VInsight Platform Installation & Administrator's Guide

Release 1.0

Published December 2013

1.0 Edition

Dependable Networking and Computing (DNC)

CONTENTS

Chapter One

1. Introduction

VInsight is a research infrastructure to enable measurement and evaluation for the advancement of the networked vehicular sensing network, which is targeted primarily for research and educational use. The VInsight project, started in 2013, is an open-source project developing *VInsight.*

The purpose of this document is to introduce new VInsight users to the system in a structured way. It is sometimes difficult for new users to gain essential information for running simulations, and duplicating VInsight on theirs demands. We, in this document, will introduce and explain key concepts and features of VInsight with several example simulations.

A few key points are worth noting at the onset:

- VInsight is open source, and can be deployed on real-world vehicles. The project strives to maintain an open research infrastructure for researchers to for enabling open innovations with understanding and investigating of real vehicle and physical link features in networked vehicle sensing and control.

- VInsight is not a entirely open source since it also relies on several open source libraries or software like OMF/OML, gpsd, iperf, and openxc. No source code of those libraries or software is included in package VInsight . The goal of using common and open source libraries/software helps resolve many incompatibility issues.

1.1. About VInsight

VInsight has been developed to provide an open, extensible research infrastructure for research and education use. In brief, it provides basic building blocks and entities of how to deploy, run experiments in networked VSC networks, and collect experimental data for further analysis. In contrast to other software or tools, VInsight enables real-time sensing and archival of real-world vehicle behavior and link characteristics by users. These real-time and archival sensing data can serve as a basis for vehicle modeling (e.g., on fuel economy) in the development of innovative networked VSC solutions, and they also can serve as realistic traffic input to evaluating VSC-oriented wireless networking solutions.

Below are a few distinguishing features of VInsight.

- VInsight is designed as a set of entities that can be combined together and also with other software libraries.
- VInsight is primarily used on Linux system. As of right now, we only have source code support for Ubuntu based distribution. IF you have a distribution you want supported, please send us email vinsight_dnc@gmail.com.
- VInsight is not an officially supported software product of any company. Support for VInsight is done on a best-effort manner.

1.2. For VInsight Users

In VInsight, the source code is written in C, with calling some Linux command. It is mainly because of the requirement of hardware configuration and controlling, which we will further explain by combining installation of the sophisticated drivers.

For those familiar with C/C++ programming and linux shell script programming, the most source code is easily understandable. Users can glean the essential information from this document and convert the information into working experiments.

Besides, we hope users can have some basic knowledge on open source libraries/software such as OpenXC, OMF/OML, GPSD, and Iperf, for better understanding of VInsight. If you are new to those libraries and software, we recommend visiting the following websites.

- OpenXC: http://openxcplatform.com/
- OMF/OML: http://mytestbed.net/projects/omf/wiki/An_Introduction_to_OMF
- GPSD: http://catb.org/gpsd/
- Iperf: http://openmaniak.com/iperf.php

We also encourage people to contribute to VInsight by developing new entities, debugging or maintaining existing entities, and sharing results and experience.

1.3. Document organization

This document assumes that new users might initially follow a path such as the following:

- Try to download and build a copy;
- Try to run a few sample programs:
- Look at experiment output/trace files, and try to customize the experiments

As a result, we have tried to organize this document along the above board sequences of events.

1.4. Conventions

The following acronyms and abbreviations have been used in this document:

Table 1: Acronyms and Abbreviations

Acronym	Description
VSC	Vehicular Sensing and Control
WiMAX	Worldwide Interoperability for Microwave Access
OMF	Orbit Management Framework
OML	OMF Measurement Library
OEDL	OMF Experiment Description Language
GPSD	GPS service Daemon
OS	Operating System
EE	Experiment Engine
ED	Experiment Description

Chapter Two

2. Installation

This section is aimed at getting a user to configure a machine with VInsight. It covers supported platforms, prerequisites, way to obtain VInsight, and ways to verify your build.

2.1 Overview

VInsight is built as a system of software and hardware that work together. It is distributed as source code and requires the target system needs to have physical devices: CAN translator, GPS, and WiMAX dongle, and a software environment to compile the libraries and source code, then build the VInsight. VInsight could in principle be distributed as pre-built libraries for selected systems, and in the future it may be distributed that way, but at present, we actually do the work by editing VInsight itself. If someone would like to undertake the job of making pre-built libraries and packages for operating systems, please contact us.

In the following, we will first explains prerequisites that are required to support VInsight and also provide the commands to install them on the OS.

2.2 Installation Prerequisites

VInsight has a number of dependencies. In this section, we provide instructions of how those dependencies are installed and the information necessary to configure the software environment of VInsight. Those dependencies include:

- WiMAX dongle driver
- OpenXC
- GPSD
- OML library
- Iperf

Note: Some links that might be useful are also provided as reference.

2.2.1 WiMAX Dongle Driver Installation

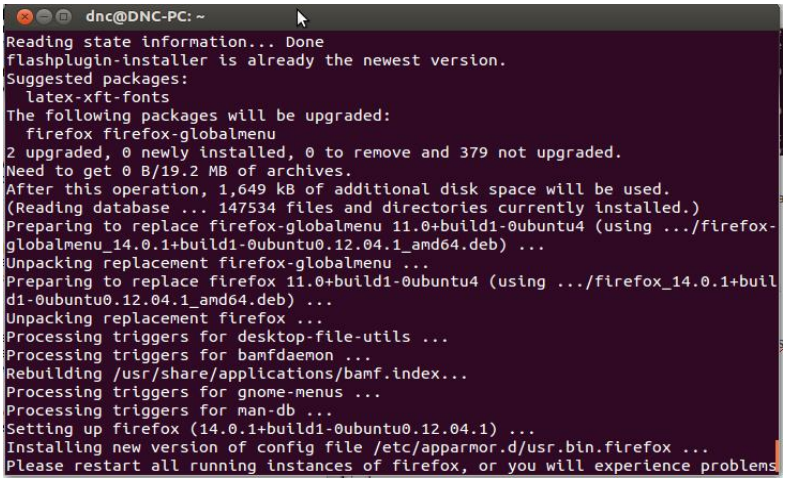
In VInsight, Teltonika device UM6225 is used for mobile platform communication in WiMAX network. The following steps are described to explain how the drive is installed on a Linux OS.

1. Insert the Teltonika device into one USB port and then wait for 5 minutes to give the driver time to install. This process is conducted implicitly.

```
%check the message that contains “Attached scsi CD-ROM[device]”
$ dmesg
$ eject
```

2. Update firefox and the flash plugin, run:

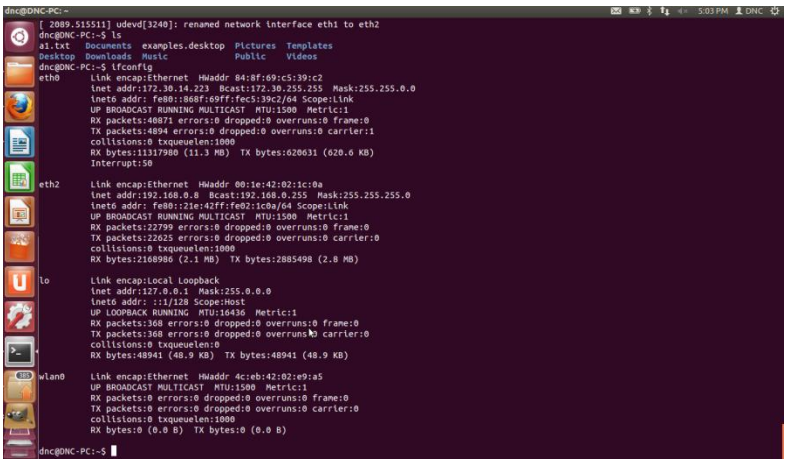
```
$ sudo apt-get install firefox flashplugin-installer
```



3. Open up the WebUI in Firefox or Chrome by navigating to 192.168.0.1. Please note that the restart operation is needed sometimes.

Login as an administrator by ctrl+shirt+clicking the Teltonika banner at the top of the page.
Username: admin
Password: EifAFKt8/genipass

4. Use the command of “ifconfig” to check the status of the adapter.



5. After the device is set up, the USB device will appear as a wired Ethernet device with the IP address of 192.168.0.1. To verify it, you can check the network connection; the wired connection that is established by the dongle should be listed. Here is an example.

```
wired network (Teltonika UM62XX)
wired connection 1
```

6. The device can be configured to look for networks on a number of channels under the wimax->channels tab. Below are the detailed steps.

i. click on wimax tab

ii. click on channels sub-tab

iii. add your base stations information e.g., Frequency(kHZ):2596000, Bandwidth:10000, FFT:1024

iv. press the button of Apply to apply the setting to the device.

7. If needed be, it is possible to set the device with a static IP address

8. Open the terminal, then input the following commands.

```
$telnet 192.168.0.1 700
  username:admin
  password: admin01
/# ifconfig icc0 [desired address e.g., 10.3.11.16] netmask [desired netmask e.g., 255.255.255.0]
/# export interface= icc0
/# export subnet=[desired subnet e.g., 10.3.11.1]
/# export ip=[desired address e.g., 10.3.11.16]
/# /etc/udhcpc.script bound
```

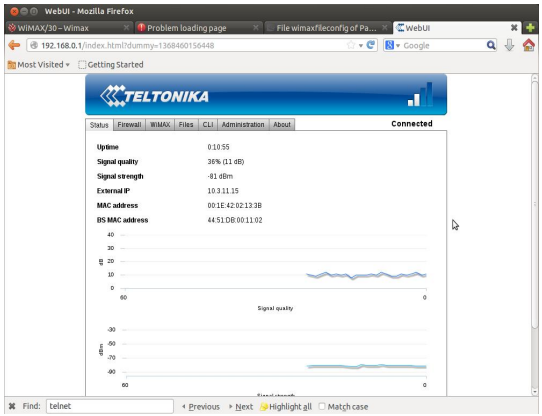
If you are planning to use the WiMAX interface for the default route, not just for traffic on its own subnet, you should add

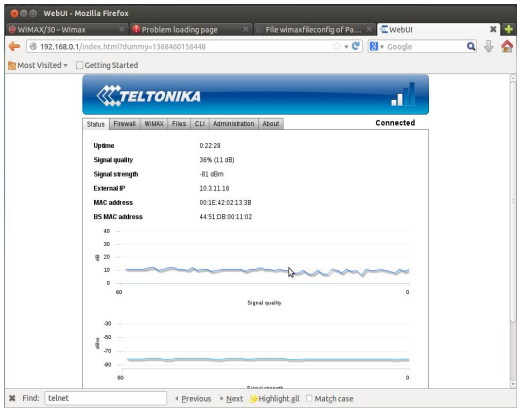
```
/#   export router =[router IP e.g., 10.3.11.1]
/#   /etc/udhcpc.script bound
```

If there is a warning message after the bound command, try to reconfigure the WiMAX interface to fix that

```
/ # ifconfig icc0 10.3.11.16 netmask 255.255.255.0 up
```

To verify the configuration, we compare the two IP addresses in the following figures to check if the new ip address has been successfully set and used to communicate with the BS.



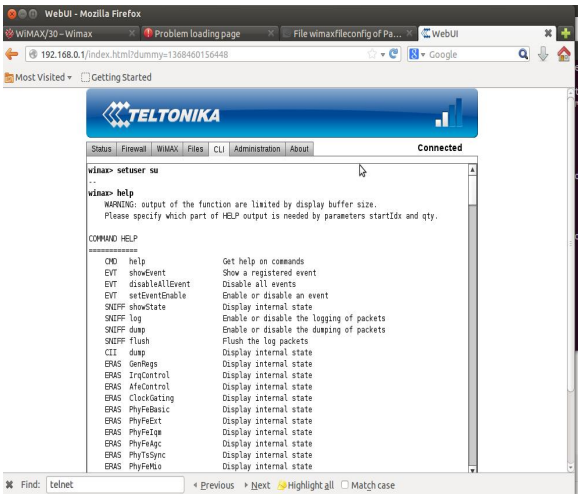


9. In some cases like measurement or application development with the wimax interface, it may need to get more information about the physical device or have an option to set the parameters according to requirements of users or applications. The tab of CLI (command line interface) is designed to serve this purpose. The operations are as follows.

i. switch to the CLI tab

ii. type " setuser su" to set the users'

iii. then type "help" and get the information shown in the figure.



iv. instead of typing commands in a browse, it is possible to get the information in a terminal to facility the measurement test or other applications.

v. create a file withe name of "teltonika_cmd.sh"

vi. add command of " wget --http-user admin --http-password EifAFKt8 -qO - "http://192.168.0.1/cgi/cli?$@" " to the file and save

vii. run it with parameter of [command]. Here is an example.

```
$ /teltonika_cmd.sh help
```

Reference:

http://wimax.orbit-lab.org/wiki/WiMAX/30

2.2.2 OpenXC Installation

OpenXC is an open source hardware and software developed by Ford Motor Company to enable user to read and translate the vehicle internal data in real-time like vehicle speed, fuel consumption, odometer, and steering wheel angle. It currently supports a growing list of Ford vehicles (https://docs.google.com/spreadsheet/ccc?key=0Ajz-75u_7nEydFJxUG4yOVZ1NXJlcjNvdzdSTDdyY0E#gid=2).

1. Switch to un-root user
2. Install Git from your distribution's package manager

```
$ sudo apt-get install git
```

3. Clone the vi-firmwave repository :

```
$ git clone https://github.com/openxc/vi-firmware
```

4. Clone the repository and install the development version like so:

```
$ git clone https://github.com/openxc/openxc-python
```

5. Installation python openxc

```
$ cd openxc-python
$pip install -e .
/*Install openxc-python library with pip or easy_install*/
$ pip install -U openxc
```

```
/*Command Line Tools*/
openxc-control options and arguments
openxc-dashboard options and arguments
openxc-dump options and arguments
openxc-gps options and arguments
openxc-trace-split options and arguments
```

6. Install vi-firmware

```
/*run the bootstrap.sh script. Un-root user is required*/
$ cd vi-firemware
$ script/bootstrap.sh
/*copy the example “CAN passthrough” implementation of “signals.cpp” to “signals.cpp”*/
$ cd src/
$ cp signals.cpp.example-passthrough signals.cpp
/*compile it ! by default this will compile for the chipKIT vehicle interface*/
$ make
```

Reference:

http://python.openxcplatform.com/en/latest/installation.html
http://vi-firmware.openxcplatform.com/en/latest/installation/installation.html
http://vi-firmware.openxcplatform.com/en/latest/definitions/definitions.html

2.2.3 GPSD Installation

GPSD is a service daemon that monitors GPSes attached to a host computer through serial or USB port, making all data on the GPS device available to be quired by the user,

Below are two ways to install gpsd and its client applications to use the GPS device on the linux OS system.

- For end user, download the gpsd packages via apt-get. After installation, use the available applications to check its status in a real-time manner
- For developer, compile the source code and design the programs based on the example codes

1. Quick Guide for GPSD installation

i. download the source file from http://git.savannah.gnu.org/cgit/gpsd.git.
ii. wE choosed the file of release-3.9.tar.gz
iii. extract the file and open the build.txt
iv. look through the build.txt and run the following commands

```
$ sudo apt-get install build-essential
$ sudo apt-get install chrpath
$ sudo apt-get install scons
$ scons && scons testregress && sudo scons udev-install
```

v. install the gpsd and gpsd-client packages

```
$ sudo apt-get install gpsd gpsd-clients
```

vi. connect the GPS device to the computer via USB port and check its connection status;

```
/*the dmesg and grep combination can also be used to show all serial ports which are represented by the string tty.*/
$ dmesg| grep tty
[        0.000000] console [tty0] enabled
[      16.961759] Bluetooth: RFCOMM TTY layer initialized
[14403.088147] cdc_acm 2-1.2:1.1: ttyACM0: USB ACM device
```

vii. Start the gpsd service deamon with specific parameters. Here is an example.

```
/*    change the device number as appropriate if you need to use a different port.*/
$ gpsd -Nn -D2 /dev/ttyACM0
```

where "-N" is run the daemon in foreground;
"-n" is to inform gpsd not to wait for the client to connect before polling whatever gps data is associated with it;
"-D" is to configure the daemon to run in a debug mode

2. Install the GPSD with the source code.

i. Get the source code from http://download.savannah.gnu.org/releases/gpsd/.. The latest version (e.g.,gpsd_3.7) is recommended.

ii. Extract the GPSD source code file and open it.

```
$ sudo    tar -xzvf gpsd_3.7.tar.gz
$ cd gpsd_3.7
```

iii. Construct the gpsd files

```
$ sudo apt-get install scons
$ sudo scons && scons testregress && sudo scons udev- install
```

iv. Run the GPSD service

```
$ sudo ./gpsd -Nn -D2 /dev/ttyACM0
```

v. Compile the client applications. An example is given below.

```
gcc -o cgps.o -c -D_GNU_SOURCE -Wextra -Wall -Wno-uninitialized -Wno-missing-field-initializers -Wcast-align -Wmissing-declarations -Wmissing-prototypes -Wstrict-prototypes -Wpointer-arith -Wreturn-type -O2 cgps.c
```

2.2.4 OMF/OML Installation

OMF is a GENI-supported open source framework for control, measurement, and management, and it is developed by WINLAB and NICTA. OML is a measurement library and used in conjunction with OMF for measurement collection. The use of OMF/OML is to configure the base stations and collect measurements in combination with VInsight.

1. Install required lib files

Note :append the source address to the /etc/apt/sources.list

```
$ deb http://pkg.mytestbed.net/ubuntu precise/
$ sudo apt-get update
$ sudo apt-get install sqlite3
$ sudo apt-get install libocomm0
```

2. Installing and configuring the OML Server package

```
$ sudo apt-get install oml2-server
```

3. To start or stop the server manually, use the following commands.

```
Stop the server:
$ sudo /etc/init.d/oml2-server stop
Start the server again:
$ sudo /etc/init.d/oml2-server start
Cycle the server (stop, then start):
$ sudo /etc/init.d/oml2-server restart
```

4. Installing the OML development packages

```
$sudo apt-get install liboml2-dev
```

5. Installing the proxy server

```
$sudo apt-get install oml2 proxy-server
```

2.2.5 Iperf Installation

Iperf is a tool used to measure maximum TCP and UDP bandwidth performance. It reports bandwidth, delay jitter, datagram loss.

1. The easiest way to install iperf is to run the command:

```
$ apt-get install iperf
```

2. The iperf can also be installed by manually compiling the source code.

```
$ apt-get install build-essential
$ wget http://stats.es.net/software/iperf-3.0.1.tar.gz
$ tar zxvf iperf-3.0.1.tar.gz
$ cd iperf-3.0.1
$ ./configure
$ make
```

2.3 VInsight Installation

From this point forward, we are going to assume that the reader is working in a well-configured Linux system attached with essential devices, i.e., WiMAX dongle, GPS, and OpenXC.

The VInsight source code is available at http://www.cs.wayne.edu/ ~ hzhang/group/software/VInsight.zip. You can also download it form GitHub with the command:

```
$ git clone git://github.com/VInsightDNC/VInsight
```

Once you have the distribution,on Linux, unpack it using unzip. That will create a new directory 'VInsight' with the source files and documentation.

VInsight complies cleanly on the system. Use 'make' to compile the source code.

```
$ unzip VInsight.zip -d .
$ cd VInsight
$ make
/*start VInsight*/
```

To install VInsight, use 'make install', which will install in/usr/bin/. To recompile, the easiest way is to start over. Do 'make clean', then' make'. If there are any problems, please report them to vinsight_dnc@gmail.com and we will try to fix them quickly.

Now, you can run 'vinsight' to start the platform.

```
$ vinsight
```

Chapter Three

3. Getting Started

In this chapter, we will show you how to describe, instrument, run, and access the results of your experiments with VInsight. It is intended for researchers, experimenters and students, who are interested in conducting experiments on VInsight.

3.1 Conceptual Overview

The first thing we need to do before actually starting to look at or write VInsight is to understand a few core concepts and terms in the project.

- Experiment. The experiment typically runs on the resources of VInsight to evaluate the ideas or solutions of the networked VSC network.
- Experiment description. It describes the resource requirements and configurations that need to be applied on the mobile platforms (i.e., vehicles), and tasks to perform with the required resources to realize the experiment.
- Experiment Engine. The EE runs as a daemon on the mobile platform to control the resource and deploy the experiments based on experiment description. Specifically, on behalf of mobile platform, it manages the entities experiment controller, resource manager, data collection, data transmission, platform health monitoring.
- Experiment controller. The experiment controller is a entity to set up,execute and clean up the experiment.
- Job. It is created based on the experiment description. All the jobs are sorted regarding the priority and executed by the sorted order.
- Database. There are two types of databases: job database and OML database. The first one is used to store jobs sent to the mobile platform. The second one is to keep the experimental results and real-time sensing data.
- Resource manager. The resource manager is a entity to manage the software and hardware resource.With the experiment description, the resource manager reserves and configures the resource for experiments.
- Data collection. The entity data collection is to collect the real time measurement data to serve the experiments or applications on the mobile platform.
- Data transmission. The data transmission is a entity, being responsible for transmitting the data between the mobile platform and the base station server.
- Platform health monitoring.The entity platform health monitoring is used to keep track of hardware and software of the mobile platform, monitoring their status.

3.2 Experiment Description for Your First Experiment

This section provides a detailed description of how to use OEDL to describe an experiment. As a new user, you would better to start with the OEDL basic tutorials. Besides, some general entry-level programming knowledge is also required.

Note: For more examples or information on using the OEDL commands, please refer to the website below http://mytestbed.net/projects/omf52/wiki/OEDL-5-2

The ED describes a simple experiment, as shown in the following.

```
1   defApplication('hybridWrapper', 'hybrid_sensing_App') {|app|
2   app.shortDescription = "This is a simple application sensing the internal vehicle status"
3   app.defProperty('exid',1,'experiment id')
4   app.defProperty('mode',3,'hybrid mode')
5   app.defProperty('start time',1 , 'the earliest available time')
6   app.defProperty('end time',600,'the time that the application should be terminated')
7   }
8   defGroup('sender',[1,1]) { |node|
9     node.type = "sender"
10    node.mode = "3"
11    node.oml = "true"
12    node.addApplication('hybridWarpper')
13    }
14
15    defGroup('receiver',[0,0]) { |node|
16    node.type = "receiver"
17    node.mode = "0"
18    node.IP ="10.3.11.28"
19    node.port ="5001"
20    node.addApplication('hybridWarpper')
21    }
22    whenAllUp(){ |node|
23    wait 10
24    allGroups.startApplications
25    wait 600
26    Experiment.done
27    }
```

In this example, we

1) define a new application, which will include both the openxc-sensing and wimax network measurement

2) define a group of nodes that will run the application and return the output

3) run the application based on the experiment description and return the results/outputs to the

- Line 1-7: we define a new application with a set of configuration. app.defProperty defines the property of the application, where
  - the first field is the name of the property;
  - the second field is the initial value of the property;
  - the third field is the textual description of the property.
- Line 2: gives a short description of the application.
- Line 3: list the identifier of the experiment.
- Line 4: set the mode to 3
  - mode 0 refers to the server that listers to the sender and cooperates with the sender to complete the application
  - mode 1 refers to the openxc sensing
  - mode 2 refers to the wimax network sensing
  - mode 3 refers to the hybrid mode of the modes 1 and 2
- Line 5: set the start time of the application. If it is 1, the EE will create one job and insert it to the joblist at the earliest available time-slot.
- Line 6: set the end time of the application. Here it is set to 600 sec. That means, the application should be stopped after the application runs 600 sec.
- Line 8-13: define a group called 'sender' that contains one nodes. In this example, there is only one sender. [x,y] describes a single or multiple resources at location x@y. In VInsight, the OMF server splits the entire 2-D space into m*n units. The mobile platforms that reside in the unit [x,y] are marked as [x,y].
  - Line 11: set the transmission mode of the experimental results
    - True: refer to the oml based transmission method
    - False: refer to the normal transmission method, i.e., Tcp/IP based method
- Line 15-20: define a group called 'receiver' that contains several nodes. In this example, the base station server is the receiver
  - Line 18: set the IP address of the receiver
  - Line 19: set the port of the receiver
- Line 22-26: define a experiment. When all the nodes are ready, execute the application
  - Line 23: wait for 10 sec to make sure the hardware are ready
  - Line 24: run the experiment
  - Line 25: wait for 600 sec for experiment execution
  - Line 26: terminate the experiment

Note: the results/outputs will temporarily stored on the local nodes and then sent to the OMF server when the WiMAX network turns to be available while all the experiments or applications stop. This experiment description can be changed based on the users' experiment requirements.

3.3 VInsight Source Code

With the VInsight source code, you will have a directory called VInsight under 'Download' or package unzipped directory. Change into the VInsight directory, and you should see a file named main.c. This is the file that is the main interface of the experiment engine. In the following, we will

take a look at the source code by starting with the main.c.

3.3.1 Main Function

This is just the declaration of the main function of VInsight. It is the first function run.

```
int main(int argc, char *argv[])
```

The next line is used to enable essential daemons with initialization of basic variables and databases for the mobile platform.

```
initialization();
```

```
/*Definition*/
void initialization(void)
{
   printf("************************************************************\n");
   printf("*****************Welcome to VInsight*********************\n");
   printf("************************************************************\n");
   // initialize the thread subsystem
   thread_init();
   // initialize setting to defaults
   setting_init();

   //    perform any cleanup when quiting the experiment or thread
}
```

The next 7 lines define a loop to detect and recovery failures of the hardware or software of the mobile platform.

```
 while(1)
   {
      sleep(5);
      if(stop_flag == true) {
         terminate_all();
      }
   }
```

The next line is to call the function cleanup() to terminate all the software or resources occupied by vinsight when the program exits.

3.3.2 Start_engine Function

The function start_engine is to create threads resource_manage, experiment_controller, health_monitoring, listen_to_server, initial the lists and create the local database to store the experiment output.

```
void start_engine()
{
```

```
// initial the job list and create the local database to store the experiment output
  jobs_list_init();
  msg_list_init();
  outputs_list_init();

  stop_flag=false;
  reserve_flag=false;

 // create the threads
 th_id[0]=pthread_create(&entities[0],NULL,resource_manager,NULL);
 th_id[1]=pthread_create(&entities[1],NULL,experiment_controller,NULL);
 th_id[2]=pthread_create(&entities[2],NULL,health_monitoring,NULL);
 th_id[3]=pthread_create(&entities[3],NULL,processing_messages,NULL);

 //wait for all threads to complete
 sleep(5);

}
```

The following two lines are used to set the variable stop_flag and reserve_flag for emergency stop and resource reservation, respectively.

Then four threads are created for resource management,experiment control, health monitoring, and processing_messages.

3.3.3 Resource_manager Function

The function resource_manager focus on the reservation and maintenance of the resources belong to the mobile platform.

```
void *resource_manager(void *arg)
{
   printf("s1-resource manager.\n");
   thread_set();
   device_init();
   while(1){
   pthread_mutex_lock(&sys_task_mutex);
   if(reserve_flag == true && jobs_num == 0){
          device_close();
          device_init();
          reserve_flag = false;
      }
      sleep(5);
      pthread_mutex_unlock(&sys_task_mutex);
   }
```

```
}
```

Where, the function thread_set() runs to set the state of the thread to cancelable. device_init() is to check the status of devices attached to the mobile platform. Note that different number of devices can be connected to the mobile platform. The users can change the function device_init() in order to tailor VInsight based on the specific conditions.

If reserver_flag is set to true and no jobs is submitted to the mobile platform, the devices are reset for maintenance.

3.3.4 Experiment_controller Function

The function experiment_controller is responsibly for reading the jobs from list and execute them according to their types: normal experiment,maintenance,and emergency stop.

```
void *experiment_controller(void* arg)
{
   printf("s-2 experiment_controller\n");
   thread_set();
   while(1)
   {
      if(execute_jobs()==false)
      {
         printf("ERROR: Experiment can not be started due to the device failure!\n");
         printf("ERROR: Please check the devices\n");
      }
       // have a short time rest, then execute the next job
         sleep(5);
   }
}
```

Then a short time rest will be triggered to ensure the devices to be released or reset to the right state.

3.3.5 Health_monitoring Function

To check the health of the software and hardware of the mobile platform, the function health_monitoring is triggered when the experiment engine starts.

```
void *health_monitoring(void* arg)
{
   job *temp;
   printf("s-5 health_monitoring\n");
```

```
  thread_set();
  while(true)
  {
   sleep(HEARTBEAT_INTERVAL);
   //check if there is any health monitoring job in the list
   if(heartbeat_existence()==false)
   {
      temp=(job *)malloc(sizeof(job));
      temp->priority=MAINTENANCE;
      temp->status=SUBMITTED;
      sorted_insert(temp);
   }
  }
}
```

For efficient resource utilization, the function heartbeat_existence will be run to check if there is any maintenance jobs in the list before submitting a new one.

3.3.6 Data_collection Function

According to the experiment description, each experiment runs the function data_collection by collecting real-time sensing data in the networked VSC network.

```
void *data_collection(void* arg)
{
  job *item=(job *)arg;
  printf("s-3 data_collection\n");
  thread_set();
  // configure the devices
  device_config(item);
  while(item->mode[0]==true ||item->mode[1]==true)
  {
      if(item->mode[0]== true){
      while(is_gps_ready()==false) {}
      store_gps_data(item);
      }
      if(item->mode[1]==true)    {
         signal_status(item);
         link_delay(item);
     }
      sleep(1);
  }
}
```

The functions store_gps_data, signal_status, and link_delay are used to record the vehicle position

and WiMAX network link quality.

3.3.7 Data_transmission Function

There are two ways of transmitting the experimental result to the base station server or other mobile platform.

oml_data_trans(msg *current) defines and implements the data transmission on behalf of users. After data processing, the data streams are forwarded by the OML client to the OMF server. The OMF server stores them in a SQL database created for the experiment.

normal_data_trans(msg *current) is developed for users who have not had the OMF/OML on their system. In such a condition, the
mobile platforms act as servers and send the live data stream to the base station server or other mobile platforms via TCP connections.

```
void data_transmission(msg *current)
{
   int mode;
   mode = job_mode;
     switch(mode)
     {
        case OML:
                oml_data_trans(current);
                break;
        case UN_OML:
                normal_data_trans(current);
                break;
        default:
                break;
      }
}
```

3.3.8 Processing_messages Function

The EE listens to the server via this function, and creates a thread to process the messages received from the base station.

```
void *processing_messages(void* arg)
{
   send_connection_msg();
   while(1)
   {
```

```
    if(is_messages_list_empty()==false)
    {
      handle_msgs();
    }
  }
}
```

The function send_connection_msg() is used to build connection between the mobile platform and the base station server for control/management message transmission.

The function handle_msgs() is to processing the msgs received or issued by other threads.

3.4 WiMAX Measurement Tool

This function illustrates the basic building blocks of the WiMAX measurement tools.

```
int main(int argc, char *argv[])
{
  job *item;
  int duration;
  // read the arguments
  if(argc <5)
  {
    printf("Usage: viwimaxtoo filename Server_IP Server_Port Duration\n");
    return 0;
  }

  //initial the wimax dongle

  if(wimax_dongle_init()==false)
  {
    printf("Please check the WiMAX dongle.\n");
    return 0;
  }

  //create a new job with several setting for wimax network measuremnt
  item=(job *)malloc(sizeof(struct JOB));

  memcpy(item->output,argv[1], strlen(argv[1]));
  memcpy(IP, argv[2],strlen(argv[2]));
  item->mode[1]=true;

  duration=atoi(argv[4]);
```

```
    // create a new thread for experiment execution
    ex_th_id=pthread_create(&ex_thread,NULL,data_collection,(void *)item);

    sleep(duration);

    device_close();

    pthread_cancel(ex_thread);
}
```

To provide a usage guide of viwimaxtool, the following commends are firstly conducted.

```
if(argc <5)
    {
        printf("Usage: viwimaxtool filename Server_IP Server_Port Duration\n");
        return 0;
    }
```

filename indicts the name of the file used to restore the measured data.
Server_IP refers to the IP address of the server to which the mobile platform connects.
Server_port refers to the port of the server that listen to the connection and link quality tests
Duration denotes the time length that the WiMAX network measurement conducts.

It is then followed by checking the availability of the WiMAX dongle which is used to connect to the base station or other mobile platforms.

```
    if(wimax_dongle_init()==false)
    {
        printf("Please check the WiMAX dongle.\n");
        return 0;
    }
```

As long as the wimax dongle is attached to the mobile planform with the right driver mentioned in Chapter 2, the return value of function wimax_donle_init() should be true. Before collecting the data, a structured variable item is created to carry the configuration parameters of the WiMAX network measurement tests.

NOTE: job is a basic concept in VInsight. Please refer to the pervious subsections or the source code for a basic understanding.

The function data_collection is activated when the thread is created. All input to this function is obtained from variable item and all output is written into the file with the name of “filename”.

```
ex_th_id=pthread_create(&ex_thread,NULL,data_collection,(void *)item);
```

The function main waits for a curtain time which is indicated by duration.

```
    sleep(duration);
```

Then it terminates the created thread by using the command:

```
pthread_cancel(ex_thread);
```

Towards better understanding of Vinsight, please look at the other functions in the source code.

3.5 Testing VInsight

You can run the tests of the VInsight on the mobile platform side by running “vinsight”,

```
$ vinsight
```

On the server side, you can start the server to receive the mobile platforms’ connection request and perform controlling and management on the experiments executed on the mobile platforms.

```
$ viserver
```

Besides, the WiMAX network measurement tool is also open to use by users. The simplest way to use is typing

```
$viwimaxtool filename Server_IP Server_Port Duration
```

Chapter Four

4. Conclusion

This document is intended as a living document. There might be some error or mistake with this document. We hope and expect it to be in a better shape and cover more and more of elements of VInsight over time.