FFI-NOTAT

Eksternnotat 20/02876

Running norBMS over virtual radios

Authors
Tore Jørgen Berg
Prosjektnummer 1558
November 23th, 2020

Approvers
Åshild G Solheim, *Research Manager*
Jan Erik Voldhaug, *Director of Research*
The document is electronically approved and therefore has no handwritten signature.

Keywords
Radiokommunikasjon, mobilkommunikasjon, IKT

Summary
The Norwegian Army is dependent on radios for command and control on the tactical operating field. A radio network emulator that assists in training of the personnel and in testing new computer applications before deployment, may increase the efficiency and the reliability on the operating field.
Using the Extendable Mobile Ad-hoc Emulator (EMANE) open source framework, we developed a virtual radio network that is able to serve IP traffic from real applications in real-time. The virtual radio implemented is a typical military long-range narrowband radio.
The network emulator, implemented on a Linux server with a large number of processors, simulates up to 99 radio nodes and provides a standard IP interface to external terminals. This document outlines the software architecture and the services provided by the emulator.

Contents

1 Introduction

The Norwegian Army must use radio networks for command and control on the operating field. A radio network emulator is a tool that can be used to test command and control applications before deployment. This tool may also assist in training of the personnel.

Figure 1.1 illustrates two headquarters and a convoy attached to a shared tactical network. If we are able to build a realistic digital copy of the tactical network, operating personnel can be trained in simulators.

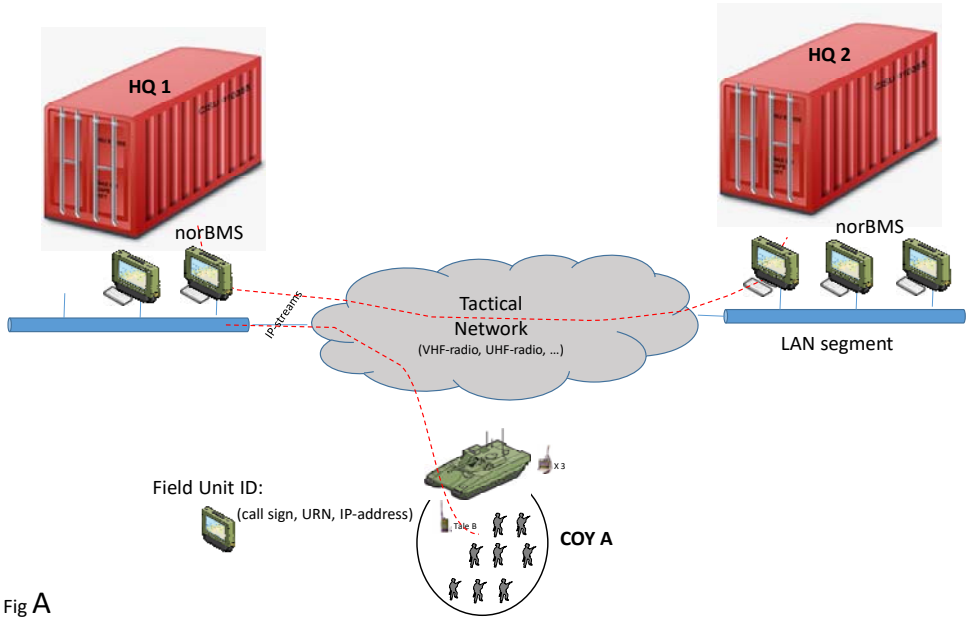
A Battle Management System (BMS) is an important computer application in the tactical area. norBMS is a BMS in use by the Norwegian Army. norBMS may communicate over a broad range of IP bearers. To achieve sufficient radio coverage in the Norwegian terrain, a narrowband VHF radio must often be used[1]. A network radio emulator should therefore implement a VHF radio which is able to serve norBMS terminals.

In FFI project "Kampnær IKT" a feasibility study was initiated to uncover the strength and weakness to mix physical terminal equipment and virtual solutions. Project "Kampnær IKT" included cloud based services as well as solutions on dedicated local servers. One task in this project was to gain experience with virtual radio network to get answers to the questions:

1. Is it possible to provide realistic IP throughput/delay performance of a narrowband tactical radio? The IP clients shall experience performance close to the real world performance.
2. How shall we adapt the radio link connectivity in real-time according to user mobility?
3. How shall we provide a GPS service to external terminals?

The scope of this document is the virtual radio network – its external interfaces and the software architecture.

[1] Military tactical radios working in the Very High Frequency (VHF) range (30 to 88 MHz) provide a IP throughput capacity below 1200 bytes/s.



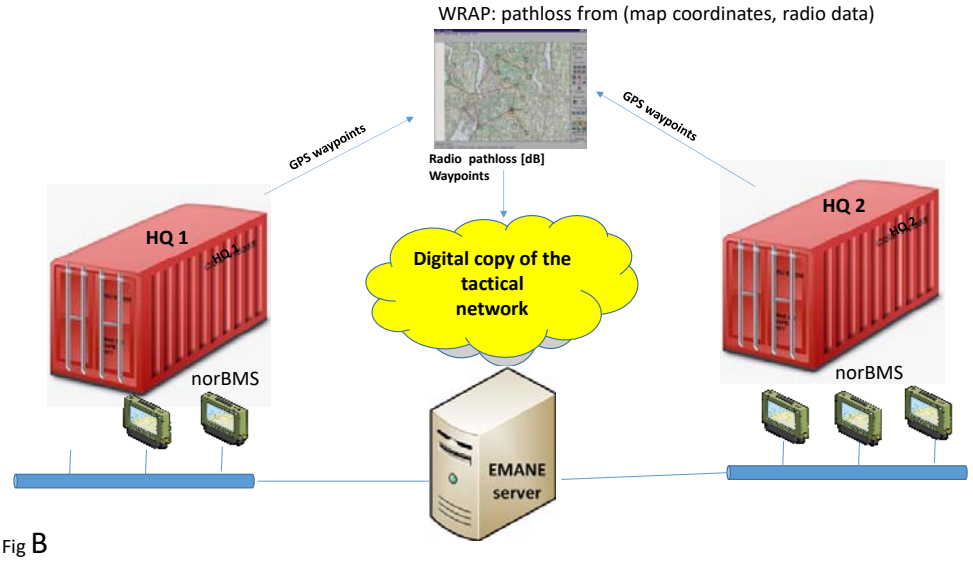


Figure 1.1 Use case example: training of operating personnel by means of a digital copy (figure B) of the tactical network in figure A.

2 Software Architecture

The Extendable Mobile Ad-hoc Emulator (EMANE) is an open source framework[2]. This framework provides software functions to build wireless network emulators that are able to serve real-time traffic from physical terminals or virtual terminals. Narrow Band Waveform (NBWF) for VHF/UHF radios is a NATO standard, references [1, 2, 3]. We have implemented a simplified NBWF radio as a software module in the EMANE framework using radio parameters from [5]. This software "product" is referred to as *eNBWF* and is a collection of Python modules and C++ programs, see Figure 2.1. We have checked the eNBWF model accuracy by means of a steady-state simulator developed in an earlier project [6][3] and conclude that the error is sufficiently low compared to the uncertainty in the radio link pathloss estimates.

External IP terminals are connected to a dedicated LAN segment, numbered as interface 4 in Figure 2.2. The EMANE server (E-server) provides an Ethernet port that gives access to the virtual radios. This port uses 802.1Q tagging (VLAN tagging). The pair (IP address, VLAN ID) identifies each radio interface. See section 2.1 for further details.

An important component in the emulator is WRAP[4] – a computer application which calculates radio link pathloss in real-time based on maps, radio models and radio data. The emulator receives synthetic waypoints from external equipment attached to the management LAN. WRAP calculates the radio link pathloss matrix and sends the matrix to the E-server. Reference [4] outlines the processes involved. The E-server also receives the waypoints and forwards these waypoints to the internal GPS simulators which emit GPS NMEA[5] messages to the terminals as described in chapter 5.

The E-server provides a service to override the WRAP pathloss matrix. This service facilitates tests with network jamming, or tests with well-defined static network topologies.

To test how computer applications tolerate a congested network is important. The run-time system has a service for implementing synthetic traffic generators[6] that may run without or in addition to the incoming external traffic.

The design of the E-server is based on link level relaying of the IP traffic between the external terminals and the virtual radios. If a terminal does not support tagging, a managed switch must

[2] https://github.com/adjacentlink/emane/wiki. eNBWF is based on version 1.2.1.
[3] This report is available online at https://www.ffi.no/en.
[4] WRAP is a commercial product, see https://wrap.se.
[5] https://www.gpsinformation.org.
[6] This service is implemented by using the open source generator MGEN, https://wrap.nrl.navy.mil.

be used to insert the correct tag. The benefit is a simple network layer setup – just set the IP address and the default gateway on the external terminals in the same manner as when connecting to a physical IP radio. Section 2.1 explains how the E-server executes internal routing.

Besides EMANE, this project uses many other open source frameworks and applications: Django, PySide2, Eclipse, MGEN and Qt. eNBWF uses only one software application with a license cost, WRAP .

The EMANE server uses three LAN segments, referred to as green, blue and red. Their purpose is as follows:

The green segment: This segment is dedicated for management traffic (WRAP traffic, NTP traffic and other processes used to control and monitor the emulator).

The blue segment: This segment is dedicated to input/output of real-time traffic between the emulator and the external terminals.

The red segment: This segment is dedicated to EMANE internal real-time traffic and cannot be accessed externally. The “radio RF waves” between the virtual radios are sent on this segment. If the emulator needs more processing power, more E-servers can be attached to this segment.

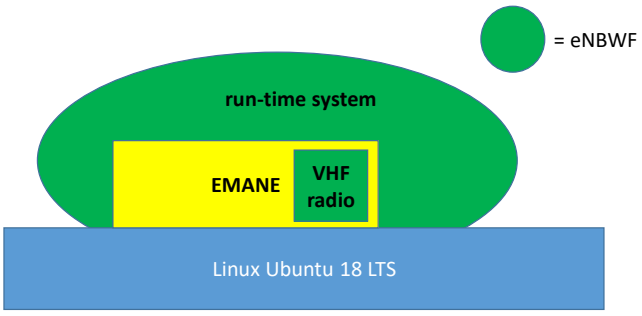


Figure 2.1 The green colour specifies the software implemented by eNBWF. The “VHF radio” is a C++ module within the EMANE core that runs in real-time. The “run-time system” is the Python modules designed to create the data structures, interfaces and processes required. All the eNBWF processes run in the Ubuntu user space.

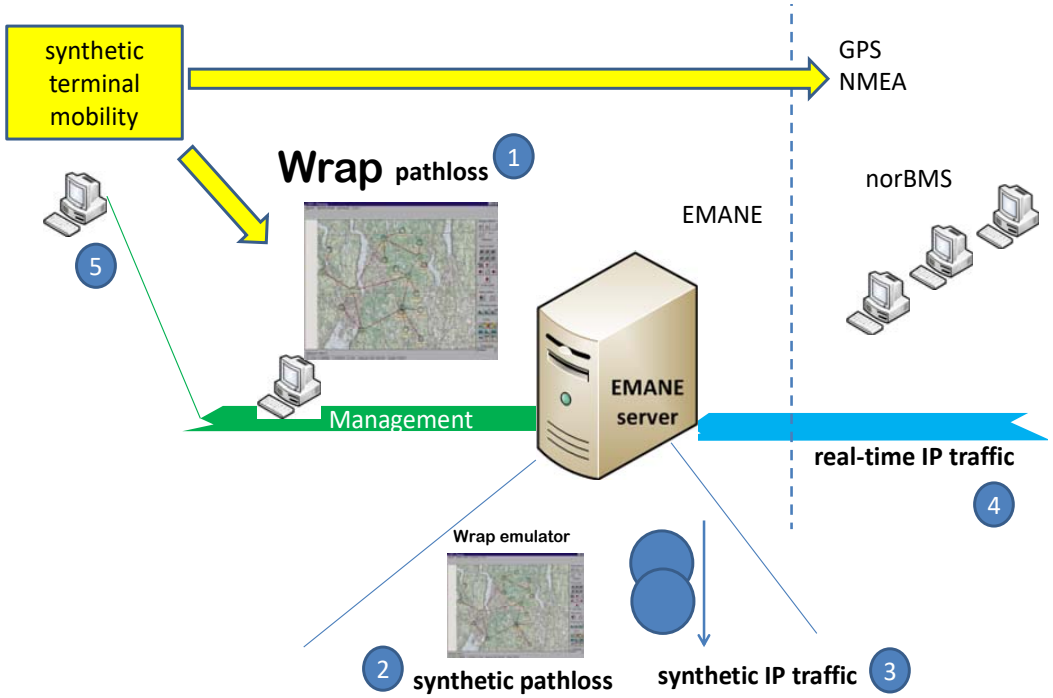


Figure 2.2 Overview. Radio link pathloss (1) is calculated on a dedicated PC and sent to the virtual radios over the management LAN. Pathloss values may also be set manually (2) which may be useful to simulate network jamming. External terminals are attached to a dedicated LAN segment (4). It is possible to simulate heavy traffic load scenarios by sending additional synthetic traffic (3).

2.1 VLAN tagging and internal routing

This section explains how the external IP traffic flows inside the EMANE server. The example used is two norBMS terminals attached to two radios. Figure 2.3 shows the IP addresses and the VLAN IDs used for this case. eNBWF demands a static binding[7] between the IP addresses and the VLAN IDs.

IP packets arrive over the blue LAN segment shown in Figure 2.2, which is the physical Ethernet device named *enp0s25* in Figure 2.4. Inside the host, each VLAN is connected to a container over a Linux bridge. A virtual radio is uniquely identified by a VLAN ID and the external IP address is only visible inside the container where the radio exists. Radio *Rn* is connected to IP client *Tn* via the virtual Ethernet device *eth3*. All the radios use the same device name but these devices have different IP addresses.

[7] The parameters cannot change in run-time but must be specified at the time when the top level data structure is created.

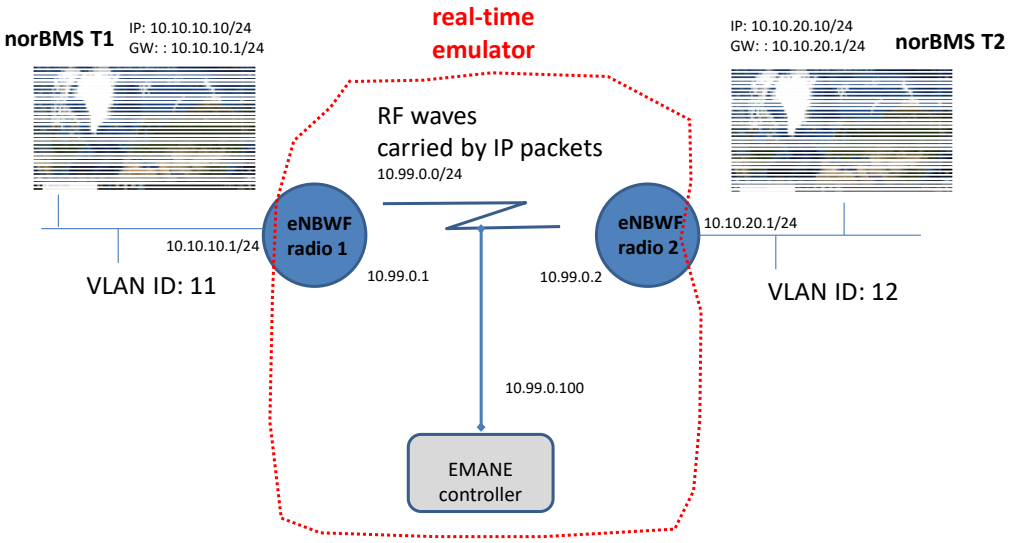


Figure 2.3 A two-node network serving two norBMS terminals.

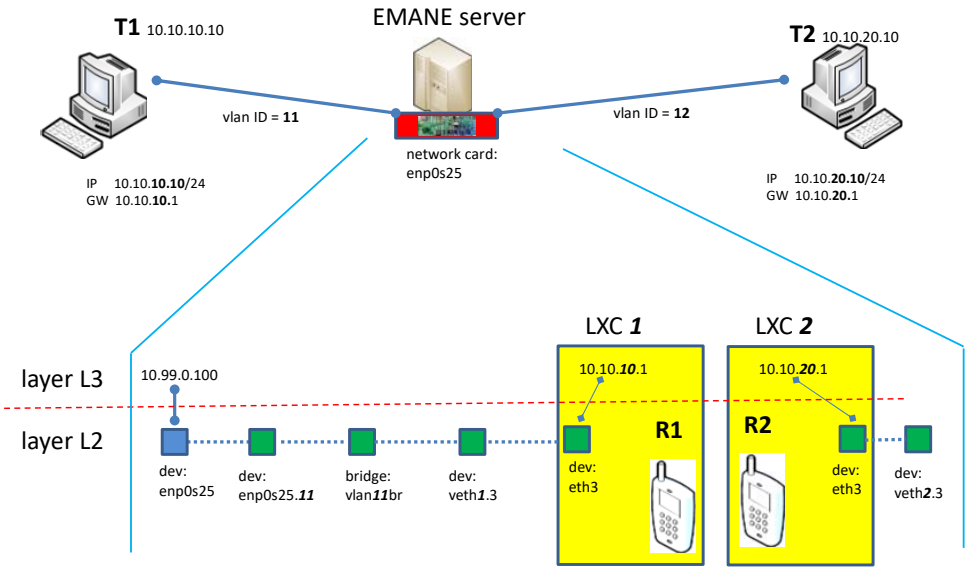


Figure 2.4 Layer 2 routing to the virtual Ethernet device eth3 inside the Linux Containers (LXCs).

The external traffic may now flow in/out of the containers and the next step is to connect to the radios. EMANE uses a Linux tunnel for packet transport between the user environment and the radio. The virtual device named *emane0* in Figure 2.5 is created for this purpose and this device

must be assigned a static IP address before run-time. eNBWF uses the default addresses inserted by the EMANE framework. The routing table in LXC *n* contains the IP addresses that can be reached by all the remote *emane0* devices in the network. The ARP cache in LXC *n* holds the pair (IP address, Ethernet address) for all the *emane0* devices that exist in the radio network.

Example: Incoming packet on LXC1::eth3 with destination 10.10.20.10. LXC1 route lookup: 10.10.20.10 → via 10.100.0.2 dev emane0. LXC1 ARP lookup: 10.100.0.2 →HW 02:02:00:00:00:02. Then the packet is sent down on emane0 and radio R1 inserts NEM IDs (source, destination) = (1, 2).

The radios are attached to the on-the-air (OTA) channel via the virtual device *eth1*. **Every packet** that arrives on *emane0* is sent down as a **multicast packet** on *eth1*. There is a fixed one-to-one relationship between the tuple (*emane0* IP address, *eth1* IP address, *eth3* IP address and *NEM ID*) specified in the main data structure. All the radios receive all packets and each radio must decide what to do with an incoming packet based on the NEM ID[8] and the signal-to-noise ratio.

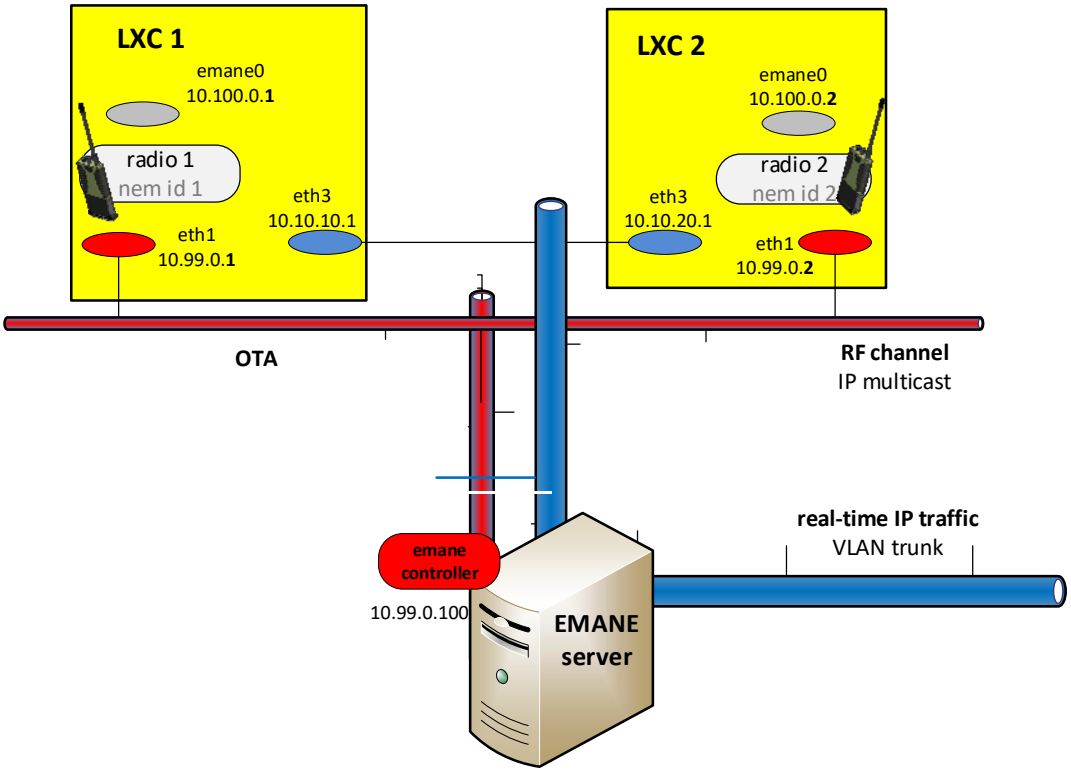


Figure 2.5 EMANE internal routing. Note the static binding eth1-emane0-eth3-nemid.

[8] IP addresses are not available in the C++ radio modules and each radio is addressed by an integer.

3 The run-time system

The run-time system is the software designed to *start* eNBWF, *monitor* eNBWF during run-time and *stop* eNBWF. The run-time system is implemented from scratch and is written in Python 3, a script based program language supporting type checking[9].

To run an experiment, you must conduct a number of steps **in sequence**:

- Specify the input data (e.g., network size and the services to use)
- Start the emulator
- Start the monitors
- Stop the emulator

A short description of each step is given in the next paragraphs.

Specify the input data
You specify a digital copy of the target tactical radio network in this step. This is done by changing the parameters in the data structure in Figure 3.1. The directory named *templates* contains the files that specify the experiment. This should be regarded as a read-only directory the first time you run an experiment. Later you can modify the files to fit your needs.

The directory named *testname* in the figure is assigned a name selected by you (e.g. *myFirstNovemberTest*). This directory contains all the files that the containers/processes use at run-time. The template directory is never modified by any eNBWF processes. A good practice is to save the *testname* directory together with the emulation output data to know exactly what you have emulated. Further information is given in section 7.1.

Start the emulator
An experiment is conducted by running a number of scripts in sequence. Some scripts are designed to be executed in a container (*wrapperFOO.py*), others on the server (*FOOmanager.py*), see Figure 3.2. To provide a friendly user interface, all scripts can be executed from a main Python module named *blab*, see section 7.2.

The eNBWF start-up process is complex and involves a large data structure and many computer processes. The data structure specifies how many virtual radios to create and the start-up

[9] https://www.python.org.

process makes one LXC[10] instance for each radio. When the LXC is up and running, the start-up process starts the EMANE process in the container.

*Start the monitor*s

Many processes and interfaces are involved and finding the cause of an error may be difficult. Monitors are tools that perform sanity checks and raise alarms upon errors. They also assist in debugging. Chapter 6 presents the monitors implemented.

Stop the emulator

It is important to do a graceful shutdown of all the running processes. If not, one or more run-time files may block new start attempts. The eNBWF stop scripts terminate all processes and delete all run-time files in the correct order. Further information is given in section 7.6.

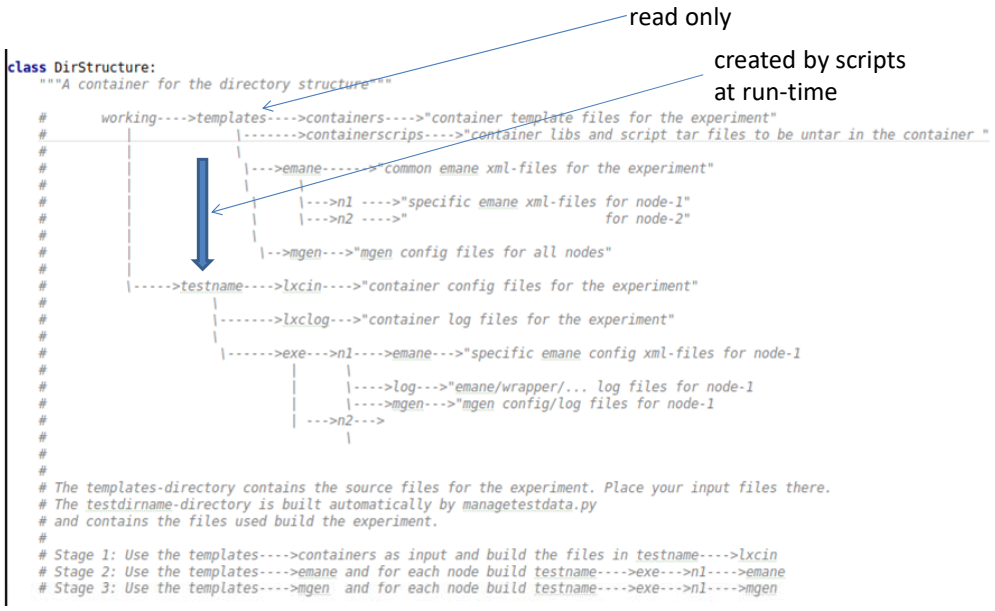


Figure 3.1 The eNBWF top level data structure which is specified in the Python file nbwfconstants.py.

[10] Linux container, see http://linuxcontainers.org/lxc.

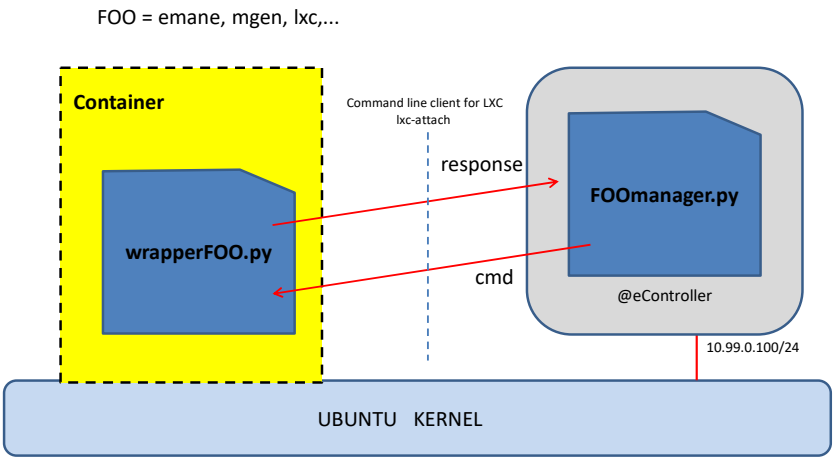


Figure 3.2 The Python module FOOmanager, which runs on the host, starts a complementary module wrapperFOO in one or more containers.

4 The WRAP server

Pathloss matrixes may be sent to the virtual radios from external equipment attached to the management segment in Figure 4.1. A server named “WRAP server” listens for incoming data on a dedicated port. This server is a part of the eNBWF run-time system and runs as a standard Linux daemon in the user space. When the server receives a pathloss matrix, it splits the matrix and then uses an EMANE service to set the pathloss tables implemented in the virtual radios addressed by the matrix.

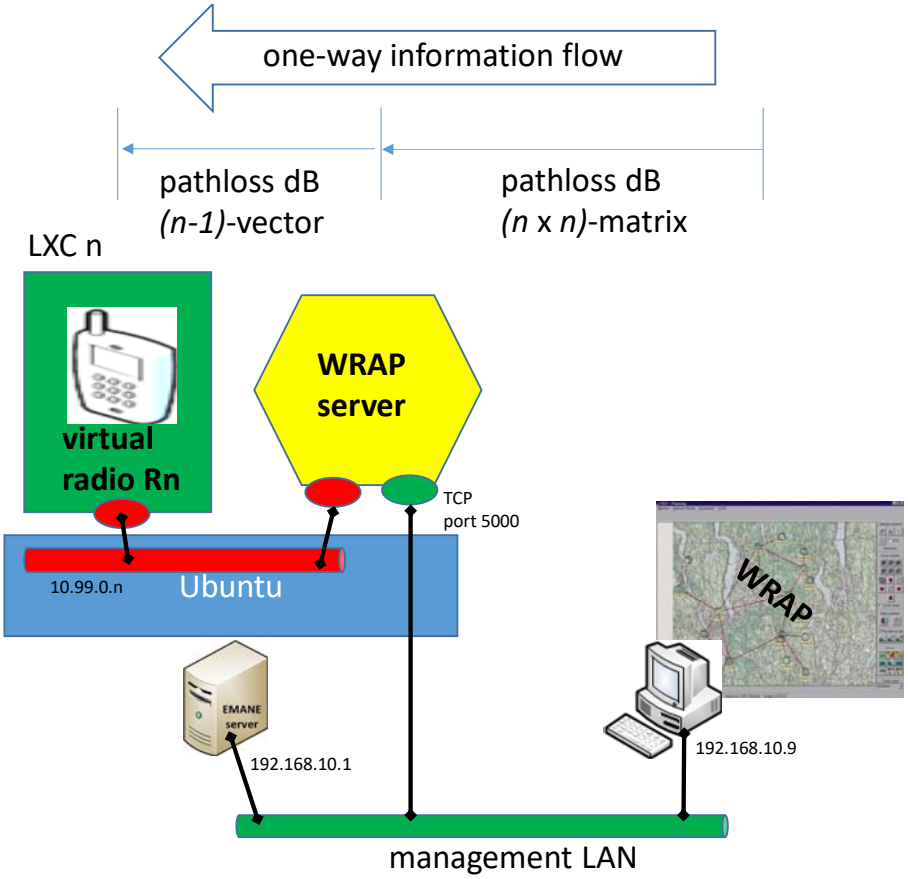


Figure 4.1 The WRAP PC sends the radio link pathloss matrix to the WRAP server which listens on port 5000 on the management LAN. Internally the WRAP server and the virtual radios communicate over a bridge painted in red colour.

The following script starts the WRAP server:

```
$ sudo ./python –m nbwf.wrapserver start
                --topdir /home/tore/project/norBMS/testNorBms/emane/netN2
                --testname march11
```

Hint: Use wrapserver –h to get information about the options.

The pathloss matrix signalling format is specified in the Python class *lxcGps::GpsSignalCsv*.

5 GPS service

norBMS requires GPS information for time synchronisation and for blue force tracking. For this reason the virtual radios provide a GPS service that emits GPS NMEA messages to the terminals, see Figure 5.1. The message rate is 1 packet/s to each terminal but this is not a problem since these messages are sent on the blue LAN segment only, and not on the radio channel.

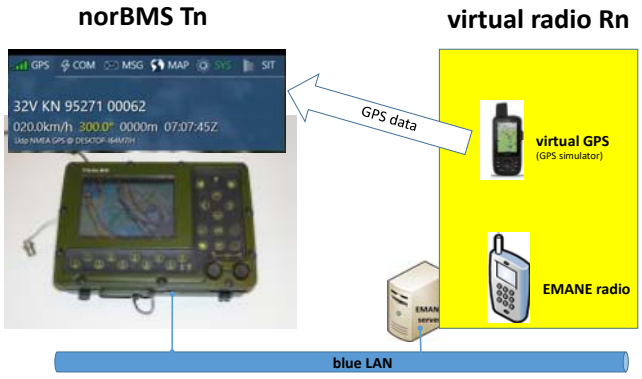


Figure 5.1 The terminals attached to the blue LAN segment may subscribe to the GPS service. A GPS simulator sends 1 packets/s and inserts the E-server's wall clock. The result is an accurate time synchronisation between the norBMS applications.

The GPS simulators may receive waypoints from external sources. In Figure 5.2, the GPS simulators get input from virtual generated forces. The eNBWF software also supports functions to set the geographic positions manually, or from GPX files.

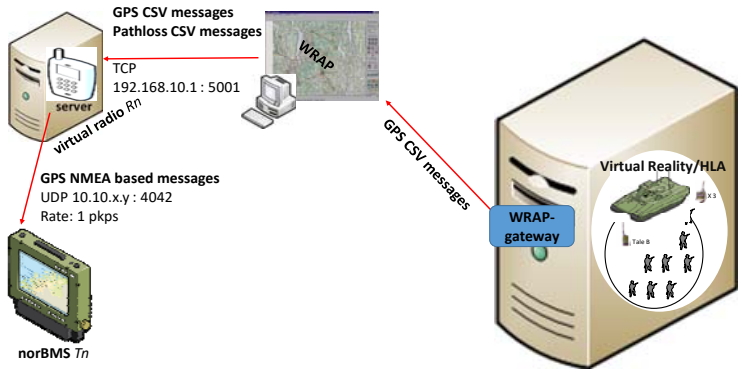


Figure 5.2 Example: A virtual reality generates flows of synthetic GPS signals.

eNBWF is designed to forward GPS information transparently from the WRAP PC to the norBMS terminals as shown in Figure 5.3. The Python module named *lxcGps* forwards the attributes in Figure 5.4 but inserts time stamps from the E-server's wall clock. The NMEA message rate is fixed at 1 packet/s regardless of the output rate from the WRAP PC. This guarantees that all norBMS applications have common date and time.

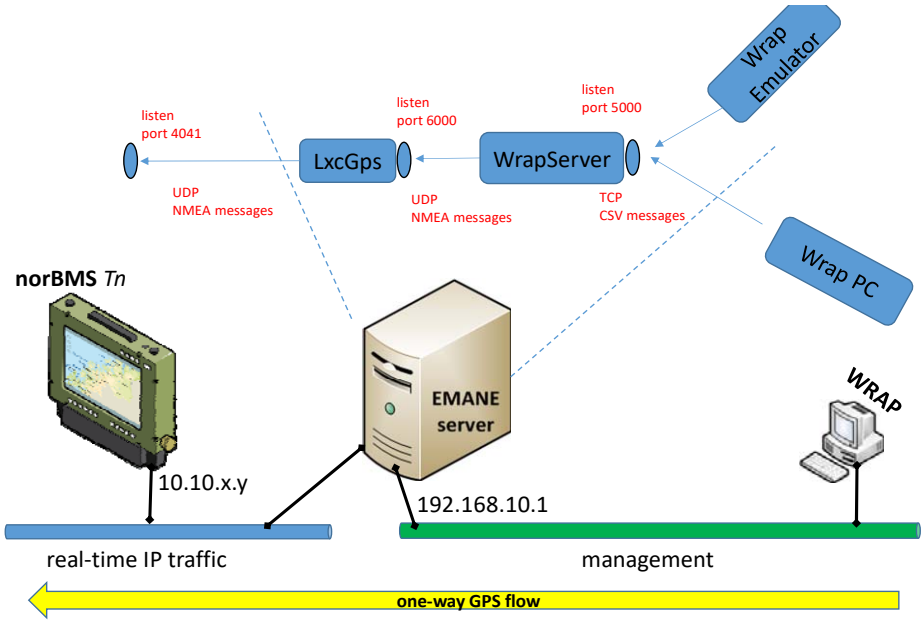


Figure 5.3 GPS message flow. The WRAP server, a process inside the E-server, converts the GPS CSV messages to NMEA format and inserts the GPS time stamp from its wall clock.

```python
# noinspection PyMethodMayBeStatic
class GpsSignalCsv:
    """
    GPS message ascii string without space or control characters.
    Format 1:
        "gps,seq=1,StationName=1,lat=59.9,latp=N,long=10.7,longp=E,ele=10.0,speed=15.0,course=200.0"
    Format 2: (single line)
        "gps,seq=1,StationName=1,59.9,N,10.7,E,10.0,15.0,200.0"
    """
    # Sentences supported
    nmea_sentences: List[str] = ['rmc']
    msg_type_names: List[str] = ['gps']

    # The attributes below shall be codes as CSV in sequence
    # *** start of sequence
    msg_name: str = 'gps'
    seq_no: int = 0  # incremented for each signal sent (per station?)
    terminal_number: int = 1  # destination terminal number as defined for pathloss
    latitude_degrees: float = 59.9  # oslo
    latitude_pole: str = 'N'  # N,S. oslo [59.9, N]
    longitude_degrees: float = 10.7  # oslo
    longitude_pole: str = 'E'  # W,E. oslo [10.7, E]
    elevation_meter: float = 10.0
    speed_kmh: float = 15.0
    course_degrees: float = 200.0
    # *** end ****
```

Figure 5.4 The GPS CSV message format between the WRAP PC and the E-server.

5.1 E-server internals

Figure 5.5 illustrates the internal components for handling GPS information. The Python module named *lxcGps* emits GPS signals to the norBMS terminals. *lxcGps* receives GPS signals from the *WrapServer* over the red LAN segment. The emulator must have functions to start and stop the *lxcGps* processes. The module *lxcGpsManager* performs these tasks by activating functions provided by *WrapperGps*. Since the *lxcGps* processes run inside the containers, the information exchange flows via the Linux system call *lxc-attach*.

The Python module *lxcgpsmanager*, which can run inside an EMANE console only, implements test sequences that can be used to generate periodic GPS CSV messages to the *lxcGps* module serving the norBMS terminals. The example below is useful for debugging the GPS flow from radio R1 to a norBMS T1.

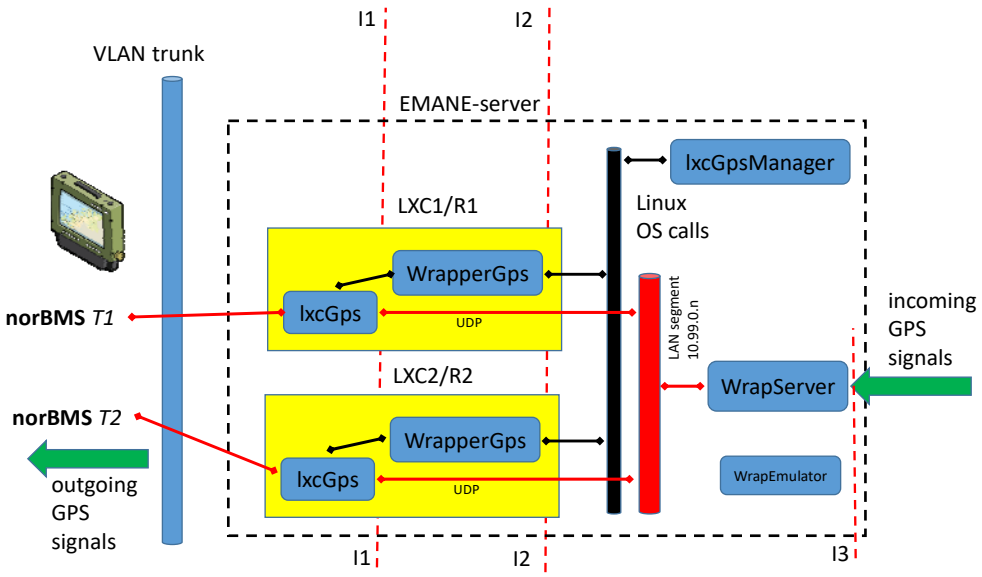


Figure 5.5 E-server internals. The GPS information paths are painted in red. The control paths are painted in black.

Step 1: To check that the GPS simulator runs inside radio R1
Do as follows on the EMANE server:

```
$ sudo ./python –m nbwf.lxcgpsmanager status –lxcn 1
```

If the GPS is running in radio R1, take the next step.

Step 2: Generate a test sequence from R1 to T1

```
$ sudo ./python –m nbwf.lxcgpsmanager oslobergen –trace True –ip 10.10.10.10 –port 4041 –wait 5
```

The options {--ip, --port} refer to parameters on the target norBMS terminal. The terminal sign on the norBMS map moves periodically between Oslo and Bergen staying 5s (--wait) in each position. Use –h to see all options.

6 E-server monitors

During emulation many processes and virtual Ethernet interfaces are involved. eNBWF provides services that monitor the E-server internal processes and emit alarms when any fault is detected. The most useful monitor is shown in Figure 6.1 – this is the 10,000-foot-view of the eNBWF state. If one of the LED buttons turns red, an error has occurred. Depending on the error, one or more radio services may have stopped. To locate the cause of the error, additional Python modules provide functions to locate errors. By clicking on a button, the corresponding component is checked immediately. An internal watch dog performs checks periodically and the LED at the upper right shows its heart beat – if this LED stops moving the status panel has crashed and must be restarted.

The functions of the buttons are as follows:

vlans: green colour means that all VLANs are up and running, and the E-server may handle external IP traffic.

emane: green colour means that all the EMANE processes are up and running inside the containers. If the lxc button becomes red, the emane button must also be red since an EMANE process only can exist inside a container.

lxc: green colour means that all containers are up and running.

RF channel: green colour means that the on-the-air LAN segment is up and the radio waves may propagate between the radios. However, be aware of that this does not imply that the IP packets will be received by the receiver – the pathloss may be too high, see section 7.2.

gps: green colour means that all GPS simulators are running.

wrap server: green colour means that the wrap server is running and accepts pathloss matrices and GPS information from the WRAP PC

wrap pc: green colour means that the wrap server has contact with the WRAP PC. The WRAP PC shall send is-alive messages periodically. If not, the button times out and changes to red.

The status panel is implemented by means of the Python package PySide2[11] and is started on the E-server as described in section 7.5. If this status panel shall be accessed from a PC attached to the management LAN, a remote desktop solution must be used. We have tested *noMachine*[12] successfully.

[11] https://wiki.qt.io/Qt_for_Python.
[12] www.nomachine.com.

A standard WEB browser can also be used to check the emulator, see Figure 6.2. This WEB server is implemented by means of Django[13].

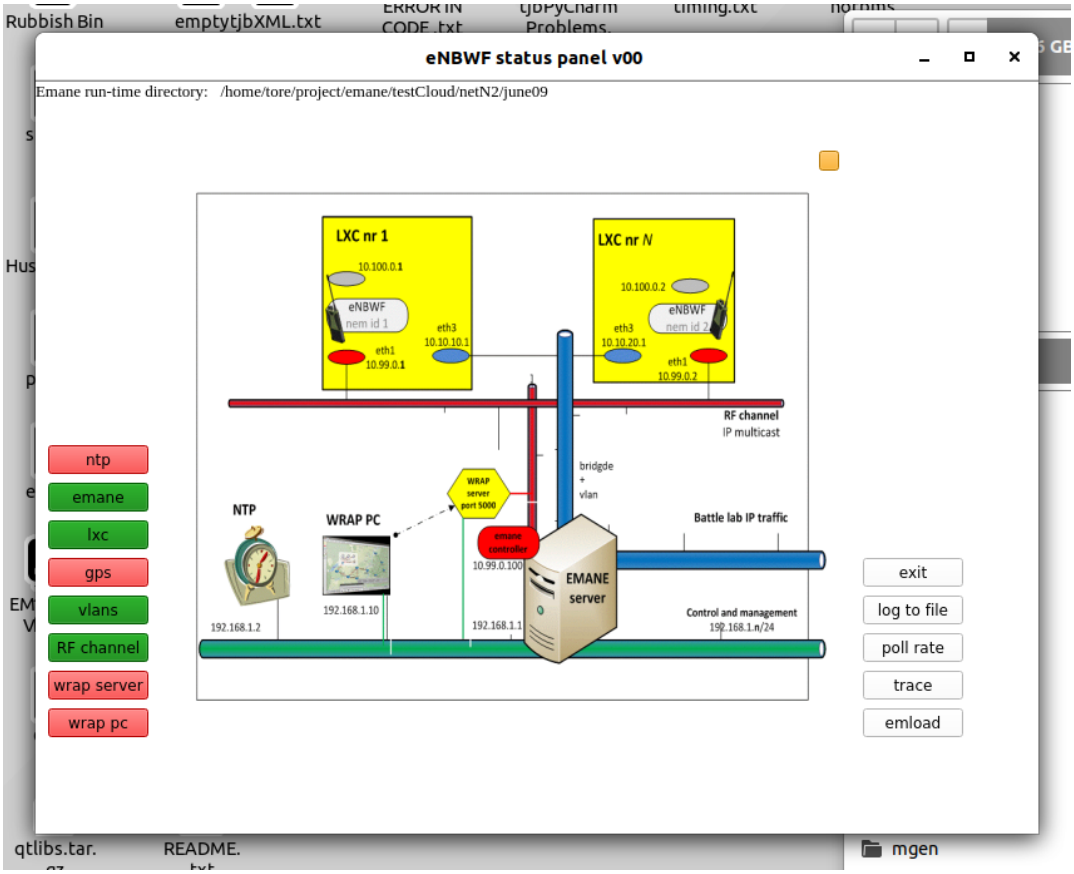


Figure 6.1 The eNBWF status panel. The buttons at the left give status information for the component named.

[13] https://www.djangoproject.com.

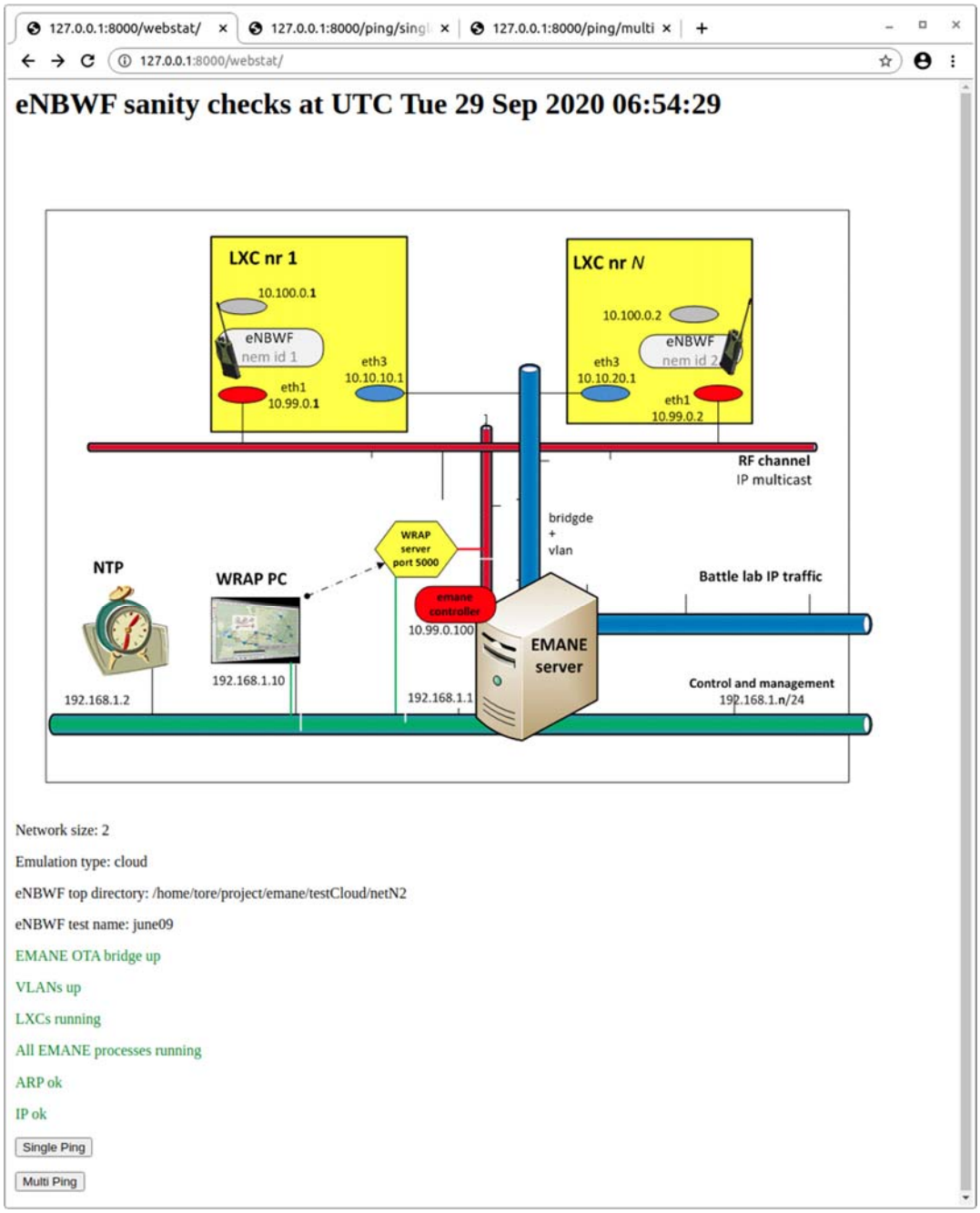


Figure 6.2 Status checks via WEB browser. Information text in black letters. Successful tests in green letters while faults are printed in red.

6.1 Radio link debugging

The tool that keeps the Internet going is ping. eNBWF provides a ping tool to test the radio links between the nodes. If the {emane, lxc, RF channel}-buttons in Figure 6.1 are green and this ping fails then the radio link pathloss is too high – the received signal is too weak. The ping test runs inside the containers with direct access to the *emane0* tunnel, see Figure 6.3. By running the script *nbwfpathloss* on the E-server, the radio link from R*1* to R*2*…R*n* can be tested. Further information is given in section 7.2.

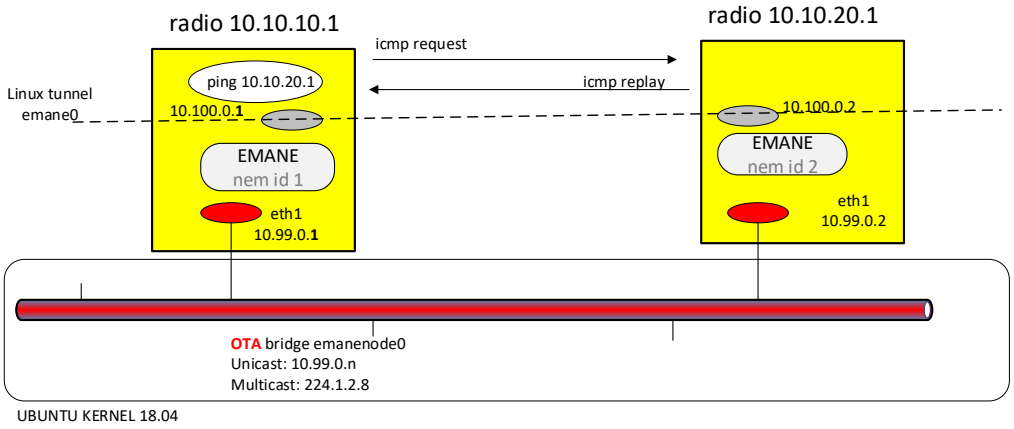


Figure 6.3 An easy method to validate radio connectivity is to apply the eNBWF ping tool.

7 Learning to Drive

This chapter explains how to build the data structure and run the eNBWF radio network. The example used below is a simple network with two radios that shall serve two norBMS terminals.

The directories referred to in this chapter are:

EMANE_DIR = /home/tore/git/emane (or any location on your computer)
NBWF_DIR = $EMANE_DIR/src/models/mac/nbwf
USER_DIR = $HOME/… (any location outside the EMANE_DIR tree)

To drive the eNBWF emulator means to execute the sections below in sequence.

7.1 Step 1 Make run-time directory

The first step is to create the top level data structure. The data files can be tailored to the user's need at a later stage. The eNBWF source tree is protected from the user of the emulator and a script must be used to copy system files to the user's working directory. Do as follows:

```
$ cd $USER_DIR
$ ./python –m nwbf.testdirectory make –type blab –src $NBWF_DIR –testdir netN2 –n 2
```

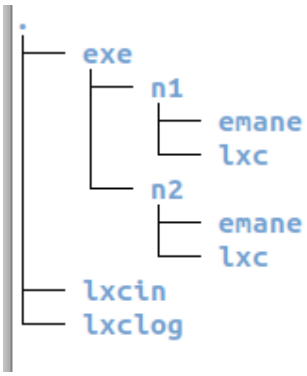
Now you have a copy of the eNBWF configuration files under the *netN2* directory. The option "--type" tells that traffic from external IP sources shall be served.

The *testdir/templates* directory contains all the configuration files, and these files can be modified by the user. For example, the internal IP addresses must match the external IP addresses.

When the user's template files are ready, they must transformed into a run-time directory structure by the script below:

```
$ ./python –m nbwf.managetestdata make –topdir $USER_DIR –testname april29 –n 2
```

The directory named *april29* contains a complete run-time file structure and each container works on a dedicated data area *exe/nN* as shown below:



The directory *lxin* contains the configuration files for the containers while the *lxclog* directory holds the LXC log files. If a file size in the *lxclog* directory is greater than zero, an error has occurred.

The file *exe/n1/emane/emane.log* contains the most important radio data for radio R1.

7.2 Step 2 Start the emulator

This step starts the emulator with the input data specified in step 1:

$ sudo ./python –m nbwf.**blab** start –topdir $USER_DIR –testname april29 –pathlossdb 10 –n 2

```
Loading emulator data from: /home/tore/project/emane/testingBlab/netN2/april29
*** Starting BRIDGES and VLANs ***
OTA bridge created: emanenode0
Number of VLANs created: 2
*** Starting LXCs ***
Host ARP setup finished
Container started: node-1
Container started: node-2
*** Starting EMANE ***
EMANE started in node: node-1
EMANE started in node: node-2
ARP script finished in node: node-1
ARP script finished in node: node-2
Pathloss set to 10dB
```

Best practise is to start emulation with excellent radio conditions and test that all the radios are up. Here the pathloss is set to 10 dB (option *–pathlossdb*) which gives excellent radio conditions regardless of the transmitter power settings. Here comes a ping test:

```
nbwfping.py ping --src 1 --n 2 --c 2
node-1 OK
node-2 OK
```

The ping test goes from 1 →{1, 2} and therefore the local interface in radio 1 is also tested. You can change the pathloss setting as follows:

$ sudo ./python –m nbwf.**nbwfpathloss** setall –pathlossdb 999 –n 2

A new ping test gives:

```
nbwfping.py ping --src 1 --n 2 --c 2
node-1 OK
node-2 ERROR
```

The connection to radio R2 is down since 999 dB will take down any radio link.

7.3 Step 3 Start the GPS service

The GPS service is implemented by one or more GPS simulators. This step is necessary only if the GPS service is required. To start a GPS simulator in the radios 1..2 use:

```
$ sudo ./python –m nbwf.lxcgpsmanager start –n 2
```

7.4 Step 4 Start the WRAP server

The WRAP server is needed when:

1) Pathloss and/or waypoints shall be received from external sources.

2) The WRAP emulator shall run inside the E-server.

If case 2) is the choice (--dryrun) then use:

```
$ sudo ./python –m nwbf.wrapserver start –topdir $USER_DIR –testname april29
                                                    –dryrun True
```

```
wrap-server bind to host::port 0.0.0.0::5000
Server is listening
LXC udp socket ready: ('10.99.0.1', 6000)
LXC udp socket ready: ('10.99.0.2', 6000)
```

The WRAP server listens on the E-server loopback on port 5000. Internally the WRAP server talks to the radios on port 6000. The WRAP emulator simulates the WRAP PC and you can use it to send synthetic waypoints to the WRAP server. For example:

```
$ sudo ./python –m nbwf.wrapemulator gps –n 2 –waypoint oslobergen –dryrun True
```

Or you can set the pathloss:

```
$ sudo ./python –m nbwf.wrapemulator pathloss –db 10 –n 2 –dryrun True
```

Hint: Both the wrapemulator and the wrapserver have many options. Try the “-h” option.

7.5 Step 5 Status checks

This is an optional step. A fast method to do a sanity check is to use the following script:

$ sudo ./python –m nbwf.**blab** status –topdir $USER_DIR –testname april29

```
blab.py status --topdir=/home/tore/project/emane/testingBlab/netN2 --testname april29
Loading emulator data from: /home/tore/project/emane/testingBlab/netN2/april29
Emulation type is: blab with 2 nodes
OTA bridge is UP: emanenode0
All vlans UP
All bridges UP
Number of running containers: 2
Number of directories in /var/lib/lxc: 2
Number of running EMANE-processes: 2
Tested ARP status OK in 2 nodes
Tested IP status OK in 2 nodes
Missing LXC GPS processes in: ['node-1', 'node-2']
```

In this case the GPS simulators do not run but the other processes are running. The status panel in Figure 6.1 is started as follows:

$ sudo ./python –m nbwf.**statuspanel** start –topdir $USER_DIR –testname april29

The third option is to monitor the emulator from a browser running on an external IP terminal, see Figure 6.2. Then the WEB server must be started as follows:

$ sudo ./python –m nbwf.**webserver** start –topdir $USER_DIR –testname april29

7.6 Step 6 Terminate

The eNBWF processes must be terminated in the correct order. If not, system files that should be deleted will block new start attempts. Use the following to terminate:

$ sudo ./python –m nbwf.**blab** stop –topdir $USER_DIR –testname april29

```
blab.py stop --topdir=/home/tore/project/emane/testingBlab/netN2 --testname april29
Loading emulator data from: /home/tore/project/emane/testingBlab/netN2/april29
*** Stopping EMANE ***
*** Stopping LXCs ***
/var/lib/lxc is empty
*** Stopping BRIDGES and VLANs ***
Bridge deleted: emanenode0
```

If new start attempts fail due to existing system files, try “blab stop” twice. If this script does not solve the problem, the files must be deleted manually.

8 Discussion and Conclusion

EMANE is an open source framework for building virtual radio networks. eNBWF is our implementation of a military tactical radio as an EMANE plug-in module (Figure 2.1). eNBWF was developed as a part of a proof-of-concept that aimed to study what can be achieved with virtual networks and cloud based solutions. Other reports will discuss the benefits of virtual solutions. The scope of this document is the eNBWF services and software.

eNBWF is a minimum viable product[14], which means that only the services required serving the norBMS application are implemented. The norBMS clients can be attached directly to the eNBWF radios – only the usual norBMS set-up procedure must be followed.

EMANE gave us the opportunity to implement an emulator in a shorter time and we ended up with an emulator that provides realistic IP throughput/delay performance of a narrow band tactical radio. The model accuracy has been validated against a steady-state simulator [5, 6]. The accuracy is sufficiently low for testing IP based application.

The radio pathloss strongly affects the radio link quality as the users move in the terrain. To model this effect we used a pathloss prediction tool (WRAP) that takes real maps and waypoints as input, calculates the pathloss in real-time and forwards the pathloss values to the digital copy in Figure 1.1. We tested this concept by using computer-generated forces[15] that generated waypoints on interface 5 in Figure 2.2. This test was limited to five mobile users.

The norBMS application needs information from GPS. Many military VHF radios have an internal GPS and we applied the same architecture to the virtual radios as explained in chapter 5. The GPS simulators received the waypoints from the external source but inserted the time tag from the EMANE server's internal wall clock. By this approach, we achieved precise time synchronisation between the external applications.

EMANE has sufficient quality to serve as a framework for further development of the eNBWF. The company Adjacentlink maintains EMANE actively, bug fixes are released regularly, and new radio models are developed. We have performed stability tests lasting up to three weeks without observing any problems – no memory leaks nor crashes. A network as large as 99 radios has been successfully tested with the use of synthetic IP traffic (interface 3 in Figure 2.2)[16].

[14] More development is needed before we can use the term prototype.
[15] https://www.mak.com/products/simulate/vr-forces
[16] Without mobile nodes.

Acronyms

ARP	Address resolution protocol
BMS	Battle management systems
CSV	Comma separated value
dBm	Decibel with reference to one milliwatt
EMANE	Extendable Mobile Ad-hoc Emulator
eNBWF	EMANE based implementation of NBWF
E-server	EMANE server
GPS	Global positioning system
GPX	GPS exchange format
HQ	Head quarter
ICMP	Internet control message protocol
ID	Identifier
IP	Internet protocol
kbps	Kilo bit per second
LAN	Local area network
LED	Light emitting diode
LXC	Linux container
MGEN	IP traffic generator from www.navy.mil
NBWF	Narrow band waveform
NEM	Network emulator module
NEM ID	NEM identifier (virtual radio identifier)
NMEA	National marine electronics association
PC	Personal computer
pkps	Packets/s
RSSI	Received signal strength indicator
TCP	Transmission control protocol
TG	Traffic generator
UDP	User datagram protocol
UTC	Coordinated universal time
VLAN	Virtual LAN
VLAN ID	VLAN identifier
WRAP	Name of a commercial product

References

[1] NATO standard AComP-5630, NARROWBAND WAVEFORM FOR VHF/UHF RADIOS - HEAD SPECIFICATION, Edition A Version 1, December 2016

[2] NATO standard AComP-5631, NARROWBAND WAVEFORM FOR VHF/UHF RADIOS - PHYSICAL LAYER AND PROPAGATION MODELS, Edition A Version 1, December 2016

[3] NATO standard AComP-5631, NARROWBAND WAVEFORM FOR VHF/UHF RADIOS – LINK LAYER, Edition A Version 1, December 2016

[4] Andre Douzette, «Simulering av propagasjonstap mellom mobile noder», FFI internnotat, 2. juli 2020, (in Norwegian).

[5] Bjørnar Libæk and Bjørn Solberg, A simulator model of the NATO Narrowband Waveform physical layer, FFI-notat 2011/00533, Norwegian Defence Research Establishment (FFI), 19. October 2011.

[6] Tore J Berg, NATO Narrowband Waveform (NBWF) - Performance Analysis of Complex Networks, FFI-rapport 2015/00402, Norwegian Defence Research Establishment (FFI), March 2015. Available online at www.ffi.no.

About FFI

The Norwegian Defence Research Establishment (FFI) was founded 11th of April 1946. It is organised as an administrative agency subordinate to the Ministry of Defence.

FFI's MISSION

FFI is the prime institution responsible for defence related research in Norway. Its principal mission is to carry out research and development to meet the requirements of the Armed Forces. FFI has the role of chief adviser to the political and military leadership. In particular, the institute shall focus on aspects of the development in science and technology that can influence our security policy or defence planning.

FFI's VISION

FFI turns knowledge and ideas into an efficient defence.

FFI's CHARACTERISTICS

Creative, daring, broad-minded and responsible.

Om FFI

Forsvarets forskningsinstitutt ble etablert 11. april 1946. Instituttet er organisert som et forvaltningsorgan med særskilte fullmakter underlagt Forsvarsdepartementet.

FFIs FORMÅL

Forsvarets forskningsinstitutt er Forsvarets sentrale forskningsinstitusjon og har som formål å drive forskning og utvikling for Forsvarets behov. Videre er FFI rådgiver overfor Forsvarets strategiske ledelse. Spesielt skal instituttet følge opp trekk ved vitenskapelig og militærteknisk utvikling som kan påvirke forutsetningene for sikkerhetspolitikken eller forsvarsplanleggingen.

FFIs VISJON

FFI gjør kunnskap og ideer til et effektivt forsvar.

FFIs VERDIER

Skapende, drivende, vidsynt og ansvarlig.

FFIs organisasjon