provide the SDK.
Normally, we can release the SDK if customer order 10pcs Gateway as MOQ! PLs let me know if the SDK is good enough for you?*

Ken

헐, 10대를 사면 SDK를 준다고 SDK에는 뭐가 들어가 있는지도 모르는데 ... 껌 안되겠군 ...
지금까지의 방법으로 해결책을 모색해 보는 수 밖에 ...

지금부터 **wg_noise_handshake_begin_session()** 함수에서 die하는 이유를 찾아 보자.

Shit~ 하루 원종일 시도해 보았는데.... 뾰족한 답이 없다.

여기다가 오늘 try한 내용을 모두 답을 수는 없는 노릇이고 ... 결국, kernel source만 있으면 해결되는 문제인데 ... 껌

아래 site는 위의 kernel panic 에러를 분석해 주는 내용을 담고 있는데, 초심자가 이해하기는 쉽지 않을 수 있다. 그래도 무슨 얘긴지 한번 훑어 보시라 ...

http://wiki.dreamrunner.org/public_html/Embedded-System/kernel/AnalyzeLinuxKernelCrashesOnMIPS.html

Analyze Linux kernel crashes on the MIPS platform - Chromium

Analyze Linux kernel crash x +

① 주의 요함 | wiki.dreamrunner.org/public_html/Embedded-System/kernel/AnalyzeLinuxKernelCrashesOnMIPS.html

Table of Contents

Analyze Linux kernel crashes on the MIPS platform

Introduction¹

The aim of this post is to illustrate the analysis of Linux kernel crashes by studying a few real-life examples. The examples are coming from a MIPS platform, but the general approach is applicable to other architectures.

Tools

Any general purpose disassembler is sufficient. We'll use objdump with '-d' option here.

If a binary was built with debugging information, objdump -S can display source code intermixed with disassembly **1**. Also, **addr2line** can be used to match addresses with source code file names and lines.

In order to interpret disassembly, we need to have the **MIPS Instruction Reference** and the **Compiler Register Usage information** at hand **2**, so please keep these pages open while reading further material.

If you are not familiar with how the virtual memory space is divided on MIPS, please refer to 'Virtual Memory Layout' **3** in the last section.

Format of a crash report

Here is an example of a crash report:

```
CPU 0 Unable to handle kernel paging request at virtual address 00000000, epc == 8023af0, ra == 8023b024
Oops[#1]:
Cpu 0
$ 0 : 00000000 1000fc00 8555fc54 00000000
$ 4 : 00000000 00000000 0000000b 00000001
$ 8 : 00000008 800445f4 00000000 00000000
```

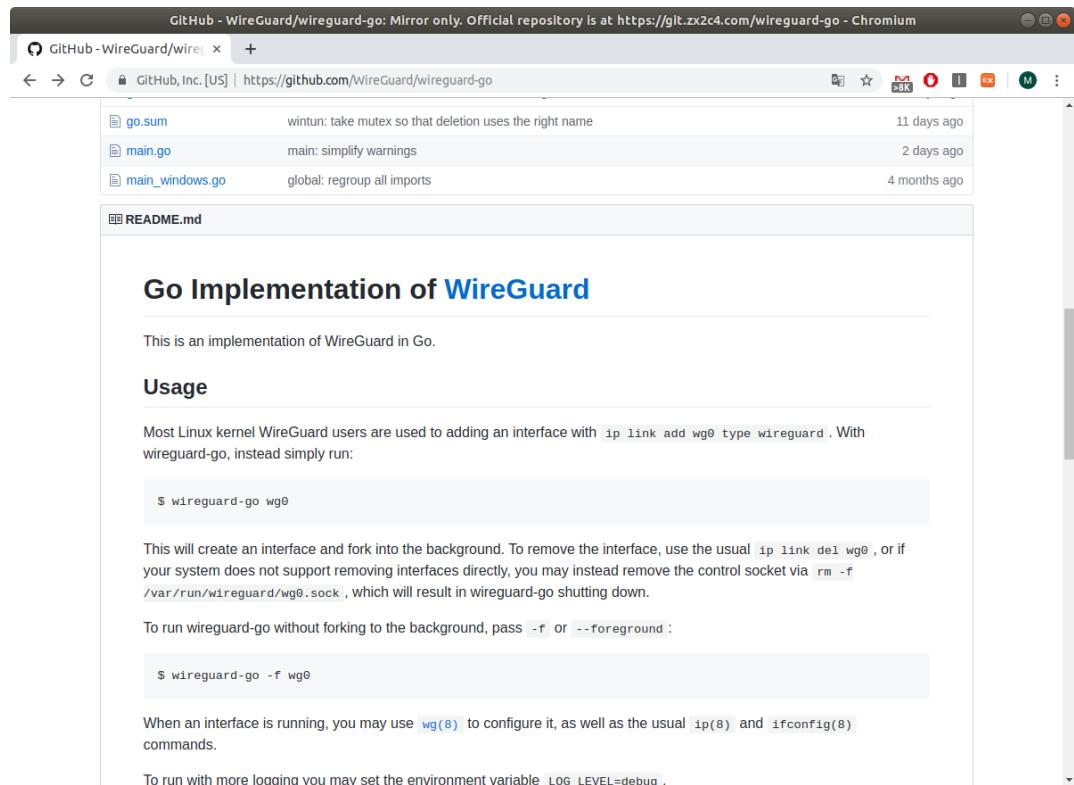
[그림 9.9] MIPS에서의 linux kernel panic 분석 site[참고용]

5) WireGuard-Go porting 하기

이렇게 끝내면 너무 싱겁지 않은가? 궁하면 통하는 법 ~ **kernel module**로 안된다면, **Wireguard-Go**로 접근해 보자.

LoRa Gateway의 경우는 많은 양의 data를 암호화할 필요가 없다. 따라서 다소 느린 wireguard-go로 접근하는 것도 나쁘지 않을 것 같다. 더군다나 packet forwarder(일종의 proxy임)가 패킷을 LoRa Server로 전달하는 구조이므로 NAT를 태울 필요도 없는 상황이다. :)

<wireguard-go code download>



[그림 9.10] Wireguard-Go page

```
$ git clone https://github.com/WireGuard/wireguard-go
```

```
$ cd wireguard-go/
```

< MIPS용 cross-compile 하기 >

```
$ GOOS=linux GOARCH=mipsle go build
```

- mipsel이 아니라 mipsle이다. mipsle는 little-endian 방식을 의미한다(RAK7258의 CPU는 MediaTek MT7628AN으로 little endian 방식임).
- 아래 파일 내용 중, **wireguard-go**는 x86용으로 build(그냥 **go build**만 하면 됨)한 결과물이고, **wireguard**가 MIPS용으로 build한 녀석이다.

참고: 당연한 거지만, go code를 build하려면, go compiler가 설치되어 있어야 한다. 간단한 내용이므로 여기서는 생략한다.

```
chyi@mars:~/workspace/new_boards/WG/wireguard-go$ ls -la
합계 7900
drwxr-xr-x 11 chyi chyi 4096 9월 10 16:48 .
drwxr-xr-x  3 chyi chyi 4096 9월 10 16:44 ..
drwxr-xr-x  8 chyi chyi 4096 9월 10 16:46 .git
-rw-r--r--  1 chyi chyi    57 9월 10 16:44 .gitignore
-rw-r--r--  1 chyi chyi 1023 9월 10 16:44 COPYING
-rw-r--r--  1 chyi chyi   832 9월 10 16:44 Makefile
-rw-r--r--  1 chyi chyi 3879 9월 10 16:44 README.md
drwxr-xr-x  2 chyi chyi 4096 9월 10 16:44 device
-rw-r--r--  1 chyi chyi   244 9월 10 16:44 go.mod
-rw-r--r--  1 chyi chyi 1400 9월 10 16:44 go.sum
drwxr-xr-x  3 chyi chyi 4096 9월 10 16:44 ipc
-rw-r--r--  1 chyi chyi 6147 9월 10 16:44 main.go
-rw-r--r--  1 chyi chyi 1953 9월 10 16:44 main_windows.go
drwxr-xr-x  2 chyi chyi 4096 9월 10 16:44 ratelimiter
drwxr-xr-x  2 chyi chyi 4096 9월 10 16:44 replay
drwxr-xr-x  2 chyi chyi 4096 9월 10 16:44 rwcancel
drwxr-xr-x  2 chyi chyi 4096 9월 10 16:44 tai64n
drwxr-xr-x  2 chyi chyi 4096 9월 10 16:44 tests
drwxr-xr-x  3 chyi chyi 4096 9월 10 16:44 tun
-rwxr-xr-x  1 chyi chyi 3912243 9월 10 16:48 wireguard
-rwxr-xr-x  1 chyi chyi 4156657 9월 10 16:46 wireguard-go
```

[그림 9.11] Wireguard-Go cross-compile 결과물

Target board로 wireguard 실행 파일을 복사하자.

```
$ scp ./wireguard root@172.30.1.26:~/workspace
```

<target board>

자, 모든 것이 준비되었으니, RAK7258 target board에서 wireguard 설정을 진행해 보도록 하자.

```
# ./wireguard
```

→ Wireguard를 실행하면, 아래와 같은 경고 문구가 나온다. 내용인 즉, linux 용으로는 kernel version이 더 좋으니, 그걸 사용하라는 것이다. 근데, 어쩌겠나... 상황이 상황이니 만큼, 우리는 wireguard-go version으로 가는 수밖에 없다 ...

```

root@RAK7258:~/workspace# ./wireguard
WARNING WARNING WARNING WARNING WARNING WARNING
W G
W You are running this software on a Linux kernel, G
W which is probably unnecessary and misguided. This G
W is because the Linux kernel has built-in first G
W class support for WireGuard, and this support is G
W much more refined than this slower userspace G
W implementation. For more information on G
W installing the kernel module, please visit: G
W https://www.wireguard.com/install G
W G
WARNING WARNING WARNING WARNING WARNING WARNING WARNING
usage:
./wireguard [-f/--foreground] INTERFACE-NAME

```

[그림 9.12] wireguard 실행 파일 실행 모습

```
# ./wireguard wg0
```

→ wg0 interface를 생성해 보자.

```
# ifconfig -a
```

→ wg0 interface 확인

```
# ip address add dev wg0 10.1.1.30/24
```

→ Wireguard ip 주소로 10.1.1.30을 지정하자.

Wireguard-go에서도 wg tool은 여전히 사용된다.

```
# export LD_LIBRARY_PATH=/root/workspace:$LD_LIBRARY_PATH
```

→ wg를 위해 필요한 LD(Linking Dynamic) library(libmnl.so) path를 지정한다.

```
# ./wg genkey | tee ./privatekey | ./wg pubkey > ./publickey
```

→ private/public key를 생성한다.

```
root@RAK7258:~/workspace# cat privatekey  
aBgqqjOLRA9Z3DCbagMPqpxn8y6MB6kwL+kYj/2CWHQ=  
root@RAK7258:~/workspace#  
root@RAK7258:~/workspace#  
root@RAK7258:~/workspace# cat publickey  
uOpUfobitCH4CMNL0ga95+mW21DP7kyQvR+vv6UoI3A=  
root@RAK7258:~/workspace#  
root@RAK7258:~/workspace#
```

[그림 9.13] private/public key 내용

```
# ./wg set wg0 listen-port 51820 private-key ./privatekey peer  
gXINPhrlpbOSr4rMD9tF614mVFvF/GqyONJo4kmNB3I= allowed-ips 10.1.1.1/32  
endpoint 13.125.60.224:59760
```

- AWS(10.1.1.1)와의 wireguard 연결 설정을 해 보자.
- AWS 설정은 잘 아는 내용이므로 생략 :)

```
# ip link set up dev wg0
```

- 끝으로 wg0 interface를 up해 주자. Kernel version은 이곳에서 전사(die)했는데, go version은 kernel module과는 상관이 없는 구조이므로 당연히 안죽는다 :)

```

RX bytes:1518308 (1.4 MiB) TX bytes:837728 (818.0 KiB)

lo      Link encap:Local Loopback
        inet addr:127.0.0.1 Mask:255.0.0.0
        UP LOOPBACK RUNNING MTU:65536 Metric:1
        RX packets:64 errors:0 dropped:0 overruns:0 frame:0
        TX packets:64 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:7327 (7.1 KiB) TX bytes:7327 (7.1 KiB)

mon0    Link encap:Ethernet HWaddr 60:C5:A8:76:13:0C
        BROADCAST MULTICAST MTU:1500 Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

ra0     Link encap:Ethernet HWaddr 60:C5:A8:76:13:0C
        UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
        RX packets:99177 errors:0 dropped:0 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:26009548 (24.8 MiB) TX bytes:0 (0.0 B)
        Interrupt:6

wg0     Link encap:UNSPEC HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
        inet addr:10.1.1.30 P-t-P:10.1.1.30 Mask:255.255.255.0
        UP POINTOPOINT RUNNING NOARP MULTICAST MTU:1420 Metric:1
        RX packets:2223 errors:0 dropped:0 overruns:0 frame:0
        TX packets:2223 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:500
        RX bytes:186732 (182.3 KiB) TX bytes:186732 (182.3 KiB)

```

[그림 9.14] wg0에 ip 주소 10.1.1.30 설정 모습

ping 10.1.1.1

→ AWS SPN과 ping 연결을 시도해 본다. OK, 동작한다.

```

PING 10.1.1.1 (10.1.1.1): 56 data bytes
64 bytes from 10.1.1.1: seq=0 ttl=64 time=3.934 ms
64 bytes from 10.1.1.1: seq=1 ttl=64 time=3.986 ms
64 bytes from 10.1.1.1: seq=2 ttl=64 time=3.818 ms
64 bytes from 10.1.1.1: seq=3 ttl=64 time=3.707 ms
^C
--- 10.1.1.1 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 3.707/3.861/3.986 ms

```

```
root@RAK7258:~/workspace# export LD_LIBRARY_PATH=/root/workspace:$LD_LIBRARY_PATH
root@RAK7258:~/workspace# ./wg
interface: wg0
  public key: uOpUfobitCH4CMNL0ga95+mW21DP7kyQvR+vv6UoI3A=
  private key: (hidden)
  listening port: 51820

peer: gXLNPhrIpboSr4rMD9tF614mVFvF/GqyONJo4kmNB3I=
  endpoint: 13.125.60.224:59760
  allowed ips: 10.1.1.1/32
  latest handshake: 25 seconds ago
  transfer: 30.05 KiB received, 30.20 KiB sent
root@RAK7258:~/workspace#
```

[그림 9.15] wg show 명령 실행 모습

휴 ~ 다음주 부터는 SPNBox-300(RAK7258 LoRa Gateway)을 modeling 작업을 시작할 수 있겠다.

6) RAK7258 LoRa Gateway와 LoRa Server를 wireguard로 연결하기

Wireguard porting이 마무리 되었으니, LoRa Gateway ↔ LoRa Server 간을 wireguard로 연결해 보도록 하자.

<Testbed>

RAK7258	↔	AP04	↔	LoRa Server
172.30.1.26				172.30.1.46
10.1.1.30				10.1.1.19

<Target Board>

```
# ./wireguard wg0
# ip address add dev wg0 10.1.1.30/24
# export LD_LIBRARY_PATH=/root/workspace:$LD_LIBRARY_PATH
# ./wg genkey | tee ./privatekey | ./wg pubkey > ./publickey
# ./wg set wg0 listen-port 51820 private-key ./privatekey peer
LIUD6hsHyrJtHifSQfN9BB6xGtqPAK0/M+YjAP0lc2M= allowed-ips 10.1.1.19/32
```

endpoint 172.30.1.46:59760

```
# ip link set up dev wg0
```

```
# ping 10.1.1.19
```

- OK

<LoRa Server - Ubuntu 18.04 - 설정>

```
$ sudo ip link add wg0 type wireguard
```

```
$ sudo ip address add dev wg0 10.1.1.19/24
```

```
$ sudo wg genkey | tee ./privatekey | wg pubkey > ./publickey
```

```
$ sudo wg set wg0 listen-port 59760 private-key ./privatekey peer
uOpUfobitCH4CMNLOga95+mW21DP7kyQvR+vv6UoI3A= allowed-ips 10.1.1.30/32
endpoint 172.30.1.26:51820
```

```
$ sudo ip link set up dev wg0
```

```
$ ping 10.1.1.30
```

- OK

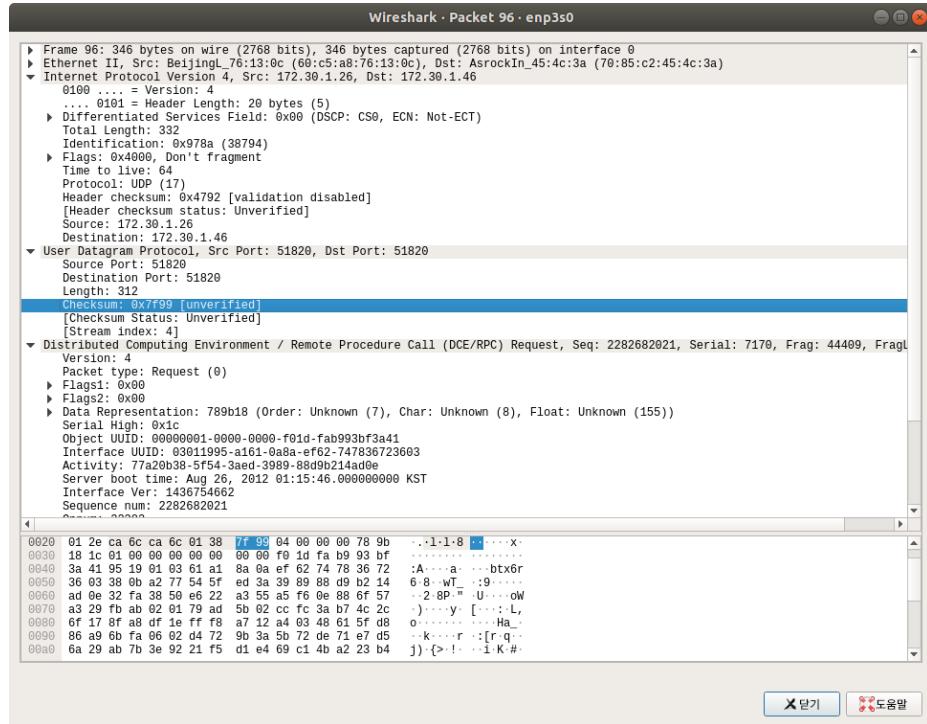
어라, LoRa Gateway의 Packet Forwarder ⇒ Gateway Configuration ⇒ Server

Address를 10.1.1.19(wireguard peer IP)로 지정하기만 하면 간단히 끝날 줄 알았는데, 동작을 안한다. 일단 LoRaServer log를 확인해 보니, 패킷이 아예 안 올라온다. 또한 LoRaServer 쪽에서 tcpdump를 해 보니, 분명히 LoRa Gateway로 부터 packet이 들어 오나, 응답을 못하고 있다. 왜 일까 ?

```
9월 11 15:03:38 mars loraserver[2099]: time="2019-09-11T15:03:38+09:00" level=info msg="finished unary call with code OK" grpc.code=OK grpc.method=GetGateway grpc.service=ns.NetworkServerService grpc.start_time="2019-09-11T15:03:38+09:00" grpc.time_ms=0.752 peer.address="127.0.0.1:37616" span.kind=server system=grpc
9월 11 15:03:38 mars loraserver[2099]: time="2019-09-11T15:03:38+09:00" level=info msg="finished unary call with code OK" grpc.code=OK grpc.method=GetGatewayStats grpc.service=ns.NetworkServerService grpc.start_time="2019-09-11T15:03:38+09:00" grpc.time_ms=0.229 peer.address="127.0.0.1:37616" span.kind=server system=grpc
```

[그림 9.16] LoRaServer log 출력 모습

Peer 쪽에서 wireshark로 wireguard packet을 capture해 보자. 아래 그림을 보면, Checksum을 verify 못한다는 내용이 보이는데, spnbox 간 통신 시(정상적인 경우)에도 동일한 결과를 뿌려주는 것으로 보아서, 이게 원인은 아닌 것 같다. 그럼 도대체 뭐가 문제란 말인가?



[그림 9.17] LoRaServer가 설치된 PC에서 wireshark을 실행한 모습

다음 내용은 Peer 쪽에서 본 wg0 interface의 상태인데, **dropped packet**이 보인다. Wireguard-go에서 encryption해서 전달된 패킷에 문제가 있는 건 아닐까? ping이나 ssh 등을 해 보면 문제가 없는데.... 왜 얘(lora packet) 만 그럴까? UDP만 안되는 것인가... 정말 이상하다.

```
wg0: flags=209<UP,POINTOPOINT,RUNNING,NOARP> mtu 1300
inet 10.1.1.100 netmask 255.255.255.0 destination 10.1.1.100
  unspec 00-00-00-00-00-00-00-00-00-00-00-00 txqueuelen 1000
  (UNSPEC)
RX packets 6016 bytes 943504 (943.5 KB)
```

```
RX errors 643 dropped 788 overruns 0 frame 643
TX packets 6504 bytes 791464 (791.4 KB)
TX errors 61 dropped 12 overruns 0 carrier 0 collisions 0
```

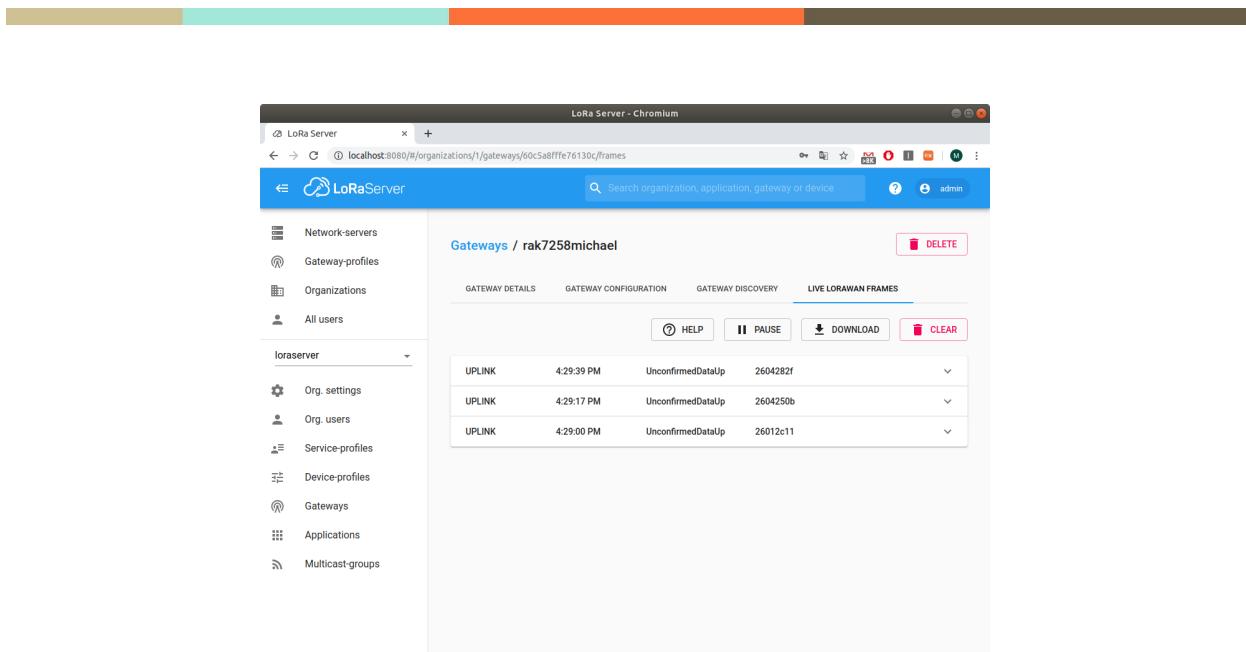
이쯤되면 Wireguard-go code를 의심해 보지 않을 수 없다. 정확히 어디간 문제인지는 시간을 두고 좀 더 살펴 보기로 하자. Wireguard-go가 문제인지를 파악하기 위해 다른 방법을 시도해 볼 필요가 있다.

아래 내용은 Mesh VPN 테스트 과정에서 원인이 파악되어 다시 정리한 것이다. wireguard-go 환경에서 lora packet이 교환되지 않은 이유는 아래 NAT rule로 해결될 수 있다. LoRa packet은 source ip가 192.168.230.1로 고정되어 있는 듯 보인다. 근데, wg0 interface를 통해 복호화가 되려면 tun0 interface를 타야 하는데, 이를 위해서는 source ip가 wg0 interface의 IP여야만 한다.

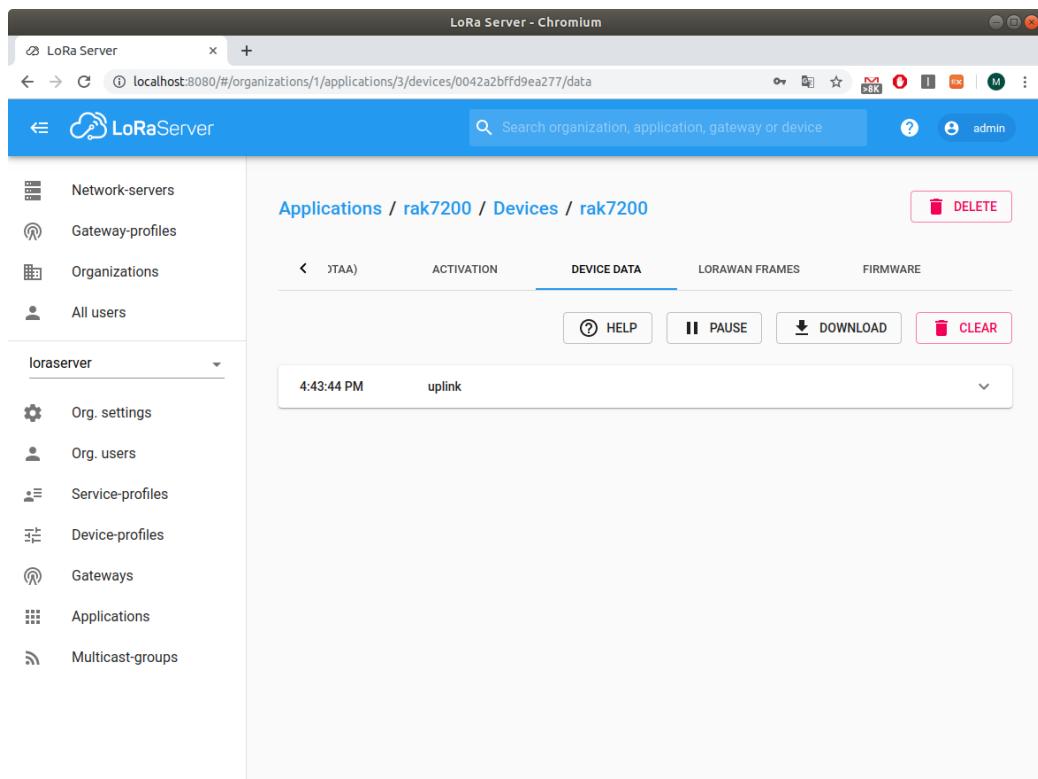
<target board>

```
root@RAK7258:~/workspace/tinc_output# iptables -t nat -A POSTROUTING -o wg0 -j MASQUERADE
root@RAK7258:~/workspace/tinc_output# iptables -I FORWARD -i wg0 -j ACCEPT
root@RAK7258:~/workspace/tinc_output# iptables -I FORWARD -o wg0 -j ACCEPT
```

Good... 위의 내용을 적용해 보니 역시 packet이 들어온다.



[그림 9.18] LoRa Server gateway 화면 - 정상



[그림 9.19] LoRa Server application/device 화면 - 정상

어라, 근데 loraserver log도 그렇고, wg0 interface 내용도 그렇고 error 패킷이 있는 것 같다.

```
9월 16 16:35:09 mars loraserver[2045]: time="2019-09-16T16:35:09+09:00" level=info msg="metrics saved" aggregation="[MINUTE HOUR DAY MONTH]" name="gw:60c5a8ffffe76130c"
9월 16 16:35:10 mars loraserver[2045]: time="2019-09-16T16:35:10+09:00" level=info msg="gateway/mqtt: uplink frame received"
9월 16 16:35:10 mars loraserver[2045]: time="2019-09-16T16:35:10+09:00" level=error msg="processing uplink frame error" data_base64="QBEsASaABwACZz1VglTaCDerx1/ybblobKwKSa04Q/35HW8TkM0aUA=" error="get device-session error: device-session does not exist or invalid fcnt or mic"
```

[그림 9.20] loraserver log 확인 - uplink frame error

```
wg0: flags=209<UP,POINTOPOINT,RUNNING,NOARP> mtu 1420
      inet 10.1.1.19 netmask 255.255.255.0 destination 10.1.1.19
        unspec 00-00-00-00-00-00-00-00-00-00-00-00 txqueuelen 1000 (UNSPEC)
          RX packets 795 bytes 104268 (104.2 KB)
          RX errors 57 dropped 0 overruns 0 frame 57
          TX packets 799 bytes 97636 (97.6 KB)
          TX errors 13 dropped 0 overruns 0 carrier 0 collisions 0
```

[그림 9.21] wg0 interface TX/RX packet error

뭔가 좀 불안하다... 암튼, MeshVPN을 적극 활용해 보도록 하자.

8) MeshVPN(Tinc) Porting하기

Wireguard.go 마저 문제(원인 파악이 되기 전 상태)라면, 이번에는 **Tinc**를 고려해 보도록 하자. n2n(p2p spn)도 검토 대상일 수 있으나, n2n은 상대적으로 보안성이 떨어진다. 따라서, tinc를 LoRa Gateway에 porting해 보도록 하겠다. Tinc가 이렇게 다시 검토 대상이 될 줄은 몰랐다 :) 다시 검토하면서 느낀 거지만, tinc 꽤 괜찮은 tool이다. 앞으로 **Kernel porting**에 문제가 있는 경우는 tinc로 가는 것도 좋을 듯 싶다.



[그림 9.22] tinc logo

<Testbed>

RAK7258(172.30.1.19) ⇔ AP04 ⇔ LoRa Server(172.30.1.14)

vpn IP: 192.168.2.1/24	192.168.2.1/24
------------------------	----------------

먼저 LoRa Server(x86_64 PC)에서 tinc 설정을 해 보도록 하자.

<my PC/Ubuntu 18.04 - LoRaServer>

LoRaServer가 설치되어 있는 PC에서 tinc를 구동시켜 보자. 아래 site에서 tinc1.1pre17 version을 download 하도록 한다.

<https://www.tinc-vpn.org/download/>

참고: 이 버전은 stable 버전은 아니지만, ECC(Ed25519) 공개키 암호 알고리즘을 지원하므로 사용하기로 한다.

```
$ cd tinc-1.1pre17/
$ ./configure
$ make
$ sudo make install
chyi@earth:~/workspace/download$ cd /usr/local/sbin/
chyi@earth:/usr/local/sbin$ ls -la
합계 2348
drwxr-xr-x 2 root root 4096 9월 13 16:12 .
```

```
drwxr-xr-x 11 root root 4096 5월 11 17:49 ..
-rwxr-xr-x 1 root root 990440 9월 13 16:12 tinc
-rwxr-xr-x 1 root root 1403536 9월 13 16:12 tincd
```

\$ sudo tinc -n mesh init earth

- Tinc configuration file 생성 및 key(public/private) 생성
- tinc -n NETNAME init NAME

```
chyi@earth:~/workspace/LoRa/TINC/x86_tinc/tinc-1.1pre17$ sudo tinc -n mesh init earth
Generating 2048 bits keys:
.....+++++ p
.....+++++ q
Done.
Generating Ed25519 keypair:
Done.
```

[그림 9.23] tinc 초기화 명령 실행

```
chyi@earth:/usr/local/etc/tinc/mesh$ ls -la
```

합계 28

```
drwxr-xr-x 3 root root 4096 9월 13 16:13 .
drwxr-xr-x 4 root root 4096 9월 13 16:13 ..
-rw----- 1 root root 200 9월 13 16:13 ed25519_key.priv      # ed25519 private key
drwxr-xr-x 2 root root 4096 9월 13 16:13 hosts            # rsa & ed25519 public key &
ip information
-rw----- 1 root root 1675 9월 13 16:13 rsa_key.priv       # rsa private key
-rwxr-xr-x 1 root root 140 9월 13 16:13 tinc-up          # tinc interface up script
-rw-r--r-- 1 root root 13 9월 13 16:13 tinc.conf        # tinc configuration
```

\$ sudo tinc -n mesh add Subnet 192.168.2.1/32

- vpn ip 주소 설정(네트워크 설정), allowed-ips 개념임.
- 참고: Subnet을 여러 개 등록해 주어야 한다면, 위의 명령을 여러 차례 수행해 주면 된다.

\$ sudo tinc -n mesh add Address 172.30.1.14

- vpn public ip address(my endpoint) 설정

→ Tinc는 listen port가 655로 지정되어 있으며, Port 명령을 사용하여 변경할 수 있다.

\$ sudo tinc -n mesh edit tinc-up

→ Tinc interface up script 설정 변경

```
#!/bin/sh

#echo 'Unconfigured tinc-up script, please edit '$0'!'

ifconfig $INTERFACE 192.168.2.1 netmask 255.255.255.0 mtu 1400
~
```

[그림 9.24] tinc-up script 변경 모습

\$ sudo vi /usr/local/etc/tinc/mesh/tinc.conf

- 몇가지 필요한 설정을 추가한다.
- 나중에 언제라도 필요한 설정을 추가/변경할 수 있다.
- **ConnectTo = rak7258** 을 추가해 줄 수 있는데, 이 경우, client 처럼 동작한다. Server로 동작하게 하고 싶으면, 넣어 줄 필요가 없다.
- 따라서 client/server를 구분 안하게 하려면, **AutoConnect = yes**를 넣어 주는게 좋다.

```
Name = earth
Interface = tun0
~
```

[그림 9.25] tinc.conf 수정 모습

이 부분에서 peer와 자신의 public key를 교환해야 한다. 즉, my host file(earth)을 rak7258 쪽에 복사하고, rak7258 host file을 내 쪽에 복사해야 한다. 이 host file에는 peer의 RSA/ECC public key 및 peer의 network 정보(vpn ip & endpoint 정보)가 포함되어 있다.

/usr/local/etc/tinc/mesh/hosts/earth ⇔ /usr/local/etc/tinc/mesh/hosts/rak7258

```
root@RAK7258:/usr/local/etc/tinc/mesh/hosts# ls -la
drwxr-xr-x  2 root  root      0 Sep 16 04:11 .
drwxr-xr-x  3 root  root      0 Sep 16 02:55 ..
-rw-r--r--  1 root  root   513 Sep 16 04:11 earth
-rw-r--r--  1 root  root   535 Sep 16 03:54 rak7258
```

참고: tinc는 ssh를 사용하여 hosts 파일을 교환하도록 하고 있다.

```
$ sudo mkdir -p /usr/local/var/run
```

이 디렉토리가 없으면, 아래 명령 tinc daemon 구동시 아래 메시지를 출력하게 된다.

```
Cannot write control socket cookie file /usr/local/var/run/tinc.mesh.pid: No such file or
directory
Terminating
Error starting tincd
```

```
$ sudo tinc -n mesh start
```

→ Tinc daemon(tincd)를 시작한다.

```
chyi@earth:~/workspace/LoRa/TINC/x86_tinc/tinc-1.1pre17$ sudo tinc -n mesh start
tincd 1.1pre17 (Sep 13 2019 16:12:39) starting, debug level 0
/dev/net/tun is a Linux tun/tap device (tun mode)
Ready
```

[그림 9.26] tinc daemon 구동 모습

```
chyi@earth:~/workspace/LoRa/TINC/x86_tinc/tinc-1.1pre17$ ps aux|grep tincd
root  15133  0.0  0.0 22672 2872 ?    Ss 16:24 0:00 tincd -n mesh
chyi  15314  0.0  0.0 22820 1104 pts/2  S+ 16:25 0:00 grep --color=auto tincd
```

Tincd가 정상적으로 구동되면, 아래와 같이 tun0 interface가 보일 것이다.

```

ether 98:83:89:9a:9b:c6 txqueuelen 1000 (Ethernet)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
        inet6 ::1 prefixlen 128 scopeid 0x10<host>
            loop txqueuelen 1000 (Local Loopback)
            RX packets 68949 bytes 11890437 (11.8 MB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 68949 bytes 11890437 (11.8 MB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

tun0: flags=4305<UP,POINTOPOINT,RUNNING,NOARP,MULTICAST> mtu 1500
    inet 192.168.2.1 netmask 255.255.255.0 destination 192.168.2.1
        inet6 fe80::d033:2d3a:21c7:3ae5 prefixlen 64 scopeid 0x20<link>
            unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00 txqueuelen 500 (UNSPEC)
            RX packets 0 bytes 0 (0.0 B)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 14 bytes 776 (776.0 B)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wlp1s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.30.1.14 netmask 255.255.255.0 broadcast 172.30.1.255
        inet6 fe80::e921:c0c7:3b45:2231 prefixlen 64 scopeid 0x20<link>
            ether f8:a2:d6:80:91:39 txqueuelen 1000 (Ethernet)
            RX packets 172243 bytes 222839746 (222.8 MB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 53441 bytes 23716645 (23.7 MB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

```

[그림 9.27] tun0 interface 생성 모습

```

root@RAK7258:~/workspace/tinc_output# netstat -nr
Kernel IP routing table
Destination     Gateway         Genmask        Flags   MSS Window irtt Iface
0.0.0.0         172.30.1.254   0.0.0.0       UG        0 0          0 eth0.2
169.254.0.0     0.0.0.0        255.255.0.0   U         0 0          0 eth0.2
172.30.1.0      0.0.0.0        255.255.255.0 U         0 0          0 eth0.2
172.30.1.254   0.0.0.0        255.255.255.255 UH       0 0          0 eth0.2
192.168.2.0     0.0.0.0        255.255.255.0   U        0 0          0 tun0
192.168.230.0   0.0.0.0        255.255.255.0   U        0 0          0 br-lan
root@RAK7258:~/workspace/tinc_output#

```

[그림 9.28] tun0를 포함한 routing table

이제 모든 설정이 정상적으로 진행되었는지 확인하기 위해 ping을 시도해 보도록 하자.

\$ ping 192.168.2.2

물론 이게 성공하려면, 아래 RAK7258에서의 tinc 설정이 완료된 상태라야만 한다.

```
chyi@mars:~/workspace/test/09162019$ ping 192.168.2.2
PING 192.168.2.2 (192.168.2.2) 56(84) bytes of data.
64 bytes from 192.168.2.2: icmp_seq=1 ttl=64 time=1.39 ms
64 bytes from 192.168.2.2: icmp_seq=2 ttl=64 time=1.28 ms
64 bytes from 192.168.2.2: icmp_seq=3 ttl=64 time=1.29 ms
^C
--- 192.168.2.2 ping statistics ---
4 packets transmitted, 3 received, 25% packet loss, time 3004ms
rtt min/avg/max/mdev = 1.289/1.325/1.394/0.064 ms
```

[그림 9.29] tinc peer로 ping 모습

LoRa Server 쪽이 준비되었으니, 다음으로 RAK7258에서의 tinc 설정을 진행해 보도록 하겠다. 이를 위해서는 먼저 tinc를 RAK7258(mips)용으로 cross-compile을 해 주는 과정이 선결되어야 한다.

<cross-compile 환경 설정하기>

<toolchain path>

```
$ export
PATH=/home/chyi/workspace/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628/
wiswrt/15.05-rak-rc2/staging_dir/toolchain-mipsel_24kec+dsp_gcc-4.8-linaro_uClibc-0.
9.33.2/bin:$PATH
```

<staging dir>

```
$ export
STAGING_DIR=/home/chyi/workspace/RAKWireless/RAK831-LoRaGateway-OpenWRT-
MT7628/wiswrt/15.05-rak-rc2/staging_dir
```

<arch & compiler>

```
$ export ARCH=mips
$ export CROSS_COMPILE=mipsel-openwrt-linux-
```

<lzo library cross-compilation>

- download lzo-2.10.tar.gz from
<http://www.oberhumer.com/opensource/lzo/#download>

```
$ tar xvzf lzo-2.10.tar.gz  
$ cd lzo-2.10/  
$ mkdir output  
$ ./configure --host=mipsel-openwrt-linux --prefix=$(pwd)/output --enable-shared  
$ make  
$ make install
```

<openssl cross-compilation>

```
$ git clone https://github.com/openssl/openssl  
$ cd openssl  
$ mkdir -p output  
  
$ ./Configure linux-generic32 shared no-sse2 no-zlib no-async --prefix=$(pwd)/output  
--openssldir=$(pwd)/output/openssl --cross-compile-prefix=mipsel-openwrt-linux-  
$ make clean  
$ make depend  
$ make  
$ make install
```

<libncurses cross-compilation>

→ <https://invisible-island.net/ncurses/>

```
$ tar xvzf ncurses.tar.gz  
$ cd ncurses-6.1/  
$ mkdir output  
$ ./configure --host=mipsel-openwrt-linux --prefix=$(pwd)/output --enable-shared  
$ make  
$ make install
```

→ install 끝 부분에서 에러 발생하나, 무시해도 됨.

<libreadline cross-compilation>

→ <https://tiswww.cwru.edu/php/chet/readline/rltop.html>

```
$ tar xvzf readline-8.0.tar.gz
$ cd readline-8.0/
$ mkdir output
$ ./configure --host=mipsel-openwrt-linux --prefix=$(pwd)/output --enable-shared
$ make
$ make install
```

<tinc cross-compilation>

→ download tinc-1.1pre17.tar.gz

```
$ sudo apt-get install texinfo
```

- 이거를 설치 안하면, 아래 make 시 에러가 발생한다.
- sudo apt-get install libssl-dev zlib1g-dev liblzo2-dev build-essential automake autoconf gettext texinfo

```
$ tar xvzf tinc-1.1pre17.tar.gz
```

```
$ cd tinc-1.1pre17
```

```
$ mkdir output
```

```
$ export
```

```
CC=/home/chyi/workspace/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628/wi
swrt/15.05-rak-rc2/staging_dir/toolchain-mipsel_24kec+dsp_gcc-4.8-linaro_uClibc-0.9.3
3.2/bin/mipsel-openwrt-linux-gcc
```

```
$ ./configure --host=mipsel-openwrt-linux --prefix=$(pwd)/output
--with-openssl-lib=$(pwd)/../openssl/output/lib
--with-openssl-include=$(pwd)/../openssl/output/include
--with-lzo-lib=$(pwd)/../lzo-2.10/output/lib
--with-lzo-include=$(pwd)/../lzo-2.10/output/include
--with-curses-lib=$(pwd)/../ncurses-6.1/output/lib
--with-curses-include=$(pwd)/../ncurses-6.1/output/include
```

```
--with-readline-lib=$(pwd)/../readline-8.0/output/lib
--with-readline-include=$(pwd)/../readline-8.0/output/include

⇒

$ ./configure --host=mipsel-openwrt-linux --prefix=$(pwd)/output
--with-openssl-lib=$(pwd)/../openssl/output/lib
--with-openssl-include=$(pwd)/../openssl/output/include
--with-lzo-lib=$(pwd)/../lzo-2.10/output/lib
--with-lzo-include=$(pwd)/../lzo-2.10/output/include --disable-zlib --disable-curses
--disable-readline

⇒

$ ./configure --host=mipsel-openwrt-linux
--with-openssl-lib=$(pwd)/../openssl/output/lib
--with-openssl-include=$(pwd)/../openssl/output/include
--with-lzo-lib=$(pwd)/../lzo-2.10/output/lib
--with-lzo-include=$(pwd)/../lzo-2.10/output/include --disable-zlib --disable-curses
--disable-readline
```

주의 1: 앞서 `ncurses`, `readline`을 `cross-compile`해 두었으나, 이상하게도 적용할 경우, `header`를 못 찾겠다고 한다. 따라서 일단은 위에서와 같이 2개의 `library`를 `disable` 시킨 상태에서 `build`해 보도록 하자. 실제 동작에 아무런 영향이 없다.

주의 2: `tinc configure` 수행 시, `--prefix=$(pwd)/output`를 넣어주게 되면, 실행시 문제가 되니 제거해야 한다.

```
$ make
$ make install
→ --prefix option을 뺀 경우에는 make install을 하지 말도록 하자.
```

이제 부터는 RAK7258 target board에서 `tinc`을 구동시켜 보도록 하자. 아래 내용은 target board의 `/root/workspace/tinc_output` 아래에 `tinc` 관련 파일을 옮겨 놓았다고 가정하고 작성한 것이다.

주의: target board에 올릴 때에는 **library & binary** 파일에 대해 `strip`을 적용해 주어야 한다(target board flash memory의 공간이 부족하므로).

```
/home/chyi/workspace/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628/wiswrt/15
.05-rak-rc2/staging_dir/toolchain-mipsel_24kec+dsp_gcc-4.8-linaro_uClibc-0.9.33.2/bin/mi
psel-openwrt-linux-strip libcrypto.so.3 liblzo2.so.2.0.0 libreadline.so.8.0 libssl.so.3
tinc tincd
```

<target board - RAK7258>

```
# export LD_LIBRARY_PATH=/root/workspace/tinc_output:$LD_LIBRARY_PATH
# mkdir -p /usr/local/etc/
# cd /root/workspace/tinc_output
# ./tinc -n mesh init rak7258
```

- Tinc configuration file 생성 및 key(public/private) 생성
- 명령 형식: tinc -n NETNAME init NAME

```
root@RAK7258:~/workspace/tinc_output# ./tinc -n mesh init rak7258
Generating 2048 bits keys:
.....+++++ p
.....+++++ q
Done.
Generating Ed25519 keypair:
Done.
Warning: could not bind to port 655. Please change tinc's Port manually. ↵ 요거 일단 무시
```

```
root@RAK7258:~/workspace/tinc_output# cd /usr/local/etc
root@RAK7258:/usr/local/etc# ls -la
drwxr-xr-x 3 root root 0 Sep 16 02:55 .
drwxr-xr-x 3 root root 0 Sep 16 02:55 ..
drwxr-xr-x 3 root root 0 Sep 16 02:55 tinc
root@RAK7258:/usr/local/etc# cd tinc/
root@RAK7258:/usr/local/etc/tinc# ls -la
drwxr-xr-x 3 root root 0 Sep 16 02:55 .
drwxr-xr-x 3 root root 0 Sep 16 02:55 ..
drwxr-xr-x 3 root root 0 Sep 16 02:55 mesh
root@RAK7258:/usr/local/etc/tinc# cd mesh/
root@RAK7258:/usr/local/etc/tinc/mesh# ls -la
drwxr-xr-x 3 root root 0 Sep 16 02:55 .
drwxr-xr-x 3 root root 0 Sep 16 02:55 ..
-rw----- 1 root root 200 Sep 16 02:55 ed25519_key.priv
drwxr-xr-x 2 root root 0 Sep 16 03:52 hosts
-rw----- 1 root root 1679 Sep 16 02:55 rsa_key.priv
-rw-r--r-- 1 root root 140 Sep 16 02:55 tinc-up
-rw-r--r-- 1 root root 15 Sep 16 02:55 tinc.conf
root@RAK7258:/usr/local/etc/tinc/mesh#
```

[그림 9.30] tinc 초기화 후 생성된 파일

```
# ./tinc -n mesh add Subnet 192.168.2.2/32
```

→ vpn ip 주소 설정(네트워크 설정)

```
# ./tinc -n mesh add Address 172.30.1.26
```

→ vpn public ip address(my endpoint) 설정

→ 참고: endpoint port는 655로 지정(default)되어 있으나, 변경하고자 한다면 **tinc -n mesh add Port <portnum>** 명령을 사용하면 된다.

```
# ./tinc -n mesh edit tinc-up
```

→ Tinc interface up script 설정 변경

```
#!/bin/sh

#echo 'Unconfigured tinc-up script, please edit '$0'!

ifconfig $INTERFACE <your vpn IP address> netmask <netmask of whole VPN>
ifconfig $INTERFACE 192.168.2.2 netmask 255.255.255.0 mtu 1400
```

```
# vi /usr/local/etc/tinc/mesh/tinc.conf
```

- 몇가지 필요한 설정을 추가한다.
- 나중에 이 부분에 필요한 설정을 추가할 수 있다.
- **ConnectTo = earth** 을 추가해 줄 수 있는데, 이 경우, client 처럼 동작한다. Server로 동작하게 하고 싶으면, 넣어 줄 필요가 없다.
- 보다 편리한 방법으로 **AutoConnect = yes**를 사용하는 것이 좋다.

```
Name = rak7258
Interface = tun0
AutoConnect = yes
~
```

[그림 9.31] tinc.conf 수정 모습

이 부분에서 peer와 자신의 public key를 교환해야 한다. 즉, my host file(rak7258)을 earth(LoRaServer) 쪽에 복사하고, earth host file을 내 쪽에 복사해야 한다. 이 host file에는 peer의 RSA/ECC public key 및 peer의 network 정보(vpn ip & endpoint 정보)가 포함되어 있다.

/usr/local/etc/tinc/mesh/hosts/rak7258 ⇔ /usr/local/etc/tinc/mesh/hosts/earth

```
# mkdir -p /usr/local/var/run
```

→ 이 디렉토리가 없으면 다음 명령 실행시 에러가 발생한다.

```
# ./tinc -n mesh start
```

→ Tinc daemon(tincd)를 시작한다.

```
root@RAK7258:~/workspace/tinc_output# ./tinc -n mesh start
tincd 1.1pre17 (Sep 16 2019 11:53:37) starting, debug level 0
/dev/net/tun is a Linux tun/tap device (tun mode)
Ready
```

```
11693 root      22752 S      lora_pkt_fwd -g /var/etc/global_conf.json
11722 root      5536 S      ./tincd -n mesh
11743 root      1500 R      ps
root@RAK7258:~/workspace/tinc_output# █
```

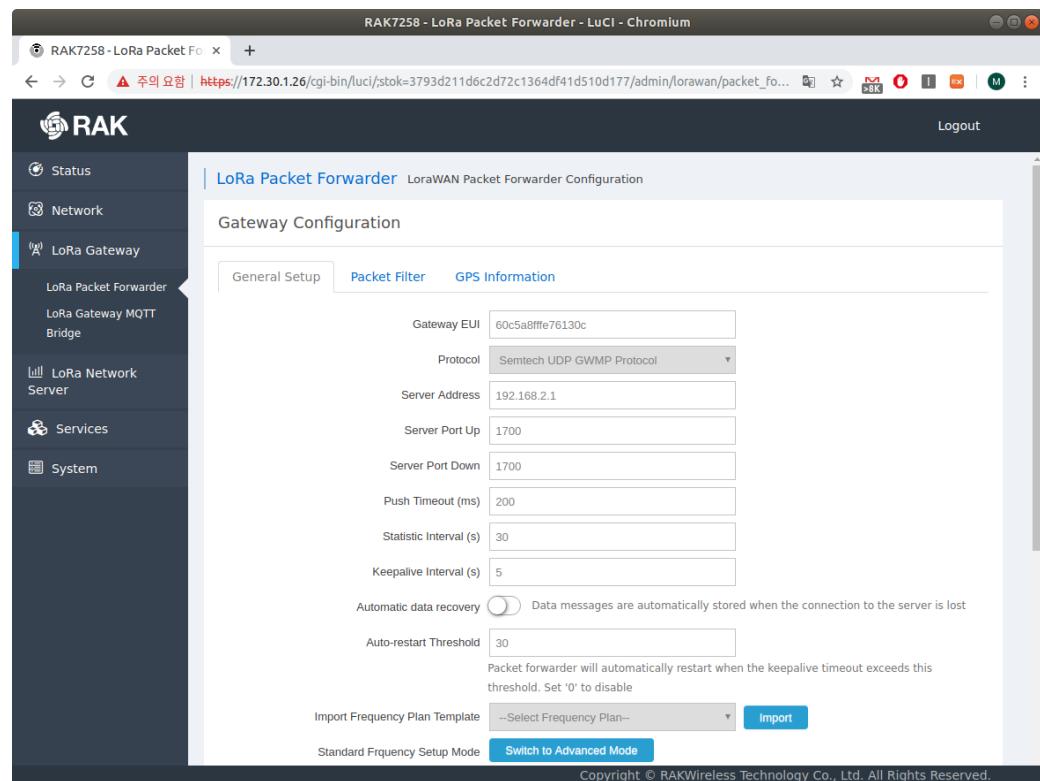
[그림 9.32] tinc daemon 구동 모습

```
# ping 192.168.2.1
```

```
root@RAK7258:/usr/local/etc/tinc/mesh/hosts# ping 192.168.2.1
PING 192.168.2.1 (192.168.2.1): 56 data bytes
64 bytes from 192.168.2.1: seq=0 ttl=64 time=1.834 ms
64 bytes from 192.168.2.1: seq=1 ttl=64 time=1.832 ms
64 bytes from 192.168.2.1: seq=2 ttl=64 time=1.824 ms
64 bytes from 192.168.2.1: seq=3 ttl=64 time=1.647 ms
^C
--- 192.168.2.1 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 1.647/1.784/1.834 ms
```

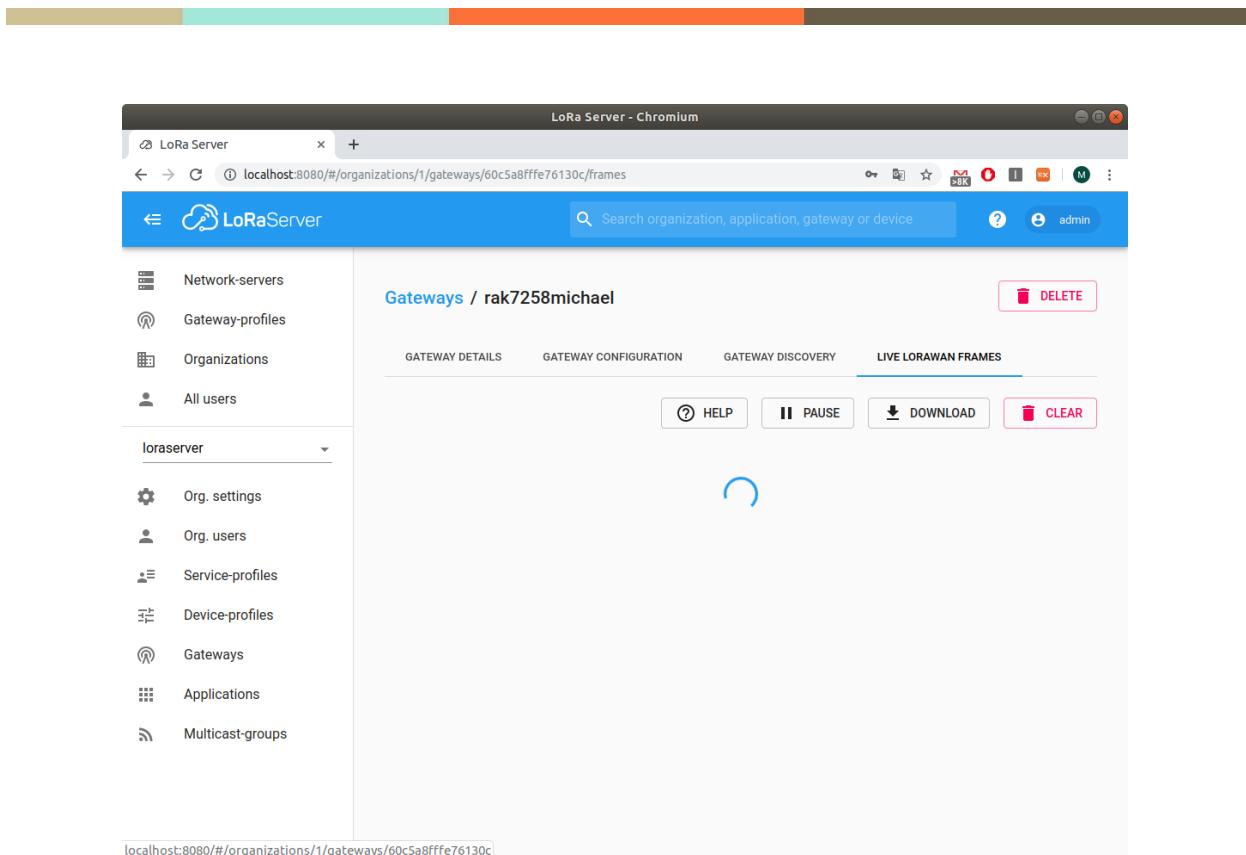
[그림 9.33] tinc peer로 ping 하는 모습

여기까지 정상적이라면, 다음으로는 lora packet이 tinc vpn 위에서 LoRa Server로 정상적으로 전달되는지를 확인해 볼 차례이다. 아래와 같이 LoRa Gateway의 packet forwarder server 주소를 192.168.2.1(peer VPN ip)로 설정하자.



[그림 9.34] LoRa Gateway packet forwarder 설정 변경

이후, LoRa Server에서 패킷이 수신되는지를 보자.



[그림 9.35] LoRa Server gateway 패킷 수신 화면

어랍쇼~ tun0 interface 상태(TX/RX error)도 정상인데, 아무것도 안 들어온다.

```
tun0: flags=4305<UP,POINTOPOINT,RUNNING,NOARP,MULTICAST> mtu 1460
      inet 192.168.2.1 netmask 255.255.255.0 destination 192.168.2.1
      inet6 fe80::8af:2e15:2110:6cf7 prefixlen 64 scopeid 0x20<link>
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00 txqueuelen 500 (UNSPEC)
          RX packets 1261 bytes 117442 (117.4 KB)
          RX errors 0 dropped 0 overruns 0 frame 0
          TX packets 1074 bytes 89892 (89.8 KB)
          TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
chyi@mars:~/workspace/test/09162019$
```

[그림 9.36] tun0 interface 상태 확인

LoRa Server 쪽에서 tcpdump를 해 보면, 분명히 tinc(udp 655) packet이 오고 간다.

```
chyi@mars:/usr/local/etc/tinc/mesh/hosts$ sudo tcpdump udp

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp3s0, link-type EN10MB (Ethernet), capture size 262144 bytes
14:14:45.982910 IP 172.30.1.26.tinc > mars.tinc: UDP, length 73
14:14:45.983113 IP 172.30.1.26.tinc > mars.tinc: UDP, length 51
14:14:45.983167 IP mars.tinc > 172.30.1.26.tinc: UDP, length 51
14:14:46.438122 IP mars.tinc > 172.30.1.26.tinc: UDP, length 51
14:14:46.438737 IP 172.30.1.26.tinc > mars.tinc: UDP, length 51
14:14:46.476627 IP 172.30.1.26.tinc > mars.tinc: UDP, length 51
14:14:46.476815 IP mars.tinc > 172.30.1.26.tinc: UDP, length 51
14:14:46.839129 IP 172.30.1.211.2111 > 224.0.0.68.1968: UDP, length 36
14:14:47.444164 IP mars.tinc > 172.30.1.26.tinc: UDP, length 51
14:14:47.444203 IP mars.tinc > 172.30.1.26.tinc: UDP, length 51
14:14:47.444922 IP 172.30.1.26.tinc > mars.tinc: UDP, length 51
14:14:47.573807 IP 172.30.1.26.tinc > mars.tinc: UDP, length 51
14:14:47.573983 IP mars.tinc > 172.30.1.26.tinc: UDP, length 51
14:14:48.461553 IP mars.tinc > 172.30.1.26.tinc: UDP, length 51
14:14:48.462203 IP 172.30.1.26.tinc > mars.tinc: UDP, length 51
14:14:48.589589 IP 172.30.1.26.tinc > mars.tinc: UDP, length 51
14:14:48.589760 IP mars.tinc > 172.30.1.26.tinc: UDP, length 51
^C
17 packets captured
17 packets received by filter
0 packets dropped by kernel
```

[그림 9.37] tcpdump udp 내용 확인(정상)

tun0 interface를 통해 ping packet을 확인해 보면 정상이다.

```
chyi@mars:/usr/local/etc/tinc/mesh/hosts$ sudo tcpdump -i tun0 icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on tun0, link-type RAW (Raw IP), capture size 262144 bytes

13:51:22.202179 IP 192.168.2.2 > mars: ICMP echo request, id 22582, seq 0, length 64
13:51:22.202195 IP mars > 192.168.2.2: ICMP echo reply, id 22582, seq 0, length 64
13:51:23.204741 IP 192.168.2.2 > mars: ICMP echo request, id 22582, seq 1, length 64
13:51:23.204755 IP mars > 192.168.2.2: ICMP echo reply, id 22582, seq 1, length 64
13:51:24.204891 IP 192.168.2.2 > mars: ICMP echo request, id 22582, seq 2, length 64
13:51:24.204905 IP mars > 192.168.2.2: ICMP echo reply, id 22582, seq 2, length 64
13:51:25.205102 IP 192.168.2.2 > mars: ICMP echo request, id 22582, seq 3, length 64
13:51:25.205115 IP mars > 192.168.2.2: ICMP echo reply, id 22582, seq 3, length 64
13:51:26.205346 IP 192.168.2.2 > mars: ICMP echo request, id 22582, seq 4, length 64
13:51:26.205360 IP mars > 192.168.2.2: ICMP echo reply, id 22582, seq 4, length 64
```

[그림 9.38] tcpdump icmp 내용 확인(정상)

tun0 interface를 통해 ssh packet을 확인해 본 결과 역시 정상이다.

```
chyi@mars:/usr/local/etc/tinc/mesh/hosts$ sudo tcpdump -i tun0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on tun0, link-type RAW (Raw IP), capture size 262144 bytes

14:19:45.131229 IP 192.168.230.1.48589 > mars.1700: [|radius]
14:19:49.477352 IP 192.168.230.1.37602 > mars.1700: RADIUS, Access-Accept (2), id: 0x31 length: 210
14:19:50.331244 IP 192.168.230.1.48589 > mars.1700: [|radius]
14:19:52.158014 IP 192.168.230.1.37602 > mars.1700: RADIUS, Access-Accept (2), id: 0x05 length: 223
14:19:53.144940 IP 192.168.230.1.37602 > mars.1700: RADIUS, Access-Accept (2), id: 0x58 length: 133
14:19:53.693078 IP 192.168.230.1.37602 > mars.1700: RADIUS, Access-Accept (2), id: 0x5e length: 240
14:19:55.531211 IP 192.168.230.1.48589 > mars.1700: [|radius]
14:19:56.431989 IP 192.168.2.2.49729 > mars.ssh: Flags [S], seq 1592964145, win 27200, options [mss 1360,s
ackOK,TS val 1686175 ecr 0,nop,wscale 4], length 0
14:19:56.432036 IP mars.ssh > 192.168.2.2.49729: Flags [S.], seq 981157746, ack 1592964146, win 64704, opt
ions [mss 1360,sackOK,TS val 822762232 ecr 1686175,nop,wscale 7], length 0
14:19:56.433213 IP 192.168.2.2.49729 > mars.ssh: Flags [.], ack 1, win 1700, options [nop,nop,TS val 16861
75 ecr 822762232], length 0
14:19:56.438174 IP mars.ssh > 192.168.2.2.49729: Flags [P.], seq 1:42, ack 1, win 506, options [nop,nop,TS
val 822762238 ecr 1686175], length 41
14:19:56.439350 IP 192.168.2.2.49729 > mars.ssh: Flags [.], ack 42, win 1700, options [nop,nop,TS val 1686
175 ecr 822762238], length 0
14:19:57.186190 IP 192.168.2.2.49729 > mars.ssh: Flags [P.], seq 1:635, ack 42, win 1700, options [nop,nop
,TS val 1686250 ecr 822762238], length 634
14:19:57.186210 IP mars.ssh > 192.168.2.2.49729: Flags [.], ack 635, win 502, options [nop,nop,TS val 8227
62986 ecr 1686250], length 0
14:19:57.186856 IP mars.ssh > 192.168.2.2.49729: Flags [P.], seq 42:1122, ack 635, win 502, options [nop,n
op,TS val 822762987 ecr 1686250], length 1080
14:19:57.188115 IP 192.168.2.2.49729 > mars.ssh: Flags [.], ack 1122, win 1835, options [nop,nop,TS val 16
86250 ecr 822762987], length 0
```

[그림 9.39] tcpdump ssh 내용 확인(정상)

그렇다면, tun0 interface를 통해 LoRa packet을 확인해 보면 어떨까? 어라 source ip가 192.168.2.2가 아니라, 192.168.230.1이다. 아, NAT가 필요해 보인다. 즉, source ip 192.168.230.1을 192.168.2.2로 바꾼 상태로 내 보내야, 수신(복호화) 시, tun0 interface를 타게 될 것이다.

```
chyi@mars:/usr/local/etc/tinc/mesh/hosts$ sudo tcpdump -i tun0
[sudo] chyi의 암호:
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on tun0, link-type RAW (Raw IP), capture size 262144 bytes

13:42:51.696401 IP 192.168.230.1.44151 > mars.1700: RADIUS, Access-Accept (2), id: 0x7c length: 133
13:42:52.909448 IP 192.168.230.1.43021 > mars.1700: [|radius]
13:42:57.377169 IP 192.168.230.1.44151 > mars.1700: RADIUS, Access-Accept (2), id: 0x1b length: 275
13:42:58.109329 IP 192.168.230.1.43021 > mars.1700: [|radius]
13:43:03.309364 IP 192.168.230.1.43021 > mars.1700: [|radius]
13:43:04.389215 IP 192.168.230.1.44151 > mars.1700: RADIUS, Access-Accept (2), id: 0x2e length: 276
13:43:08.509262 IP 192.168.230.1.43021 > mars.1700: [|radius]
13:43:13.665042 IP 192.168.230.1.44151 > mars.1700: RADIUS, Access-Accept (2), id: 0xc9 length: 241
13:43:13.709402 IP 192.168.230.1.43021 > mars.1700: [|radius]
13:43:18.550441 IP 192.168.230.1.44151 > mars.1700: RADIUS, Access-Accept (2), id: 0xd length: 208
13:43:18.909393 IP 192.168.230.1.43021 > mars.1700: [|radius]
13:43:21.705636 IP 192.168.230.1.44151 > mars.1700: RADIUS, Access-Accept (2), id: 0x25 length: 133
13:43:24.109293 IP 192.168.230.1.43021 > mars.1700: [|radius]
```

[그림 9.40] tcpdump lora packet 내용 확인(비정상)

<target board>

root@RAK7258:~/workspace/tinc_output# **iptables -t nat -A POSTROUTING -o tun0 -j MASQUERADE**

root@RAK7258:~/workspace/tinc_output# **iptables -I FORWARD -i tun0 -j ACCEPT**

root@RAK7258:~/workspace/tinc_output# **iptables -I FORWARD -o tun0 -j ACCEPT**

OK, 드디어 packet이 들어온다. :)

```
chyil@mars:/usr/local/etc/tinc/mesh/hosts$ sudo tcpdump -i tun0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on tun0, link-type RAW (Raw IP), capture size 262144 bytes
14:28:00.580563 IP 192.168.2.2.33882 > mars.1700: RADIUS, Access-Accept (2), id: 0xaa length: 203
14:28:00.581175 IP mars.1700 > 192.168.2.2.33882: [|radius]
14:28:00.749667 IP 192.168.2.2.33882 > mars.1700: RADIUS, Access-Accept (2), id: 0xac length: 210
14:28:00.750229 IP mars.1700 > 192.168.2.2.33882: [|radius]
14:28:01.689612 IP 192.168.2.2.59455 > mars.1700: [|radius]
14:28:01.689697 IP mars.1700 > 192.168.2.2.59455: [|radius]
14:28:03.890510 IP 192.168.2.2.33882 > mars.1700: RADIUS, Access-Accept (2), id: 0xaf length: 135
14:28:03.890990 IP mars.1700 > 192.168.2.2.33882: [|radius]
14:28:06.889735 IP 192.168.2.2.59455 > mars.1700: [|radius]
14:28:06.889958 IP mars.1700 > 192.168.2.2.59455: [|radius]
^C
10 packets captured
10 packets received by filter
0 packets dropped by kernel
```

[그림 9.41] tcpdump lora packet 내용 확인(정상)

The screenshot shows the LoRa Server interface in a Chromium browser window. The URL is `localhost:8080/#/organizations/1/gateways/60c5a8ffe76130c/frames`. The left sidebar shows navigation options like Network-servers, Gateway-profiles, Organizations, All users, and a dropdown for 'loraserver'. The main content area is titled 'Gateways / rak7258michael'. It has tabs for GATEWAY DETAILS, GATEWAY CONFIGURATION, GATEWAY DISCOVERY, and LIVE LORAWAN FRAMES (which is selected). Below these tabs are buttons for HELP, PAUSE, DOWNLOAD, and CLEAR. A table lists received frames:

	DOWNLINK	2:26:18 PM	UnconfirmedDataDown	002665cb	▼
1	UPLINK	2:26:18 PM	UnconfirmedDataUp	002665cb	▼
2	UPLINK	2:26:14 PM	UnconfirmedDataUp	1a00bb72	▼
3	UPLINK	2:26:08 PM	ConfirmedDataUp	1b0f12dd	▼
4	UPLINK	2:26:00 PM	UnconfirmedDataUp	2604250b	▼
5	UPLINK	2:25:42 PM	UnconfirmedDataUp	26042ca8	▼
6	UPLINK	2:25:33 PM	ConfirmedDataUp	1b0511ee	▼
7	UPLINK	2:25:27 PM	UnconfirmedDataUp	2604250b	▼
8	UPLINK	2:25:26 PM	UnconfirmedDataUp	1a0027c2	▼
9	UPLINK	2:25:24 PM	UnconfirmedDataUp	00421e74	▼

[그림 9.42] LoRa Server gateway 패킷 수신 화면 - 정상 상태

The screenshot shows the LoRa Server interface in a Chromium browser window. The URL is `localhost:8080/#/organizations/1/applications/3/devices/0042a2bffd9ea277/data`. The left sidebar shows navigation options like Network-servers, Gateway-profiles, Organizations, All users, and a dropdown for 'loraserver'. The main content area is titled 'Applications / rak7200 / Devices / rak7200'. It has tabs for MJS, CONFIGURATION, KEYS (OTAA), ACTIVATION, and DEVICE DATA (which is selected). Below these tabs are buttons for HELP, PAUSE, DOWNLOAD, and CLEAR. A table shows device data:

	2:29:19 PM	uplink	^
1	adr: true		
2	applicationID: "3"		
3	applicationName: "rak7200"		
4	data: "CAIBgANx/vAAP4ABYYAAAABAAEAgK7Awb/WQ=="		
5	devEUI: "0042a2bffd9ea277"		
6	deviceName: "rak7200"		
7	fCnt: 2		
8	fPort: 2		
9	rxInfo: [] 1 item		
10	▼ 0: [] 5 keys		
11	gatewayID: "60c5a8ffe76130c"		
12	lorASNR: 10		
13	location: [] 3 keys		
14	altitude: 3		
15	latitude: 37.5371163		
16	longitude: 127.0078127		
17	name: "rak7258michael"		
18	rssi: -54		
19	▼ txInfo: [] 2 keys		

[그림 9.43] LoRa Server application/device 패킷 수신 화면 - 정상 상태

이 대목에서 집고 넘어갈 부분이 있다. Wireguard-go에서도 NAT 문제로 인해 통신이 안되었을 가능성성이 있다. 따라서 다시 앞으로 돌아가서 이 부분을 적용해 보아야 겠다 :)

9) Mesh VPN 기능을 포함하는 CLI tool(vtysh) cross-compile하기

이제 부터는 Mesh VPN 기능을 포함한 vtysh를 OpenWrt 환경에서 build하는 절차를 소개하고자 한다.

참고: 이 대목에서 vtysh에 Mesh VPN 기능을 추가시키는 절차를 설명하지는 않는다(이미 2.0 version에서 작업한 내용임).

<Ubuntu 16.04>

```
$ cd ~/workspace/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628
```

```
$ build/envsetup.sh
```

...

```
$ cp -R vtysh/
```

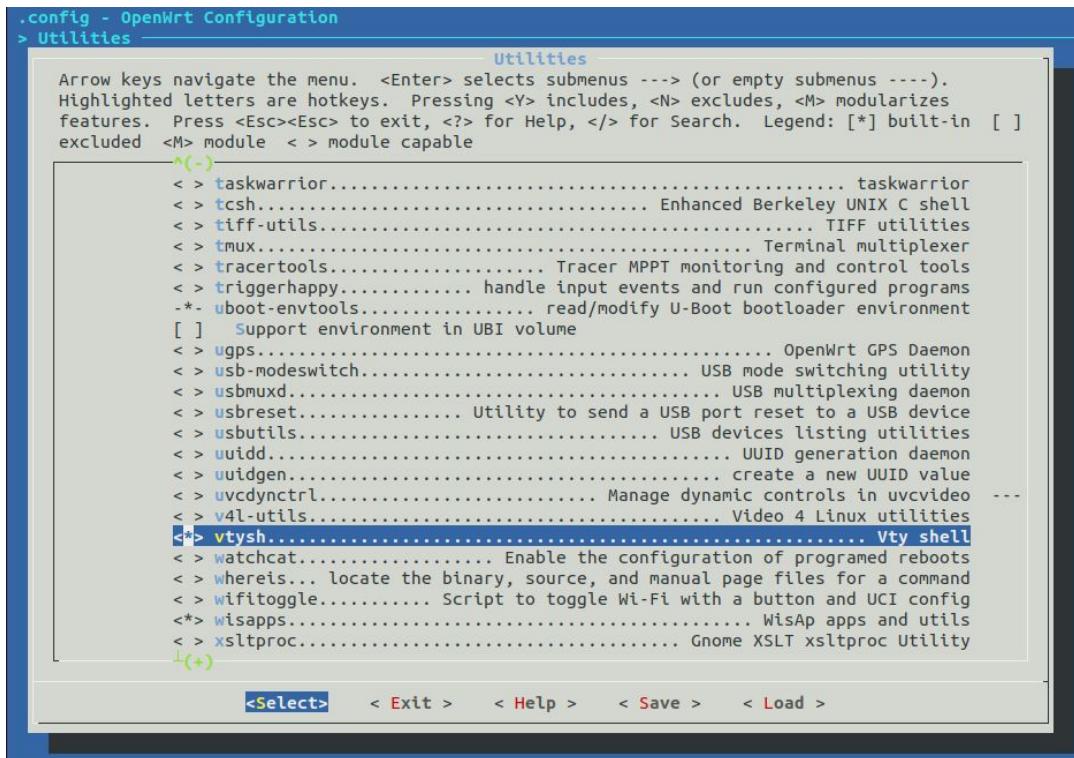
```
~/workspace/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628/wiswrt/15.05-rak
-rc2/package/utils
```

- vtysh source code(spnbox/system/cli/openwrt/vtysh)를 package/utils 아래로 복사한다.

- 주의: build/envsetup.sh 실행 상태에서 파일 복사를 해 주어야 한다. 사전에 해 버리면 build/envsetup.sh이 실행되는 과정에서 모두 지워지게 된다.

```
$ build/envsetup.sh
```

- build/envsetup.sh 마지막 부분에서 make menuconfig 자동 실행되게 하자.
- 아래 vtysh 메뉴를 선택(enable)하자.



[그림 9.44] make menuconfig - vtysh 선택

```
$ cd
~/workspace/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628/wiswrt/15.05-rak
-rc2/package/utils
```

```
$ cd vtysh/src
$ cp Makefile.rak7258 Makefile
```

```
$ make
... build 가 한참 진행된다...
```

<여기서 잠깐 - vtysh만 build하려면 ...>

```
$ build/envsetup.sh
$ pwd
~/workspace/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628
```

```
$ cd wiswrt/15.05-rak-rc2  
$ make package/utils/vtysh/clean -j1 V=s  
$ make package/utils/vtysh/compile -j1 V=s
```

```
$ cd wiswrt/15.05-rak-rc2/build_dir
```

```
chyi@twoip-desktop:~/workspace/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628/wiswrt/15.05-rak-rc2/build_dir$ find . -name "vtysh" -print
```

```
./target-mipsel_24kec+dsp_uClibc-0.9.33.2/vtysh  
./target-mipsel_24kec+dsp_uClibc-0.9.33.2/vtysh/ipkg-ramips_24kec/vtysh  
.target-mipsel_24kec+dsp_uClibc-0.9.33.2/vtysh/ipkg-ramips_24kec/vtysh/sbin/vtysh  
./target-mipsel_24kec+dsp_uClibc-0.9.33.2/vtysh/vtysh  
./target-mipsel_24kec+dsp_uClibc-0.9.33.2/root-ramips/sbin/vtysh
```

```
chyi@twoip-desktop:~/workspace/RAKWireless/RAK831-LoRaGateway-OpenWRT-MT7628/wiswrt/15.05-rak-rc2/build_dir$ scp  
.target-mipsel_24kec+dsp_uClibc-0.9.33.2/vtysh/ipkg-ramips_24kec/vtysh/sbin/vtysh  
root@172.30.1.26:~/workspace
```

→ Build된 결과 파일(vtysh)을 target board(172.30.1.26)로 올린 후, 아래와 같이 실행한다.

```

root@RAK7258:~/workspace# ./vtysh
Build On Sep 18 2019 15:28:05
RAK7258> en
RAK7258# configure terminal
RAK7258(config)#
  bridge      add/modify the bridge information
  date        Set the date
  enable      Modify enable password parameters
  exit        Exit current mode and down to previous mode
  factory     Go back to the factory default state
  hostname   Set system's network name
  ip          IP information set
  meshvpn    Configure meshvpn tunnel
  nameserver Config the dns server
  no         Negate a command or set its defaults
  p2p        Configure p2p tunnel
  password   Modify password parameters
  ping       Send echo messages
  se          Configure SoftEther VPN
  sfirewall  Configure spn firewall rules
  show       Show running system information
  spn        Configure SPN rules
  ssh        Open a ssh connection
  swupgrade  spnbox software upgrade
  write      Write running configuration to memory, network, or terminal
RAK7258(config)# meshvpn
  add        Add a new option for mesh vpn
  editpeer  Edit peer information
  id        Specify a meshvpn id
  start     Start the meshvpn daemon
RAK7258(config)#

```

[그림 9.45] RAK7258 vtysh 실행 모습

참고1: 다른 2ston_XXX binary 처럼 spnbox build script 내에서 vtysh를 build하려면, openwrt source가 준비되어 있어야 한다. 아니면, x86 series용 vtysh처럼 source를 재구성할 경우, 위와 같이 복잡하게 build할 필요 없이, 다른 2ston_XXX 처럼 쉽게 build가 가능하다. 현재 AP series가 모두 openwrt 기반으로 build하는 형태라서 위와 같이 (다소 복잡하게) build를 진행했을 뿐이다.

참고2: vtysh code 정리 필요하다 - L3 spn 기능 제거해야 함.

10) RAK7258 기반 SPNBox-300 model 작업하기

이 부분에 대한 세세한 설명은 아래 shell script로 대신하기로 한다.

```

2ston_spnbox_prj/spnbox/build_spnbox.sh ⇒
2ston_spnbox_prj/spnbox/build/build_lora_series.sh ⇒
2ston_spnbox_prj/spnbox/build/rakwireless_7258.sh ⇒

```

2ston_spnbox_prj/spnbox/build/**openwrt-repack.sh**

```
chyi@mars:~/workspace/spn/2ston_spnbox_prj/spnbox$ ./build_spnbox.sh
=====
** 2STON SPNBox/Cloud Image Generator **
=====

Would you like to:
 1) create the SPNBox image for Project Boards
 2) create the SPNBox image for Access Points
 3) create the SPNBox image for LoRa Gateway
 4) create the SPNBox image for X86 Series boxes
 5) create the SPNBox image for Server/Cloud Series
 6) create the SPNBox image for New OSes
 7) login to AWS EC2
 p) print the list of some packages needed to build spnbox
 i) get some Information
 q) Quit this menu

Please select one of the above (7 or p/i/q): 3

-----
** 2STON SPNBox LoRa Gateways **
-----

 1. SPNBox-300      - MIPS32 RAKWireless RAK7258 board
 2. SPNBox-310      - MIPS32 MatchX MX1702 board
 3. SPNBox-320      - MIPS32 RAKWireless RAK7249 board
 4. Login to AWS EC2
 r. Return to main menu

Please select one of the above (1-4 or r): 1
```

[그림 9.46] SPNBox-300 image build menu

참고: LoRa Gateway 기반 SPNBox 모델명 작명에 관하여...

SPNBox-300 : RAK7258

SPNBox-310: MatchX MX1702

SPNBox-320: RAK7249

(*) 위 세 모델 모두 SPNBox-300 series로 구분한 이유는 각각의 LoRa Gateway에 사용된 CPU가 모두 MT7268 계열로 SPNBox-200용과 동일하기 때문이다.

```

Scanning dependencies of target test_suite_gcm.aes192_de
[100%] Building C object tests/CMakeFiles/test_suite_gcm.aes192_de.dir/test_suite_gcm.aes192_de
.c.o
[100%] Linking C executable test_suite_gcm.aes192_de
[100%] Built target test_suite_gcm.aes192_de
Let's use a prebuilt binary which was made on Ubuntu 16.04

>>> Do you want to remove the existing lzo output files ?(y/n)n
#####
#>>> Do you want to remove the existing openssl output files ?(y/n)n
#####
#>>> Do you want to remove the existing ncurses output files ?(y/n)n
#####
#>>> Do you want to remove the existing readline output files ?(y/n)n
#####
#>>> Do you want to remove the existing mesh spn output files ?(y/n)y
#####

```

[그림 9.47] SPNBox-300 image build 과정 중 y/n 선택

참고: mesh vpn build를 위해서는 몇가지 library(lzo, openssl 등)가 필요한데, 시간이 오래 걸릴 수 있으니 매번 build할 필요는 없다.

Build가 정상적으로 진행될 경우, 아래와 같은 결과물을 얻게 된다.

```

chyi@mars:~/workspace/spn/2ston_spnbox_prj/spnbox/output$ ls -la
합계 14884
drwxr-xr-x 2 chyi chyi 4096 9월 18 19:44 .
drwxr-xr-x 7 chyi chyi 4096 9월 18 19:03 ..
-rw-r--r-- 1 chyi chyi 2167600 9월 18 19:44 2ston_spnbox.bin
-rw-r--r-- 1 root root 10878980 9월 18 19:44 spnbox-LoRaGateway_1.1.0050_Release_r183.bin
-rw-r--r-- 1 chyi chyi 2167507 9월 18 19:43 spnbox_install.tar.gz
-rw-r--r-- 1 chyi chyi 864 9월 18 19:44 update.bin
-rw-r--r-- 1 chyi chyi 9 9월 18 19:43 version.txt
chyi@mars:~/workspace/spn/2ston_spnbox_prj/spnbox/output$ 

```

[그림 9.48] SPNBox-300 image build menu

이제 아래 파일을 가지고 RAK7258 LoRa Gateway를 upgrade해 볼 차례이다. [TBD]

spnbox-LoRaGateway_1.1.0050_Release_r183.bin

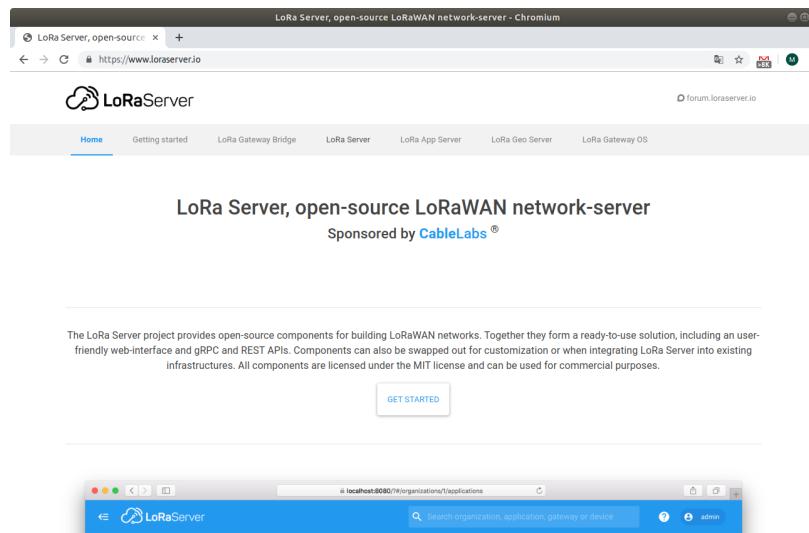
참고: 아직 완벽하게 준비된 것은 아니니, 이 부분 테스트는 추후(다른 내용 좀 더 검토 후) 진행하기로 하자.

그 전에 **spnbox_install.tar.gz** 파일을 target board에 올려, 수동으로 설치하는 것을 먼저 테스트해 보기로 하자. [TBD]

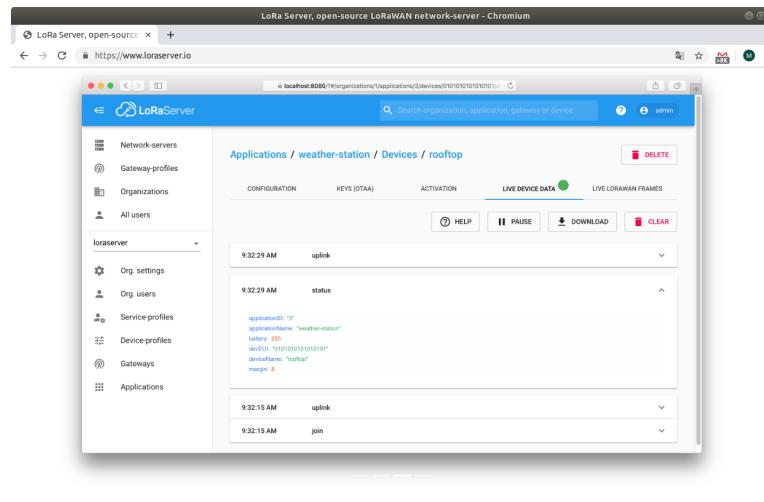
10. LoRaServer Project 1 - 설치 & 운용

이번 장에서는 TTN과 유사하지만 open source project인 LoRaServer project(<https://www.loraserver.io>)를 소개해 보도록 하겠다. Home page도 그렇고, lora app server WebUI 화면도 그렇고 ... 전체적으로 완성도가 매우 높은 project로 보인다. 앞서 5장에서 이미 LoRaServer를 RAK7258 & RAK7200과 연동시켜 본 바가 있다. 설명의 순서가 뒤바뀐 느낌은 있으나, 어쩔 수 없다. ㅋ

"The LoRa Server project provides open-source components for building LoRaWAN networks. Together they form a ready-to-use solution, including an user-friendly web-interface and gRPC and REST APIs. Components can also be swapped out for customization or when integrating LoRa Server into existing infrastructures. All components are licensed under the MIT license and can be used for commercial purposes."

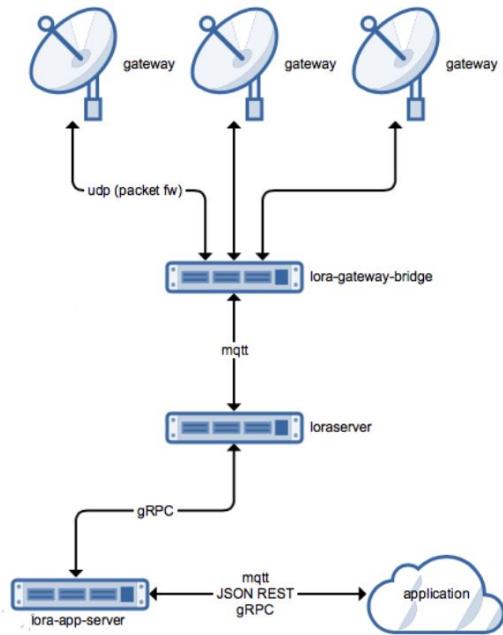


[그림 10.1] LoRaServer Project 소개(1)

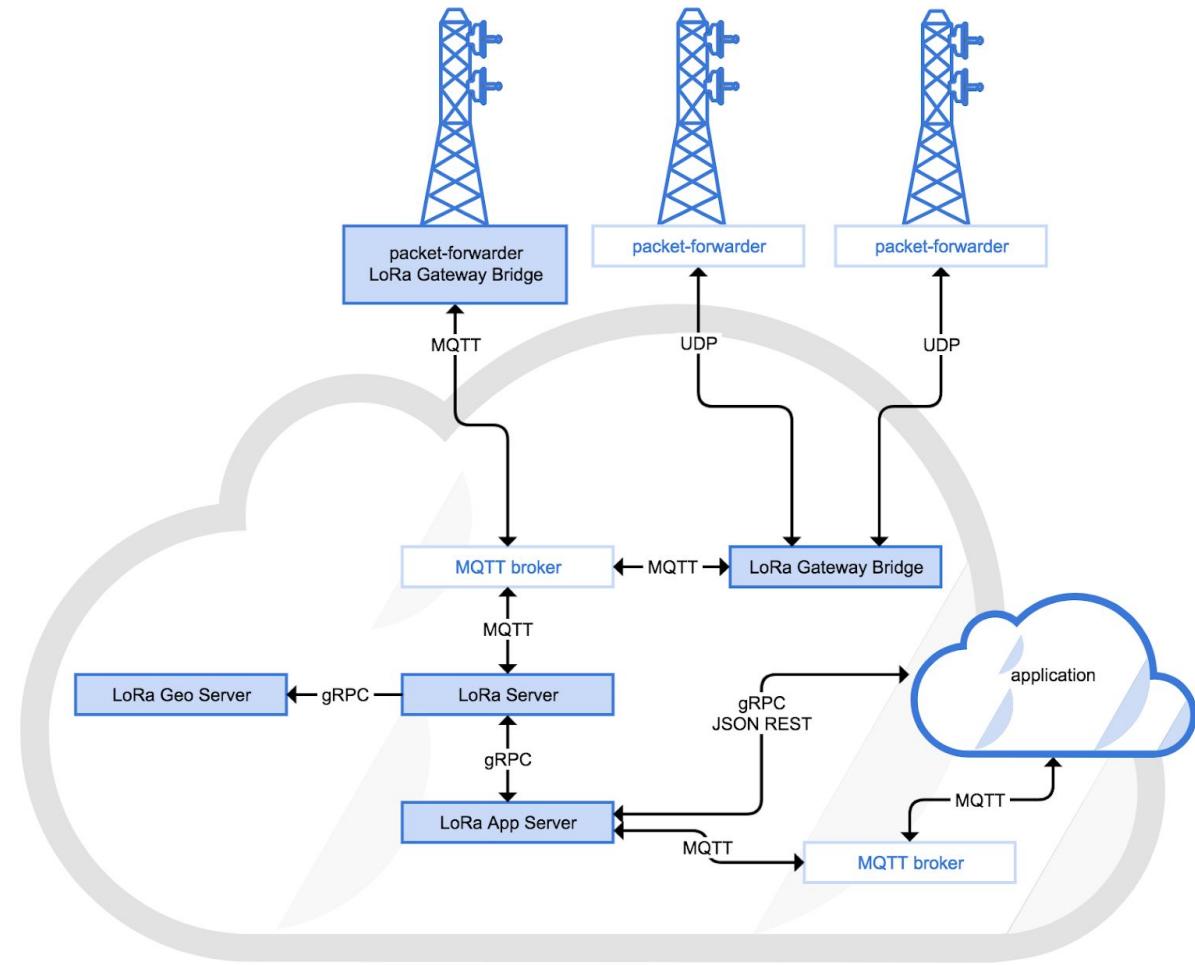


[그림 10.2] LoRaServer Project 소개(2)

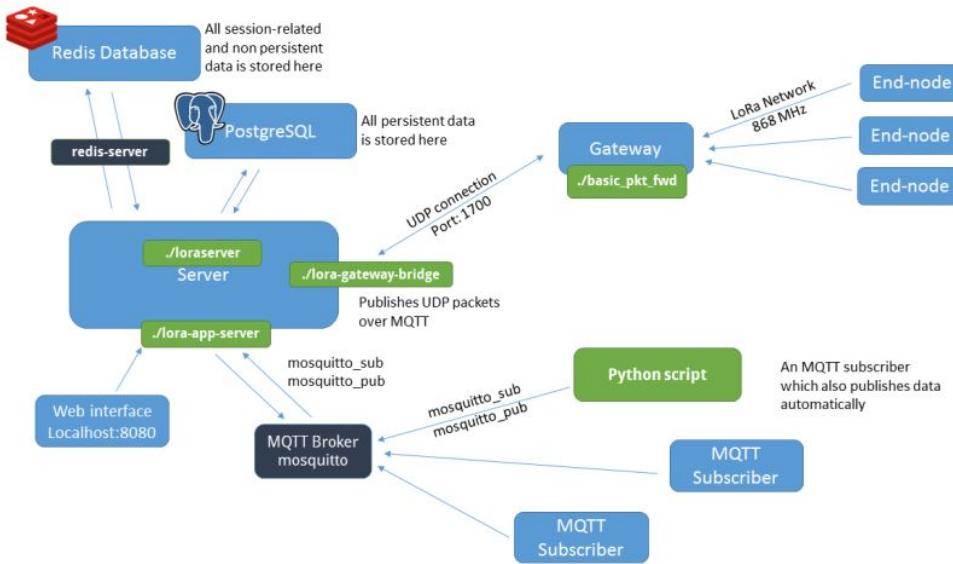
아래 두장의 그림은 LoRa Server의 S/W 구조를 한눈에 파악하는데 도움을 준다. 굳이 부연 설명을 하지는 않겠다. 척보면 앱니다^^



[그림 10.3] LoRaServer 아키텍처(1)



[그림 10.4] LoRaServer 아키텍쳐(2)



[그림 10.5] LoRaServer 아키텍쳐(3)

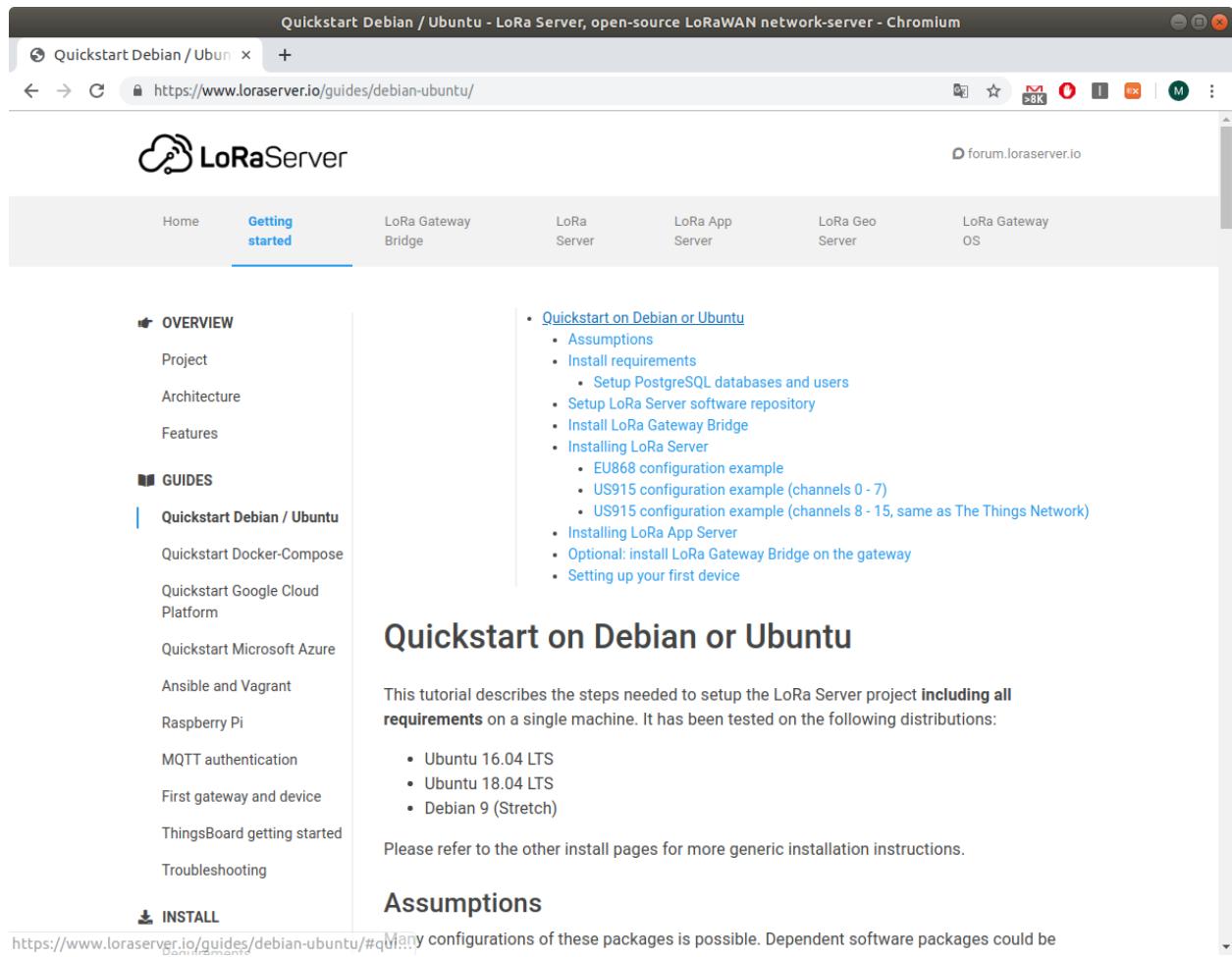
우리가 나중에 Viewer code(WebUI, android/iOS app)를 작성한다면, 그 위치는 그림 10.5 하단 우측에 있는 MQTT Subscriber 정도가 될 것이다. 왜 그런지 잘 생각해 보기 바란다. (나중에 정리한 것이지만) gRPC 혹은 REST API로 연동할 수도 있겠다.

참고: **Viewer code가 별도로 필요한 이유** - LoRa App Server WebUI도 매우 훌륭하기는 하나, 일반 사용자가 보기에는 조금 어려움이 느껴진다. 따라서 일전에 만들었던 Texaking demo project 수준으로 light한 UI(app 포함)를 만들 필요가 있다. **이게 우리의 경쟁력을 높여 줄 것으로 믿는다 :)**

1) Ubuntu 18.04에 LoRaServer 설치하기

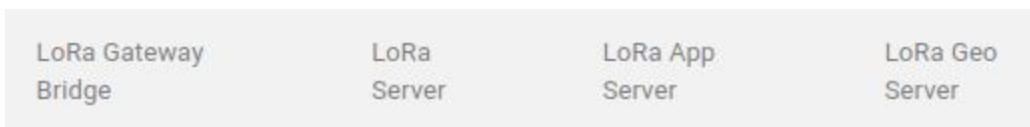
이 절에서 설명하는 내용은 아래 site의 내용을 기초로 하였다. 따라서 보다 자세한 내용을 위해서는 반드시 아래 site의 내용을 꼼꼼히 살펴 보기 바란다.

<https://www.loraserver.io/guides/debian-ubuntu/>



[그림 10.6] Quick Start Debian/Ubuntu page

LoRaServer는 아래와 같이 4개의 component로 구성되어 있다. 이중 **LoRa Server**가 LoRa network server에 해당하며, **LoRa App Server**가 LoRa Application Server에 해당한다. **LoRa Gateway Bridge**는 글자그대로 LoRa Gateway(예: RAK7258)와 LoRa network server를 연결 시켜 주는 논석(packet format 변화가 일어남)이며, **LoRa Geo Server**는 LoRaServer project에만 있는 Geolocation(위치 정보) 관련 server가 되겠다.



위의 site 내용을 그대로 따라하기만 하면 설치가 정상적으로 되므로, 여기서는 중복해서 그 과정을 설명하지는 않는다.

(나중에 정리한 것임 - 반드시 필요한 내용만 간략히 정리함)

```
$ sudo apt update
```

<mosquitto MQTT broker 설치>

```
$ sudo apt install mosquitto mosquitto-clients redis-server redis-tools postgresql
```

<Postgres DB table 생성>

```
$ sudo -u postgres psql
```

```
$ sudo -u postgres psql
```

```
psql (10.10 (Ubuntu 10.10-0ubuntu0.18.04.1))
```

```
Type "help" for help
```

```
postgres=# create role loraserver_as with login password 'spnbox1234';
```

```
CREATE ROLE
```

```
postgres=# create role loraserver_ns with login password 'spnbox1234';
```

```
postgres=# create database loraserver_as with owner loraserver_as;
```

```
CREATE DATABASE
```

```
postgres=# create database loraserver_ns with owner loraserver_ns;
```

```
CREATE DATABASE
```

```
postgres=# \c loraserver_as
```

```
You are now connected to database "loraserver_as" as user "postgres".
```

```
loraserver_as=# create extension pg_trgm;
```

```
CREATE EXTENSION
```

```
loraserver_as=# create extension hstore;
```

```
CREATE EXTENSION
```

```
loraserver_as=# \q
```

<LoRa Server repository 설정>

```
$ sudo apt install apt-transport-https dirmngr
```

```
$ sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys
```

```
1CE2AFD36DBCCA00
```

```
$ sudo echo "deb https://artifacts.loraserver.io/packages/3.x/deb stable main" |
```

```
sudo tee /etc/apt/sources.list.d/loraserver.list
```

<LoRa Gateway Bridge 설치>

```
$ sudo apt install lora-gateway-bridge
```

```
$ sudo systemctl start lora-gateway-bridge
```

```
$ sudo systemctl enable lora-gateway-bridge
```

<LoRa Server 설치/설정 변경>

```
$ sudo apt install loraserver
```

```
$ sudo vi /etc/loraserver/loraserver.toml
```

→ 수정 or 추가한 부분만 정리

```
dsn="postgres://loraserver_ns:spnbox1234@localhost/loraserver_ns?sslmode=disable"
```

```
#michael@2019.10.07 --
name="KR_920_923"
```

```
#added by michael@2019.10.07 --
enabled_uplink_channels=[0, 1, 2, 3, 4, 5, 6, 7]
```

```
#blocked by michael@2019.10.07 --
# [[network_server.network_settings.extra_channels]]
# frequency=867100000
# min_dr=0
# max_dr=5
#
# [[network_server.network_settings.extra_channels]]
# frequency=867300000
# min_dr=0
# max_dr=5
#
# [[network_server.network_settings.extra_channels]]
# frequency=867500000
# min_dr=0
# max_dr=5
#
# [[network_server.network_settings.extra_channels]]
# frequency=867700000
# min_dr=0
# max_dr=5
#
# [[network_server.network_settings.extra_channels]]
# frequency=867900000
# min_dr=0
# max_dr=5
```

```
$ sudo systemctl start loraserver
$ sudo systemctl enable loraserver
```

<LoRa App Server 설치/설정>

```
$ sudo apt install lora-app-server
$ sudo vi /etc/lora-app-server/lora-app-server.toml
→ 수정 or 추가한 부분만 정리
```

```
#michael@2019.10.07 --
dsn="postgres://loraserver_as:spnbox1234@localhost/loraserver_as?sslmode=disable"
```

```
#bind="0.0.0.0:8080"
#michael@2019.10.07 --
bind="0.0.0.0:7194"  
  
#michael@2019.10.07 --
jwt_secret="spnbox1234"
```

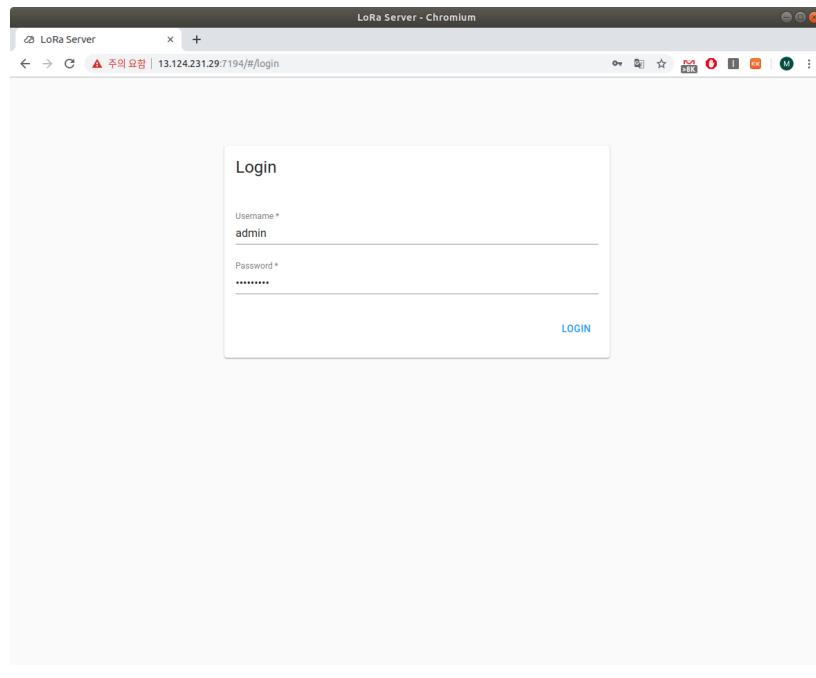
```
$ sudo systemctl start lora-app-server
$ sudo systemctl enable lora-app-server
```

지금까지 내용은 AWS EC2(Ubuntu 18.04, 13.124.231.29)에서 진행한 것임.

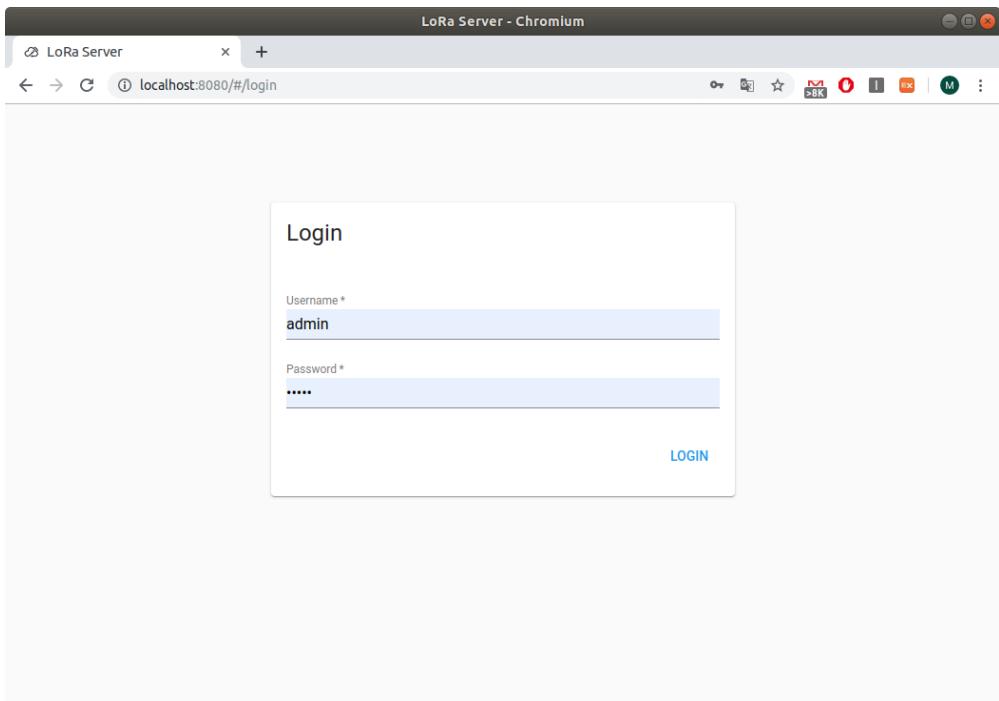
<AWS EC2>

<http://13.124.231.29:7194>

id/pass: admin/spn)(*^%&



정상 설치가 되었는지를 확인하기 위해서는 아래와 같이 <http://localhost:8080> 으로 접속(LoRa App Server에 접속)해 보면 된다.



[그림 10.7] LoRa App Server 로긴 화면(id/pas: admin/admin)

아래 내용은 몇몇 LoRa process의 로그를 살펴 보는 방법을 정리해 놓은 것인데, 실제 로그 내용을 보면 어디에 문제가 있는지를 예상해 볼 수 있다.

```
$ sudo journalctl -f -n 100 -u loraserver
→ LoRa Server log 확인하기

$ sudo journalctl -f -n 100 -u lora-app-server
→ LoRa App Server log 확인하기
$ sudo journalctl -f -n 100 -u lora-gateway-bridge
→ LoRa Gateway Bridge log 확인하기
```

만일, 정상 동작이 안된다면, 아래 주요 configuration file을 수정해 주는 것도 답이 될 수 있다.

```
$ sudo vi /etc/loraserver/loraserver.toml
→ LoRa Server configuration file
→ 툭Sir/name="KR_920_923" 추가해 주어야 함.
```

```
$ sudo vi /etc/lora-app-server/lora-app-server.toml  
→ LoRa App Server configuration file
```

```
$ sudo vi /etc/lora-gateway-bridge/lora-gateway-bridge.toml  
→ LoRa Gateway Bridge configuration file
```

2) LoRaServer Build 하기

이번에는 LoRa Server, LoRa App Server, LoRa Gateway Bridge 등을 직접 build해 보기로 하자.
나중에 source code를 수정해야 할 수도 있으니 말이다~ ㅋ

2-1) LoRa Server build 하기

LoRa Server는 Go 언어로 구현되어 있다. 따라서 사전에 Go가 설치되어 있어야 한다(Go 설치 방법은 생략함).

Build 관련 자세한 사항은 site를 참조하기 바란다.

<https://www.loraserver.io/loraserver/community/source/>

Docker를 사용하지 않고, 직접 build하는 절차는 다음과 같다(실제로 해 보면, make test에서 멈춰서는 문제만 있을 뿐, 모두 정상적으로 build가 이루어진다).

참고: Docker를 이용하면 build 과정 중 환경적인 문제로 인한 build 실패 문제를 해결할 수 있는 장점이 있다. 알아두면 좋다^^. 하지만, 여기서는 docker 없이 직접 build해 볼 것이다.

```
# install development requirements
make dev-requirements

# run the tests
make test

# compile
make build

# compile snapshot builds for supported architectures
make snapshot
```

2-2) LoRa App Server build 하기

LoRa App Server는 Go 언어와 Node.js로 구현되어 있다. 따라서 사전에 Go와 Node.js가 설치되어 있어야 한다.

<https://www.loraserver.io/lora-app-server/community/source/>

먼저 아래 site 내용을 참조하여 node.js를 설치해 보자(꼭 여기를 참조해야 하는 것은 아님).

<https://tecadmin.net/install-latest-nodejs-npm-on-ubuntu/>

```
$ sudo apt-get install curl
$ curl -sL https://deb.nodesource.com/setup_10.x | sudo -E bash -
$ sudo apt-get install nodejs
$ node -v
v10.16.3
$ npm -v
6.9.0
```

다음으로 "Go protocol buffer support" 관련 패키지를 설치해야 한다(요 부분이 처음에 조금 복잡하게 느껴졌다). 여기서는 여러가지 version 중에서 C++ version을 설치하기로 한다.

참고: **Protocol Buffers - Google's data interchange format** 이라고 한다. (나중에 정리한 것이지만) Protocol Buffers framework은 gRPC에서 주로 된다.

<Go Protocol Buffer 설치 과정>

```
https://github.com/protocolbuffers/protobuf/blob/master/src/README.md
```

-> C++ version으로 설치해 보자.

```
$ sudo apt-get install autoconf automake libtool curl make g++ unzip  
$ git clone https://github.com/protocolbuffers/protobuf.git
```

```
$ cd protobuf/  
$ git submodule update --init --recursive
```

```
$ ./autogen.sh
```

```
$ ./configure  
$ make  
$ make check
```

```
$ sudo make install  
$ sudo ldconfig
```

이후 아래 내용을 실행하면 정상적으로 build가 이루어지게 된다. 위의 과정(node.js 및 Go protocol buffer)이 잘못 진행된 경우, make api에서 예상치 못한 에러가 발생 하니 주의하기 바란다(한번 build해 보라, 무슨 얘기인지 금방 알 수 있다).

```
# install all requirements
make dev-requirements ui-requirements

# cleanup workspace
make clean

# generate the API source-code (run this after changing the .proto files)
make api

# run the tests
make test

# compile (this will also compile the ui and generate the static files)
make build

# compile snapshot builds for supported architectures (this will also compile th
make snapshot
```

2-3) LoRa Gateway Bridge build 하기

LoRa Gateway Bridge는 LoRa packet-forwarder protocol을 LoRa Server common protocol(JSON & protobuf)로 바꿔주는 역할을 한다. 역시 Go 언어로 작성되어 있다.

<https://www.loraserver.io/lora-gateway-bridge/community/source/>

```
# install development requirements
make dev-requirements

# run the tests
make test

# compile
make build

# compile snapshot for supported architectures (using goreleaser)
make snapshot
```

참고: LoRa Server source code를 build해 보니, x86은 물론이고, ARM(32, 64bit용) 용 binary도 함께 생성된다. 캬~ 암만 봐도 잘 만들었던 말이지~

11. LoRaServer Project 2 - External Interface

이번 장에서는 LoRa App Server와 2ip viewer(OLoRa)간의 interface(gRPC, REST API, MQTT ...)와 관련하여 **집중 분석(?)**해 보고자 한다.

Go 언어 study 합시다~

참고: 집중 분석은 담당자를 별도로 지정해 드릴테니, 그 때 해 보는 걸로 합시다 :)

본 장에서 설명하는 내용은 아래 page 내용을 기준으로 작성한 것이다(Rough하게 대략적인 방향만 정리하였음).

<https://www.loraserver.io/lora-app-server/integrate/sending-receiving/>

1) Event Types

LoRa App Server \Leftrightarrow User Application을 통합하기 위해 필요한 External Interface(혹은 Application Integration 방식)를 이해하기 위해서는, 먼저 LoRa App Server에서 정의하고 있는 Event Types(**Uplink, Status, Join, Ack, Error**)을 파악할 필요가 있다. 즉, External Interface를 통해 주고 받는 data & format이 무엇인지를 사전에 파악해야만 실제 Integration(LoRa App Server \Leftrightarrow User Application) 작업을 할 수 있을 것이다.

<https://www.loraserver.io/lora-app-server/integrate/sending-receiving/#event-types>

The screenshot shows a web browser window with the title "Sending / receiving data - LoRa App Server documentation - Chromium". The URL in the address bar is "loraserver.io/lora-app-server/integrate/sending-receiving/#event-types". The page content is titled "Event types" and includes a section for "Uplink". It describes uplink application payloads containing data and meta-data. Below this, there is a JSON code snippet illustrating the structure of an uplink payload. The left sidebar contains navigation links for "Frame logging", "Event logging", "Firmware updates", "INTEGRATE" (with "Sending / receiving data" selected), "API", "Authentication", "gRPC", "RESTful JSON", "METRICS" (with "Prometheus" selected), "COMMUNITY & SUPPORT" (with "Support" and "Contribute" selected), and "Source".

```
{
    "applicationID": "123",
    "applicationName": "temperature-sensor",
    "deviceName": "garden-sensor",
    "devEUI": "0202020202020202",
    "rxInfo": [
        {
            "gatewayID": "0303030303030303", // ID of the receiving gateway
            "name": "rooftop-gateway", // name of the receiving gateway
            "time": "2016-11-25T16:24:37.295915988Z", // time when the package was received (GPS)
            "rssi": -57, // signal strength (dBm)
            "loRaSNR": 10, // signal to noise ratio
            "location": {
                "latitude": 52.3740364, // latitude of the receiving gateway
                "longitude": 4.9144401, // longitude of the receiving gateway
                "altitude": 10.5, // altitude of the receiving gateway
            }
        },
        ...
    ],
    "txInfo": {
        "frequency": 868100000, // frequency used for transmission
        "dr": 5 // data-rate used for transmission
    },
    "adr": false, // device ADR status
    "fCnt": 10, // frame-counter
    "fPort": 5, // FPort
    "data": "...", // base64 encoded payload (decrypted)
    "object": {
        "temperatureSensor": {"1": 25}, // decoded object (when application coded has been configured)
        "humiditySensor": {"1": 32}
    },
    "tags": { // User-provided tags (optional)
        "key": "value"
    }
}
```

[그림 11.1] Application 통합을 위한 Event Types 5가지

지금 부터는 이러한 Event Types을 기초로 할 때, **User Application ↔ LoRa App Server 간 상호 통신을 하기 위한 몇 가지 IPC(Inter Process Communication) 방법**을 하나씩 소개해 보도록 하겠다. 미리 얘기하자면, 우리는 (모든 방법을 다 소화할 필요는 없으므로) 이 중 가장 효과적인 방법(예: MQTT)을 하나 선택하여 구현하면 될 것으로 보인다.

2) gRPC - google이 만든 RPC framework

gRPC는 google의 여러 cloud product에서 사용하기 위해 만든 open source RPC(remote procedure call) framework이다. Google이 만들어서 g가 앞에 붙은 모양(추측)이다. gRPC를 지원하는 programming language로는 아래와 같은 것들이 있다.

<gRPC site>

<https://www.grpc.io/docs/guides/>

C++, Java (incl. support for Android), Objective-C (for iOS), Python, Ruby, Go, C#, Node.js

gRPC by Language		
C++	Java	Python
Quick Start Guide gRPC Basics Tutorial API Reference	Quick Start Guide gRPC Basics Tutorial API Reference Generated Code Reference	Quick Start Guide gRPC Basics Tutorial API Reference Generated Code Reference
GO	Ruby	C#
Quick Start Guide gRPC Basics Tutorial API Reference Generated Code Reference	Quick Start Guide gRPC Basics Tutorial API Reference	Quick Start Guide gRPC Basics Tutorial API Reference
Node.js	Android Java	Objective-C
Quick Start Guide gRPC Basics Tutorial API Reference	Quick Start Guide gRPC Basics Tutorial API Reference Generated Code Reference	Quick Start Guide (the "grpc-dotnet" version)
PHP	Dart	Web
Quick Start Guide gRPC Basics Tutorial API Reference	Quick Start Guide gRPC Basics Tutorial	Quick Start Guide gRPC Basics Tutorial

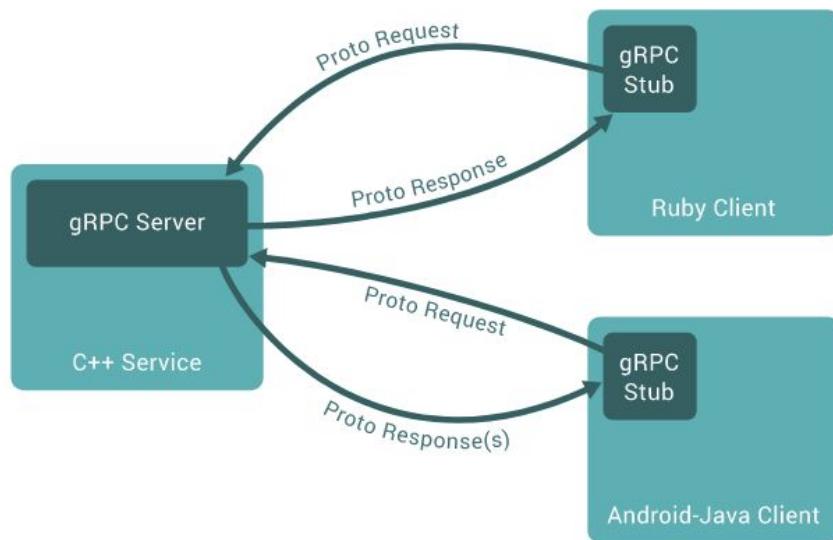
[그림 11.2] gRPC 지원 programming language

gRPC는 (이 역시 google에서 만든) Protocol Buffer라는 것(framework)을 기반으로 동작한다.

<Protocol Buffers site>

<https://developers.google.com/protocol-buffers/>

아래 그림은 gRPC의 동작 원리(개요)를 표현하고 있는데, 정확한 동작 원리를 이해하기 위해서는 실제 예제를 통해 그 사용법을 파악해 볼 필요가 있다.



[그림 11.3] gRPC 개요도

[숙제] 아래 site를 통해 다양한 programming language 환경에서의 gRPC 예제를 직접 들려보기 바란다.

The screenshot shows a browser window titled "gRPC - Chromium" displaying the gRPC tutorial page for C++. The URL in the address bar is grpc.io/docs/tutorials/basic/cpp/. The page has a dark teal header with the "gRPC" logo and navigation links for ABOUT, DOCS, BLOG, COMMUNITY, PACKAGES, and FAQ. Below the header, there's a "Documentation" section with a "TUTORIALS" link. On the left, a sidebar lists various tutorials: Tutorials, Auth - Objective-C, Async - C++, Basic, Android, C#, C++, Dart, Go, and Java. The main content area is titled "gRPC Basics - C++". It describes the tutorial as providing a basic introduction for C++ programmers. It lists what you'll learn: defining a service in a .proto file, generating server and client code using the protocol buffer compiler, and using the C++ gRPC API to write a simple client and server. It also notes that the example uses the proto3 version of the protocol buffers language.

[그림 11.4] gRPC Tutorial Page

참고: RPC는 (이미 사라져 버린) SUN 사에서 처음 그 개념을 소개한 바 있으며, 이미 Android 등에서도 그 기능이 널리 사용되고 있을 만큼, 아주 보편적인 IPC(Inter Process Communication) 중 하나이다. 하지만 상대적으로 다른 IPC 기법과 비교해 난이도가 높은게 사실이다. 따라서 꼭 한번 사용해 보시길 권한다~

3) REST API

REST API(HTTP 기반)는 보편적으로 널리 알려진 만큼, 본 문서에서는 부가 설명은 생략하도록 하겠다. 아래 site를 포함하여 여러 인터넷 site를 살펴보기 바란다.

<https://mangkyu.tistory.com/46>

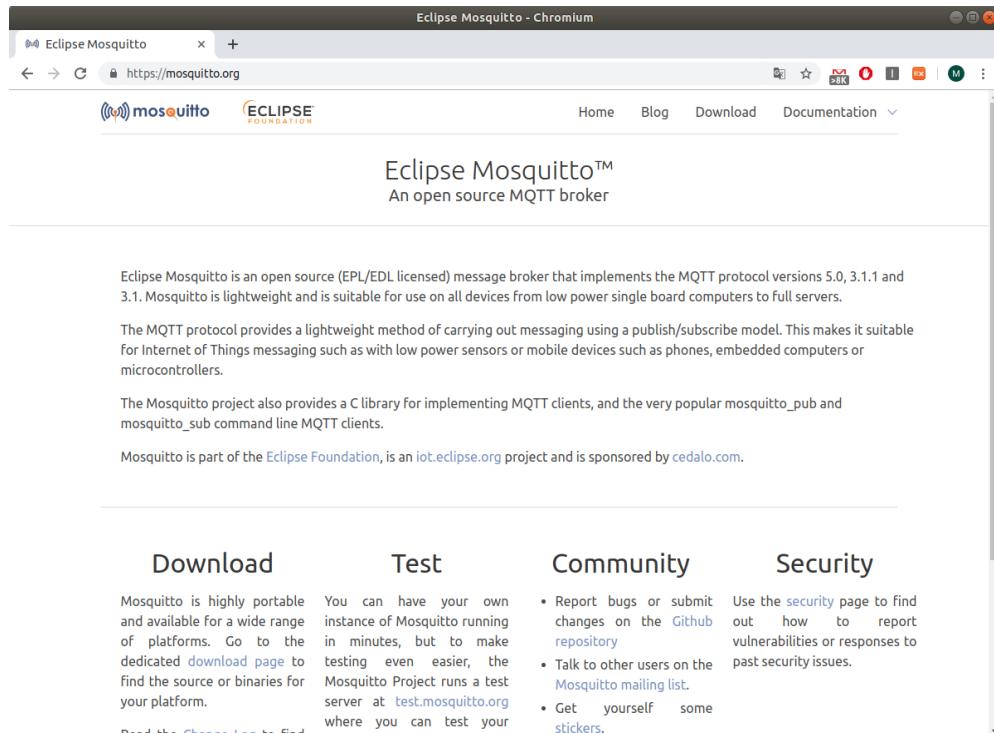
The screenshot shows a web browser window titled "LoRa App Server REST API" at the URL "localhost:8080/api". The page has a green header bar with the title and a "JWT TOKEN" input field. Below the header is a sub-header "LoRa App Server REST API" and a note about the usage of the API. The main content is a table listing various REST API endpoints for the ApplicationService:

ApplicationService		Show/Hide	List Operations	Expand Operations
GET	/api/applications	List lists the available applications.		
POST	/api/applications	Create creates the given application.		
PUT	/api/applications/{application.id}	Update updates the given application.		
GET	/api/applications/{application_id}/integrations	ListIntegrations lists all configured integrations.		
DELETE	/api/applications/{application_id}/integrations/http	DeleteIntegration deletes the HTTP application-integration.		
GET	/api/applications/{application_id}/integrations/http	GetHTTPIntegration returns the HTTP application-integration.		
DELETE	/api/applications/{application_id}/integrations/influxdb	DeleteInfluxDBIntegration deletes the InfluxDB application-integration.		
GET	/api/applications/{application_id}/integrations/influxdb	GetInfluxDBIntegration returns the InfluxDB application-integration.		
DELETE	/api/applications/{id}	Delete deletes the given application.		
GET	/api/applications/{id}	Get returns the requested application.		
POST	/api/applications/{integration.application_id}/integrations/http	CreateHTTPIntegration creates a HTTP application-integration.		
PUT	/api/applications/{integration.application_id}/integrations/http	UpdateHTTPIntegration updates the HTTP application-integration.		
POST	/api/applications/{integration.application_id}/integrations/influxdb	CreateInfluxDBIntegration create an InfluxDB application-integration.		

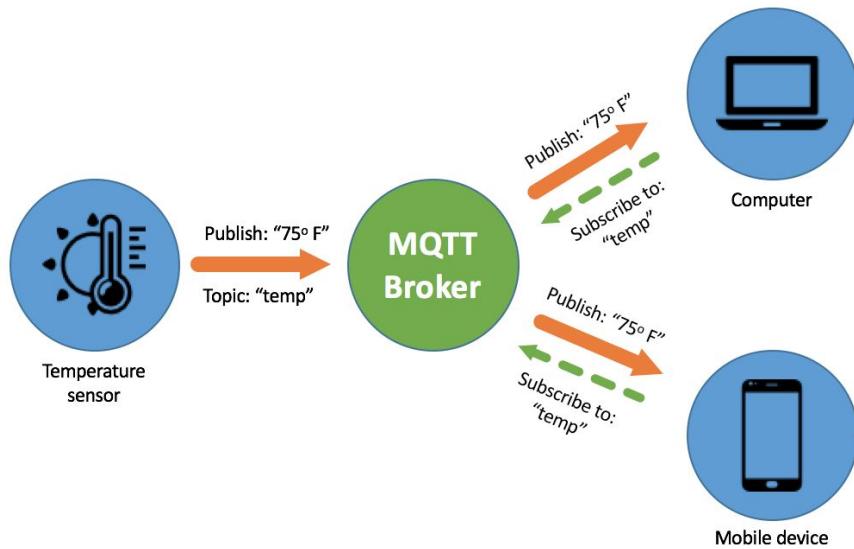
[그림 11.5] LoRa App Server에서 제공하는 REST API

4) MQTT - Let's use Mosquitto MQTT broker

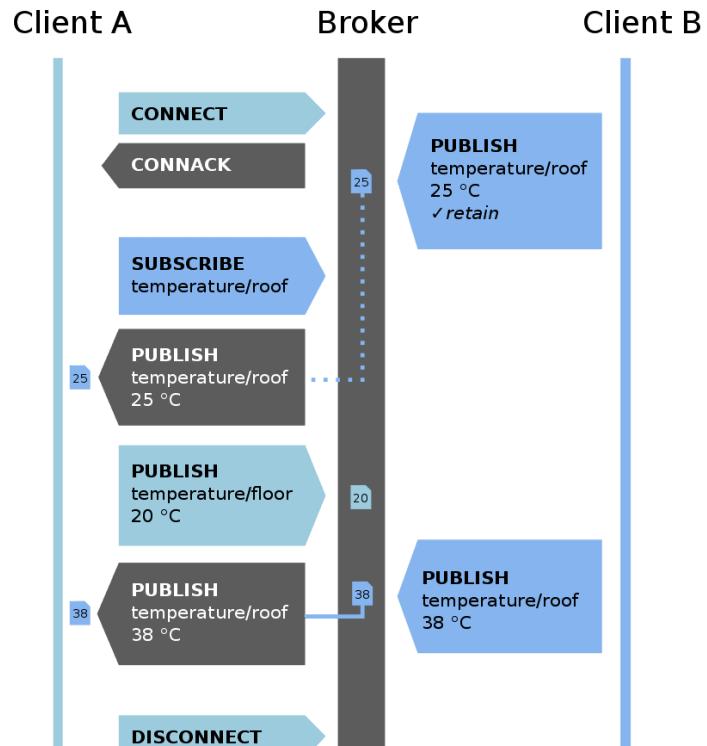
8장의 LoRa Server 설치 과정에는 **Mosquitto MQTT Broker**를 설치하는 부분이 이미 포함되어 있다. (개인적인 소견이긴 하지만) gRPC는 상대적으로 난이도가 있고, REST API는 융통성(?)이 다소 떨어진다고 판단되는 바, 앞으로 우리(2ip OLoRa viewer)는 MQTT 방식을 사용하여 접근하는 것이 맞을 듯 보인다. 하지만, 결론이 이렇다고 해서 gRPC, REST API를 사용해 보지도 않는 것은 문제가 있으니, 꼭 한번씩 사용해 보길 권한다.



[그림 11.6] Eclipse Mosquitto™ An open source MQTT broker 공식 site



[그림 11.7] MQTT broker



[그림 11.8] MQTT Protocol

<MQTT Mosquitto 사용법>

Mosquitto 사용법과 관련해서 몇가지 web site를 모아 보았다. 각각의 내용을 살펴 보기 바란다.

<http://iot.knu.ac.kr/tech/CPL-TR-16-02-MQTT.pdf>

<https://blog.neonkid.xyz/127>

https://wnsgml972.github.io/mqtt/mqtt_ubuntu-install.html

참고: Android, iOS, PHP 등에는 이미 MQTT client(publisher, subscriber)를 지원하는 함수(API)가 제공되고 있다.

본론으로 돌아와서, LoRa App Server MQTT integration과 관련해서는

<https://www.loraserver.io>의 아래 페이지를 참조하기 바란다.

<https://www.loraserver.io/lora-app-server/integrate/sending-receiving/mqtt/>

<Ubuntu 18.04 - LoRa Server가 설치되어 있는 PC>

\$ sudo mosquitto_sub -v -t "#" -h localhost -p 1883

- 이렇게 하면 모든 mqtt data를 수신할 수 있다.
- 추후 좀 더 다양한 사용 방법을 정리해 보도록 하자.

- `-v` - verbose output - includes the *topic* of the message
- `-t "#"` - any message. `"#"` is a multi-level wildcard. Other possibilities include:
 - `"gateway/#"` - any gateway messages
 - `"application/#"` - any application messages
- `-u` - The user to log into mosquitto with
- `-P` - The password for the user
- `-h` - The host to log in to
- `-p` - The mosquitto port

5) PostgreSQL

여기서 PostgreSQL database를 Integration interface로 보는 이유는 "모든 event가 PostgreSQL database에 쓰여 지고, application에서는 이 DB로부터 값을 읽어 갈 수 있기 때문"이다.

Event를 저장할 db table과 schema와 관련해서는 아래 page를 참조하기 바란다.

<https://www.loraserver.io/lora-app-server/integrate/sending-receiving/postgresql/>

12. ThingsBoard Integration

이번 장에서는 **ThingsBoard**와 **LoRa App Server**를 integration 시키는 작업을 해 보도록 하겠다. ThingsBoard는 우리가 앞으로 만들고자 하는 OLoRa viewer(desktop version) 쯤에 해당하는 녀석(?)이라고 보면 된다. 물론, 아래 내용을 보면 알겠지만, 훨씬 복잡하고 매우 잘 만들어져 있으니 비교 대상이라기보다는 배움의 대상이라고 보는게 적절할 것 같다 :)

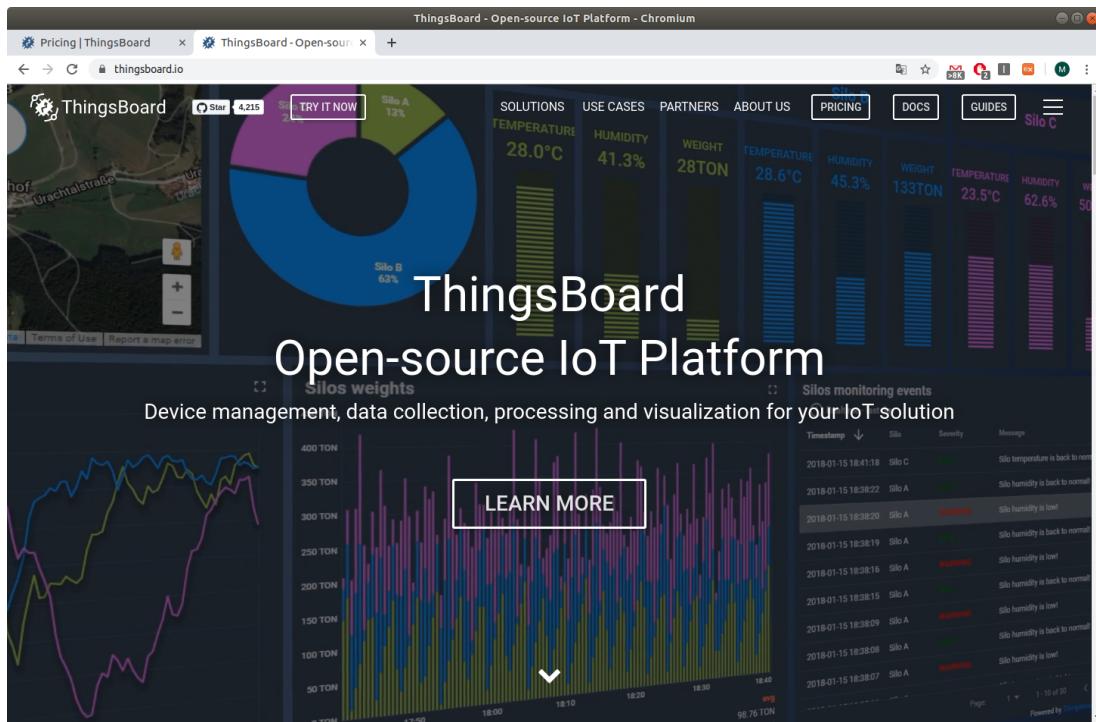
ThingsBoard ~ 잘 활용하면 LoRa 이외의 Project(예: NF 산소발생기 project)에서도 사용 가능할 것으로 보인다.

(나중에 정리한 것임) ThingsBoard == Our OLoRa !!!

1) ThingsBoard 개요

일전에 IoT 관련 서적에서 본 일이 있는데, 이렇게 LoRa App Server와 엑일 줄은 미쳐 생각 못했다.

<https://thingsboard.io>



[그림 12.1] ThingsBoard Home Page

What is ThingsBoard?

ThingsBoard is an open-source IoT platform that enables rapid development, management and scaling of IoT projects. Our goal is to provide the out-of-the-box IoT cloud or on-premises solution that will enable server-side infrastructure for your IoT applications.

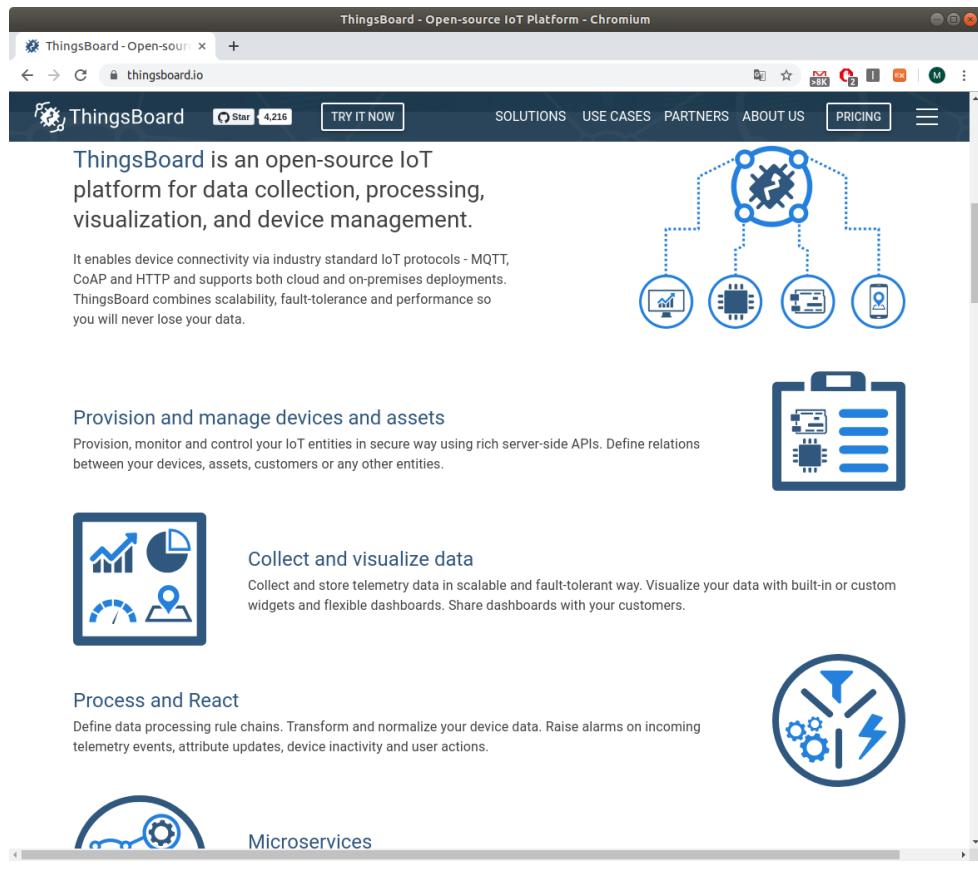
Features

With ThingsBoard, you are able to:

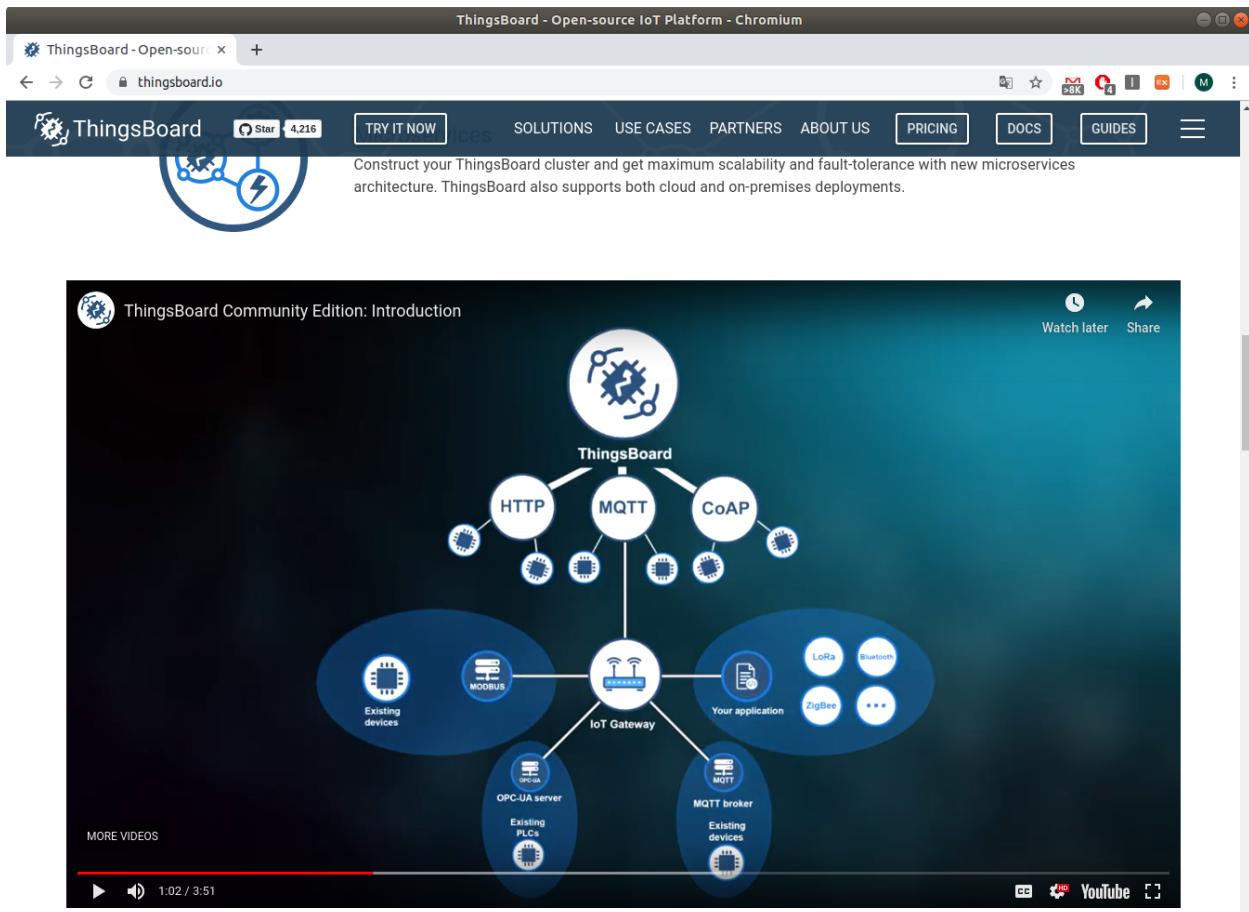
- Provision devices, assets and customers and define relations between them.
- Collect and visualize data from devices and assets.
- Analyze incoming telemetry and trigger alarms with complex event processing.
- Control your devices using remote procedure calls (RPC).
- Build work-flows based on device life-cycle event, REST API event, RPC request, etc
- Design dynamic and responsive dashboards and present device or asset telemetry and insights to your customers
- Enable use-case specific features using customizable rule chains.
- Push device data to other systems.
- Much more...

See [ThingsBoard features list](#) for more features and useful links to the specific feature documentation.

[그림 12.2] ThingsBoard 란(1) ?

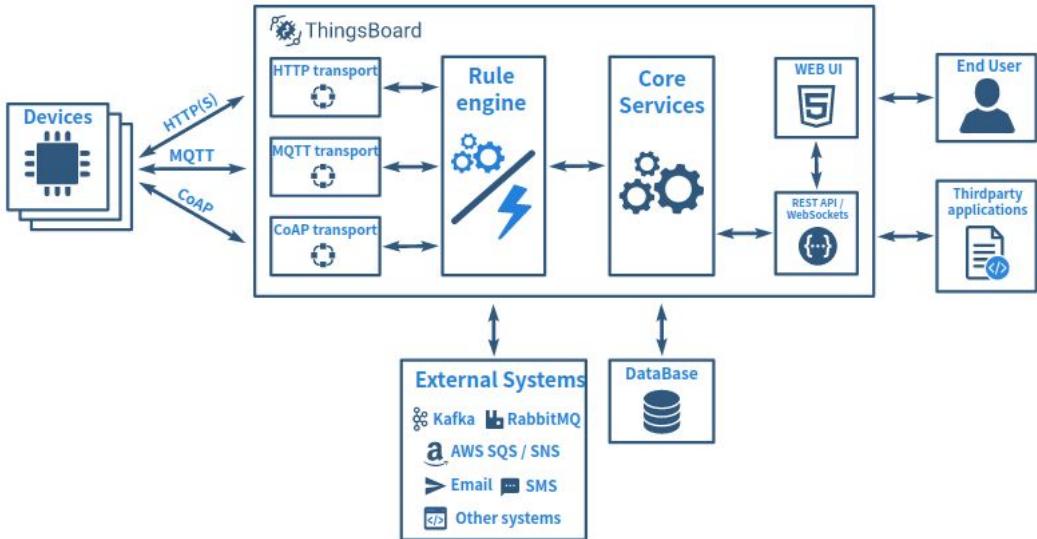


[그림 12.3] ThingsBoard 란(2) ?

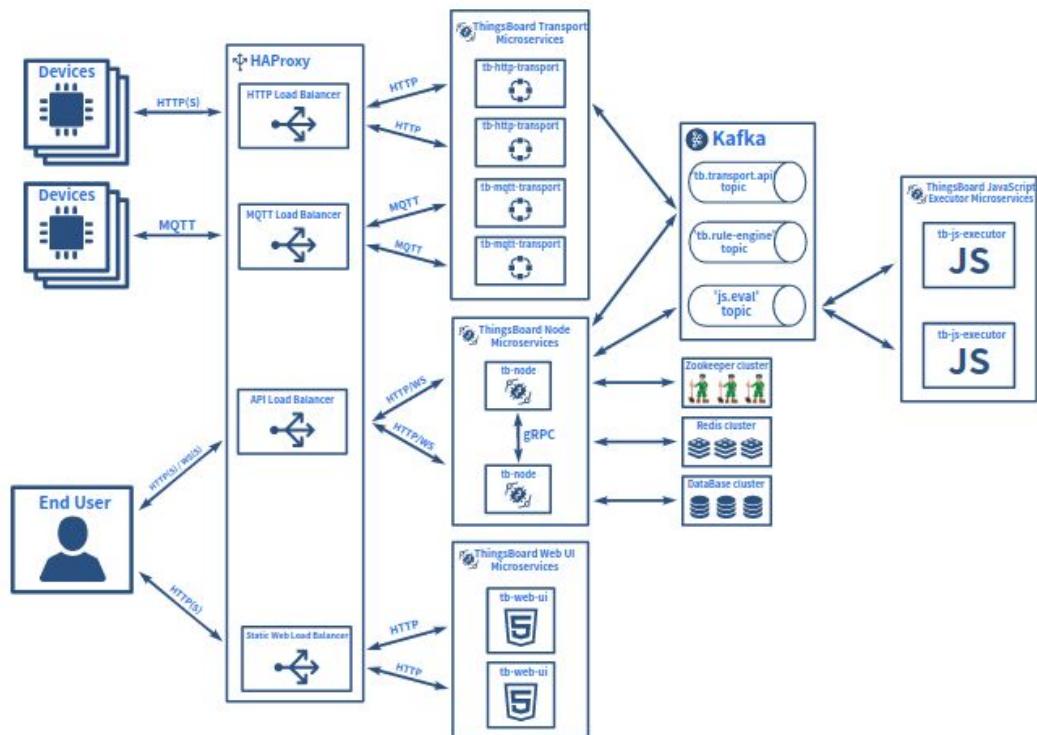


[그림 12.4] ThingsBoard 아키텍처

"ThingsBoard back-end is written in Java, but we also have some micro-services based on Node.js. ThingsBoard front-end is a SPA based on Angular JS framework."



[그림 12.5] ThingsBoard Monolithic 아키텍쳐



[그림 12.6] ThingsBoard Microservices 아키텍처

2) ThingsBoard 설치

각설하고 지금 부터는 ThingsBoard 설치로 바로 들어가 보겠다. 아래 site를 보니 Ubuntu 18.04 server를 기준으로 설치 과정이 설명되어 있고, myPC(Ubuntu 18.04 desktop)에는 이미 LoRaServer(불행히도 접속 방법 - <http://localhost:8080> - 이 ThingsBoard와 동일함)가 설치되어 있는 관계로, SPNBox-1800(Ubuntu 18.04 server 설치)에 ThingsBoard를 설치하기로 하겠다. 나중에는 AWS EC2에 설치해 보는 것도 좋은 대안이 될 것 같긴하다. **이 글을 읽고 있는 분들 중에 ThingsBoard를 설치해 보고 싶은 분들은 SPNBox-1600 or 1100 정도에 설치해 보면 좋을 것 같다.**

<https://thingsboard.io/docs/user-guide/install/ubuntu/?ubuntuThingsboardDatabase=postgresql>

참고 1: ThingsBoard는 Ubuntu는 물론이고, CentOS, Windows, Raspberry Pi3 등에서도 설치가 가능하다. 또한 Docker(가상 환경)를 이용한다면, apple 등에서도 설치 가능한 듯 보인다.

참고 2: ThingsBoard는 CE(Community Edition)과 PE(Professional Edition)가 존재하는데, 예상대로 PE는 license key를 요구한다. 따라서 여기서는 CE를 설치해 보기로 한다.

설치 절차는 매우 간단한데, 이를 요약해 보면 다음과 같다(아래 빨간색 부분만 설치해주면 기본적인 설치가 가능함). HTTPS 접속 관련 설정은 추후 진행해도 된다(NextCloud 설치 시 활용했던 Let's Encrypt 인증서 설치 과정을 밟아 주면 됨).

- **Prerequisites**
- **Step 1. Install Java 8 (OpenJDK)**
- **Step 2. ThingsBoard service installation**
- **Step 3. Configure ThingsBoard database**
- **Step 4. [Optional] Memory update for slow machines (1GB of RAM)**
- **Step 5. Run installation script**
- **Step 6. Start ThingsBoard service**
- **Post-installation steps**
- **Troubleshooting**

<Step 1> OpenJDK(Java 8) 설치하기

```
$ sudo apt update
```

```
$ sudo apt install openjdk-8-jdk
```

- Code를 대충 훑어 보니, Java를 기반으로 만들어진 듯 하다. 따라서 JDK를 설치해야 한다.
- Source code 위치: <https://github.com/thingsboard/thingsboard>

```
$ sudo update-alternatives --config java
```

There is only one alternative in link group java (providing /usr/bin/java):

/usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java

Nothing to configure.

```
$ java -version
```

openjdk version "1.8.0_222"

OpenJDK Runtime Environment (build 1.8.0_222-8u222-b10-1ubuntu1~18.04.1-b10)

OpenJDK 64-Bit Server VM (build 25.222-b10, mixed mode)

<Step 2> ThingsBoard service 설치하기

```
$ wget
```

<https://github.com/thingsboard/thingsboard/releases/download/v2.4.1/thingsboard-2.4.1.deb>

- Debian package를 내려 받는다.

```
--2019-09-20 01:02:48--
https://github.com/thingsboard/thingsboard/releases/download/v2.4.1/thingsboard-2.4.1.deb
Resolving github.com (github.com)... 52.78.231.108
Connecting to github.com (github.com)|52.78.231.108|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location:
https://github-production-release-asset-2e65be.s3.amazonaws.com/75277003/221da880-d644-11e9-8df3-b89d600f5b4d?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20190920%2Fsus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20190920T010248Z&X-Amz-Expires=300&X-Amz-Signature=dd11e9c208f9a2eac2afb658e256a6dc1ba8e16b0d41b150427f383127274c70&X-Amz-SignedHeaders=host&actor_id=0&response-content-disposition=attachment%3B%20filename%3Dthingsboard-2.4.1.deb&response-content-type=application%2Foctet-stream [following]
--2019-09-20 01:02:48--
https://github-production-release-asset-2e65be.s3.amazonaws.com/75277003/221da880-d644-11e9-8df3-b89d600f5b4d?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20190920%2Fsus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20190920T010248Z&X-Amz-Expires=300&X-Amz-Signature=dd11e9c208f9a2eac2afb658e256a6dc1ba8e16b0d41b150427f383127274c70&X-Amz-SignedHeaders=host
```

```
ost&actor_id=0&response-content-disposition=attachment%3B%20filename%3Dthingsboard-2.4.1.deb&resp
onse-content-type=application%2Foctet-stream

Resolving github-production-release-asset-2e65be.s3.amazonaws.com
(github-production-release-asset-2e65be.s3.amazonaws.com)... 52.216.80.128

Connecting to github-production-release-asset-2e65be.s3.amazonaws.com
(github-production-release-asset-2e65be.s3.amazonaws.com)|52.216.80.128|:443... connected.

HTTP request sent, awaiting response... 200 OK

Length: 116445540 (111M) [application/octet-stream]

Saving to: 'thingsboard-2.4.1.deb'

thingsboard-2.4.1.deb    100%[=====] 111.05M 7.70MB/s in 23s

2019-09-20 01:03:12 (4.93 MB/s) - 'thingsboard-2.4.1.deb' saved [116445540/116445540]
```

\$ sudo dpkg -i thingsboard-2.4.1.deb

→ Debian package를 설치한다.

```
Selecting previously unselected package thingsboard.

(Reading database ... 168432 files and directories currently installed.)

Preparing to unpack thingsboard-2.4.1.deb ...
Adding group `thingsboard' (GID 118) ...
Done.

Unpacking thingsboard (2.4.1-1) ...
Setting up thingsboard (2.4.1-1) ...
Processing triggers for systemd (237-3ubuntu10.29) ...
Processing triggers for ureadahead (0.100.0-20) ...
```

<Step 3> ThingsBoard DB 설치 및 설정(db table 생성)

\$ sudo apt-get update

→ 앞서 이미 수행했으므로 안해도 된다.

\$ sudo apt-get install postgresql postgresql-contrib

→ DB로 PostgreSQL을 설치한다. 요즘은 (mySQL이 상용 버전화 되면서)PostgreSQL이 나름 대세인듯 보인다.

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  linux-headers-4.15.0-60 linux-headers-4.15.0-60-generic linux-image-4.15.0-60-generic
  linux-modules-4.15.0-60-generic linux-modules-extra-4.15.0-60-generic
Use 'sudo apt autoremove' to remove them.

The following additional packages will be installed:
  libpq5 postgresql-10 postgresql-client-10 postgresql-client-common postgresql-common sysstat

Suggested packages:
  postgresql-doc locales-all postgresql-doc-10 libjson-perl isag

The following NEW packages will be installed:
  libpq5 postgresql postgresql-10 postgresql-client-10 postgresql-client-common postgresql-common
  postgresql-contrib sysstat

0 upgraded, 8 newly installed, 0 to remove and 113 not upgraded.

Need to get 5293 kB of archives.

After this operation, 20.9 MB of additional disk space will be used.

Do you want to continue? [Y/n] y

Get:1 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libpq5 amd64 10.10-0ubuntu0.18.04.1 [108 kB]
Get:2 http://archive.ubuntu.com/ubuntu bionic/main amd64 postgresql-client-common all 190 [29.5 kB]
Get:3 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 postgresql-client-10 amd64 10.10-0ubuntu0.18.04.1 [935 kB]
Get:4 http://archive.ubuntu.com/ubuntu bionic/main amd64 postgresql-common all 190 [157 kB]
Get:5 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 postgresql-10 amd64 10.10-0ubuntu0.18.04.1 [3758 kB]
Get:6 http://archive.ubuntu.com/ubuntu bionic/main amd64 postgresql all 10+190 [5784 B]
Get:7 http://archive.ubuntu.com/ubuntu bionic/main amd64 postgresql-contrib all 10+190 [5796 B]
Get:8 http://archive.ubuntu.com/ubuntu bionic/main amd64 sysstat amd64 11.6.1-1 [295 kB]

Fetched 5293 kB in 11s (470 kB/s)

Preconfiguring packages ...

Selecting previously unselected package libpq5:amd64.
(Reading database ... 168518 files and directories currently installed.)

Preparing to unpack .../0-libpq5_10.10-0ubuntu0.18.04.1_amd64.deb ...
Unpacking libpq5:amd64 (10.10-0ubuntu0.18.04.1) ...
Selecting previously unselected package postgresql-client-common.
Preparing to unpack .../1-postgresql-client-common_190_all.deb ...
Unpacking postgresql-client-common (190) ...
```

```
Selecting previously unselected package postgresql-client-10.  
Preparing to unpack .../2-postgresql-client-10_10.10-0ubuntu0.18.04.1_amd64.deb ...  
Unpacking postgresql-client-10 (10.10-0ubuntu0.18.04.1) ...  
Selecting previously unselected package postgresql-common.  
Preparing to unpack .../3-postgresql-common_190_all.deb ...  
Adding 'diversion of /usr/bin/pg_config to /usr/bin/pg_config.libpq-dev by postgresql-common'  
Unpacking postgresql-common (190) ...  
Selecting previously unselected package postgresql-10.  
Preparing to unpack .../4-postgresql-10_10.10-0ubuntu0.18.04.1_amd64.deb ...  
Unpacking postgresql-10 (10.10-0ubuntu0.18.04.1) ...  
Selecting previously unselected package postgresql.  
Preparing to unpack .../5-postgresql_10+190_all.deb ...  
Unpacking postgresql (10+190) ...  
Selecting previously unselected package postgresql-contrib.  
Preparing to unpack .../6-postgresql-contrib_10+190_all.deb ...  
Unpacking postgresql-contrib (10+190) ...  
Selecting previously unselected package sysstat.  
Preparing to unpack .../7-sysstat_11.6.1-1_amd64.deb ...  
Unpacking sysstat (11.6.1-1) ...  
Setting up sysstat (11.6.1-1) ...  
  
Creating config file /etc/default/sysstat with new version  
update-alternatives: using /usr/bin/sar.sysstat to provide /usr/bin/sar (sar) in auto mode  
Created symlink /etc/systemd/system/multi-user.target.wants/sysstat.service → /lib/systemd/system/sysstat.service.  
Processing triggers for ureadahead (0.100.0-20) ...  
Setting up libpq5:amd64 (10.10-0ubuntu0.18.04.1) ...  
Processing triggers for libc-bin (2.27-3ubuntu1) ...  
Setting up postgresql-client-common (190) ...  
Processing triggers for systemd (237-3ubuntu10.29) ...  
Setting up postgresql-common (190) ...  
Adding user postgres to group ssl-cert  
  
Creating config file /etc/postgresql-common/createcluster.conf with new version  
Building PostgreSQL dictionaries from installed myspell/hunspell packages...  
Removing obsolete dictionary files:  
Created symlink /etc/systemd/system/multi-user.target.wants/postgresql.service →  
/lib/systemd/system/postgresql.service.
```

```
Unsafe symlinks encountered in /var/run/postgresql, refusing.  
Unsafe symlinks encountered in /var/log/postgresql, refusing.  
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...  
Setting up postgresql-client-10 (10.10-0ubuntu0.18.04.1) ...  
update-alternatives: using /usr/share/postgresql/10/man/man1/psql.1.gz to provide /usr/share/man/man1/psql.1.gz  
(psql.1.gz) in auto mode  
Setting up postgresql-10 (10.10-0ubuntu0.18.04.1) ...  
Creating new PostgreSQL cluster 10/main ...  
/usr/lib/postgresql/10/bin/initdb -D /var/lib/postgresql/10/main --auth-local peer --auth-host md5  
The files belonging to this database system will be owned by user "postgres".  
This user must also own the server process.  
  
The database cluster will be initialized with locale "en_US.UTF-8".  
The default database encoding has accordingly been set to "UTF8".  
The default text search configuration will be set to "english".  
  
Data page checksums are disabled.  
  
fixing permissions on existing directory /var/lib/postgresql/10/main ... ok  
creating subdirectories ... ok  
selecting default max_connections ... 100  
selecting default shared_buffers ... 128MB  
selecting default timezone ... Etc/UTC  
selecting dynamic shared memory implementation ... posix  
creating configuration files ... ok  
running bootstrap script ... ok  
performing post-bootstrap initialization ... ok  
syncing data to disk ... ok  
  
Success. You can now start the database server using:  
  
/usr/lib/postgresql/10/bin/pg_ctl -D /var/lib/postgresql/10/main -l logfile start  
  
Ver Cluster Port Status Owner  Data directory      Log file  
10 main  5432 down  postgres /var/lib/postgresql/10/main /var/log/postgresql/postgresql-10-main.log  
update-alternatives: using /usr/share/postgresql/10/man/man1/postmaster.1.gz to provide  
/usr/share/man/man1/postmaster.1.gz (postmaster.1.gz) in auto mode
```



```
Setting up postgresql (10+190) ...
Setting up postgresql-contrib (10+190) ...
Processing triggers for systemd (237-3ubuntu10.29) ...
Processing triggers for ureadahead (0.100.0-20) ...
```

\$ sudo service postgresql start

→ Postgresql을 구동시키자.

\$ ps aux|grep postgresql

```
postgres 27372 0.0 0.3 318756 27632 ? S 01:05 0:00
/usr/lib/postgresql/10/bin/postgres -D /var/lib/postgresql/10/main -c
config_file=/etc/postgresql/10/main/postgresql.conf

spnbox 27523 0.0 0.0 14856 1044 pts/1 S+ 01:06 0:00 grep --color=auto postgresql
```

\$ sudo su - postgres

→ Postgres 사용자 패스워드 설정을 한다.

```
postgres@spnbox-c1037:~$ psql
psql (10.10 (Ubuntu 10.10-0ubuntu0.18.04.1))
Type "help" for help.
```

postgres=# \password

Enter new password: ← spnbox1234 (꼭 이 값을 넣어야 하는 것은 당연히 아님)

Enter it again: ← spnbox1234

postgres=# \q

postgres=# ← Press **Ctrl+D**

postgres@spnbox-c1037:~\$ logout

\$ psql -U postgres -d postgres -h 127.0.0.1 -W

→ ThingsBoard DB table을 생성한다.

Password for user postgres: ← spnbox1234

psql (10.10 (Ubuntu 10.10-0ubuntu0.18.04.1))

SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)

Type "help" for help.

```
postgres=# CREATE DATABASE thingsboard;
```

```
CREATE DATABASE
```

```
postgres=# \q
```

```
$ sudo vi /etc/thingsboard/conf/thingsboard.conf
```

→ thingsboard configuration을 수정한다. 아래 내용을 추가하자.

```
...
#added by michael@2019.09.20 --
# DB Configuration
export DATABASE_ENTITIES_TYPE=mysql
export DATABASE_TS_TYPE=mysql
export SPRING_JPA_DATABASE_PLATFORM=org.hibernate.dialect.PostgreSQLDialect
export SPRING_DRIVER_CLASS_NAME=org.postgresql.Driver
export SPRING_DATASOURCE_URL=jdbc:postgresql://localhost:5432/thingsboard
export SPRING_DATASOURCE_USERNAME=postgres
export SPRING_DATASOURCE_PASSWORD=spnbox1234
```

<Step 5> 설치 script 실행하기

```
$ sudo /usr/share/thingsboard/bin/install/install.sh --loadDemo
```

↳ (나중에 안 것이지만) --loadDemo를 빼고 설치해야 겠다. 넣고 설치할 경우, ThingsBoard 측에서 미리 등록해 둔 사용자 계정 정보를 통해서만(?) login 가능한 것 같다.

```
=====
:: ThingsBoard ::    (v2.4.1)
=====

Starting ThingsBoard Installation...
Installing DataBase schema for entities...
Installing SQL DataBase schema part: schema-entities.sql
Installing SQL DataBase schema indexes part: schema-entities-idx.sql
```

Installing DataBase schema for timeseries...

Installing SQL DataBase schema part: schema-ts.sql

Loading system data...

Loading demo data...

Installation finished successfully!

ThingsBoard installed successfully!

<Step 6> ThingsBoard 서비스 시작하기

\$ sudo service thingsboard start

→ ThingsBoard 서비스를 시작한다.

\$ ps aux|grep thingsboard

→ 아래 java로 실행되는 부분이 main인 것 같다.

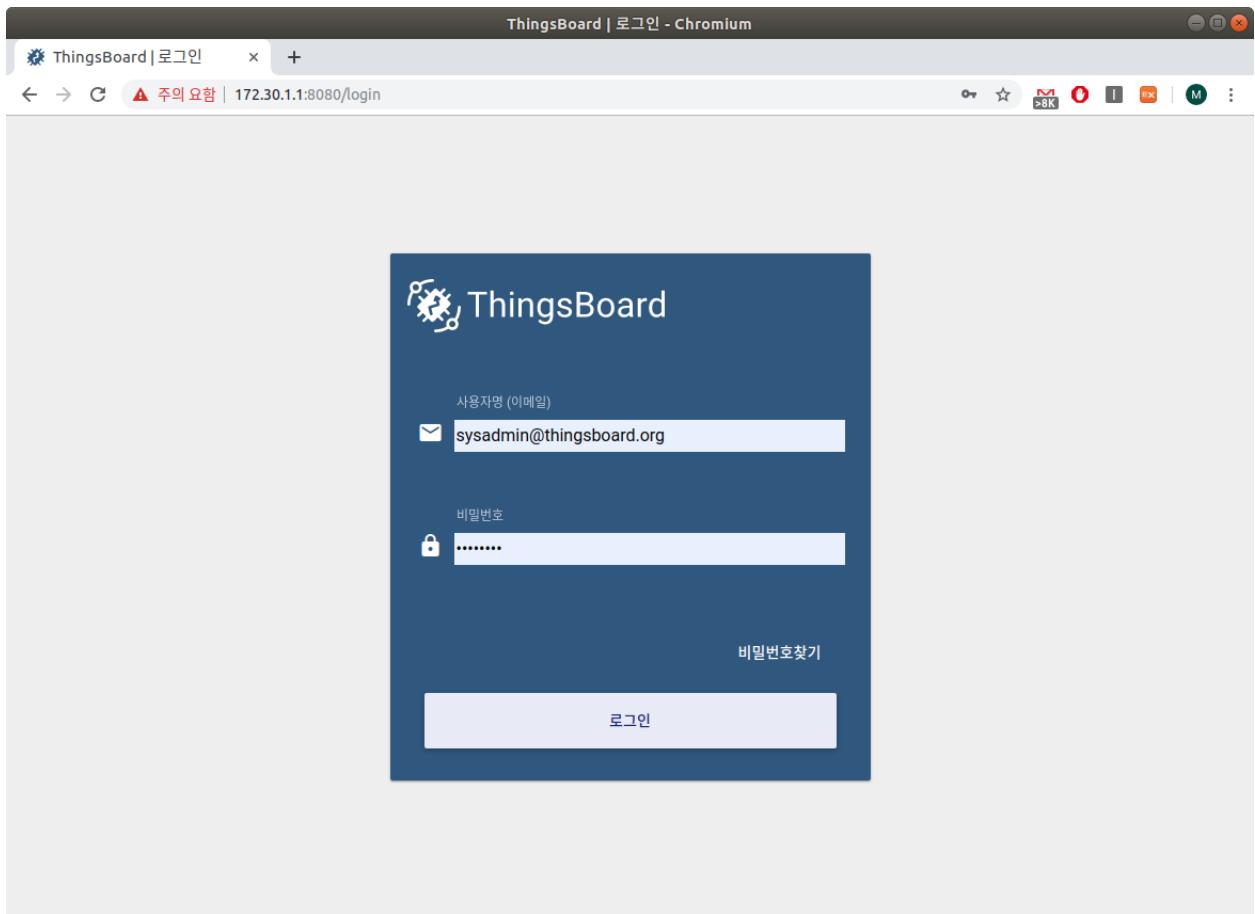
```
thingsb+ 27863 0.0 0.0 20180 3724 ? Ss 01:15 0:00 /bin/bash /usr/share/thingsboard/bin/thingsboard.jar
thingsb+ 27890 200 3.3 4502676 273532 ? SI 01:15 0:34 /usr/bin/java
-Dsun.misc.URLClassPath.disableJarChecking=true -Dplatform=deb -Dinstall.data_dir=/usr/share/thingsboard/data
-Xloggc:/var/log/thingsboard/gc.log -XX:+IgnoreUnrecognizedVMOptions -XX:+HeapDumpOnOutOfMemoryError
-XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:+PrintHeapAtGC -XX:+PrintTenuringDistribution
-XX:+PrintGCApplicationStoppedTime -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=10 -XX:GCLogFileSize=10M
-XX:-UseBiasedLocking -XX:+UseTLAB -XX:+ResizeTLAB -XX:+PerfDisableSharedMem -XX:+UseCondCardMark
-XX:CMSWaitDuration=10000 -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled
-XX:+CMSParallelInitialMarkEnabled -XX:+CMSEdenChunksRecordAlways -XX:CMSInitiatingOccupancyFraction=75
-XX:+UseCMSInitiatingOccupancyOnly -jar /usr/share/thingsboard/bin/thingsboard.jar

postgres 27915 1.0 0.1 320124 15132 ? Rs 01:15 0:00 postgres: 10/main: postgres thingsboard 127.0.0.1(34550)
BIND

postgres 27919 0.0 0.1 319580 10244 ? Ss 01:15 0:00 postgres: 10/main: postgres thingsboard 127.0.0.1(34552) idle
postgres 27920 0.0 0.1 319580 10244 ? Ss 01:15 0:00 postgres: 10/main: postgres thingsboard 127.0.0.1(34554) idle
postgres 27921 0.0 0.1 319580 10244 ? Ss 01:15 0:00 postgres: 10/main: postgres thingsboard 127.0.0.1(34556) idle
...
```

자 이제 모든 (기본) 설치가 마무리 되었으니, ThingsBoard WebUI를 띄워 볼 차례이다.
SPNBox-1800의 WAN ip가 172.30.1.1이므로, 아래와 같이 접속해 본다.

<http://172.30.1.1:8080/>

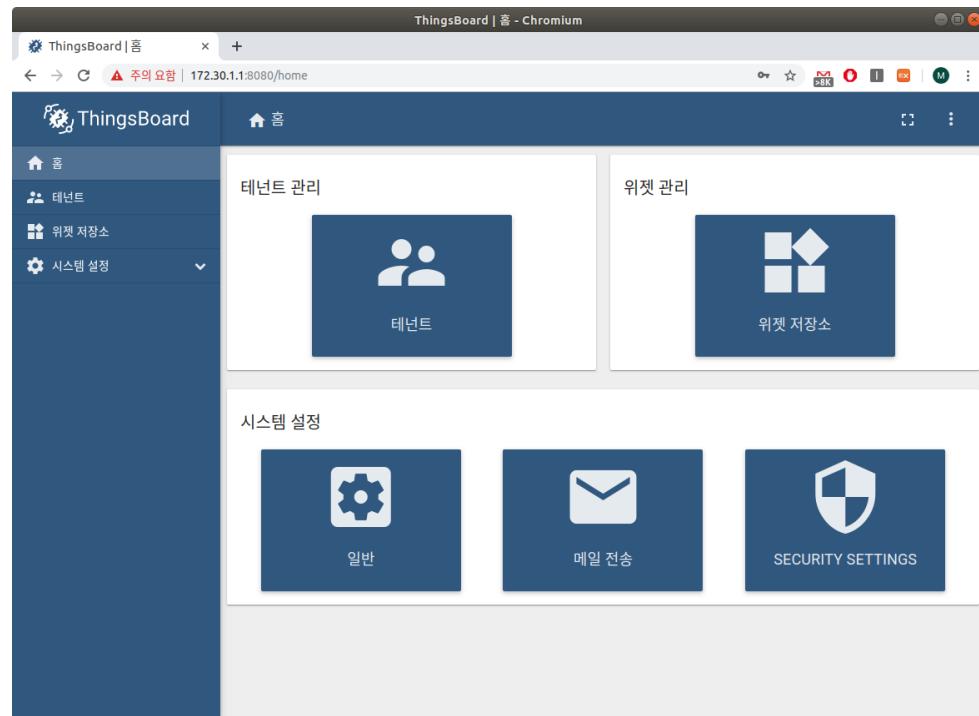


[그림 12.7] ThingsBoard WebUI 화면(1) - login 화면

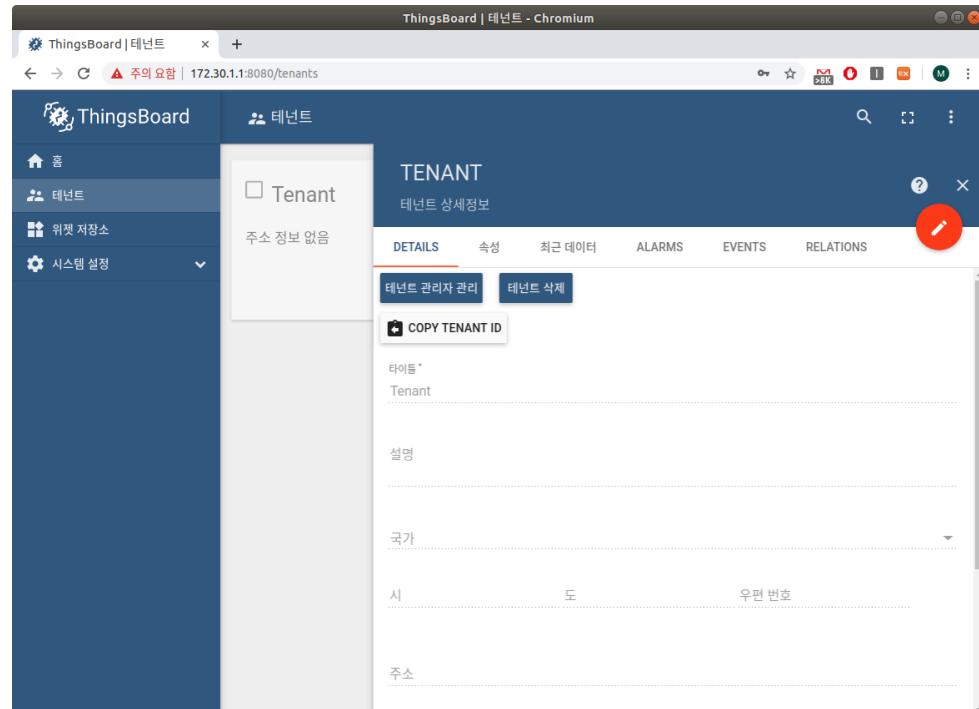
Id, password를 입력해야 하는데, <Step5>에서 --loadDemo option을 주어 설치한 탓에 아래 id/passwd를 사용하여 로긴해 볼 수 밖에 없다. 로긴 후, 신규 id를 등록할 수 있는지는 확인해 볼 일이다. 위의 그림에서는 sysadmin으로 로긴해 보았다.

- System Administrator: sysadmin@thingsboard.org / sysadmin
- Tenant Administrator: tenant@thingsboard.org / tenant
- Customer User: customer@thingsboard.org / customer

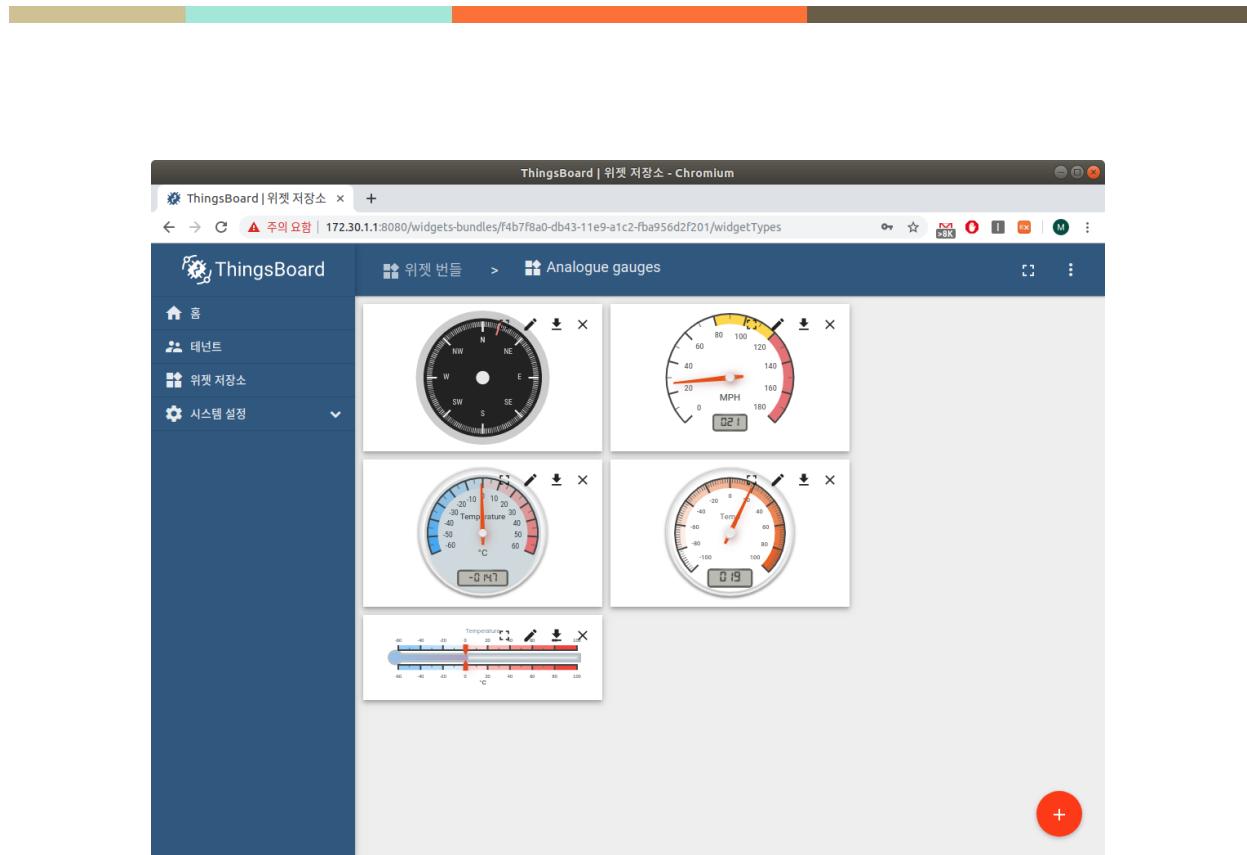
와우~ 테넌트 관리, 위젯 관리, 시스템 설정 화면이 보이고, 위젯 관리로 들어가 보면 매우 다양한 위젯이 보인다. :)



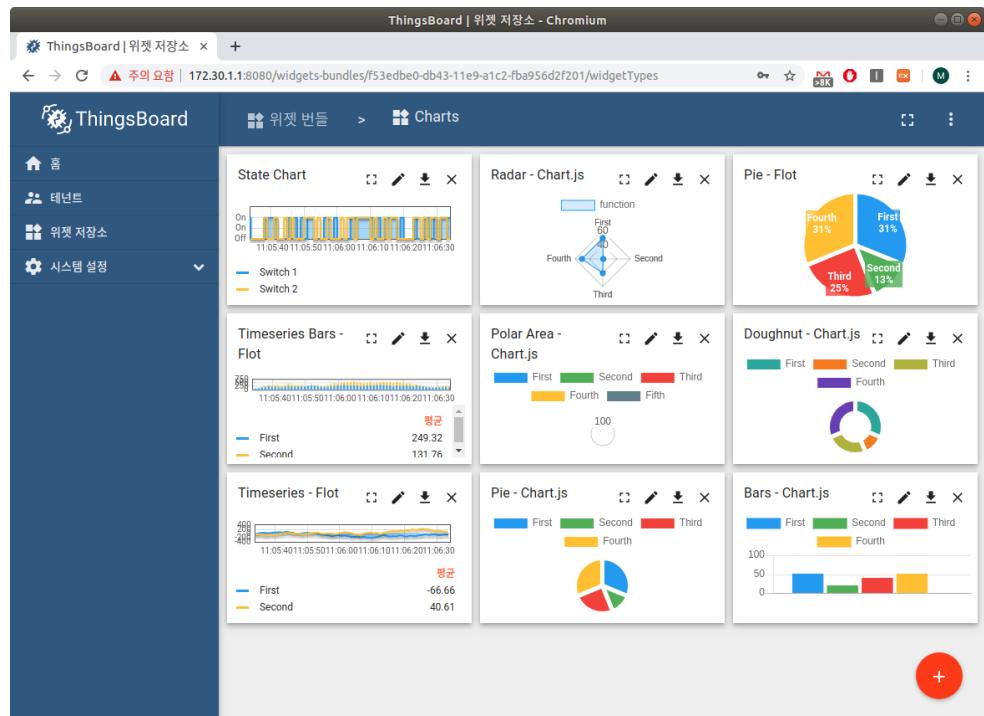
[그림 12.8] ThingsBoard WebUI 화면(2) - main 화면



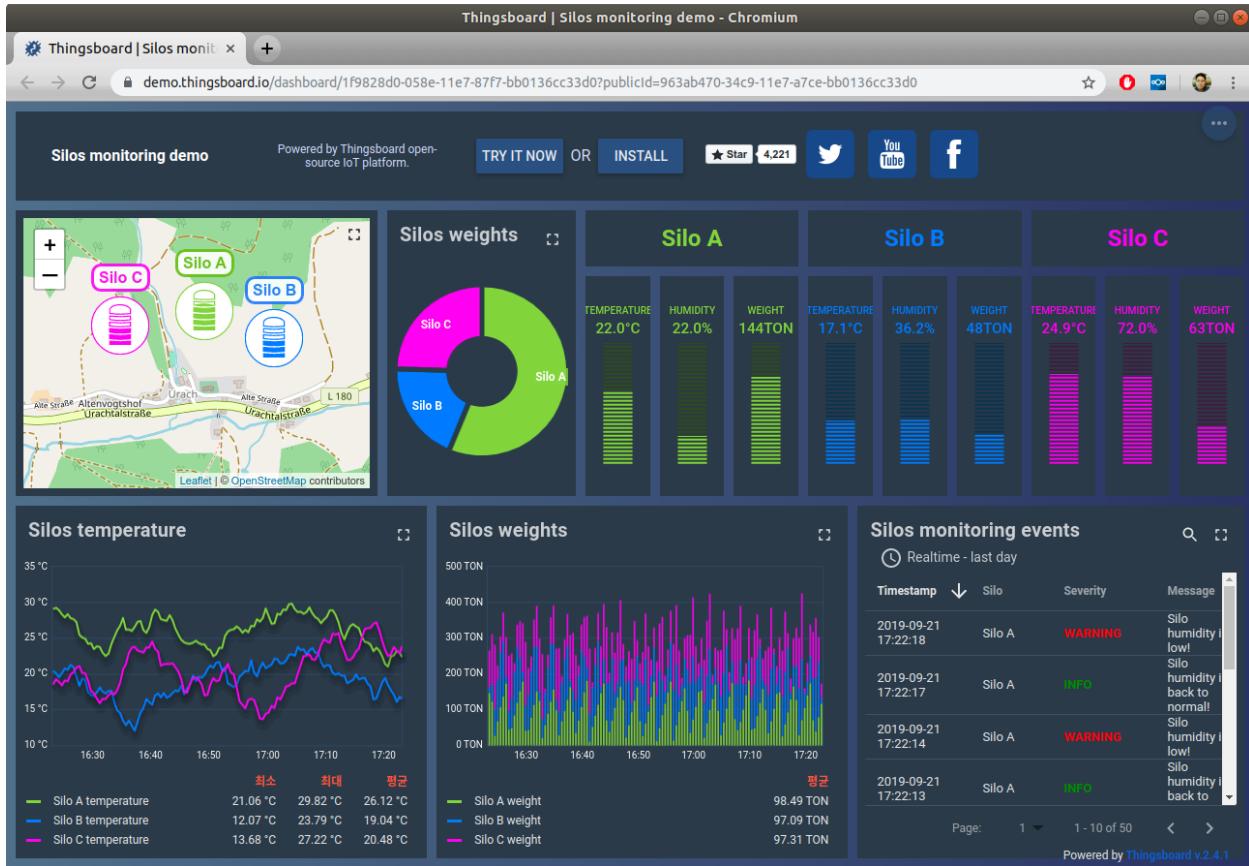
[그림 12.9] ThingsBoard WebUI 화면(3) - 테넌트 설정 화면



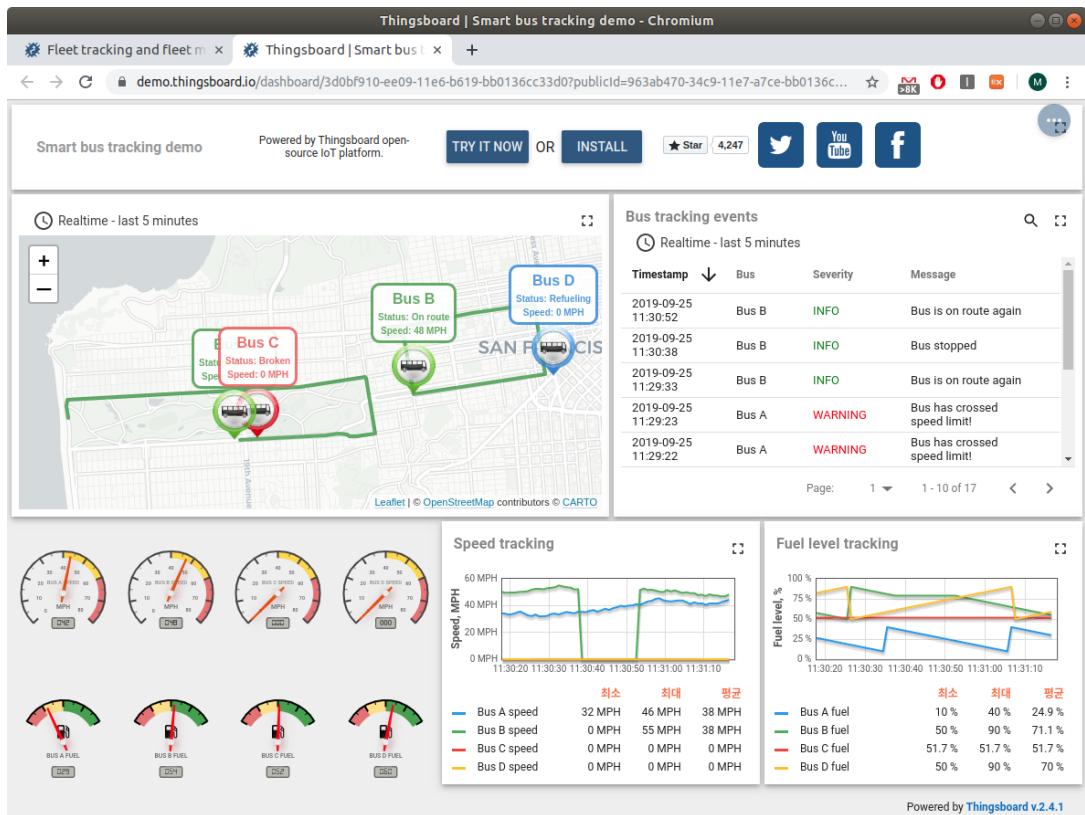
[그림 12.10] ThingsBoard WebUI 화면(4) - 위젯(1)



[그림 12.11] ThingsBoard WebUI 화면(5) - 위젯(2)



[그림 12.12] ThingsBoard를 이용한 Smart Farm Demo Site



[그림 12.13] ThingsBoard를 이용한 Bus Tracking Demo Site

근데, 어찌 설정해야 하는지 사용법을 좀 익힐 필요가 있겠다.

우선 아래 thingsboard page의 video 동영상을 면밀히 살펴 볼 필요가 있다.

<https://thingsboard.io/docs/getting-started-guides/helloworld/#video-tutorial>

또한 아래 site 내용도 함께 참조해 보자. 한글로 쉽게 잘 설명되어 있다. :)

<https://embedscope.com/112>

전체적인 설정 과정을 (아주 간단히) 요약해 보면 다음과 같다.

<정리 1>

- 1) sysadmin으로 login
- 2) tenant 추가
ex) michael@2ipco.com/*********
- 3) customer 추가
ex) NF

- 4) asset 추가 후, customer NF에 할당
 5) device 추가 후, customer NF에 할당
 6) dashboard 추가 후, customer NF에 할당
 7) Rule chain 관련(= Rule Nodes + 그들간의 관계 설정)
 → rule chain은 root rule chain과 사용자 정의 rule chain으로 구분
 → 상호간에 연결하여 사용하면 됨.
 → 즉, root rule chain ⇒ 사용자 정의 rule chain or 사용자 정의 rule chain ⇒ root rule chain 형태로 운용 가능
 → 사용자 rule chain에는 내가 원하는 조건 설정(추가) 가능
 8) REST API 이용하여 mobile app(2ip)과 연결 가능함.
 → 추후 mobile app 만들고 ThingsBoard와 연결해 보자.

<여기서 잠깐 ! - Rule Engine에 관하여>

- ThingsBoard를 강력하게 만들어 주는 기능
- [그림 11.5] ThingsBoard Monolithic 아키텍쳐 내용 중 Rule Engine 부분 확인

"**ThingsBoard Rule Engine** is a highly customizable and configurable system for complex event processing. With rule engine you are able **to filter, enrich and transform incoming messages originated by IoT devices** and related assets. You are also able **to trigger various actions, for example, notifications or communication with external systems.**"

⇒ Rule Engine이란 IoT Device로부터 들어오는 message(data)를 가공(filter, enrich, transform)해서 내보내는 역할을 담당하는 것으로, 가공된 message를 내보낼 때는 alarm, mail, SNS 등 특정 action을 수행할 수 있다. 또한, 가공과정에는 javascript를 사용하여 보다 고차원(?)적인 조건을 걸 수 있도록 설계되어 있다.

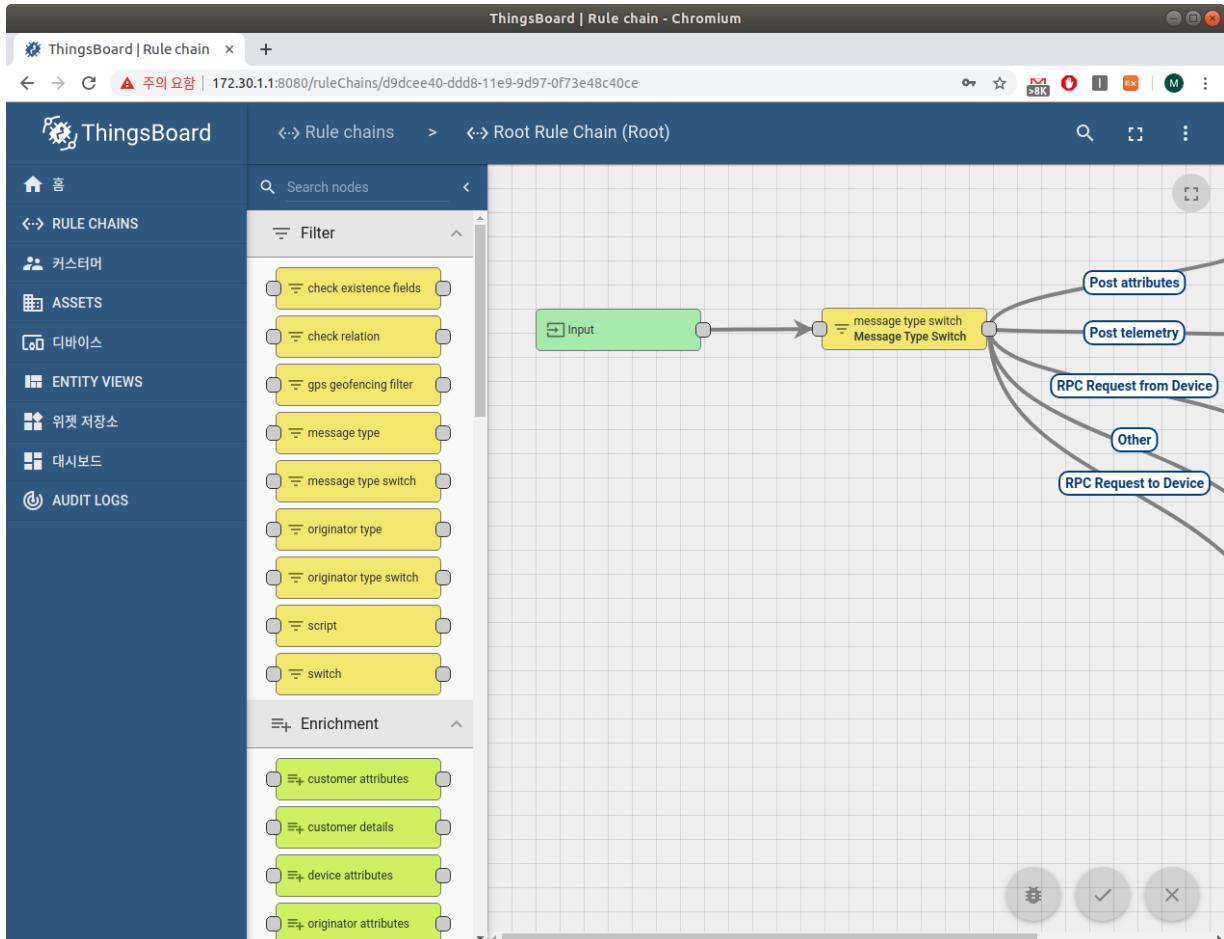
<https://thingsboard.io/docs/user-guide/rule-engine-2-0/overview/>

<용어 정의>

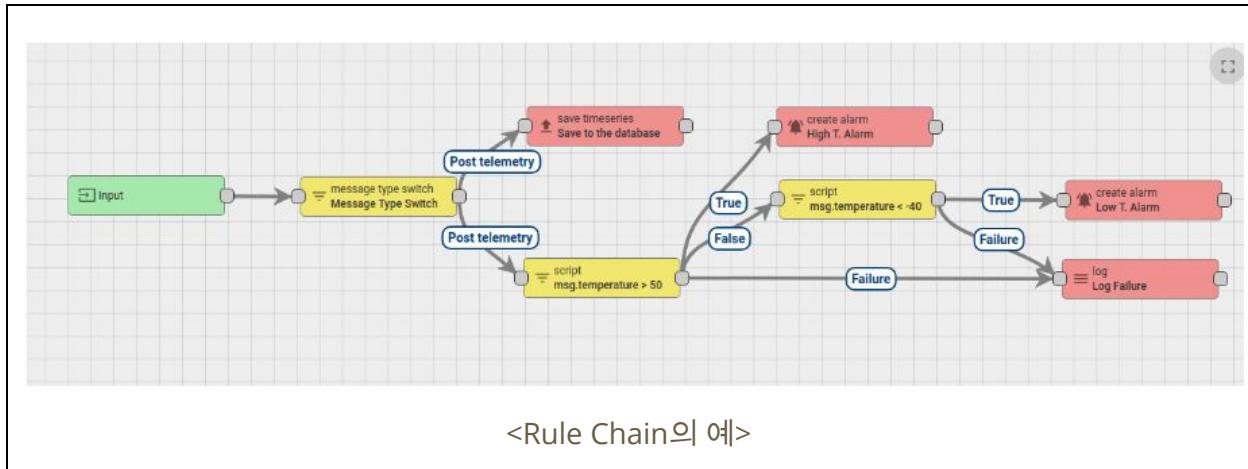
Rule Node: Rule Engine의 기본 구성요소로, 하나의 incoming message(입력 메시지)를 처리하고, 한개 이상의 outgoing message(출력 메시지)를 만들어 내는 역할을 담당한다. Rule Node type으로는 Filter Nodes, Enrichment Nodes, Transformation Nodes, Action Nodes, External Nodes 등 총 5가지가 있다.

Rule Chain: 여러개의 Rule Node를 정의하고, 이들을 서로 묶어(relation 설정) 놓은 것을 말한다.

참고: ThingsBoard를 설치하면 **Root Rule Chain**이 기 포함되어 있으며, 사용자는 자유롭게 **Rule Chain**을 추가해서 Root Rule Chain과 연결(앞, 뒤 상관 없음)하여 사용할 수 있다. 이 부분이 Rule Engine의 핵심적인 내용이다.



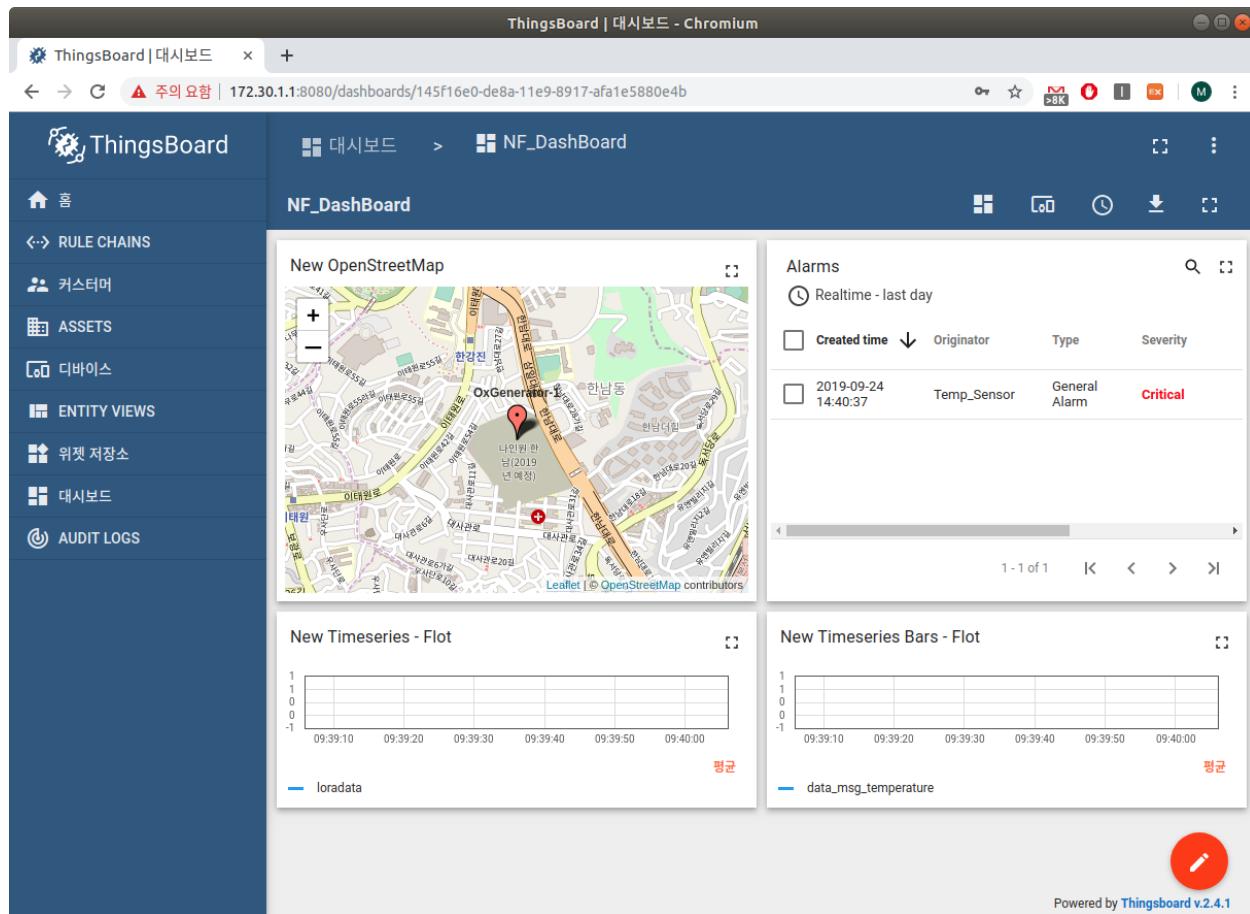
<Rule Nodes의 예 - 좌측 중앙>



위의 내용을 참조하여 (가칭)NF용 dashboard를 간단하게 만들어 보았다(만드는 과정은 생략).

<http://172.30.1.1:8080>

(Id/passwd: michael@2ipco.com/******/)



[그림 12.14] NF project용 dashboard 생성 예

3) ThingsBoard ↔ LoRa App Server 연동

아래 LoRa Server page에 ThingsBoard와 연동하는 방법이 설명되어 있다. 지금부터는 이 내용을 토대로 LoRa App Server와 ThingsBoard를 연결시켜 보도록 하겠다. **이 부분이 본 문서를 작성하는 또 다른 중요한 이유가 되겠다.**

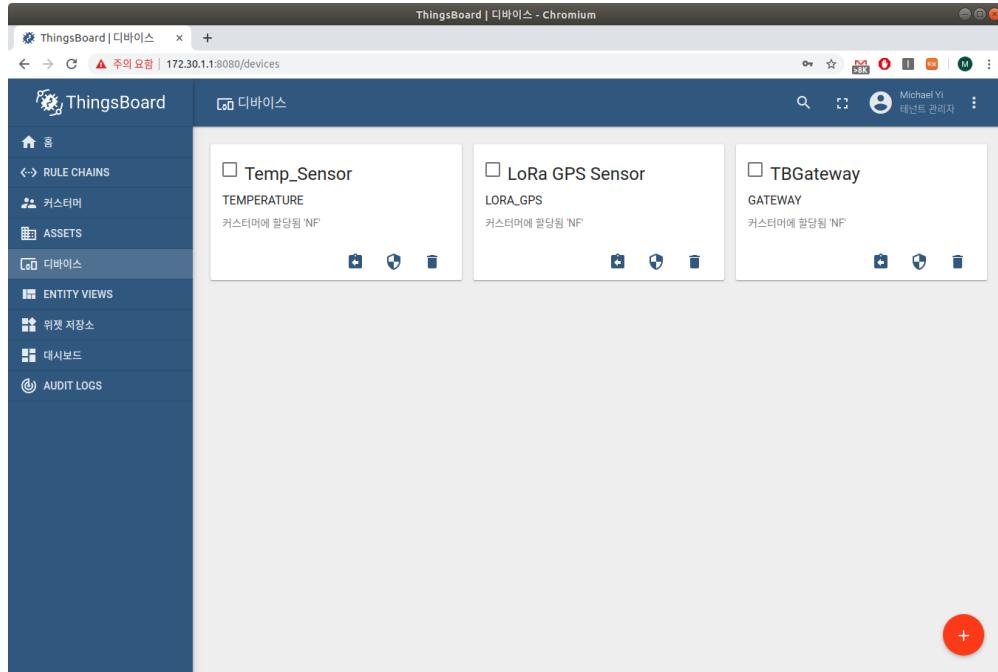
<https://www.loraserver.io/guides/thingsboard/>

<https://www.loraserver.io/lora-app-server/integrate/sending-receiving/thingsboard/>

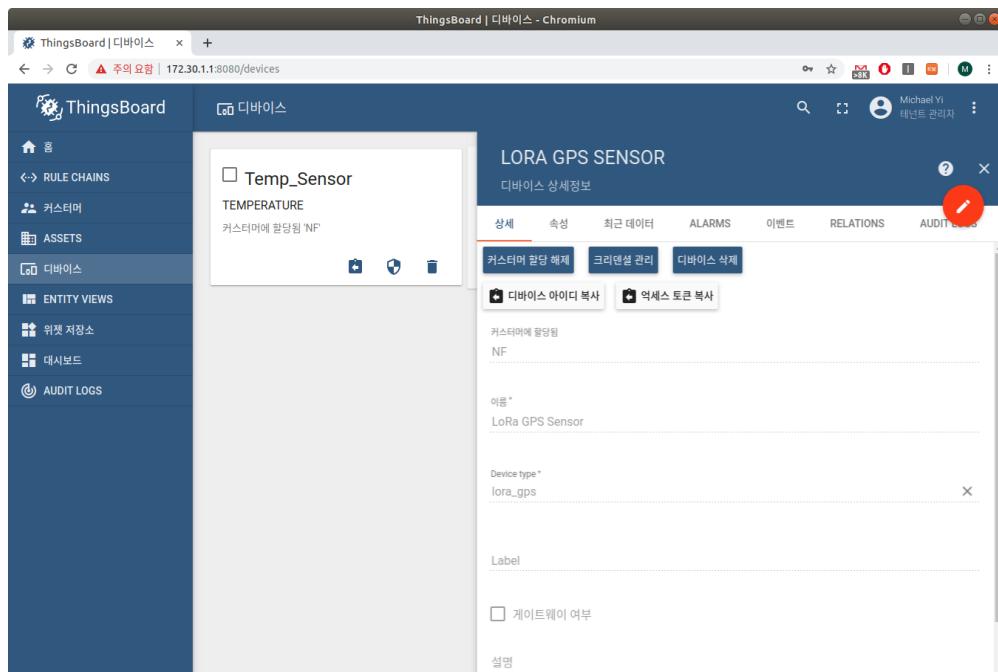
화면을 캡쳐하고, 세부적인 부분을 정리하는 것은 매우 시간이 많이 소요되는 일이다. 따라서 아래 내용에는 모든 것을 일일히 정리하기보다는, 중요하다고 생각되는 부분만이 정리되어 있으니, 이를 감안하고 읽어 주기 바란다.

<Step 1 - ThingsBoard Device 추가>

→ LoRa GPS Sensor라는 이름의 ThingsBoard 디바이스를 하나 추가하자.



[그림 12.15] LoRa GPS Sensor 디바이스 추가(1)



[그림 12.16] LoRa GPS Sensor 디바이스 추가(2)

<Step 2 - CODEC 설정 변경>

- Device-profiles / rak7200(device 선택) / CODEC ⇒ Payload codec 선택
- [결론부터 얘기하자면] 여기가 문제다. 결국 이 부분을 해결해야만 ThingsBoard 화면에 결과가 정상 출력될 것인데 ...
- (나중에 정리한 것임) 이 설정을 Cayenne LPP로 변경해야만 한다.

```

6 //return [];
7 //var msg = { temperature: 55};
8 // return { msg: bytes };
9 for (var i = 0; i < bytes.length; ++i) {
10 myDataString += String.fromCharCode(bytes[i]);
11 }
12 //var values = decodedString.split(",")
13 var values = myDataString.split(",")
14 return {"temperature": values[0], "pressure": values[1]};
15 }

The function must have the signature function Decode(fPort, bytes) and must return an object. LoRa App Server will convert this object to JSON.

```

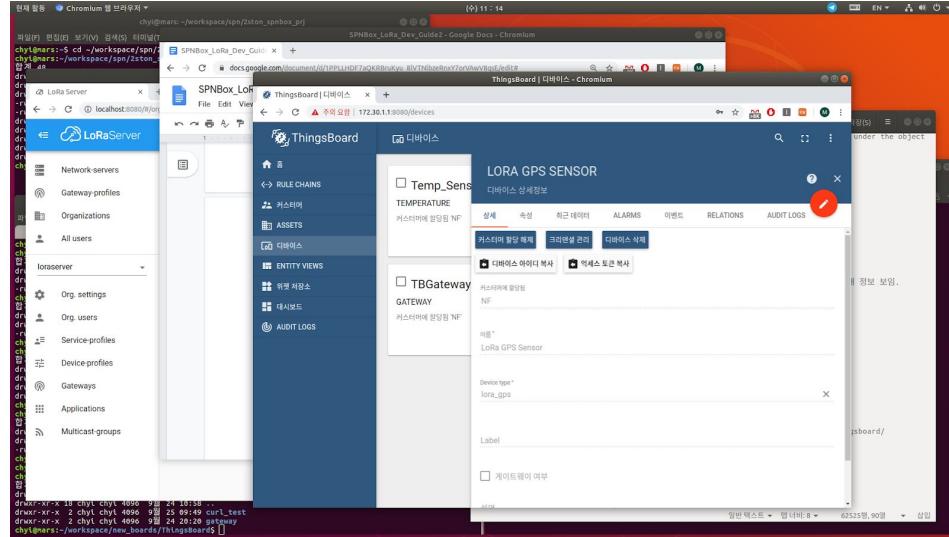
[그림 12.17] LoRa App Server CODEC 설정 화면

<참고 사항>

- 1) thingsboard는 json format으로 data가 들어오기를 원한다.
- 2) lora app server는 whole lorawan datapacket을 통째로 보내는 듯 보인다.
- 3) Lora app server에서 thingsboard로 전달되는 packet은 attribute(속성) + telemetry(원격 측정 data) 형태로 구성되어 있다.

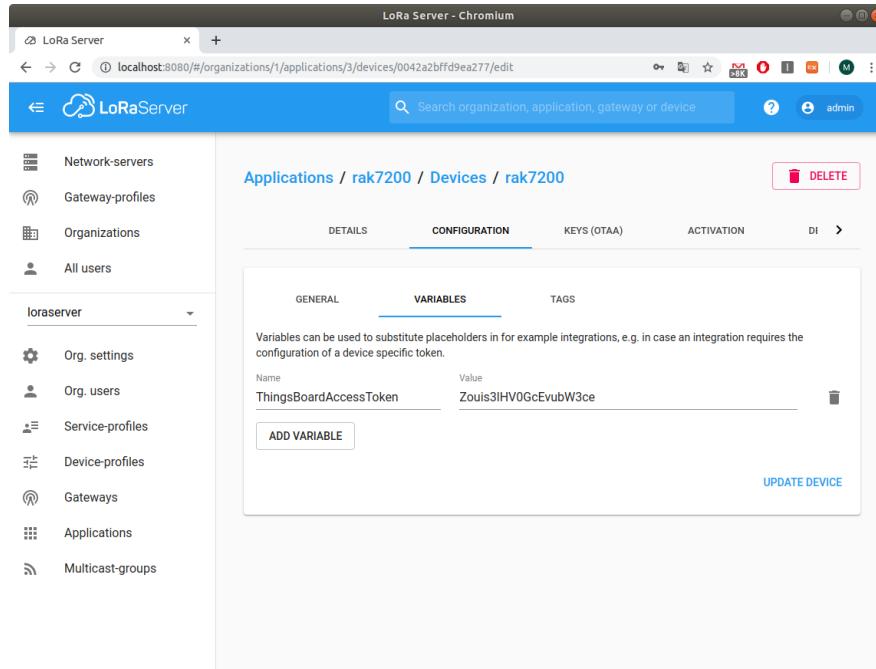
<Step 3 - ThingsBoard Device Token 복사>

→ [ThingsBoard] 디바이스 / LORA GPS SENSOR -> 액세스 토큰 복사



[그림 12.18] ThingsBoard Device Token 복사(from) - 액세스 토큰 복사 버튼 선택해야 함

→ [LoRa App Server] Applications / rak7200 / Devices / rak7200 -> CONFIGURATION -> VARIABLES

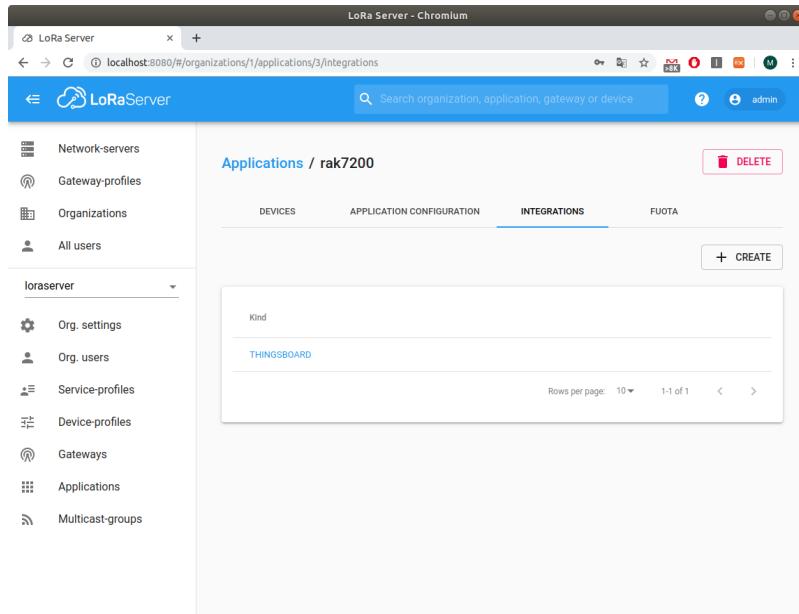


[그림 12.19] ThingsBoard Device Token 복사(to) - Value 필드

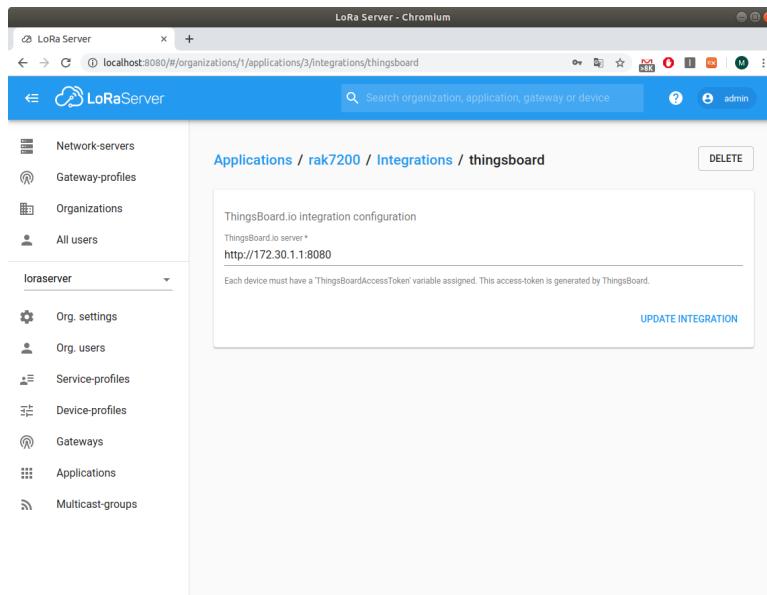
참고: Device Token은 ThingsBoard 디바이스에서 획득하는 것으로, 위의 Value 필드에 복사하여 사용하도록 한다.

<Step 4 - ThingsBoard 서버 ip 입력>

- Applications / rak7200 -> INTEGRATIONS -> Kind(THINGSBOARD) ->
http://172.30.1.1:8080



[그림 12.20] Integrations / Kind 설정 화면 - THINGSBOARD 선택



[그림 12.21] ThingsBoard IP 주소 입력

<Step 5 - ThingsBoard 디바이스 정보 확인>

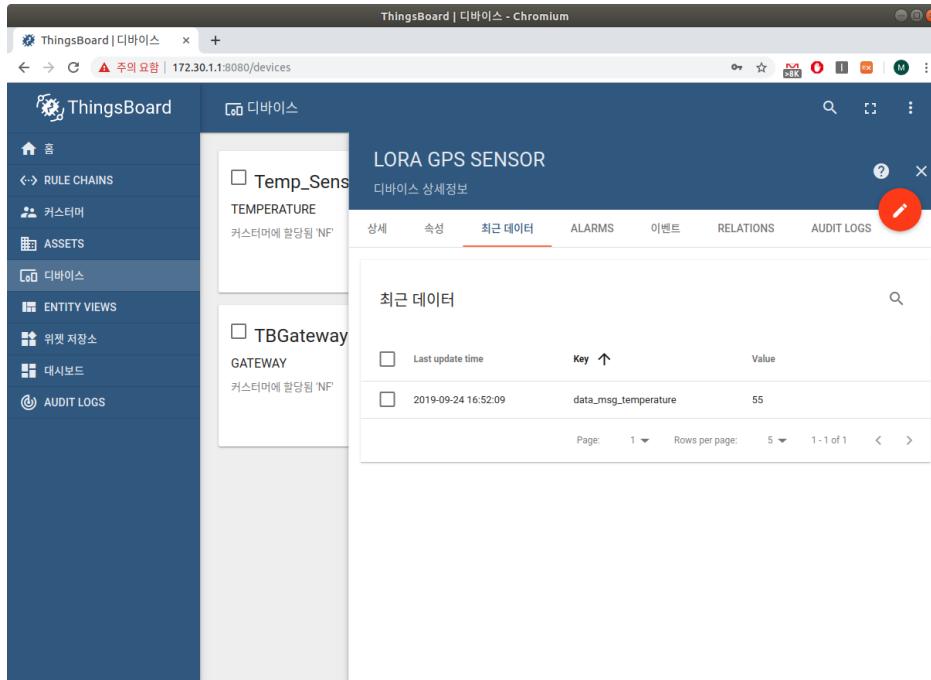
- LoRa App Server로 부터 data가 정상적으로 올라오는지를 확인해 보자.
- Device(LORA GPS Sensor)의 속성을 살펴 보니, 뭔가 없던 내용이 보인다.

속성	최근 데이터	ALARMS	이벤트	RELATIONS	AUDIT LOGS
디바이스 속성 범위					
클라이언트 속성					
Last update time	Key ↑	Value			
2019-09-25 09:56:00	application_id	3			
2019-09-25 09:56:00	application_name	rak7200			
2019-09-25 09:56:00	dev_eui	0042a2bfdf9ea277			
2019-09-25 09:56:00	device_name	rak7200			

[그림 12.22] ThingsBoard 디바이스(LORA GPS SENSOR) 속성 확인

참고: 위 내용은 LoRa packet 중 attribute에 해당하는 부분이다.

다음으로 LORA GPS Sensor 디바이스의 최근 데이터(Recent Data)를 확인해 보자. 여기에 LoRa GPS tracker 정보(telemetry)가 올라와야 하는데, 이것이 현재 안 나온다. 왜 일까? 문제는 <Step 2 - CODEC 설정 변경>과 연관이 있는 것 같다. 이 부분을 해결해 보자.



[그림 12.23] ThingsBoard 디바이스(LORA GPS SENSOR) 최신 데이터 확인

참고: 위 그림에 마치 data가 한개 올라와 있는 것처럼 보이는데, 이는 <Step 2>에서 테스트를 위해 script를 수정해서 강제로 packet을 내 보냈기 때문이다(무시하기 바람).

이 문제(telemetry data가 안 도달하는 문제)를 해결하기 위해, 아래 몇가지 site 내용을 참조해 보도록 하자.

<https://forum.loraserver.io/t/custom-payload-codec/1484/3>

 brohaar Jun '18

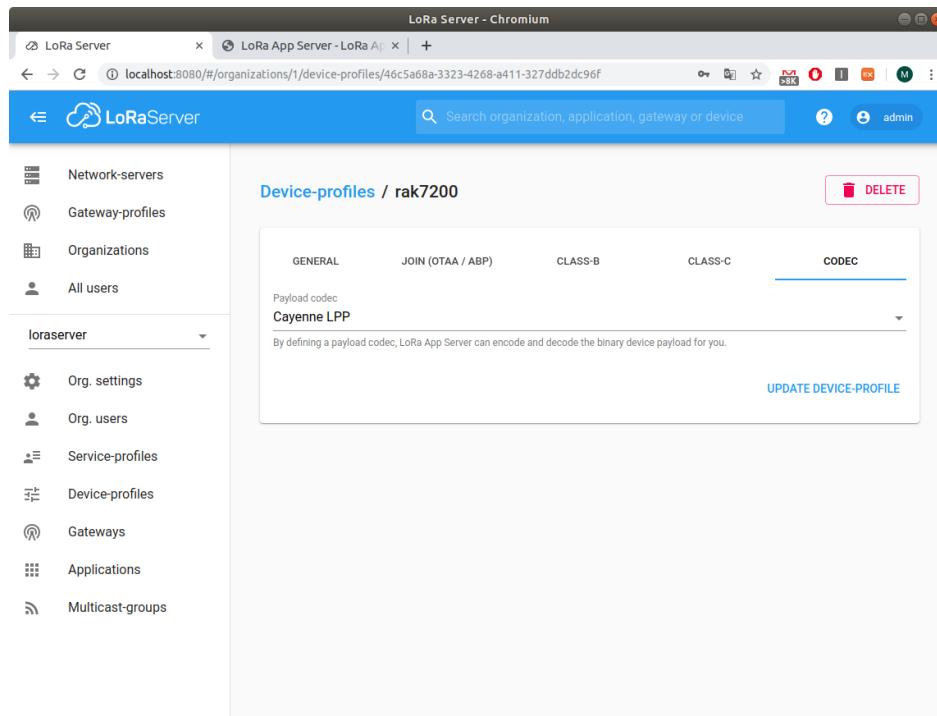
Note that Go by default encodes a byte array to base64 when encoding an object to JSON. So within the context of:

```
function Decode(fPort, bytes) {
    return {"myresult":bytes};
}
```

bytes is an array containing the decrypted payload bytes. Thus bytes[0] returns the first byte, etc...

<https://www.loraserver.io/lora-app-server/integrate/sending-receiving/thingsboard/>

(위 내용을 토대로) LoRa App Server 상에서 payload codec을 Cayenne LPP로 변경해 보자.



[그림 12.25] Payload code 설정을 Cayenne LPP로 변경

참고: Cayenne LPP와 관련해서는 아래 site 내용을 확인해 보도록 하자.

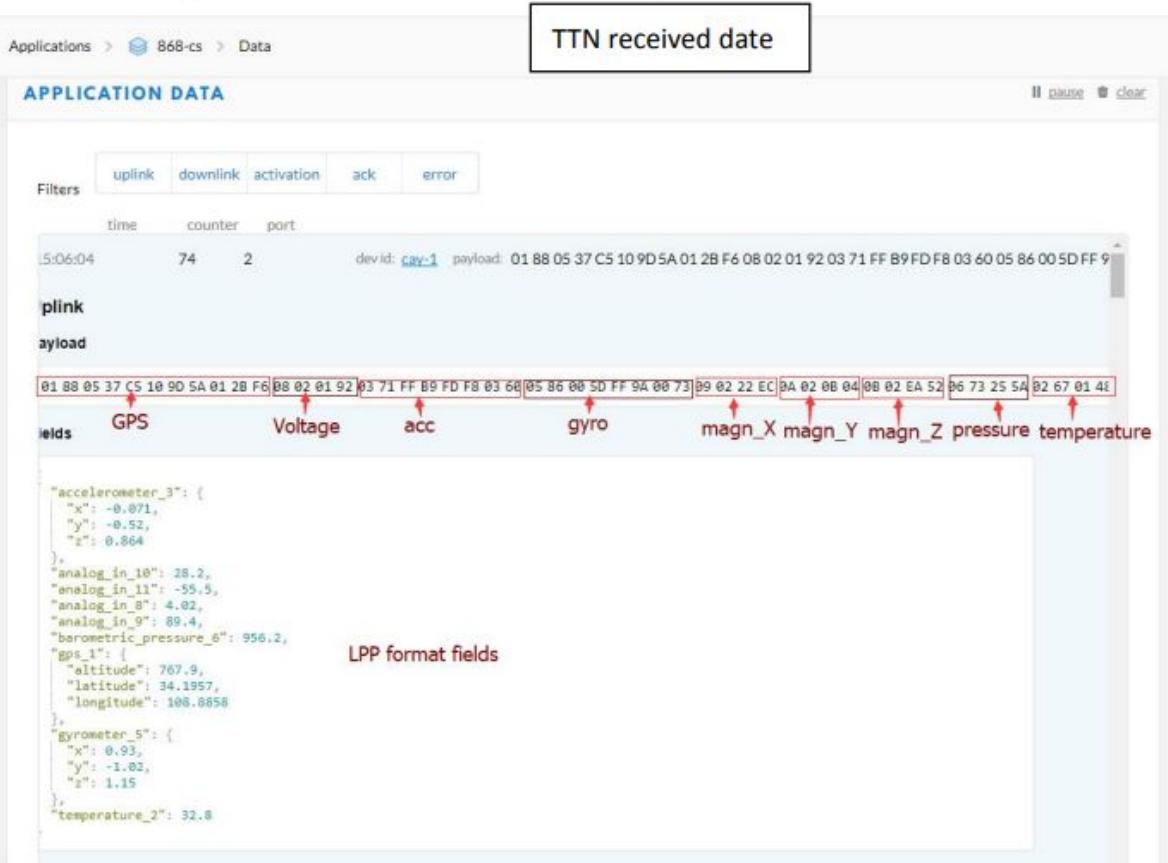
<https://developers.mydevices.com/cayenne/docs/lora/>

(나중에 정리한 것임)

아래 RAK7200 문서를 참조해 보라. RAK7200은 LPP 형태로 message를 던져주고 있다. 따라서 LoRa App Server에서 Payload Codec을 Cayenne LPP로 지정하는 것은 올바른 설정이다.

https://downloads.rakwireless.com/en/LoRa/RAK7200-Tracker/Application_Notes/RAK7200_Packet_Decode.pdf

RAK7200's payload is in LPP format, for example:



[그림 12.26] RAK7200 payload - LPP format

이 상태에서 (LoRa Server가 설치되어 있는 PC에서) mosquitto_sub program을 띄워 LoRa packet이 들어오는지 확인해 보도록 하자.

```
$ sudo mosquitto_sub -v -t "application/#" -h localhost -p 1883
```

→ 어라, 안보이던 object 정보(여기에 sensor data가 담겨 있음)가 보인다.

```
application/3/device/0042a2bffd9ea277/error {"applicationID": "3", "applicationName": "rak7200", "deviceName": "rak7200", "devEUI": "0042a2bffd9ea277", "type": "CODEC", "error": "invalid data type: 5", "fCnt": 6}
application/3/device/0042a2bffd9ea277/rx {"applicationID": "3", "applicationName": "rak7200", "deviceName": "rak7200", "devEUI": "0042a2bffd9ea277", "rxInfo": [{"gatewayID": "60c5a8ffffe76130c", "name": "rak7258michael", "rssi": -48, "loRaSNR": 9.8, "location": {"latitude": 37.5371163, "longitude": 127.0078127, "altitude": 3}}, {"txInfo": {"frequency": 922300000, "dr": 5}, "adr": true, "fCnt": 6, "fPort": 2, "data": "CAIBgQNxAAD+AAAAABYYAAAABAAEAgLPAwX+uQ=="}]
application/3/device/0042a2bffd9ea277/rx {"applicationID": "3", "applicationName": "rak7200", "deviceName": "rak7200", "devEUI": "0042a2bffd9ea277", "rxInfo": [{"gatewayID": "60c5a8ffffe76130c", "name": "rak7258michael", "rssi": -47, "loRaSNR": 7.8, "location": {"latitude": 37.5371163, "longitude": 127.0078127, "altitude": 3}}, {"txInfo": {"frequency": 922500000, "dr": 5}, "adr": true, "fCnt": 7, "fPort": 2, "data": "CAIBgQNxAAD+AAAAAAEABYYAAAABAAEAgLQAwL+rW==", "object": {"analogInput": {"3": -3.37, "4": 7.2, "8": 3.85}, "accelerometer": {"3": {"x": 0, "y": 0, "z": 0.256}}, "gyrometer": {"5": {"x": 0, "y": 0.01, "z": 0}}}]}

application/3/device/0042a2bffd9ea277/error {"applicationID": "3", "applicationName": "rak7200", "deviceName": "rak7200", "devEUI": "0042a2bffd9ea277", "type": "CODEC", "error": "invalid data type: 245", "fCnt": 8}
application/3/device/0042a2bffd9ea277/rx {"applicationID": "3", "applicationName": "rak7200", "deviceName": "rak7200", "devEUI": "0042a2bffd9ea277", "rxInfo": [{"gatewayID": "60c5a8ffffe76130c", "name": "rak7258michael", "rssi": -53, "loRaSNR": 9.8, "location": {"latitude": 37.5371163, "longitude": 127.0078127, "altitude": 3}}, {"txInfo": {"frequency": 922300000, "dr": 5}, "adr": true, "fCnt": 8, "fPort": 2, "data": "CAIBgQNxAAD/AP8ABYYAAAABAAEAgLIAvX+pQ=="}]
```

[그림 12.27] mosquitto_sub 실행 결과 - 정상 수신된 LoRa packet

<정상적으로 decoding된 LoRa packet>

```
application/3/device/0042a2bffd9ea277/rx
{"applicationID": "3", "applicationName": "rak7200", "deviceName": "rak7200", "devEUI": "0042a2bffd9ea277", "rxInfo": [{"gatewayID": "60c5a8ffffe76130c", "name": "rak7258michael", "rssi": -52, "loRaSNR": 10, "location": {"latitude": 37.5371163, "longitude": 127.0078127, "altitude": 3}}, {"txInfo": {"frequency": 922100000, "dr": 5}, "adr": true, "fCnt": 15, "fPort": 2, "data": "CAIBgQNxAAD+AAAAAP8ABYYAAAABAAEAgLXAwL+tg==", "object": {"analogInput": {"3": -3.3, "4": 7.27, "8": 3.85}, "accelerometer": {"3": {"x": 0, "y": 0, "z": -0.256}}, "gyrometer": {"5": {"x": 0, "y": 0.01, "z": 0}}}]}
```

ThingsBoard LORA GPS SENSOR 디바이스의 최근 데이터 tab에 드디어 analogInput, accelerometer, gyrometer 정보가 보인다. 어라... **근데, 계속 보이지 않고, 한번만 보이고 만다.** mosquitto_sub 실행 결과도 동일하다 ... 왜일까 ?

참고: (나중에 정리한 것임) 다른 RAK7200 GPS tracker로 해 보니 정상이다. 문제가 되는 RAK7200의 firmware를 교체해 보아야 겠다.

The screenshot shows the ThingsBoard web interface. On the left sidebar, under 'ENTITY VIEWS', there is a section for '디바이스' (Devices). It lists two entities: 'Temp_Sensor' and 'RAK7204_LoRa_Sensor'. The 'RAK7204_LoRa_Sensor' entity is selected, and its details are displayed in the main content area. The title is 'LORA GPS SENSOR' and it says '디바이스 상세정보'. Below this is a table of data logs:

날짜	측정 항목	값
2019-10-02 13:46:53	data_accelerometer_3_y	-0.985
2019-10-02 13:46:53	data_accelerometer_3_z	-0.032
2019-10-02 13:46:53	data_analog_input_10	37.95
2019-10-02 13:46:53	data_analog_input_11	-59.55
2019-09-25 16:30:17	data_analog_input_3	-3.41
2019-09-25 18:01:19	data_analog_input_4	7.28
2019-10-02 13:46:53	data_analog_input_8	3.8
2019-10-02 13:46:53	data_analog_input_9	39.6
2019-09-25 18:01:19	data_analog_output_3	-3.2
2019-10-02 13:46:53	data_gyrometer_5_x	-1.65
2019-10-02 13:46:53	data_gyrometer_5_y	0.53
2019-10-02 13:46:53	data_gyrometer_5_z	0
2019-09-24 16:52:09	data_msg_temperature	55

[그림 12.28] 정상적으로 수신된 LoRa GPS Sensor 데이터

아, 이건 ThingsBoard 쪽 문제가 아니다. RAK7258 ⇔ LoRa Server ⇔ LoRa App Server 쪽에서 발생하는 문제다. 처음부터 있던 문제인데 왜 이제 발견했을까? 어디가 문제일까?

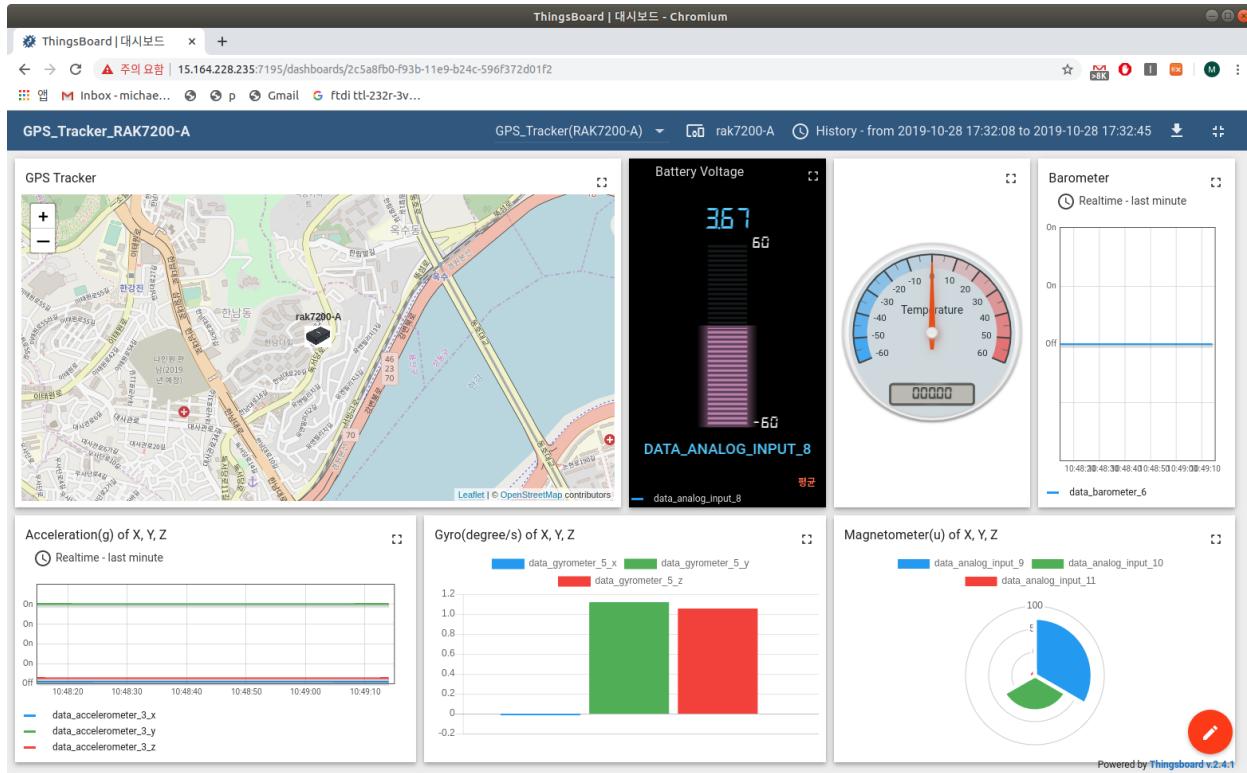
원인 분석은 좀 뒤로 미루기로 하고, 우선 최근 수신한 LoRa GPS Sensor 데이터를 위젯 형태로 표현해 보도록 하자.

The screenshot shows the ThingsBoard web interface with the URL 172.30.1.1:8080/devices. On the left sidebar, under the 'Devices' section, two devices are listed: 'Temp_Sensor' and 'RAK7204_LoRa_Sensor'. The 'RAK7204_LoRa_Sensor' card displays its entity views: 'LORA_TEMP_HUMI_PRESSURE' and 'LORA_GPS_SENSOR'. The main content area is titled 'LORA GPS SENSOR' and shows a table of recent data. The table has columns: 'Last update time', 'Key ↑', and 'Value'. Several rows are listed, with the first three having checkboxes checked. A red circle highlights the 'Recent Data' button (three dots) in the top right corner of the table header.

Last update time	Key ↑	Value
2019-10-02 13:56:58	data_accelerometer_3_x	0.032
2019-10-02 13:56:58	data_accelerometer_3_y	-0.987
2019-10-02 13:56:58	data_accelerometer_3_z	-0.02
2019-10-02 13:56:58	data_analog_input_10	-0.3
2019-10-02 13:56:58	data_analog_input_11	-94.65
2019-09-25 16:30:17	data_analog_input_3	-3.41
2019-09-25 18:01:19	data_analog_input_4	7.28
2019-10-02 13:56:58	data_analog_input_8	3.81
2019-10-02 13:56:58	data_analog_input_9	32.7
2019-09-25 18:01:19	data_analog_output_3	-3.2
2019-10-02 13:56:58	data_gyrometer_5_x	-1.7

[그림 12.29] 정상적으로 수신된 LoRa GPS Sensor 데이터 - 위젯 보기 선택

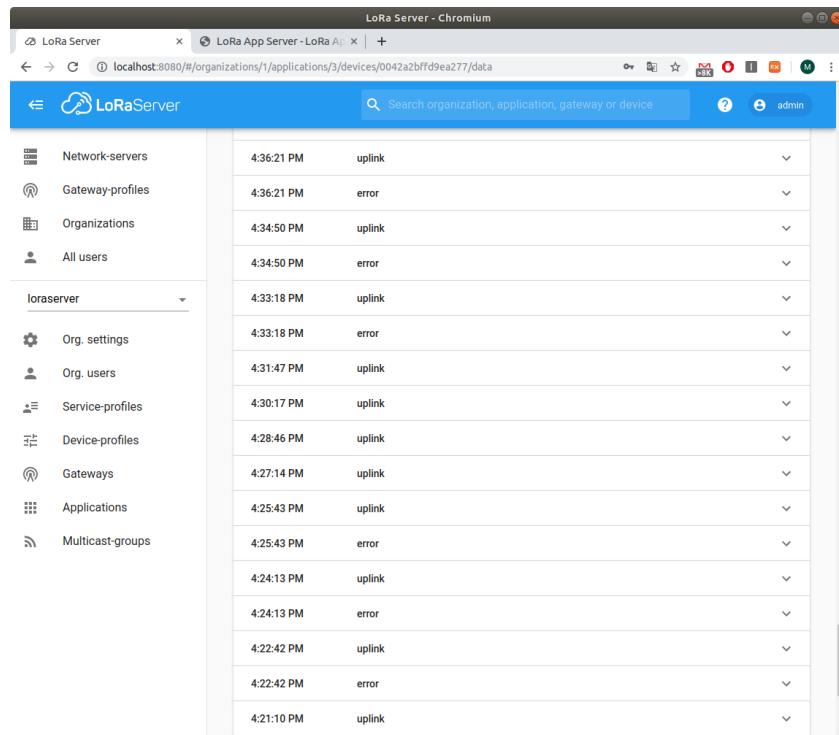
참고: Recent Data(최근 데이터)를 가지고, 위젯을 만들려면, 해당 행(row)을 체크한 후, 상단의 "위젯보기" 버튼을 선택하면 된다.



[그림 12.30] LoRa GPS Sensor 수신 데이터를 dash board 상의 위젯으로 표현한 화면
(Device - RAK7200)

4) LoRa App Server Packet 에러 발생 원인 분석

아래 비정상 packet이 발생하는 문제는 추후 다시 살펴 보기로 하자.



The screenshot shows a browser window titled "LoRa Server - Chromium" with the URL "localhost:8080/#/organizations/1/applications/3/devices/0042a2bffd9ea277/data". The main content is a table titled "LoRaServer" with a search bar at the top. The left sidebar lists various LoRa components: Network-servers, Gateway-profiles, Organizations, All users, loraserver (selected), Org. settings, Org. users, Service-profiles, Device-profiles, Gateways, Applications, and Multicast-groups. The main table displays a list of events:

	Time	Type
	4:36:21 PM	uplink
	4:36:21 PM	error
	4:34:50 PM	uplink
	4:34:50 PM	error
	4:33:18 PM	uplink
	4:33:18 PM	error
	4:31:47 PM	uplink
	4:30:17 PM	uplink
	4:28:46 PM	uplink
	4:27:14 PM	uplink
	4:25:43 PM	uplink
	4:25:43 PM	error
	4:24:13 PM	uplink
	4:24:13 PM	error
	4:22:42 PM	uplink
	4:22:42 PM	error
	4:21:10 PM	uplink

[그림 12.31] LoRa App Server Uplink error(**TBD: 이 문제 해결해야 함**)

참고: 전에는 분명 이랬던 것 같지 않은데, 이상하다. 중간에 LoRa Gateway(RAK7258) WebUI 설정을 바꾼게 영향을 주지 않았나 싶다. RAK7249로 다시 한번 확인해 보아야 겠다.

참고 1: (나중에 정리한 것임) RAK7249로 해 보니, uplink error 패킷이 보이질 않는다(즉, 정상적이다). 따라서 RAK7258 설정에 문제가 있는 것이 분명하다.

참고2: (나중에 정리한 것임) RAK7200 GPS Tracker firmware version 문제인 것 같다. 다른 RAK7200 GPS tracker로 테스트해 보니 정상이다.

LoRa App Server ⇔ ThingsBoard format 전환과 관련해서는 아래 page를 확인해 보자.
ThingsBoard PE(Professional Edition)만 지원하는 것 같은 느낌이긴 한데, 참조해 보도록 하자.

<https://thingsboard.io/docs/user-guide/integrations/>

5) Source code를 가지고 Build해 보기

아래 site를 참조하여 build를 해 보도록 하자.

<https://thingsboard.io/docs/user-guide/install/building-from-source/>

<Ubuntu 18.04 desktop PC>

\$ sudo apt-get install maven

\$ git clone <https://github.com/thingsboard/thingsboard>

→ Github에서 직접 받자.

\$ cd thingsboard

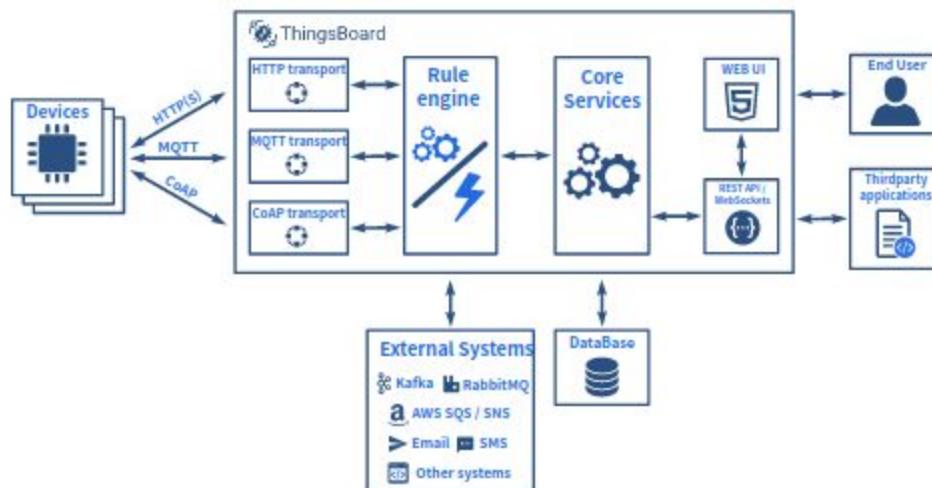
\$ mvn clean install

→ 중간 중간에 Error가 보이지만, 결국은 정상 build된다.

```
[INFO] Thingsboard Server Commons ..... SUCCESS [ 0.029 s]
[INFO] Thingsboard Server Common Transport components ..... SUCCESS [ 3.302 s]
[INFO] Thingsboard MQTT Transport Common ..... SUCCESS [ 0.683 s]
[INFO] Thingsboard HTTP Transport Common ..... SUCCESS [ 0.347 s]
[INFO] Thingsboard CoAP Transport Common ..... SUCCESS [ 0.451 s]
[INFO] Thingsboard Server Common DAO API ..... SUCCESS [ 0.571 s]
[INFO] Thingsboard Extensions ..... SUCCESS [ 0.046 s]
[INFO] Thingsboard Rule Engine API ..... SUCCESS [ 0.512 s]
[INFO] Thingsboard Server DAO Layer ..... SUCCESS [03:04 min]
[INFO] Thingsboard Rule Engine Components ..... SUCCESS [ 7.509 s]
[INFO] Thingsboard Server Transport Modules ..... SUCCESS [ 0.045 s]
[INFO] Thingsboard HTTP Transport Service ..... SUCCESS [ 13.893 s]
[INFO] Thingsboard MQTT Transport Service ..... SUCCESS [ 7.832 s]
[INFO] Thingsboard CoAP Transport Service ..... SUCCESS [ 12.175 s]
[INFO] Thingsboard Server UI ..... SUCCESS [ 57.909 s]
[INFO] Thingsboard Server Tools ..... SUCCESS [ 0.363 s]
[INFO] ThingsBoard Server Application ..... SUCCESS [13:12 min]
[INFO] ThingsBoard Microservices ..... SUCCESS [ 0.346 s]
[INFO] ThingsBoard Docker Images ..... SUCCESS [01:08 min]
[INFO] ThingsBoard JavaScript Executor Microservice ..... SUCCESS [01:54 min]
[INFO] ThingsBoard Web UI Microservice ..... SUCCESS [ 17.984 s]
[INFO] ThingsBoard Node Microservice ..... SUCCESS [ 0.137 s]
[INFO] ThingsBoard Transport Microservices ..... SUCCESS [ 0.023 s]
[INFO] ThingsBoard MQTT Transport Microservice ..... SUCCESS [ 0.273 s]
[INFO] ThingsBoard HTTP Transport Microservice ..... SUCCESS [ 0.609 s]
[INFO] ThingsBoard COAP Transport Microservice ..... SUCCESS [ 0.078 s]
[INFO] ThingsBoard Black Box Tests ..... SUCCESS [01:42 min]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 23:29 min
[INFO] Finished at: 2019-09-28T16:07:41+09:00
[INFO] -----
```

chyi@earth:~/workspace/ThingsBoard/thingsboard\$ █

```
chyi@earth:~/workspace/ThingsBoard/thingsboard/application/target$ ls -la
합계 471228
drwxr-xr-x 19 chyi chyi      4096 9월 28 16:02 .
drwxr-xr-x  5 chyi chyi      4096 9월 28 16:02 ..
drwxr-xr-x  2 chyi chyi      4096 9월 28 16:02 archive-tmp
drwxr-xr-x  3 chyi chyi      4096 9월 28 15:51 bin
drwxr-xr-x  5 chyi chyi      4096 9월 28 15:51 classes
drwxr-xr-x  4 chyi chyi      4096 9월 28 15:51 conf
drwxr-xr-x  4 chyi chyi      4096 9월 28 15:51 control
drwxr-xr-x  6 chyi chyi      4096 9월 28 15:51 data
drwxr-xr-x  2 chyi chyi      4096 9월 28 16:02 debian
drwxr-xr-x  6 chyi chyi      4096 9월 28 15:51 embeddedCassandra
drwxr-xr-x  4 chyi chyi      4096 9월 28 15:51 generated-sources
drwxr-xr-x  3 chyi chyi      4096 9월 28 15:51 generated-test-sources
drwxr-xr-x  2 chyi chyi      4096 9월 28 16:02 maven-archiver
-rw-r--r--  1 chyi chyi 4539800 9월 28 15:51 protoc-3.6.1-linux-x86_64.exe
drwxr-xr-x 10 chyi chyi      4096 9월 28 15:51 protoc-dependencies
drwxr-xr-x  2 chyi chyi      4096 9월 28 15:51 protoc-plugins
drwxr-xr-x  2 chyi chyi      4096 9월 28 16:02 surefire-reports
drwxr-xr-x  3 chyi chyi      4096 9월 28 15:51 test-classes
-rw-r--r--  1 chyi chyi 127345309 9월 28 16:02 thingsboard-2.4.1-SNAPSHOT-boot.jar
-rw-r--r--  1 chyi chyi 1021251 9월 28 16:02 thingsboard-2.4.1-SNAPSHOT.jar
-rw-r--r--  1 chyi chyi 116626654 9월 28 16:02 thingsboard-windows.zip
-rw-r--r--  1 chyi chyi 551 9월 28 16:02 thingsboard.changes
-rw-r--r--  1 chyi chyi 116446754 9월 28 16:02 thingsboard.deb
-rw-r--r--  1 chyi chyi 116453992 9월 28 16:02 thingsboard.rpm
drwxr-xr-x  4 chyi chyi      4096 9월 28 16:02 tmp
drwxr-xr-x  4 chyi chyi      4096 9월 28 16:02 windows
```



[그림 12.32] ThingsBoard Monolithic Architecture

참고: ThingsBoard는 서버 부분(Transport/Rule Engine/Core Services 등)은 Java로 구현되어 있으며, Web front end는 HTML/Angular JS(javascript)로 구현되어 있다. 용기(?)있는 자여 ~ 코드 분석을 해 보기 험 권한다.

```
chyi@earth:~/workspace/ThingsBoard/thingsboard$ ls -la
합계 128
drwxr-xr-x 15 chyi chyi 4096 9월 28 15:14 .
drwxr-xr-x  3 chyi chyi 4096 9월 28 15:14 ..
drwxr-xr-x  8 chyi chyi 4096 9월 28 15:14 .git
-rw-r--r--  1 chyi chyi   435 9월 28 15:14 .gitignore
-rw-r--r--  1 chyi chyi   410 9월 28 15:14 .travis.yml
-rw-r--r--  1 chyi chyi 11353 9월 28 15:14 LICENSE
-rw-r--r--  1 chyi chyi   2326 9월 28 15:14 README.md
drwxr-xr-x  5 chyi chyi 4096 9월 28 16:02 application
drwxr-xr-x  8 chyi chyi 4096 9월 28 15:14 common
drwxr-xr-x  4 chyi chyi 4096 9월 28 15:44 dao
drwxr-xr-x  5 chyi chyi 4096 9월 28 15:14 docker
drwxr-xr-x  2 chyi chyi 4096 9월 28 15:14 img
drwxr-xr-x  2 chyi chyi 4096 9월 28 16:20 k8s
-rw-r--r--  1 chyi chyi   577 9월 28 15:14 license-header-template.txt
drwxr-xr-x  8 chyi chyi 4096 9월 28 15:14 msa
drwxr-xr-x  4 chyi chyi 4096 9월 28 15:44 netty-mqtt
-rwxr-xr-x  1 chyi chyi 40246 9월 28 15:14 pom.xml
drwxr-xr-x  4 chyi chyi 4096 9월 28 15:14 rule-engine
drwxr-xr-x  4 chyi chyi 4096 9월 28 15:49 tools
drwxr-xr-x  5 chyi chyi 4096 9월 28 15:14 transport
drwxr-xr-x  5 chyi chyi 4096 9월 28 15:48 ui
```

[그림 12.33] ThingsBoard 소스 Tree

6) ThingsBoard 로 할 수 있는 것

헐 대박 ~ ThingsBoard를 잘 활용하면 LoRa 디바이스 관리는 물론이고, SPNBox 원격 관리, Texaking 냉장고 관리 시스템, NF 산소 발생기 관제 시스템 구축 등 많은 것들을 만들어 낼 수 있을 것 같다.

[숙제 - 12월말까지(1차)]

- Texaking 냉장고 관리 시스템을 만들어 보자.
- NF 산소 발생기 관리 시스템을 만들어 보자.
- LoRa Device 관리 시스템을 만들어 보자.
- SPN(SPNBox & Clients) 원격 관리 시스템을 만들어 보자.

참고: 위의 관리 시스템은 **Client(C로 작성) + ThingsBoard(Server/Viewer)**를 말한다. 즉, ThingsBoard로 관리 page(dashboard)를 만들고, C(다른 langauge도 OK이나, gateway에 탑재해야 하는 바 binary 크기가 작아야 함)로는 관련 정보를 전달(HTTP로 전달)하는 program을 하나 만들면 된다.

7) myDevices Cayenne Overview

<여기서 잠깐 - myDevices사가 개발한 Cayenne에 관하여>

지금 부터는 ThingsBoard와 견줄 수 있는 또 다른 IoT Platform인 Cayenne를 조금만 살펴보고자 한다.

<https://developers.mydevices.com/cayenne/docs/intro/>

How LoRaWAN™ works

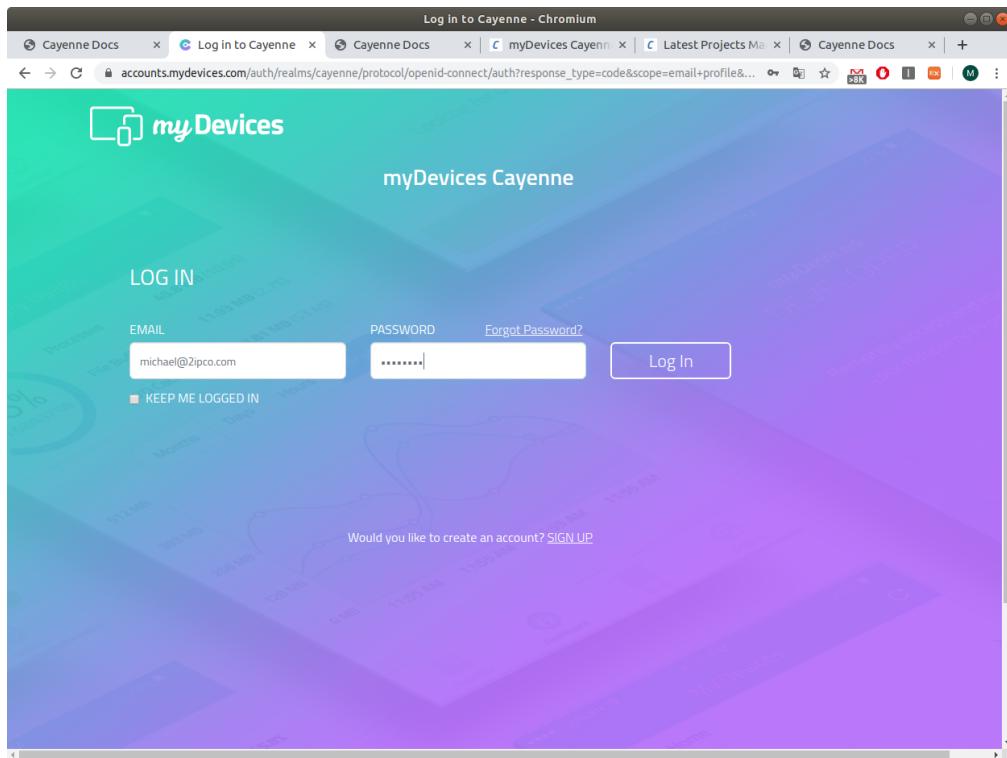
LoRaWAN™ is a protocol specification based on the LoRa technology developed by the LoRa Alliance. LoRaWAN targets the basic needs of LoRa usage for IoT by providing Addressing, Routing and Security.

Topology of a LoRaWAN network consists of several elements.

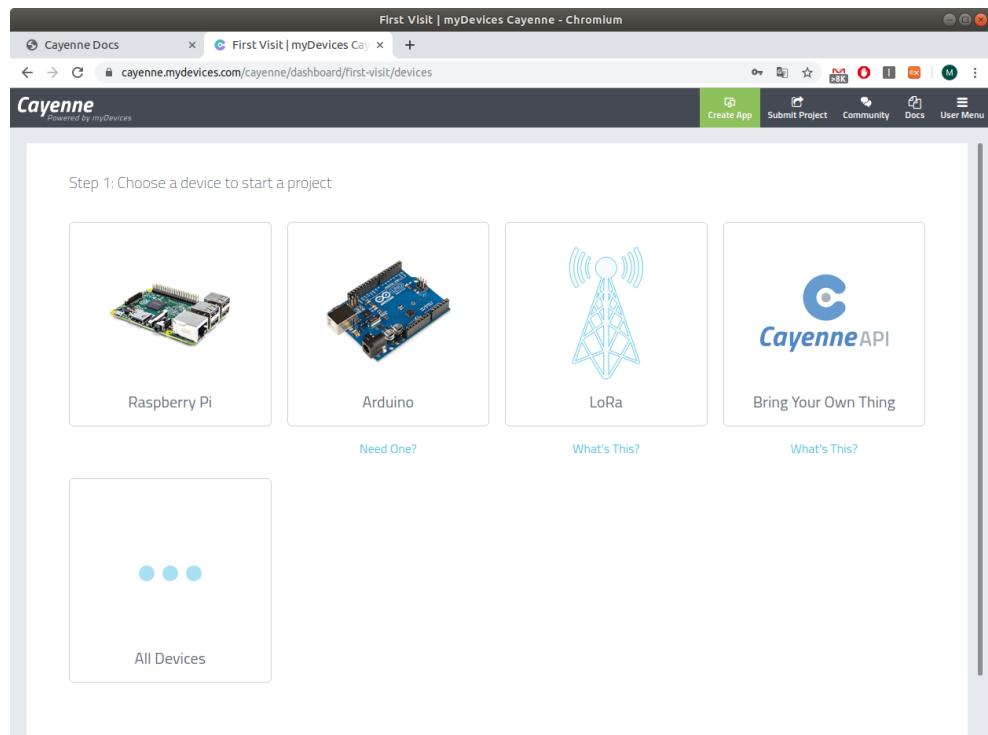
- **End Nodes:** End nodes are elements such as sensors, which are usually remotely located.
- **Concentrator / Gateway:** Gateways are access points for end nodes (e.g. sensors), aggregating data and communicating that data to a central network server via standard IP connections. Several gateways can be co-located in an area and can transparently share a single connection to the network server.
- **Network Server:** The LoRa Network Server acts to eliminate duplicate packets, manages security and data rates.
- **Application Server:** Application Servers manage payload security and performs analysis to utilize sensor data. Cayenne operates as an Application Server.

아래 site의 내용을 토대로 계정을 생성하고, 화면 설정을 해 보자.

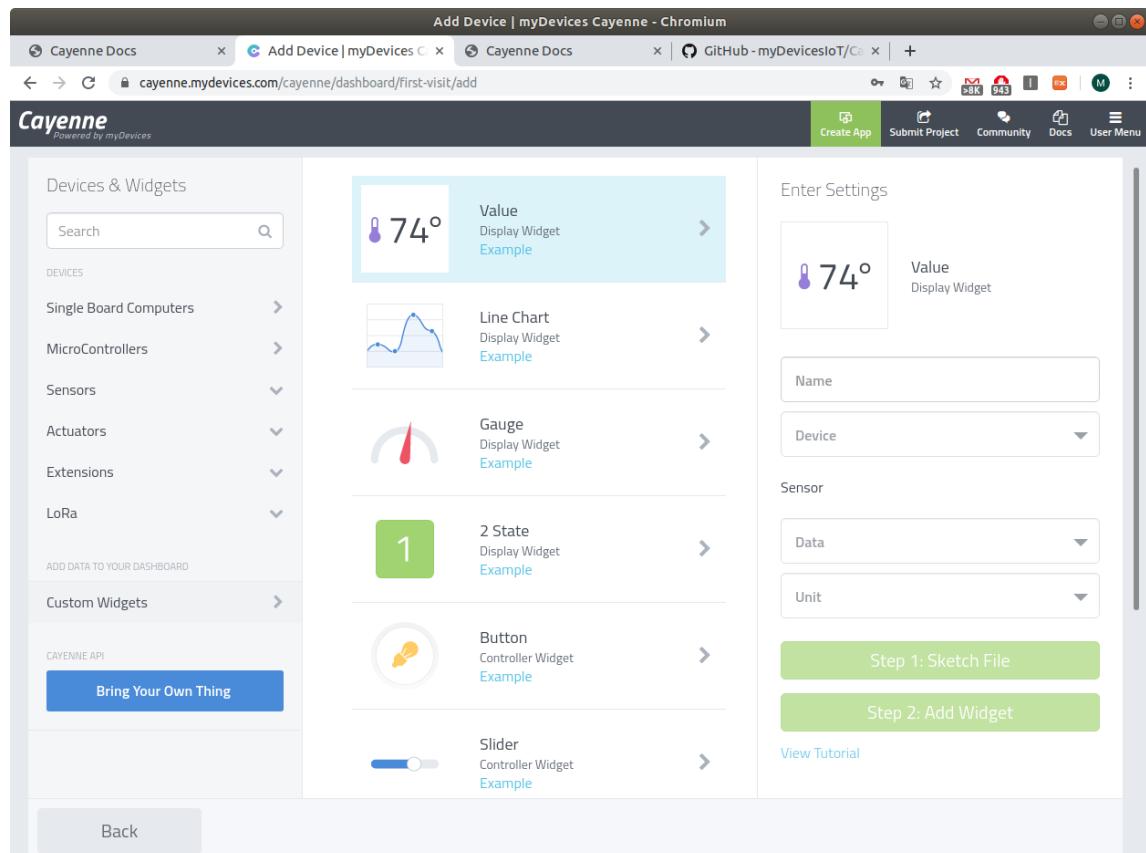
<https://developers.mydevices.com/cayenne/docs/getting-started/>



[그림 12.34] Cayenne Login 화면



[그림 12.35] Cayenne Login 후 Step 1 화면



[그림 12.36] Cayenne Devices & Widgets 화면

나름 화면이 괜찮아 보인다. 근데, 전체적으로 설정이 좀 복잡하고, 정리가 좀 덜 된 느낌이다 ...
추후 기회가 오면 다시 설정해 보기로 하고, 오늘은 여기까지만 하자.

13. Our LoRa Viewer: OLoRa

Desktop Viewer는 ThingsBoard 기반으로 가는 것이 맞겠다. 하지만 Mobile Viewer 정도는 ThingsBoard와 연계(REST API 사용)하는 형태로 간단하게 만들 수 있을 것 같고 ... 따라서 사실상 OLoRa의 의미는 크게 없어 보인다 ...

ThingsBoard == OLoRa !!!

1) 2ip Mobile Viewer

<TBD> 일단 아래 내용을 검토해 보자.

<https://thingsboard.io/docs/reference/rest-api/>

<https://demo.thingsboard.io/swagger-ui.html>

참고 사항: 만일 Android or iOS에서 ThingsBoard를 접근하는 방법(REST API)이 문제가 있다면, Texaking 형태(별도의 작은 서버 - LoRa App Server와 MQTT 등으로 통신 - 구현)로 개발할 수도 있다.

