# MONROE

**Measuring Mobile Broadband Networks in Europe**

H2020-ICT-11-2014
Project number: 644399

Deliverable D3.1
**Experimental SW as EaaS**

**Editor(s):** Miguel Peón-Quirós, Vincenzo Mancuso, Özgü Alay
**Contributor(s):** Miguel Peón-Quirós, Foivos Michelinakis, Jonas Karlsson, Ali Safari Khatouni, Mohammad Rajiullah, Andra Lutu

**Work Package:** 3 / Experiments
**Revision:** 1.0
**Date:** February 28, 2017
**Deliverable type:** R (Report)
**Dissemination level:** Confidential, only for members of the consortium (including the Commission Services)

European Commission
Horizon 2020
European Union funding
for Research & Innovation

## Abstract

Software including experimental scripts and report describing selected experiments, instructions on how to use them and expected results.

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

# Contents

# 1 Introduction

Learning and getting used to a new tool is always a time-consuming task, and designing experiments especially for large-scale complex systems such as MBB networks, requires experience and time. Therefore, one fundamental objective of MONROE consists in taking all necessary steps to let external users start using the testbed as quickly as possible and provide experiments as a service (EaaS).

This document describes the experimental software developed by the MONROE consortium that is provided as EaaS to external users. It includes a template that can serve as the basis on which external users can build their own experiments as well as full experiments that can be executed on their own to produce measurements. All the examples referenced in this document are open and can be retrieved from the MONROE GIT repository at: `https://github.com/MONROE-PROJECT/Experiments`.

Additionally, this document contains short descriptions to experiments developed by MONROE's external users. By forking the external user's experiment repository, all MONROE experiments will be made accessible from the MONROE GIT repository. However, please note that, while MONROE consortium commits to keep the MONROE experiments continuously updated in the GIT repository, experiments from external users will be updated only periodically.

The document is organized as follows. Section 2 describes the MONROE experiment template, which can be used by external users to build their own ones. Section 3 presents the experiments that form part of the core MONROE platform and that produce data that is continuously stored in the MONROE database. Then, Section 4 guides the reader through the rest of experiments that are provided as complete examples with the platform. Those experiments can be used "as is" to produce measurements, or they can serve as the basis for more complex ones. Finally, Section 5 briefly describes some experiments developed by MONROE external users that are already mature enough to be useful on their own.

# 2 Example template

This experiment template provides an example to show the structure of the experiments leveraging different features of the MONROE platform. The goal of the experiment is to download a url (file) over http using `curl` from a configurable operator while at the same time recording the GPS positions of the node. If the operator is not available at the time of execution, the experiment will fail. The code for this example is available at: `https://github.com/MONROE-PROJECT/Experiments/tree/master/experiments/http_download`.

## 2.1 Overview of the code structure

The experiment consists of one main process and two sub processes, where one process listens to modem and GPS information, and the other executes the experiment. The main process supervises the execution of its two children.

**Information sharing between processes.** Information is shared between processes via two thread-safe data structures (i.e., a Python "Manager" object). Regarding modem information, the latest metadata update (for the specified operator) is stored in a dictionary. The GPS information is continuously appended to a list as it is received.

**The metadata sub-process.** This process listens to GPS and modem messages sent on the ZeroMQ bus and updates the shared data structures.

**The experiment sub-process.** This process reads entries from the shared data structures, runs the experiment and saves its result when finished.

## 2.2 Usage and Configuration

The experiment will execute a statement similar to running `curl` with the following command line:

```
curl -o /dev/null --raw --silent --write-out "{ remote: %{remote_ip}:%{remote_port},
size: %{size_download}, speed: %{speed_download}, time: %{time_total},
time_download: %{time_starttransfer} }" --interface eth0 --max-time 100 --range 0-100
http://193.10.227.25/test/1000M.zip
```

Each experiment has a default configuration which can be overwritten during the execution of the experiment. The default configuration values for this experiment are as follows:

```
{
# The following values are specific to the monroe platform
"guid": "no.guid.in.config.file",               # Created by the scheduler
"nodeid": "no.nodeid.in.config.file",           # Created by the scheduler
"storage": 104857600,                           # Created by the scheduler
"traffic": 104857600,                           # Created by the scheduler
"script": "jonakarl/experiment-template",        # Created by the scheduler
"zmqport": "tcp://172.17.0.1:5556",
"modem_metadata_topic": "MONROE.META.DEVICE.MODEM",
"gps_metadata_topic": "MONROE.META.DEVICE.GPS",
# "dataversion": 1,                             # Version of the experiment
# "dataid": "MONROE.EXP.JONAKARL.TEMPLATE",     # Name of the experiement
"meta_grace": 120,                              # Grace period to wait for interface metadata
"exp_grace": 120,                               # Grace period before killing experiment
"meta_interval_check": 5,                       # Interval to check if interface is up
"verbosity": 2,                                 # 0="Mute", 1=error, 2=information, 3=verbose
"resultdir": "/monroe/results/",
# These values are specic for this experiment
"operator": "Telenor SE",
"url": "http://193.10.227.25/test/1000M.zip",
"size": 3*1024 - 1,                             # The maximum size in Kbytes to download
"time": 3600                                    # The maximum time in seconds for a download
}
```

However, the configuration values can also be supplied as a JSON string in the “Additional options” field of the web user interface. This allows to specify a different set of parameters for each execution of the experiment.

The values of the configuration parameters can be read by the experiment from the `/monroe/config` file. The following text shows a configuration file with per-execution (“additional options” field) options:[1]

```
{
"stop": 1486653420,
"start": 1486653120,
"traffic": 1048576,
```

[1]Entries in the `/monroe/config` file may appear in different order.

```
"script": "mikepeon/mike-depurar",
"shared": 0,
"storage": 134217728,
"resultsQuota": 0,
"guid": "sha256:3796f833f55c8dbca7e9845ea06120ccebec85c2770c0de2deb57509300efa44.165695.48.1",
"option1": "value1",
"option2": "value2",
"nodeid": "48"
}
```

## 2.3 Requirements

The following directories and files must exist and have read and write permissions for the user/process running the container:

- "`/monroe/config`," supplied by the scheduler in the nodes.
- "`resultdir`," according to the values supplied in the configuration string or the default ones (Section 2.2).

## 2.4 Output

The download will abort when either size OR time OR actual size of the "url" is downloaded. Upon completion of the experiment, a single-line JSON object similar to this (pretty printed to improve readability) will be produced:

```
{
"Bytes": 30720000,
"DataId": "313.123213.123123.123123",
"DataVersion": 1,
"DownloadTime": 2.716,
"GPSPositions": [
{
"Altitude": 225.0,
"DataId": "MONROE.META.DEVICE.GPS",
"DataVersion": 1,
"Latitude": 59.404697,
"Longitude": 13.581558,
"NMEA": "$GPGGA,094832.0,5924.281896,N,01334.893500,E,1,05,1.6,225.0,M,35.0,M,,*5D\r\n",
"SatelliteCount": 5,
"SequenceNumber": 14,
"Timestamp": 1465551728
},
{
"DataId": "MONROE.META.DEVICE.GPS",
"DataVersion": 1,
"Latitude": 59.404697,
"Longitude": 13.581558,
"NMEA": "$GPRMC,094832.0,A,5924.281896,N,01334.893500,E,0.0,,100616,0.0,E,A*2B\r\n",
"SequenceNumber": 15,
"Timestamp": 1465551728
}
],
"Guid": "sha256:15979bc2e2449b0011826c2bb8668df980da88221af3fc7916cb2eba4f2296c1.0.45.15",
"Host": "193.10.227.25",
"Iccid": "89460850007006922138",
"InterfaceName": "usb0",
"NodeId": "45",
"Operator": "Telenor SE",
"Port": "80",
"SequenceNumber": 1,
"SetupTime": 0.004,
"Speed": 11295189.0,
"TimeStamp": 1465551458.099917,
"TotalTime": 2.72
}
```

All debug/error information will be printed on `stdout`, depending on the "verbosity" variable in configuration.

## 2.5 Docker miscellaneous usage notes

- List running containers:

  ```
  docker ps
  ```

- Debug shell:

  ```
  docker run -i -t --entrypoint bash --net=host template
  ```

- Normal execution with output to `stdout`:

  ```
  docker run -i -t --net=host template
  ```

- Attach to a running container (with shell):

  ```
  docker exec -i -t [container runtime name] bash
  ```

- Get container logs (`stderr` and `stdout`):

  ```
  docker logs [container runtime name]
  ```

# 3 MONROE Base Experiments

## 3.1 ICMP Ping

The ICMP ping experiment runs continuously on each MBB operator on the node (one independent experiment is run per interface) in order to provide basic connectivity information to each network. The experiment measures IP RTT by continuously sending ping packets to a configurable server (by default `8.8.8.8`, Google's public DNS server). The experiment sends one "Echo Request" (ICMP type 8) packet per second over the specified interface until aborted. RTT is measured as the time between the echo request is sent and the echo reply (ICMP type 0) is received from the server. The experiment runs on all interfaces in parallel.

The code for this example is available at: https://github.com/MONROE-PROJECT/Experiments/tree/master/experiments/ping.

### 3.1.1 Usage and Configuration

The experiment will execute a statement similar to running `fping` with the following command line:

```
fping -I eth0 -D -c 1 -p 1000 -l 8.8.8.8
```

The experiment is designed to run as a Docker container and will not attempt to do any active network configuration. If the specified interface does not exist (i.e., is not up) when the experiment starts, it will immediately exit.

The default parameter values are:

```
{
"guid": "no.guid.in.config.file",                # Created by the scheduler
"zmqport": "tcp://172.17.0.1:5556",
"nodeid": "fake.nodeid",
"modem_metadata_topic": "MONROE.META.DEVICE.MODEM",
"server": "8.8.8.8",                             # ping target
"interval": 1000,                                # time in ms between successive packets
"dataversion": 2,
"dataid": "MONROE.EXP.PING",
"meta_grace": 120,                               # Grace period to wait for interface metadata
"ifup_interval_check": 5,                        # Interval to check if interface is up
"export_interval": 5.0,
"verbosity": 2,                                  # 0="Mute", 1=error, 2=Information, 3=verbose
"resultdir": "/monroe/results/",
"modeminterfacename": "InternalInterface",
"interfacename": "eth0",                         # Interface to run the experiment on
"interfaces_without_metadata": ["eth0", "wlan0"] # Manual metadata on these interfaces
}
```

### 3.1.2 Requirements

The following directories and files must exist and have read and write permissions for the user/process running the container:

- `/monroe/config`, supplied by the scheduler in the nodes.
- "`resultdir`," according to the values supplied in the configuration string or the default ones.

### 3.1.3 Output

The experiment will produce one of the two following single-line JSON objects, depending on whether it got a reply form the server or not. If a reply was received:

```
{
"Guid": "313.123213.123123.123123", # exp_config['guid']
"Timestamp": 23123.1212, # time.time()
"Iccid": 2332323, # meta_info["ICCID"]
"Operator": "Telia", # meta_info["Operator"]
"NodeId" : "9", # exp_config['nodeid']
"DataId": "MONROE.EXP.PING",
"DataVersion": 2,
"SequenceNumber": 70,
"Rtt": 6.47,
"Bytes": 84,
"Host": "8.8.8.8",
}
```

If the reply was not received (Bytes and RRR values are not present):

```
{
"Guid": "313.123213.123123.123123", # exp_config['guid']
"Timestamp": 23123.1212, # time.time()
"Iccid": 2332323, # meta_info["ICCID"]
"Operator": "Telia", # meta_info["Operator"]
"NodeId" : "9", # exp_config['nodeid']
"DataId": "MONROE.EXP.PING",
"DataVersion": 2,
"SequenceNumber": 71,
"Host": "8.8.8.8",
}
```

## 3.2 HTTP Download

This is a periodically scheduled experiment that monitors the download speed of each MBB operator on the node. The experiment will, over each MBB operator in sequence, download the specified url (file) with `curl` (http), presenting one result per interface. The MONROE experiment template described in Section 2 corresponds to this experiment, therefore, we will not further detail this experiment here.

## 3.3 Tstat & mPlane

The mPlane protocol provides control and data interchange for passive and active network measurement tasks. It is built around a simple workflow that can interact with different frameworks to provide the results of the measurements. This package includes an mPlane proxy and generic configuration files for Tstat.

mPlane captures traffic flow on all interfaces by means of the Tstat (http://tstat.polito.it/) probe. The mPlane container is always running as one of the default experiments on all MONROE nodes. The Tstat passive traces are stored locally on the node and are accessible by the experimenters. A detailed description and the source code are available on github (https://github.com/MONROE-PROJECT/mPlane).

### 3.3.1 Usage and Configuration

In order to run tstat, one needs to create the docker image normally and execute the container with the following command line:

```
docker run -i -t --net=host -d -v /mplane:/monroe/results -v /tstat:/monroe/tstat
    -v /etc/nodeid:/nodeid:ro monroe/mplane
```

### 3.3.2 Requirements

The script must have access to `/nodeid` and run `get_nodeid`.

### 3.3.3 Output

Tstat RRD logs and the compressed log are stored in the node at `/experiments/monroe/mplane`. Tstat logs are transfered to the MONROE server and imported into MONROE's (Cassandra) database. The structure of the database tables is available on github (https://github.com/MONROE-PROJECT/Database/blob/master/db_schema.cql).

During experiment execution, the last three Tstat logs are shared with the experiment at `/monroe/tstat`. Therefore, MONROE users can access the passive traces collected by Tstat during their experiments.

# 4 Operational MONROE Examples

MONROE provides a set of diverse fully-working examples that users can use to learn how to design and implement their own experiments, or indeed to gather real measurements. The source code for the examples is publicly available at `https://github.com/MONROE-PROJECT/Experiments`.

## 4.1 Helloworld (helloworld)

This experiment provides an easy example for using the configuration options from the scheduler, listen to and record the metadata stream (e.g., GPS and operator information), and show the experiment log functionality on a MONROE node. The experiment listens to the metadata stream and records the `nr_of_messages` first messages. The metadata messages are saved in JSON format with a custom field ("Hello") in the output directory. Additionally, the experiment prints out some debugging messages to show how these messages are logged and later retrieved via the web user interface.

The code for this example is available at: `https://github.com/MONROE-PROJECT/Experiments/tree/master/experiments/helloworld`.

### 4.1.1 Usage and Configuration

The experiment is configured with a JSON string introduced via the "Additional options" field in the web user interface. The configurable parameters and their default values are:

```
{
    "zmqport": "tcp://172.17.0.1:5556",
    "nodeid": "fake.nodeid",            # Needs to be overriden
    "metadata_topic": "MONROE.META",
    "verbosity": 2,                     # 0 = "Mute", 1=error, 2=Information, 3=verbose
    "resultdir": "/monroe/results/",
    "nr_of_messages": 3
}
```

### 4.1.2 Requirements

The following directories and files must exist and have read and write permissions for the user/process running the container:

- `/monroe/config`, supplied by the scheduler in the nodes.
- "`resultdir`," according to the values supplied in the configuration string or the default ones (Section 4.1.1).

### 4.1.3 Output

The experiment will produce a single-line JSON object similar to the following ones, depending on the metadata received ("pretty printed" here to improve readability):

```
{
    "DataId": "MONROE.META.NODE.SENSOR",
    "DataVersion": 1,
    "SequenceNumber": 58602,
    "Timestamp": 1465888420,
    "NodeId": "9",
    "Hello": "World"
}
```

The log file will contain records similar to these ones:

```
[2017-02-07 09:53:27.190338] Hello: Default config {
"metadata_topic": "MONROE.META",
"nodeid": "fake.nodeid",
"nr_of_messages": 3,
"resultdir": "/monroe/results/",
"verbosity": 2,
"zmqport": "tcp://172.17.0.1:5556"
}
[2017-02-07 09:53:27.20000] Hello: Start recording messages with configuration {
"metadata_topic": "MONROE.META",
"nodeid": "fake.nodeid",
"nr_of_messages": 3,
"resultdir": "/monroe/results/",
"verbosity": 2,
"zmqport": "tcp://172.17.0.1:5556"
}
[[2017-02-07 09:53:27.30000] Received message 1 with topic : MONROE.META.NODE.SENSOR
{
"DataId": "MONROE.META.NODE.SENSOR",
"DataVersion": 1,
"SequenceNumber": 58602,
"Timestamp": 1465888420,
"NodeId": "9",
"Hello": "World"
}
. # And so on for each metadata message received until the configured value of metadata messages
.
.
[2017-02-07 09:53:27.40000] Hello : Finished the experiment
```

## 4.2 Paris Traceroute (paris-traceroute)

This example showcases how to use the MONROE-modified version of paris-traceroute inside a container. The binary of this tool is included in the base image of MONROE.

The original version of paris-traceroute has no option to choose which interface should be used. In this version, flags to set the interface and source IP of the transmitted packets have been added. Setting the interface is obligatory; if it is not set, the program will crash (by design), since if the interface were chosen automatically, it would probably not be what the experimenter intended to use. The source IP flag is optional. Just setting the IP flag to the IP of an interface without setting the interface flag will not work either. This is done on purpose as well, as it might be possible for multiple interfaces to have the same IP within the MONROE network namespace. If the IP flag is not set, the source IP is set to the IP of the chosen interface.

The code for this example is available at: https://github.com/MONROE-PROJECT/Experiments/tree/master/experiments/paris_traceroute.

### 4.2.1 Usage and Configuration

The parameters of this experiment are provided as "Additional options" in the scheduling web interface. The following JSON string is an example of the additional options that can be passed to this container:

```
"interfaces": ["op1", "op2"], "targets": ["8.8.8.8", "www.uc3m.es"],
"traceAlgos": ["exh"], "protocol": "udp"
```

Flags:

```
-C  --nodeIPArgument              Source IP
-O  --nodeInterfaceArgument       Source interface (mandatory)
```

The paris-traceroute binary can be executed (as any normal Linux command) either without specifying a traceroute algorithm to perform a "simple" traceroute (similar to the output of the ordinary traceroute command), or with the flags `-n -a exh`, to perform an exhaustive traceroute. Exhaustive traceroutes provide more detailed and accurate paths between the host (MONROE node) and the target server that are able to detect, among others, the presence of load balancers, which create multiple paths between host and target.

#### 4.2.2 Output

The experiment output is a text file:

```
root@b59e69a56297:/# paris-traceroute -O op2 -C 192.168.1.127 8.8.8.8
traceroute [(192.168.1.127:33456) -> (8.8.8.8:33457)], protocol udp, algo hopbyhop, duration 18 s
1  192.168.1.1 (192.168.1.1)  2.946 ms   0.553 ms   0.559 ms
2  * * *
3  10.133.17.29 (10.133.17.29)  83.259 ms   136.577 ms   82.050 ms
4  10.133.17.14 (10.133.17.14)  78.783 ms   131.510 ms   79.231 ms
5  10.133.17.236 (10.133.17.236)  84.243 ms   133.024 ms   79.785 ms
6  10.133.17.3 (10.133.17.3)  81.543 ms   139.381 ms   100.263 ms
7  83.224.40.186 (83.224.40.186)  89.319 ms   188.926 ms   179.963 ms
MPLS Label 24703 TTL=254
8  83.224.40.185 (83.224.40.185)  82.710 ms   172.438 ms   147.020 ms
9  85.205.14.105 (85.205.14.105)  85.179 ms   137.514 ms   125.869 ms
10  72.14.223.169 (72.14.223.169)  85.609 ms   137.363 ms   118.063 ms
11  216.239.47.128 (216.239.47.128)  79.567 ms   146.356 ms   145.285 ms
12  209.85.243.33 (209.85.243.33)  129.615 ms   198.938 ms   269.407 ms
MPLS Label 568892 TTL=1
13  64.233.174.143 (64.233.174.143)  108.599 ms   185.810 ms   246.661 ms
MPLS Label 692130 TTL=1
14  108.170.234.47 (108.170.234.47)  111.645 ms   825.615 ms   1424.942 ms
15  * * *
16  google-public-dns-a.google.com (8.8.8.8)  103.087 ms !T2   166.279 ms !T2   224.649 ms !T2
```

#### 4.2.3 Additional remarks

Paris-traceroute instances should be run sequentially and preferably when the node is generating little traffic in general because it uses raw packet capture to detect the replies from intermediate nodes and background traffic might interfere with this process.

### 4.3 Headless Browser (headlessbrowsing)

This experiment evaluates the performance of different HTTP protocols (HTTP1.1, HTTP1.1/TLS, HTTP2) using the headless Firefox browser. The experiment uses the Selenium browser-automation framework, which enables execution of web browsing automation tests in different browsers such as Firefox and Chrome. The Selenium web-driver is used for Firefox. For a given url, HTTP protocol and source network interface, Selenium launches the native Firefox browser to visit that url.

The code for this example is available at: https://github.com/MONROE-PROJECT/Experiments/tree/master/experiments/headlessbrowsing.

### 4.3.1 Usage and Configuration

The default input values are (can be overridden by a /monroe/config):

```
{
        "guid": "no.guid.in.config.file",  # Should be overridden by scheduler
        "url": "http://193.10.227.25/test/1000M.zip",
        "size": 3*1024,  # The maximum size in Kbytes to download
        "time": 3600,  # The maximum time in seconds for a download
        "zmqport": "tcp://172.17.0.1:5556",
        "modem_metadata_topic": "MONROE.META.DEVICE.MODEM",
        "dataversion": 1,
        "dataid": "MONROE.EXP.FIREFOX.HEADLESS.BROWSING",
        "nodeid": "fake.nodeid",
        "meta_grace": 120,  # Grace period to wait for interface metadata
        "exp_grace": 120,  # Grace period before killing experiment
        "ifup_interval_check": 5,  # Interval to check if interface is up
        "time_between_experiments": 30,
        "verbosity": 2,  # 0 = "Mute", 1=error, 2=Information, 3=verbose
        "resultdir": "/monroe/results/",
        "modeminterfacename": "InternalInterface",
        "urls": ["www.wikipedia.com"],
        "http_protocols":["h1","h1s","h2"],
        "iterations": 1,
        "allowed_interfaces": ["op0",
                               "op1",
                               "op2"],  # Interfaces to run the experiment on
        "interfaces_without_metadata": [ ]  # Manual metadata on these IF
        }
```

For example, using the above inputs (default), the experiment will download wikipedia with different combinations of three different http protocols and three different source interfaces each time. So, by updating the configuration file, it is possible to input required urls, http protocols, iterations, interfaces, etc. while scheduling the experiment.

### 4.3.2 Output

This experiment generates an HTTP ARchive (HAR) file during the download of a target url that helps to find afterwards the impact of different web-page features on its overall Page Load Time (PLT).

The experiment generates a single JSON file such as:

```
{
    "DataId":"MONROE.EXP.FIREFOX.HEADLESS.BROWSING",
    "ping_min":" 55.6",
    "ping_max":"56.8",
    "NumObjects":6,
    "InterfaceName":"usb2",
    "Web load time":196,
    "PageSize":35641,
    "DataVersion":1,
    "Timestamp":1481536829.0814,
    "NWMCCMNC":22210,
    "Objects":[
{
        "objectSize":1951,
        "mimeType":"image/png",
        "startedDateTime":"2016-12-12T10:00:21.293+00:00",
        "url":"https://www.wikipedia.org/portal/wikipedia.org/assets/img/Wikipedia_wordmark.png",
        "timings":{"receive":1, "send":0, "connect":1, "dns":0, "blocked":0, "wait":60},
        "time":62
    },
    {
        "objectSize":13196,
        "mimeType":"image/png",
        "startedDateTime":"2016-12-12T10:00:21.294+00:00",
        "url":"https://www.wikipedia.org/portal/wikipedia.org/assets/img/Wikipedia-logo-v2.png",
        "timings":{"receive":52, "send":3, "connect":59, "dns":2, "blocked":0, "wait":53 },
        "time":169
    },
    {
        "objectSize":9425,
        "mimeType":"application/javascript",
        "startedDateTime":"2016-12-12T10:00:21.295+00:00",
        "url":"https://www.wikipedia.org/portal/wikipedia.org/assets/js/index-abc278face.js",
        "timings":{"receive":3, "send":0, "connect":120, "dns":0, "blocked":0, "wait":65},
        "time":188
    },
    {
        "objectSize":1164,
        "mimeType":"application/javascript",
        "startedDateTime":"2016-12-12T10:00:21.296+00:00",
        "url":"https://www.wikipedia.org/portal/wikipedia.org/assets/js/gt-ie9-c84bf66d33.js",
        "timings":{"receive":0, "send":1, "connect":64, "dns":2, "blocked":0, "wait":70 },
        "time":137
    },
    {
        "objectSize":1590,
        "mimeType":"image/png",
        "startedDateTime":"2016-12-12T10:00:21.381+00:00",
        "url":"https://www.wikipedia.org/portal/wikipedia.org/assets/img/sprite-icons.png?
         27378e2bb51199321b32dd1ac3f5cd755adc21a5",
        "timings":{"receive":1, "send":0, "connect":1, "dns":0, "blocked":0, "wait":49 },
        "time":51
    },
    {
        "objectSize":8315,
        "mimeType":"image/png",
        "startedDateTime":"2016-12-12T10:00:21.425+00:00",
        "url":"https://www.wikipedia.org/portal/wikipedia.org/assets/img/sprite-project-logos.png?
         dea6426c061216dfcba1d2d57d33f4ee315df1c2",
        "timings":{"receive":2, "send":0, "connect":8, "dns":0, "blocked":0, "wait":54 },
        "time":64
    } ],
    "IPAddress":"2.43.181.254",
    "IMSIMCCMNC":22210,
    "tracedRoutes":["192.168.96.1", "193.10.227.25", "xx.xx.xx.xx" ..... "192.168.96.1" ],
    "InternalInterface":"op0",
    "NodeId":"41",
    "ping_exp":1,
    "Protocol":"HTTP1.1",
    "SequenceNumber":1,
    "url":"www.wikipedia.org",
    "ping_avg":"56.2",
    "InternalIPAddress":"192.168.96.123",
    "Operator":"voda IT",
    "Iccid":"8939104160000392116"
}
```

## 4.4 Web replay (pReplay)

The pReplay experiment replays the dependency graph of a web site.

The traversal begins with the first activity: Loading the root HTML. After building the dependency graph, it acts for each task whose dependencies have already been met. For network tasks, it makes a request for the related url; correspondingly, for computation activities, it waits for the amount of time mentioned in the graph. Once a particular activity is finished, pReplay checks if any activities depending on that one have already met all of their dependencies and must thus be triggered. pReplay walks through the dependency graph until all activities in the graph have been visited.

The code for this example is available at: https://github.com/MONROE-PROJECT/Experiments/tree/master/experiments/preplay.

### 4.4.1 Usage and Configuration

Execute pReplay on a command line inside a container as with any other Linux command:

```
./pReplay interface_name server testfile [http|https|http2] [max-connections] [cookie-size]
```

Parameters:

- **interface_name:** Source interface for outgoing traffic.
- **server:** DNS name or IP address.
- **testfile:** Relative path to test file in JSON format.
- **protocol:**
  - http: http 1.1
  - https: http 1.1 with SSL
  - http2: http 2
- **max-connections:** Maximum amount of concurrent connections.
- **cookie-size:** Size of cookie — works with http1 only.

## 4.5 DASH Video Streaming (astream)

AStream is a Python based emulated video player to evaluate the performance of the DASH bitrate adaptation algorithms. The supported rate adaptation algorithms are:

- Basic adaptation.
- Segment Aware Rate Adaptation (SARA) [2].
- Buffer-Based Rate Adaptation (Netflix) [1].

The code for this example is available at: https://github.com/MONROE-PROJECT/Experiments/tree/master/experiments/astream.

### 4.5.1 Usage and Configuration

The experimenter can choose the rate adaptation algorithm passing a JSON string to the scheduler through the user interface (e.g., `"playback":"NETFLIX"`). The default is the basic adaptation scheme. Additionally, the user can specify the target MPD file to play (e.g., `"mpd_file":"http://128.39.37.161:8080/BigBuck-Bunny_4s.mpd"`) and the number of segments to retrieve (e.g., `"segment_limit":10`).

### 4.5.2 Output

The astream container outputs two log files:

1. Buffer logs: Epoch time, current playback time, current buffer size (in segments), current playback state.
2. Playback logs: Epoch time, playback time, segment number, segment size, playback bitrate, segment duration, weighted harmonic mean average download rate.

## 4.6 UDP Bandwidth Estimatior (udpbwestimator)

Udpbwestimator is an experiment setup to estimate available bandwidth for a particular network interface. It consists of two applications, a receiver and a traffic generator (server). The receiver initiates connections and requests the server for traffic. Then, every second, the server sends a burst of UDP packets back to back to the receiver, which follows the packet arrival times and estimates the available bandwidth.

The code for this example is available at: https://github.com/MONROE-PROJECT/Experiments/tree/master/experiments/udpbwestimator.

### 4.6.1 Usage and Configuration

The receiver accepts the following command line parameters:

-c : Number of back-to-back packets to be sent in each second.
-b : Number of bursts to be sent.
-l : Payload length in bytes.
-s : Source IP to bind to.
-o : Source port.
-d : Destination IP.
-p : Destination port.
-w : Optional, filename for writing the packet arrival times.

### 4.6.2 Output

The experiment will produce a single-line JSON object similar to the following:

```
{
"CID" : 33346602,
"DataId" : "MONROE.EXP.UDPBWESTIMATOR",
"DataVersion" : 1,
"DeviceMode" : 5,
"DeviceState" : 3,
"Guid" : "sha256:872af8c8b8f1635be6936a111b5fa838071e6f42cb317e9db1d9bb0c7db31425.93321.204.1",
"IMEI" : "864154023639966",
"IMSI" : "240016025247086",
"IMSIMCCMNC" : 24001,
"IPAddress" : "78.79.63.124",
"Iccid" : "89460120151010468086",
"InterfaceName" : "usb0",
"InternalIPAddress" : "192.168.68.118",
"InternalInterface" : "op1",
"LAC" : 2806,
"NWMCCMNC" : 24202,
"NodeId" : "204",
"Operator" : "NetCom",
"RSRP" : -72,
"RSRQ" : -7,
"RSSI" : -49,
"SequenceNumber" : 1,
"Timestamp" : 1479312368.633218,
"bw" : "48.41 38.98 36.44 50.00 30.20 45.21 47.02 37.89 44.37 28.90 25.91 38.57 48.74
       39.94 43.37 37.94 43.81 39.60 52.00 47.55 48.20 34.85 41.44 47.60 57.26 46.11
       45.66 52.04 37.43 49.67 33.56 50.35 41.11 51.63 45.33 104.01 45.73 49.95 50.37
       38.57 29.45 50.95 54.95 45.42 47.13 34.30 46.10 103.68 79.75 45.72 52.03 30.38
       50.21 36.96 71.51 54.66 39.26 44.12 45.18 39.93"
}
```

## 4.7 Traceroute running in Background (traceroute_background_experiment)

Performs traceroute periodically to various targets. This experiment is meant to be run in the background and can be run in parallel with experiments of other users. It uses the default traceroute binary distributed by the Debian repositories.

The code for this example is available at: `https://github.com/MONROE-PROJECT/Experiments/tree/master/experiments/traceroute_background_experiment`.

### 4.7.1 Usage and Configuration

To reduce experiment duration, the traceroutes can be run in parallel. The number of parallel traceroute instances is dictated by the `maxNumberOfTotalTracerouteInstances` parameter. It is possible to parallelize on a per-interface basis (i.e., `maxNumberOfTotalTracerouteInstances` per interface) or per the whole experiment (i.e., `maxNumberOfTotalTracerouteInstances` total in the experiment instance spread among all the interfaces). This behavior is controlled by the `executionMode` parameter. The available options are: serially, serialPerInterface and parallel.

Additionally, a flag can be provided to choose the protocol of the probes: default, udp, tcp and icmp.

The parameters of this experiment are provided as "Additional options" in the web user interface. An example JSON string that can be used with this container as additional options is:

```
"interfaces": ["op0", "op1", "op2"], "targets": ["www.ntua.gr", "www.uc3m.es", "Google.com",
"Facebook.com", "Youtube.com", "Baidu.com", "Yahoo.com", "Amazon.com", "Wikipedia.org",
"audio-ec.spotify.com", "mme.whatsapp.net", "sync.liverail.com", "ds.serving-sys.com",
"instagramstatic-a.akamaihd.net"], "maxNumberOfTotalTracerouteInstances": 5,
"executionMode": "parallel"
```

### 4.7.2 Output

Each traceroute produces a text file that is parsed by `outputParser.py` to generate the JSON output of this experiment. The parser is located under: https://github.com/MONROE-PROJECT/Experiments/tree/master/experiments/traceroute_background_experiment/files.

The JSON file is then imported into the MONROE database.

## 4.8 Other containers in the repositories

Our public repositories contain the source code for other Docker containers that perform various tasks in the nodes. Although they are not intended as examples, users can take a look into them to gain a deeper understanding of the platform configuration.

### 4.8.1 Container: metadata-subscriber

The subscriber is designed to listen to ZMQ messages send out by the metadata-multicaster. The subscriber attaches to a configurable ZeroMQ socket and listens to all messages that begin with the topic "MONROE.META," except the ones whose topic ends with ".UPDATE" (rebroadcasts) and/or begins with "MONROE.META.DEVICE.CONNECTIVITY." as these are redundant. All messages are updated with NodeId, but are otherwise saved verbatim as a JSON formatted file suitable for later import in the MONROE databases. The source code is available at:
https://github.com/MONROE-PROJECT/Experiments/tree/master/metadata-subscriber.

### 4.8.2 Container: tunnelbox-server

This container acts as an SSH reverse tunnel endpoint that clients can use to directly connect to their experiment containers (on any MONROE node). The purpose is to provide experimenters an interactive way of accessing an experiment running on a real MONROE node during development or debugging. The client has to supply its own public SSH key to the experiment container using the web user interface. The web user interface provides further instructions (SSH command line) to connect to the experiment container using the provided key. The source code is available at:
https://github.com/MONROE-PROJECT/Experiments/tree/master/tunnelbox-server.

### 4.8.3 Container: monroe_base

This is the container upon which all user experiments *must* be built. The container is based on Debian "jessie" with (MONROE) common experiment tools added. For a list of the tools currently installed see the folder `monroe_base.docker` in our repositories. The source code is available at:
https://github.com/MONROE-PROJECT/Experiments/tree/master/monroe_base.

# 5 Experiments of MONROE External Users

This section describes two of the experiments developed by the external users from the OPEN CAll 1 projects. We will continue to share the external user's experiments from open call projects as they become available in MONROE's git repository.

## 5.1 PATHspider-Monroe

PATHspider-monroe is a version of the PATHspider project[2] adapted for running on MONROE nodes. PATHspider is a tool for A/B testing of path transparency to certain features in the Internet. The Docker container for execution on MONROE nodes is available at:
https://github.com/mami-project/pathspider-monroe.
The tool source code is available at:
https://github.com/mami-project/pathspider.

### 5.1.1 Usage and Configuration

PATHspider can be easily used from the command line with existing plugins:

```
pathspider [-h] [-s] [-i INTERFACE] [-w WORKERS] [--input INPUTFILE]
           [--output OUTPUTFILE]
           PLUGIN ...
```

Optional arguments:

-h, - -help : Show this help message and exit.
-s, - -standalone : Run in standalone mode. This is the default mode (and currently the only supported one). In the future, mplane will be supported as a mode of operation.
-i INTERFACE, - -interface INTERFACE : The interface to use for the observer.
-w WORKERS, - -workers WORKERS : Number of workers to use.
- -input INPUTFILE : A file containing a list of remote hosts to test, with any accompanying metadata expected by the pathspider test. This file should be formatted as a comma-seperated values file.
- -output OUTPUTFILE : The file to output results data to.

The following plugins are available for use:

tfo : TCP Fast Open.
dscp : DiffServ Codepoints.
ecn : Explicit Congestion Notification.
dnsresolv : DNS resolution for hostnames to IPv4 and v6 addresses.

Interested MONROE users are kindly requested to refer to the project webpage for further information.

## 5.2 Yomo-Monroe

Yomo-Monroe has two components: Yomo-docker and Yomo-browser-plugin.

*Yomo-docker* container provides features to estimate YouTube's "Quality of Experience." For this purpose, the docker container independently performs experiments and monitors the quality at the end-user side. For more information, see: https://go.uniwue.de/yomo-docker and https://go.uniwue.de/yomo.

*yomo-browser-plugin* monitors the quality of YouTube video streaming in the browser. With the help of the results, the Quality of Experience of the end-user can be estimated. The monitored parameters are stored in log files. Analysis scripts are included. For more information, see: https://go.uniwue.de/yomo-browser-plugin and https://go.uniwue.de/yomo.

The source codes are available at: https://github.com/lsinfo3/yomo-docker.git and https://github.com/lsinfo3/yomo-browser-plugin.git.

[2]https://pathspider.net/

### 5.2.1 Usage and Configuration

Possible settings for the plugin (use “about:config” to set and change settings):

1. extensions.yomo.userid : Set a userid as string value that is included in the log files.
2. extensions.yomo.filename_suffix : Specify a log file suffix.
3. extensions.yomo.filename_prefix : Specify a log file prefix.
4. extensions.yomo.debug (bool) : Specify whether to operate in debug mode or not.
5. extensions.yomo.log_directory : Specify where to write the log files. If nothing is specified, the Windows or Linux temporary directory is used.
6. extensions.yomo.zip_log_files (bool) : If activated, log files are automatically zipped to reduce hard disk space.

### 5.2.2 Output

When a YouTube video is watched, it writes down information like title, video time, quality, estimated buffer, etc. to log files in the user TEMP directory. For example, player information can be found at lines marked with "##P##" (e.g., Player state: 3 means the video is buffering).

To easily extract information out of the log files, the attached scripts within the zip file can be used (see “parse-output.zip”). They work on Linux or in Windows with cygwin (bash scripts). For instance see “parse-output/another.example” to learn how to, first, find the video IDs that are watched within this browsing session and then, get the timestamps when the video is buffering.

## References

[1] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proceedings of the 2014 ACM Conference on SIGCOMM (SIGCOMM'14)*, pages 187–198, New York, NY, USA, 2014. ACM Press.

[2] Juluri P., Tamarapalli V., and D. Medhi. SARA: Segment aware rate adaptation algorithm for dynamic adaptive streaming over HTTP. In *ICC QoE-FI Workshop*, June 2015.