MONROE

Measuring Mobile Broadband Networks in Europe

H2020-ICT-11-2014
Project number: 644399

Deliverable D2.3
Robust Node Recovery Method

Editor(s): Stefan Alfredsson and Anna Brunstrom
Contributor(s): Roberto Monno, Ozgu Alay, Thomas Hirsch

Work Package: 2 / Hardware Design and Node Deployment
Revision: 0.1
Date: November 30, 2016
Deliverable type: R (Report)
Dissemination level: Public

Abstract

This report describes the design and implementation of robust node recovery and the associated maintenance procedures within the MONROE platform. In order to monitor the operational status of the deployed nodes and take corrective actions, a set of watchdog functionalities have been designed and implemented. Further, the procedures designed for robust node recovery enables a hard restart, automatically taking the system back to a known working baseline. Finally, to handle node failures due to hardware malfunction, procedures for identifying a hardware malfunction as well as the procedures and responsibilities needed to recover a failed node have been defined.

Participant organisation name	**Short name**
SIMULA RESEARCH LABORATORY AS *(Coordinator)*	SRL
CELERWAY COMMUNICATION AS	Celerway
TELENOR ASA	Telenor
NEXTWORKS	NXW
FUNDACION IMDEA NETWORKS	IMDEA
KARLSTADS UNIVERSITET	KaU
POLITECNICO DI TORINO	POLITO

1 Introduction

The MONROE project aims to build and operate a novel platform to perform measurements and experiments in operational mobile networks (MBBs). The main objective of the project is to design and implement an European transnational open testbed for independent, multi-homed, large-scale monitoring and assessment of performance of MBBs networks in heterogeneous environments. In order to operate such a large-scale platform, appropriate maintenance procedures are required. The required maintenance procedures involve both hardware and software maintenance, including the replacement of a node or one of its components when necessary. Robust node recovery methods are required to minimize local intervention, which would be particularly expensive.

In this document, we describe the design and implementation of robust node recovery and the associated maintenance procedures within the MONROE platform. First, in Section 2 we describe the mechanisms designed to monitor the operational status of the deployed nodes. A set of watchdog functionalities have been implemented in order to detect problems in a timely fashion and take appropriate corrective actions. Section 3 describes the procedures designed for robust node recovery. The robust node recovery procedures enables a hard restart, bringing the system back to a known working baseline without manual intervention. The procedures defined to handle a node failure due to hardware malfunction are described in Section 4. This involves defining how a hardware malfunction is identified, specifying the different hardware problems that may occur, and defining the procedures and responsibilities to recover the node or component for each possible problem. Finally, some concluding remarks are given in Section 5.

2 Node Monitoring

In order to provide a robust system where we react to the problems timely, we implemented different watchdog functionalities. In Table 1, we tabulated the *symptom*, the *root cause* and the corresponding *repair action*. If each of the listed recovery actions by the watchdog fail, we define a *final action* to resort to:

- *Flag* - the detected error is not critical. The error will be flagged and reported in the metadata and system logs. The issue will be taken care of during the regular maintenance activities.
- *Reboot* - the issue will be resolved by a reboot of the node.
- *Reinstall* - the issue will be resolved by a reinstallation of the node from the boot OS.
- *Maintenance Mode* - the issue requires intervention by the maintenance team. Base and scheduled experiments will be stopped until the issue is resolved.

For example, if the core components (e.g. multi, network-listener, autotunnel) are not running, the most likely root cause of this is a misconfiguration, or a failed update or a crash in the system. When the watchdog detects this, it will restart or reinstall the corrupt service. After the restart/reinstall, the reboot of the system will take place as an final action. On the other hand, if the node temperature is higher than 95°C, the most likely root cause of this is CPU load or high outside temperature or bad ventilation. In this case, there is no automatic repair action, therefore the node goes into maintenance mode and the maintenance team needs to manually intervene the issue.

In this deliverable, we mostly considered the robust recovery of the nodes and implemented watchdogs on the node to repair the problems as quickly as possible. In the future, we are planning to also monitor the inventory and the database to generate alerts when there is a problem with the nodes or there are gaps/anomalies in the data generated by the nodes.

Table 1: *Watchdogs.*

Symptom	Root Cause	Repair action	Final action
No working NTP peer connections	Network down at service start, unresponsive servers in pool	Restart service	Flag
cron service not running	N/A	Restart service	Flag
/dev/watchdog is not written to	system freeze, etc.	None	Reboot
No successful autotunnel connection	Various	Reboot after 24 hours of uptime	Reinstall
Core components are not running (multi, network-listener, autotunnel)	Misconfiguration, failed update, crashed	Restart service, Reinstall service	Reboot
Node temperature >95°C	CPU load, outside temperature, ventilation	None	Maintenance Mode
HDD filled (less than 2GB left)	Misconfiguration, Experiments are unable to sync and delete data	Delete log files and apt cache	Maintenance Mode
Large number of processes open	Misconfiguration, experiment misbehaving (fork bomb)	Restart running experiments	Reboot
Unresponsive internal modem	Various	Restart network listener	Maintenance Mode
Main services not running (dlb)	Misconfiguration, failed update, crashed	Restart service, Reinstall service	Maintenance Mode
Auxiliary services missing (cron, sshd)	Misconfiguration, Crash	Restart service	Reboot
Auxiliary services missing (ansible)	Misconfiguration	Reschedule in cron	Reinstall
Auxiliary services missing (rsyslog)	Misconfiguration, crash	Restart service	Flag
Container system (docker) not running	Misconfiguration, crash	Restart	Maintenance Mode
Base experiments missing	Misconfiguration	Re-download base containers	Flag
Cannot read metadata broadcast	Exporter failed, network misconfiguration	Various	Maintenance mode
Half-installed packages present	Packaging system was interrupted	Continue installation/configuration Remove broken packages	Flag
Network addresses overlap	Mifis in factory setting have the same IP range	Set random network range	Flag

3 Node Recovery

The MONROE nodes have a wide geographical distribution, and manual maintenance is therefore costly in terms of time for the nearest maintenance personnel to travel and service a node. Therefore we have defined and implemented a robust node recovery method that enables a hard restart of the node. To be more precise, a "hard restart" means a reinstallation of the operating system to a known working baseline. This method is able to recover both from file system errors that prevents system boot-ups (that can occur due to frequent sudden power loss in mobile nodes), and also due to software configurations that may lead to loss of connectivity.

To achieve this goal, the nodes have a two-stage boot loader process when a node is started. In the first stage, the BootOS is started. It is kept entirely in RAM and only uses read-only hard-drive access for it's normal operation. The BootOS verifies that the filesystem of the APU is not corrupt, and that no forced reinstallation has been requested, and then proceeds to boot the MainOS, which contains the MONROE system software. If the filesystem is corrupt and unable to be automatically repaired, or if a forced reinstallation is requested, the BootOS reinstalls an image of a known working installation.

The technical details of this method is described in the following subsections.

3.1 The APU boot process

The APU boot process follows that of a standard PC system. When power is applied, the BIOS firmware is started. The BIOS consults it's configuration parameters with regards to the device boot order, and starts from the first available device. This is typically the main storage device, in our case the 16 GB mSSD unit.

From the storage device, the master boot record is loaded from the first sector, which consults the partitioning table to find the operating system loader. The storage partitioning scheme is shown in Figure 1.

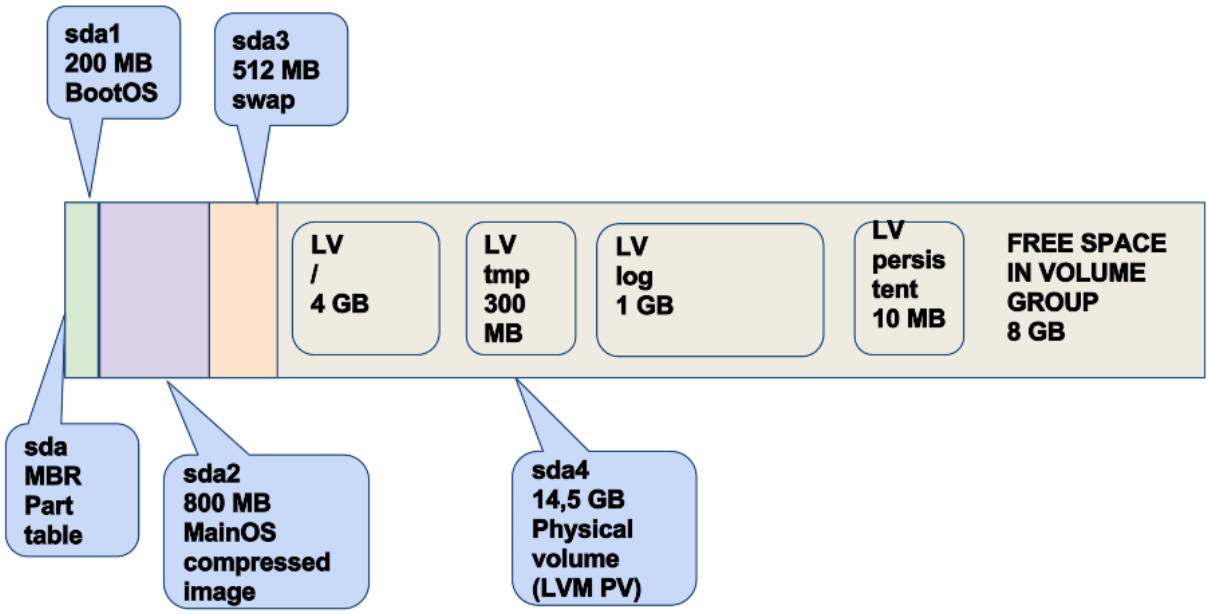

Figure 1: The node storage partitioning

The operating system loader (GRUB) is configured to load a kernel and ram disk from the first partition on the storage device `/dev/sda1`. The kernel and ram disk constitute the BootOS. The kernel is a standard Linux kernel, and is the same version used in the MainOS. The ram disk is a customized Debian initrd with added logic to validate the MainOS filesystem, try to recover from possible file system errors, or as a last resort, do a full reinstallation of the MainOS.

After a successful validation/recovery of the filesystem, the MainOS partition is mounted and that installation is chain-loaded via the `kexec` functionality, which does an in-memory replacement of the currently executing kernel and proceeds to start the new kernel.

3.2 BootOS and MainOS interaction

By design, the logic in the BootOS is kept as simple as possible, leaving the majority of the recovery logic to be handled and updated in the MainOS. This means that the BootOS should seldom need to be updated, since it is a critical operation that if it fails can render nodes unreachable and require manual maintenance, for example if power loss occurs during the update procedure.

Besides triggering recovery on a corrupted filesystem, the BootOS also has a communication channel with the MainOS, implemented as a file in the MainOS file system. In this file, the BootOS maintains a boot counter, and a boolean flag that forcefully triggers a reinstallation. During normal operations and after validating the system functionality and network reachability, the MainOS resets the boot counter variable to zero.

In the event that the MainOS fails to boot, or for some other reason does not complete the booting and system verification such that the boot counter is not reset, it is therefore incremented at every reboot. Eventually, the BootOS will determine that a maximum number of boot has been reached, without the MainOS being able to reset the counter. The BootOS then proceeds to do a reinstallation of the MainOS.

In the event that the system completes booting, but (for example) fails to attach to the mobile broadband networks on any interface during some time period, and having performed a number of reboots, the MainOS watchdog functionality can itself request a forced reinstallation to try and recover (see also Table 1 above. For example if a software update causes the network drivers to malfunction, or incorrectly configure routing, this can be remedied by a reinstallation to a last known good installation, as a last resort[1].

For the implementation of this communication channel, we re-use the grub-editenv tool to manipulate the file `/.bootos`. The file is using an simple ASCII-based format and the file is easily parsed into environment variables for processing in scripts or compiled programs. Further, the file is having a fixed-size allocation of 1 kB, to avoid problems if the storage device is full.

To use the variables in a shell script, they can be listed and evaluated with the following commmand:

```
eval $(grub-editenv /.bootos list)
```

The current value of a variable is then available as a regular environment variable, e.g. `echo $BOOTCOUNT`

The MainOS can use this to validate that the bootos has been executed, by checking that `$BOOTCOUNT` is larger than 0. If it is not, this means that the boot counter has not been incremented by the BootOS, and therefore has been bypassed, and therefore the recovery mechanism is also bypassed if it would be needed.

When recovery is initiated, the BootOS enables logging, establishes a reverse ssh tunnel to enable manual inspection in case that automatic recovery fails, and also fetches a per-node specific script to allow, for example, halting or pausing the recovery procedure.

To enable individual keys and configuration, the MainOS can put configuration files in a shared storage are on the image partition (`/dev/sda2`).

[1] The watchdog system also tries to revert package installations in an attept to recover from a failed package update.

The files needing individualization are

- The authorized_keys file which lists the keys that are allowed remote login
- A private rsa key which is used to access the tunnelbox
- The autotunnel configuration file
- The node id which determines which SSH port to use

If more files need to be individualized, this can be implemented in `local-top/graft-identity`.

3.3 BootOS internals

The MONROE BootOS is based on the standard Debian initramfs boot sequence, with the specific extensions hooked in the 'local-top' stage.

The boot flow, in pseudo-code, is the following:

```
/scripts/local-top/monroe
 - if 'install' in /proc/cmdline:
   /opt/bootos/partition-drive.sh
   /opt/bootos/install-bootos.sh
 - if 'reinstall' in /proc/cmdline:
   /opt/bootos/install-bootos.sh
 - if 'upgrade' in /proc/cmdline:
   /opt/bootos/upgrade.sh
 - if 'shell' in /proc/cmdline:
   /usr/bin/bash
 - if 'remoteshell' in /proc/cmdline:
 - else:
   /opt/bootos/bootmainos.sh # usually kexec's and never returns
```

If booting of the MainOS fails at the last step, execution falls through

```
/scripts/local-top/netdevices
   network-listener &
   dlb &
/scripts/local-top/networking
   multi-client &
/scripts/local-top/ntpdate
/scripts/local-top/logging
   stunnel &
   rsyslog &
/scripts/local-top/recovery
   /opt/bootos/recovery.sh
   /opt/bootos/reboot
```

To explain this further, at boot the kernel is started and assigned the initramdisk. This sets up the system and performs the steps in `/scripts/init-top`, `init-premount`, and `local-top` (See https://manpages.debian.org/cgi-bin/man.cgi?query=initramfs-tools)

In `local-top`, the following files exist: `graft-identity logging monroe netdevices networking ntpdate recovery`

Via a dependency/pre-requisite system, the `monroe` script is executed first, which checks if the system should boot normally, install, reinstall, upgrade or execute a debug shell for troubleshooting via serial console. If the system is instructed to install itself, it will invoke drive partitioning and installation continues booting via `/opt/bootos/partition-drive.sh` and `install-bootos.sh`.

If the system is instructed to boot, it continues booting via `/opt/bootos/bootmainos.sh`. This script does sanity checking of the MainOS filesystem and checking semaphores used for communication between the bootos and MainOS. If all is well, the MainOS kernel and initramdisk are executed via kexec and booting of the MainOS commences. If there is a problem with the file system consistency, or that the MainOS has not responded for a number of boots (or has requested a reinstallation), the script exits.

Execution then continues in the `local-top` directory, where the 'recovery' part depends on logging, networking, and ntpdate parts. After these components are started, the recovery component starts `/opt/bootos/recovery.sh` which handles reinstallation by unpacking an image of the MainOS. If the image is missing, incomplete or is corrupt, it is fetched from the image distribution server `dist.monroe-system.eu`, using the network path setup by the core components (`network-listener`, `multi`, `dlb`). Concurrently, the BootOS also enables remote syslogging and opens a reverse SSH tunnel to enable manual remote access. The BootOS also fetches a command script (if enabled and available) such that it is possible to interrupt, pause, automate or extend the recovery for unforeseen failure modes. To validate the integrity of both the system image download as well as scripts, there is a two layered security model. First, the image and scripts are transferred via a secure-shell (ssh) channel. Second, the image and scripts are signed with public-key cryptography using pgp (in practice the gnupg tool) to ensure that the image and script that are loaded and executed on the node have been signed by the project maintenance staff.

3.4 Building and installing the BootOS image

The building of the initramdisk is done in a docker container, which maps in a number of directories on the host. From here, the script `build-initramfs.sh` installs the needed software components, does various configurations, and then invokes the rebuilding of the initramfs.

The initramfs rebuild uses 'hooks', see `build-files/hooks/monroe-bootos-binaries` for copying executable files, libraries, and other files from the docker container filesystem into the initramdisk.

Apart from various package contents and executables, there are also a number of custom files copied from the `filesystem` directory; for example `filesystem/opt/bootos/` contains the files described above for handling the chained boot, recovery and installation procedures.

Additionally, the `filesystem-secret` directory contains authentication credentials, such as ssh keys and certificates for the syslog SSL tunnel. These are not available from the public repository, but are kept in the Celerway github private repository. This extra file-system is concatenated to the newly created initramdisk; this allows for separation of development of the bootos, and the credentials needed to access the monroe platform/backends.

After building, the BootOS can be installed in several ways. It is packaged as ISO-9660 image (the standard CD-ROM format) with multiboot features. It is therefore possible to boot via a CD-ROM device, or written to a USB memory stick, via network boot (PXE + memdisk). It also contains an conversion functionality, so nodes can be remotely converted from another installation or have BootOS upgrades.

The same kernel and ram disk is used for both installation, conversion and normal operation. The choice

of operation is determined via kernel parameters. Using the 'install' parameter starts the normal installation process with a clean installation, while using the 'upgrade' parameter causes parts of the existing files to be preserved (for example node certificates). If no parameter is supplied, it assumes the role of an installed system and performs the procedures to boot the MainOS.

The installer images are kept at a distribution site and are accessible via scp/rsync from `dist.monroe-system.eu:/home/dist/files/bootos`. Due to the sensitive nature of this image (it contains authentication keys to send logging data, keys to download software components, and keys for setting up remote tunneling), access is restricted to the project maintenance personnel.

3.5 MainOS image

The bootos uses a simple reinstallation strategy, where a filesystem archive is unpacked onto a newly formatted filesystem. The filesystem archive consists of a compressed and signed tar-ball of the MainOS filesystem from a new MainOS installation from the monroe-installer debian installation ISO media.

These are the steps to recreate the installation. The image can either be created from a node installation, or alternatively from a virtual machine. In the steps below, we use the QEmu virtualization engine (http://wiki.qemu.org/Main_Page) but other engines such as VirtualBox or VMWare Workstation would work just as well. At the install stage, the Ansible (https://www.ansible.com/) system provides configuration management tasks to automate installation of software components and automates the system setup.

1. Create a qemu disk image of sufficient max size:

   ```
   qemu-img create -f qcow2 ~/monroe-vm.qcow2 10G
   ```

2. Boot the installation media (created by the monroe-installer reimager script, see https://bitbucket.org/ozgualay/monroe-installer/src/).

   ```
   sudo qemu-system-x86_64 -m 2G ~/monroe-vm.qcow2 -enable-kvm -serial stdio \
                           -cdrom out/debian-8.4.0-amd64-netinst.iso
   ```

3. Let the system complete the installation, and let it run for at least 30 minutes to allow for ansible to install core components, upgrade the kernel, etc. If the installation halted, restart it with

   ```
   sudo qemu-system-x86_64 -m 2G ~/monroe-vm.qcow2 -enable-kvm -serial stdio
   ```

4. Check the ansible crontab interval (typically 20 minutes) and make sure there is sufficient time to perform the following steps before ansible-wrapper is invoked next time.

5. Clean up - IN THE VIRTUAL MACHINE, not the build host:

   ```
   systemctl stop syslog
   for log in $(find /var/log/ -type f) ; do truncate --size 0 $log; done
   apt-get install localepurge
   localepurge
   ```

```
apt-get clean
apt-get autoremove
rm -f /etc/nodeid
echo 'Monroe-X' > /etc/hostname
```

6. Capture an image of the installation:

```
sudo apt-get install libguestfs-tools # if needed
sudo mkdir /mnt/monroe
sudo guestmount -a monroe-vm.qcow2 -m /dev/Monroe-X-vg/root /mnt/monroe
sudo guestmount -a monroe-vm.qcow2 -m /dev/sda1 /mnt/monroe/boot
sudo XZ_OPT=-6 tar --one-file-system -Jcf mainos-$(date +%Y%m%d).txz \
                   -C /mnt/monroe . boot
sudo umount /mnt/monroe/boot
sudo umount /mnt/monroe
```

7. Add the digital signature:

```
gpg --sign --local-user 0x8E8812DF mainos-$(date +%Y%m%d).txz
```

   (pgp keys are currently with Stefan A, and kept in escrow with Jonas K)

8. Copy the image to the distribution server:

```
rsync -avP mainos-$(date +%Y%m%d).txz.gpg \
           root@dist.monroe-system.eu:/home/dist/files/mainos
```

9. Test the installation image on nodes, for example using

```
mount /dev/sda2 /image
rsync dist.monroe-system.eu:mainos-$(date +%Y%m%d).txz.gpg \
      /image/mainos.txz.gpg
umount /image
grub-editenv /.bootos set FORCEREINSTALL=1
reboot
```

   and the new image should be installed. This is evident from the "Monroe-X" hostname when the node is rebooted; this is updated when ansible-wrapper run.

10. If image testing is OK, the image is activated for default distribution by repointing the symlink to the newly pushed image:

```
ssh root@dist.monroe-system.eu
cd ~dist/files/mainos/
rm mainos.txz.gpg
ln -sf mainos-$(date +%Y%m%d).txz.gpg mainos.txz.gpg
```

4 Handling Hardware Failures

The MONROE node is composed of many hardware devices of different brands, quality and prices. The design of the system guarantees the integration of the hardware components in an unique schema, but cannot completely exclude failures in the hardware parts during the evolution of the project. In general terms, a hardware problem can arise from multiple and specific causes, e.g. a manufacturing defect, an improper usage (typically out of the range of the technical specifications), a high temperature of the blade, a crash, vibrations, etc.

During the assembly/deployment phase of the project, we have identified components that are more prone to generate problems due to the working conditions in which they operate. Unfortunately if/when a hardware failure occurs, the device can become unstable, not-reachable and, in any case, the results of the running applications can be affected and cannot be considered reliable.

The following sections describe the procedures we have designed to solve (or minimize) the causes of the malfunctions. These routines can be run remotely in most of the cases or on the site where the MONROE node has been deployed in the others. Sometimes it should be necessary to un-install the node, move it to the lab and identify the hardware board that is not working properly with a deep analysis of the problem. At the end, the broken component or the whole node should be replaced.

Two operative cases are analysed here: i) failures in the stationary nodes deployed at offices, houses or labs, and ii) failures in the mobile nodes installed in buses, trains or trucks.

In order to correctly manage the hardware failures and to substitute the broken hardware with a spare component, we have decided to set up a bug tracking system to store defects, errors, failures or hardware malfunctions. In short, we have decided to use the **github** open-source platform (the same we are currently using for the MONROE software repository) and its *issues* section. The responsible of the nodes in each county can open a "*ticket*" (or "*issue*") if he/she verifies a hardware failure. Each ticket should contain a title with both a clear identification of the problem and the node identifier in which the problem has been verified. Moreover the title should have the "*Hardware*" label that uniquely identifies errors in the hardware components. Of course, the ticket should report a clear description of the analysis already done, or the suspects of the sub-system that is not working properly. The procedures described below can give an indication of the different sub-systems and the problems that can be recognized.

Moreover we have also decided to distribute spare hardware devices among the partners (1 responsible per country) to minimize the shipping time.

4.1 Hardware failures in Stationary nodes

We have identified mainly 3 sub-components that can raise problems for the stationary nodes: the *power plugs and GSM socket*, the *USB hub* and the *ZTE modem*.

4.1.1 Power plugs and GSM socket

If the node is not reachable through ssh or a serial cable, a malfunction of the power supply of the device may have occurred. First verify that the device is correctly connected to the wall plug, then open the box and observe the green lights of the APU and the USB hub. In case of failure, we can have different scenarios:

- *Both APU and USB hub are off.*

  Check the leds of the GSM socket and verify that

  1. the POWER led is ON;
  2. the SWITCH led is ON;
  3. the STATUS led is a slow flashing red. Consider that a fast flashing red means that the device is continuously searching a GSM signal. In this case, it is possible that the device is placed in a bad coverage area and thereby the node should be moved to a better zone.

  If the three leds indicate a good status of the GSM socket, try to send a SMS to switch on/off the device. If the socket does not reply immediately or after a short period, it is needed to reset the component and resend a SMS. If the device is still not working, a new GSM socket should be tested. Install the new component in the stationary node and verify if this device works correctly. In this case, open a ticket in the *github* tracking system, describe (briefly) the problem you have verified and then ask for a spare component in order to replace the broken one.

- *APU is off.*

  Substitute the power plug with a new component. If the new plug works, then open a ticket to obtain a spare device. On the other hand, the APU can be damaged and the node must be replaced with a testing node. In this case, open a ticket to describe the problem and move the node in the hardware maintenance phase. In this phase, the broken device should be sent back to the Nextworks which takes care of the substitution of the component, the reassembling of the whole node and the preliminary tests of the basic functionalities of the device.

- *USB hub is off.*

  Substitute the AC/DC transformer and the related cable with spare components. Verify that the new devices are working and then open a ticket asking for spare components. Otherwise it is possible that the USB hub is broken. Try to replace the USB hub with a spare device and then open a ticket to require a new USB hub.

4.1.2 USB hub

If analysing the kernel logs of the APU system (*dmesg utility*), error messages related to the management of the usb ports can be seen, then the USB hub is working in unstable power conditions, i.e. voltage peaks or current spikes. In order to restore a stable environment, try to:

- disconnect the three ZTE modems from the output ports of the hub;
- check the connections and cables, then insert only one modem in the first port of the hub. Reboot the modem and the node. Check if the node is recovered and is working properly again;
- wait a quite long time (1 hour) to ensure a good level of the internal battery of the modem;

- repeat these steps for the others modems.

If the errors appear again, exchange the USB hub with a spare component. If the problem persist, replace the MONROE node with a testing device. The broken node should be moved to a hardware maintenance phase to be completely reassembled and tested.

4.1.3 ZTE modem

If the MONROE node is working properly and there are no errors in the internal logs of the system, but still a ZTE modem does not respond to commands, or the traffic is not forwarded correctly, or the modem is not connected to the mobile network, then try to:

- reset and then restart the ZTE modem;
- insert a new SIM card (possibly of a different operator);
- use a new modem and cable. In this case, open a ticket to request new spare devices.

4.2 Hardware failures in Mobile nodes

We have identified 3 different subsystems that can generate failures for the mobile nodes: the *DC/DC converters*, the *USB hub* and the *ZTE modem*.

4.2.1 DC/DC converters

If the MONROE node is not reachable even if the bus/train/truck is powered on or the USB ports are not detected by the operating system of the APU and there are no errors in the software modules, then a malfunction of the converters which provide the ingress power of the system may have occurred. In this case, it is necessary to move the device to the lab, open the box, switch on the node and verify the status of the lights of both the APU and the USB hub:

- APU is off. The converter X (24V, 48V or 110V depending on the type of the MONROE node) to 12V is broken.
- USB hub is off. The converter X (24V, 48V or 110V depending on the type of the MONROE node) to 5V is broken.

In both cases, the node should be replaced with a testing device. The broken node should be moved to the maintenance phase in order to be replaced with a new one (produced using spare components).

4.2.2 USB hub

There are no significant differences in respect of the procedures related to the stationary nodes. Please refer to Section 4.1.2.

4.2.3 ZTE modem

There are no significant differences in respect of the procedures related to the stationary nodes. Please refer to Section 4.1.3.

5 Conclusions

This document has described the design and implementation of robust node recovery and associated maintenance methods within the MONROE platform. Such methods are crucial for the operation of the platform in order to detect problems early, reduce manual intervention to a minimum and ensure that failed hardware components are identified and replaced in a timely manner. The implemented methods form a basis for the defined maintenance routines and maintenance activities carried out in WP4 of the MONROE project. While the initial design and implementation of the mechanisms for node recovery are soon completed, continuous optimizations and refinements are expected as part of the maintenance activities in WP4. All with the target to keep the platform available to its users for measurements and experimentation in a reliable and cost efficient manner.

Disclaimer

The views expressed in this document are solely those of the author(s). The European Commission is not responsible for any use that may be made of the information it contains.

All information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.