

# Engineering assistant internship report

By Zacharie El Abdalaoui (SPID, Robotics)

Implementation of a web interface for the Flowtester device for the CTU (Czech  
Technical University)



**ČESKÉ  
VYSOKÉ  
UČENÍ  
TECHNICKÉ  
V PRAZE**

## Summary

Abstract .....	3
Résumé .....	4
Host establishment presentation.....	5
Project Problematic.....	6
I Flowtester Configuration system .....	8
II. Pattern and Measurement configurator .....	10
Pattern Status page .....	12
The Add/Edit pattern page.....	14
III Measurements .....	17
Profile list page.....	18
Profile Status page.....	19
Profile Creation page.....	20
Conclusion .....	23
Annex.....	24
First Annex: Project Description .....	24

## Abstract

The Flowtester device is a project led by the CTU-FEE (Czech Technical University in Prague- Faculty of Electrical Engineering). This is a network benchmarking tool, based on the OpenWRT operating system, and thoroughly using the Ubus console. The device needs a proper web interface in order to start the testing phase. The Interface is a web interface programmed in the Lua programming language. Throughout this report, the solutions I have given to create this interface will be explained, as well as the tasks I was given. Flowtester configuration, patterns, and measurements profile listing, reviewing, creation, edition. Then I will explain my thoughts on the experience I have gained as well as how this will serve me in my future engineer career.

## Résumé

L'appareil Flowtester, est un outil de mesure de flux de données réseaux développé par une équipe de la CTU-FEE (Czech Technical University in Prague- Faculty of Electrical Engineering). Cet appareil fonctionne sous le système d'exploitation basé sur Linux OpenWRT. Mais OpenWRT ne fournit pas d'interface graphique à l'utilisateur. Or afin de pouvoir passer à la phase de test client, cette équipe avait besoin de développer une interface web ergonomique. Cette interface a été imaginée sous l'interface LuCi, elle-même basée sur le langage Lua. Durant ce rapport je vais exposer les solutions que j'ai apporté pour l'implémentation des différentes fonctionnalités. (De la configuration du Flowtester, à la création des pages listant les profils de mesures utilisées par l'appareil). Par la suite je reviendrai sur les compétences et l'expérience que j'ai acquise, et la mettre en relation avec ma future carrière d'ingénieur.

## Host establishment presentation

The Czech Technical University in Prague was established in 1707 by the Austro-Hungarian Emperor Joseph I, in order to provide engineering teaching. First as the Institute of Engineering Education, the CTU was a secondary school before turning into the Prague Polytechnic Institute, in 1806, and then started to be a tertiary university. The CTU got its actual name in 1920 after the establishment of Czechoslovakia.

The CTU contains eight faculties, including the Faculty of Electrical Engineering which was the hosting faculty for the internship, and three higher education institutes. In the FEE I was in contact with the Department of Telecommunications Engineering, which is working on several projects.

The Flowtester project is the project I took part during this internship. The Flowtester is a network benchmarking device created by the CTU. The tool evaluates data from TCP/IP protocols based networks, and can quickly check the status of a network, detect bottlenecks, and on the long run analyze the network state, and stability. The Flowtester can carry these tests on its own or controlled by a master unit linked with several Flowtesters, and process the data gathered from those.

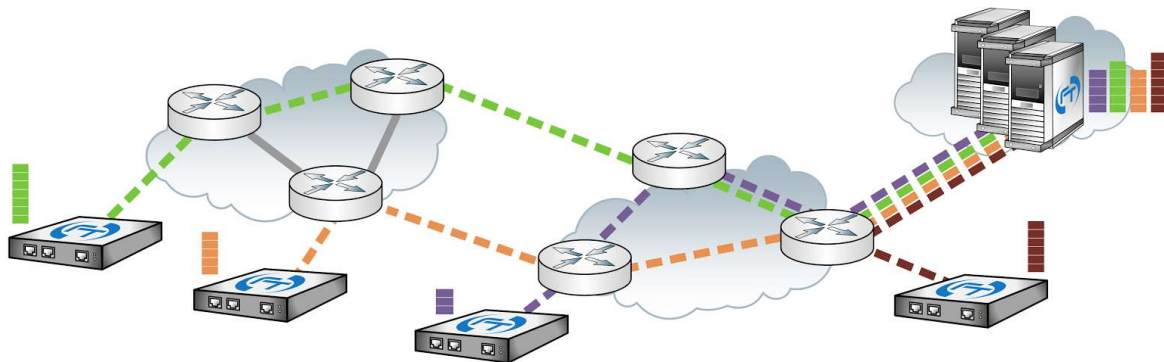
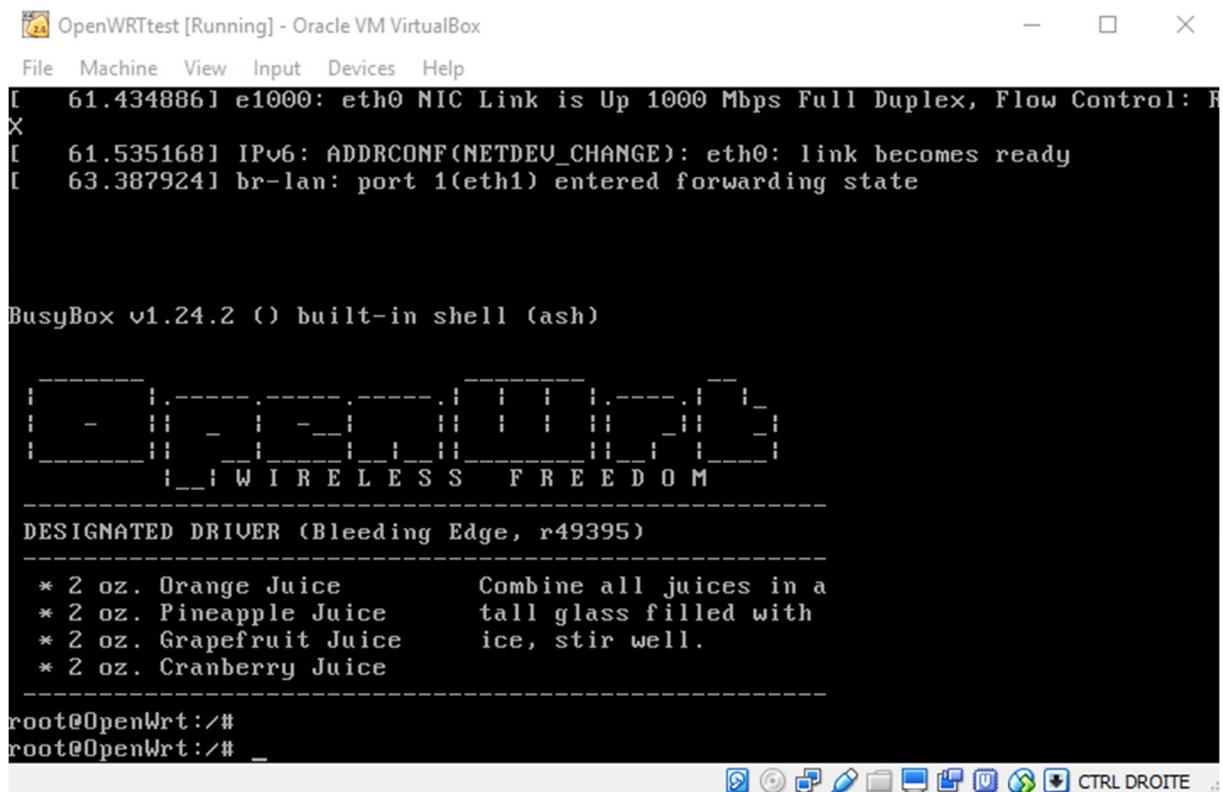


Figure 1 Example of the deployment of Flowtester devices.

The Flowtester system provides the user with time charts, Histograms, and any data useful for the maintenance of the targeted network. (For further details see the project description in the Annex section)

## Project Problematic

The Flowtester project has been undergoing a redesign from its initial form, which used to work on GNU/Linux, to its current state, which uses OpenWRT as operating system. OpenWRT is a Linux based operating system (OS) for embedded systems, which allow thorough customization for the user, and easier developing for the developer. This OS also provides an Ubus package, which is used by the Flowtester API to call the different functions implemented to realize the needed features.



```
OpenWRTtest [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
[ 61.434886] e1000: eth0 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: F
X
[ 61.535168] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
[ 63.387924] br-lan: port 1(eth1) entered forwarding state

BusyBox v1.24.2 () built-in shell (ash)

-----
 | _ | W I R E L E S S   F R E E D O M
|___|
-----
DESIGNATED DRIVER (Bleeding Edge, r49395)
-----
* 2 oz. Orange Juice      Combine all juices in a
* 2 oz. Pineapple Juice   tall glass filled with
* 2 oz. Grapefruit Juice  ice, stir well.
* 2 oz. Cranberry Juice
-----

root@OpenWrt:/#
root@OpenWrt:/#
```

Figure 2 OpenWRT OS ( Please note that this is the only interface for the user)

However, this operating system does not directly contain a user-friendly interface (as seen in fig 2), so in order to use all the flowtester devices features for the common client. The user has to use a LuCi web interface to use the flowtester device. This interface is implemented in Lua programming language (A cross platform language, based on a simplified version of the C language. Lua is designed to fit in embedded systems, and is a quick, adaptable programming code) for the most part, and completed with HTML language. The LuCi interface contains some basic features but the team needed someone to implement the Flowtester features, to start the testing phase of the device in September 2016.

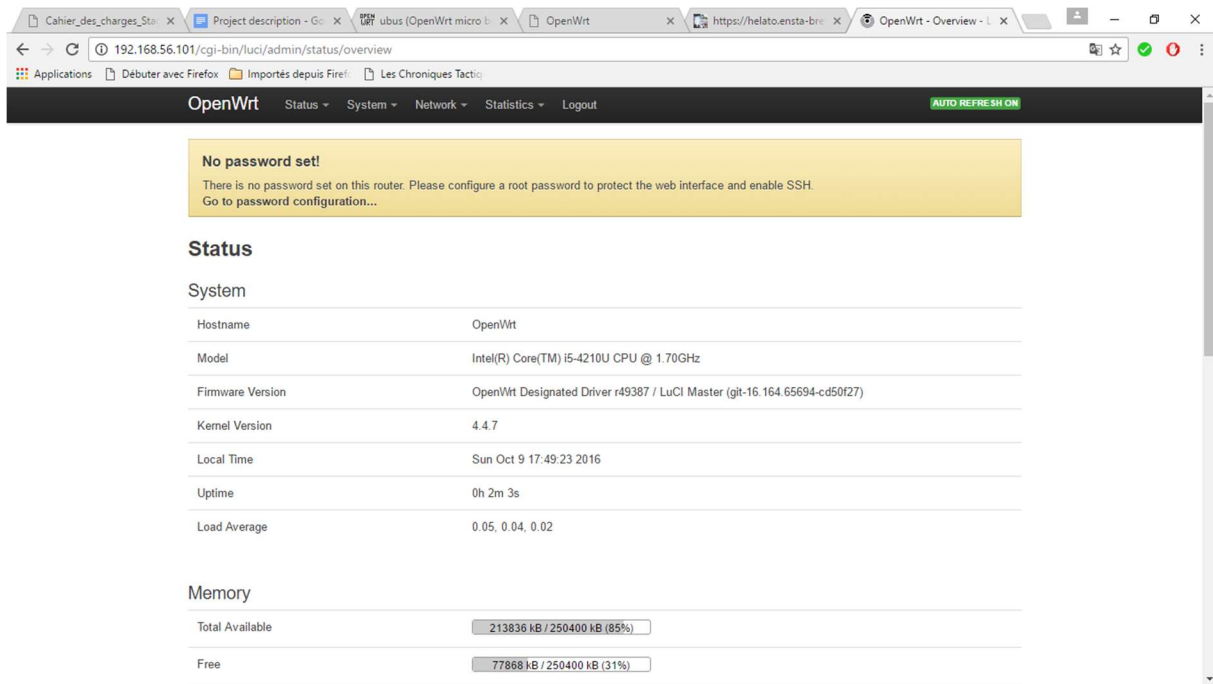


Figure 3 Basic LuCi Framework

This was the task I was due to accomplish during the length of the Internship. First I had to get used to the different OS and to learn the basics of the Lua language to proceed further. Then the creation of an Upper-menu to separate the basic functions from the Flowtester features was requested. In a second phase, the implementation of the features regarding the measuring patterns used by the Flowtester, and the measurement profiles were next. A third phase of the project was planned at the beginning but due to some delays, could not be treated.

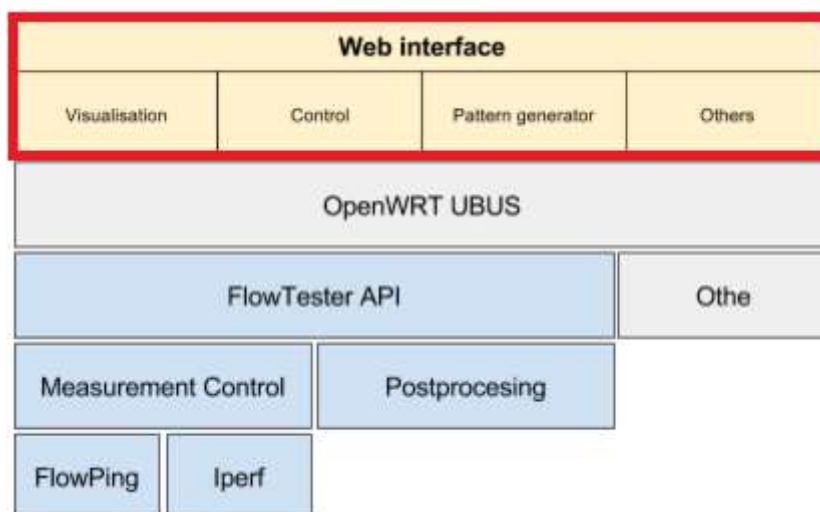


Figure 4 In red the phase of the project concerned by the Internship

## I Flowtester Configuration system

In this first part of the project as written above, the focus was put on the implementation of an Upper-menu to separate the LuCi features from the Flowtester features, as well as a web page which could control the configuration file of the Flowtester.

The main problem at the beginning was the learning time to handle the OpenWRT virtual machine I was using, and the Lua programming language (In both areas I was starting from scratch). With that being said the OpenWRT can be eluded by getting all the files from the virtual machine in an Ubuntu terminal via the command `sshfs` which allow the Ubuntu OS to import the files from the OpenWRT OS and gives a graphic interface much more comfortable to program, than the sole terminal provided by OpenWRT.

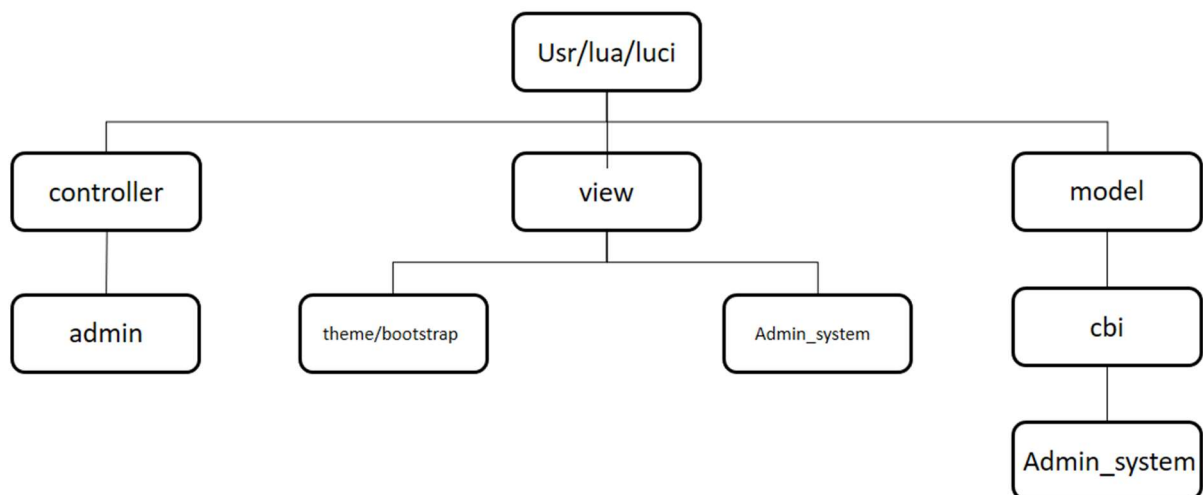


Figure 5 Simplified tree view of the system

In fig 5 I only show the relevant folders for the project, as the full tree view would be too heavy and confusing to show.

Then after getting used to the program, I could start modifying it in order to get the upper menu required. First I tried to create a menu regrouping the tabs seen on the default LuCi on one single tab, and another gathering the Flowtester features. But after several direct unsuccessful attempts, I decided to opt for buttons which would print or hide the selected menu as wished. For this I modified the `header.htm` file in `/view/theme/bootstrap`. There I implemented the button directly on the html code and modified the CSS and Javascript code in order to place and correctly print the menus on click. That created the wanted upper-menus. In order to add an upper menu, a



folder had to be created in the controller repository, as well as the Lua files managing the different path of the wished web pages on the OpenWRT system.

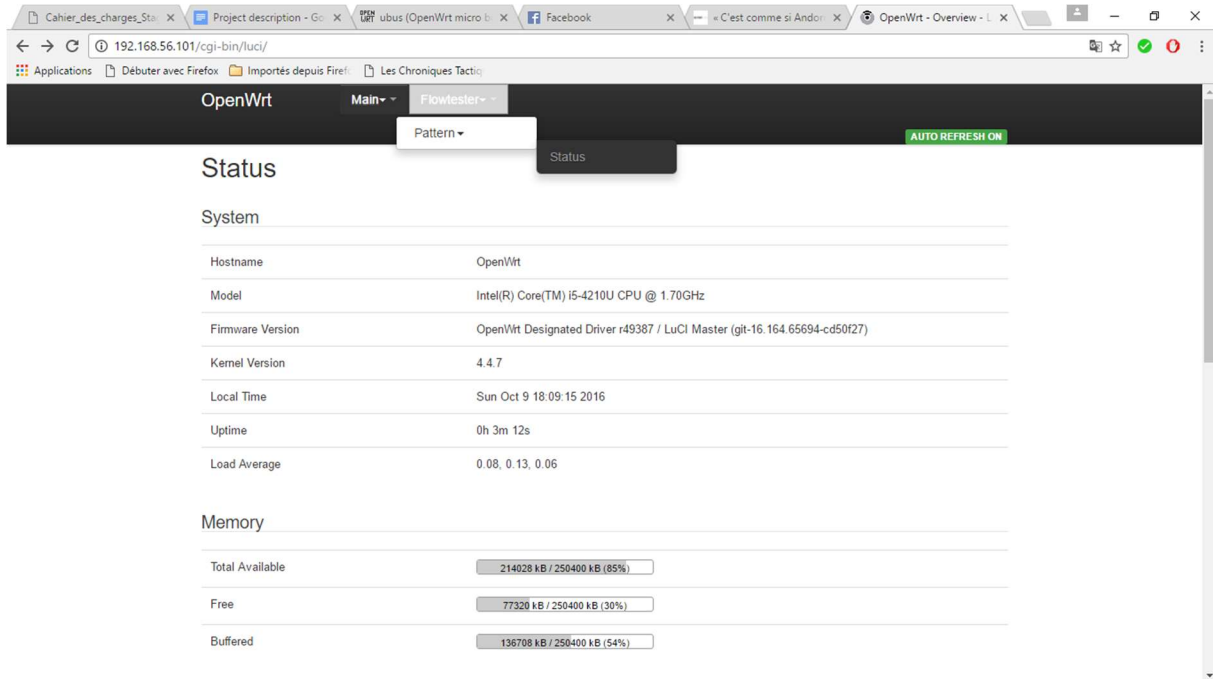


Figure 6 Final result of the upper-menus implementation

Then after I started the implementation of the configuration web page. Here the adopted method was inspired by an existing file in the LuCi interface.

First create a lua file that would get the path of the configuration file on the OpenWRT machine, then read the file and print it to the web page. For the customization, I applied the reverse strategy, reading the text on the web page, then write it down in the configuration file to change it.

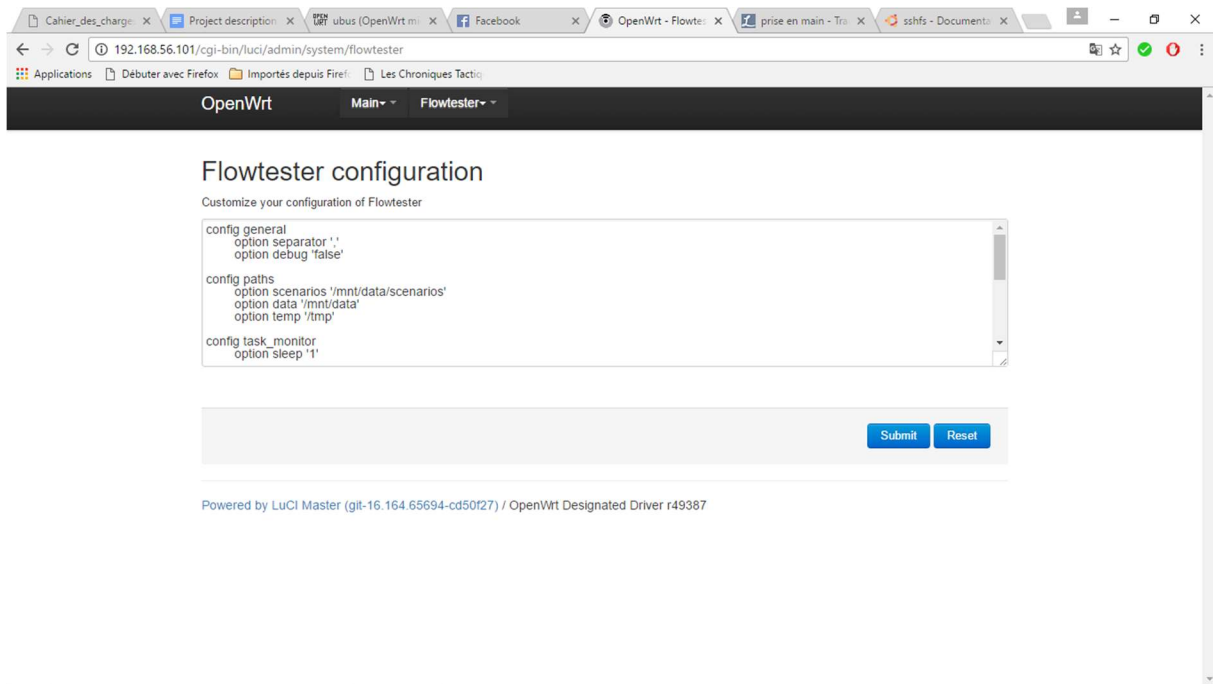


Figure 7 Configuration page

As exemplified in fig 7 the user is now able to see the configuration parameters and to modify them as he wishes.

## II. Pattern and Measurement configurator

For this part of the project, I had to create a graphical pattern generator, which could describe and manage the pattern used by the Flowtester device to monitor the system. First the web interface had to display all the different pattern listed in the OpenWRT machine, as well as buttons to edit them, remove them, and a button to redirect the user towards the Add/Edit page where he would be able to add or edit the pattern he wants. The solution had to use the Ubus console of the OpenWRT machine as simply reading file could not apply here. Indeed, the Ubus console is able to find the different patterns without knowing their location. And since the solution is supposed to apply to any device, it could not simply seek the pattern files on the machine since it is likely to change. The Ubus console is using JavaScript Object Notation to call function and returning the results, and is supposed to be called from the OpenWRT machine, not from the LuCI interface. But since Lua includes a Ubus library allowing the developer to call Ubus function directly from the Lua file, the problem was solved.

The Flowtester API already included several Ubus function concerning the patterns:

- List: List all the patterns detected on the device

```
Output: [  "pattern1": {
            "name": "pattern1",
            "filename": "/opt/flowtester/dataset/pattern1.dat",
            "modified": "DATE-TIME"
          },
  "pattern2": {
            "name": "pattern2",
            "filename": "/opt/flowtester/dataset/pattern2.dat",
            "modified": "DATE-TIME"
          },
]
```

- Read: Display all information about a specific pattern (name, path, and content)

```
Output: {
  "name": "pattern2",
  "filename": "/opt/flowtester/dataset/pattern2.dat",
  "content": [ [ 0, 10, 64 ], [60, 1024, 64], [60, 10, 150], [120, 1024,
150], [120, 10, 515] ]
}
```

- Create: Create a pattern, provided the user specifies, a name and the content of the pattern. The function does not care if the name written by the user already exists or not, so I also used it to edit the patterns when needed. This function does not contain an output.
- Delete: Delete the pattern specified by the user.

With all these functions already implemented, my task was to create the basic framework for the web interface (Which I based on other existing framework, the listing web page is based on the page displaying the processes on the status tab for instance). This consisted in the development of two pages:

- One displaying the list of all patterns in the Flowtester, allowing the user to delete patterns, or redirecting him to the second page
- The second page is used to add or edit patterns, and display further details about the concerned pattern (the table in the content area, and a graphical representation of this one)

## Pattern Status page

For this page I had to display the list of all patterns, offer the user a way to access the editing section, and the possibility to delete any patterns he wants. To allow him to do so I implemented two tabs at the top of the page:

- Status, the opened tab by default in order to display the list page
- Add/Edit, the hidden page by default, that redirects towards the Add/Edit page (If the user click on this tab he will be able to create his own pattern)

Then I created two buttons:

- Delete, to delete the pattern
- Edit to redirect the user to the same page as the Add/Edit tab but here with the possibility to get more details about the pattern he wants to edit, and the ability to edit it

The possibility of mixing Lua code directly inside html pages, allowed me to apply the pattern list pretty quickly inside the “view/admin\_system/pattern.htm” file I have created, via the function *get\_names()* calling the Ubus listing function, and putting the String in the name section in a temporary table, before the function *render\_name()* calls the Ubus reading function and print the name. The table from the first function also allows to create a for loop in which the HTML code displaying the names and the buttons is written and the second function is called to print the names in the right place.

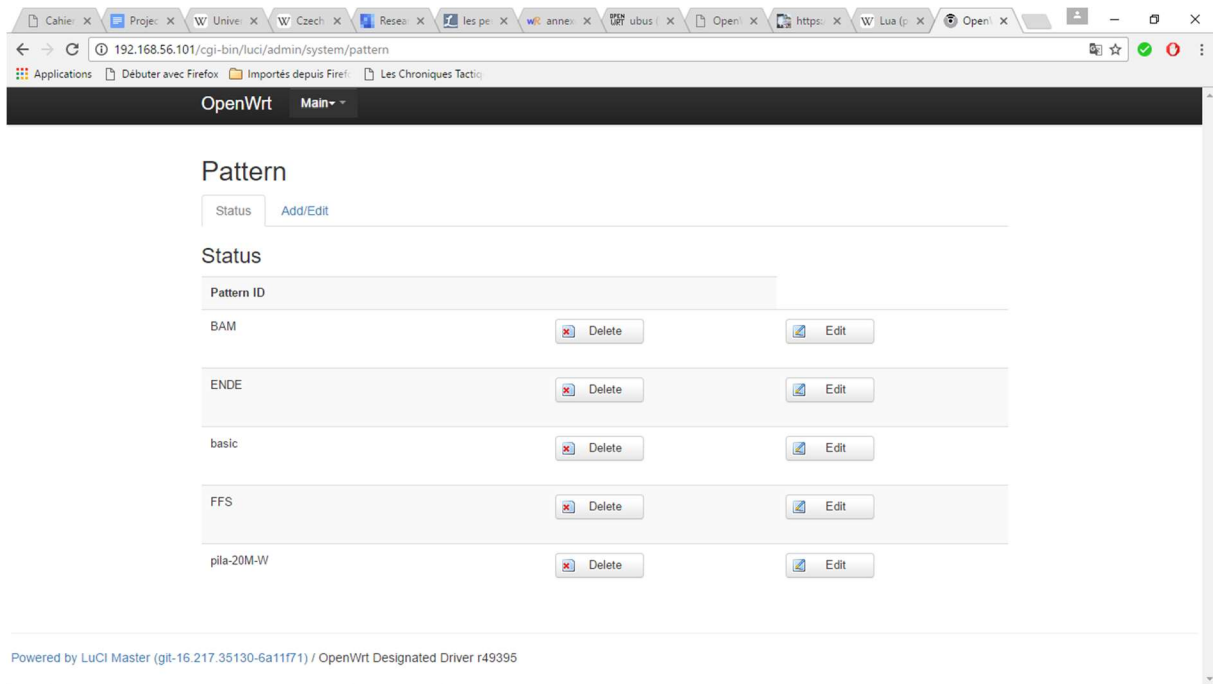


Figure 8 Result of the listing of the patterns

Regarding the link towards the Add/Edit page, the Add/Edit tab on the top of the page can redirect the user from this page to the Add/Edit page, which in this case would contain nothing (Since here the user can create his pattern from scratch).

### Pattern Editor

Add a new pattern, or edit an existing one

Status Add/Edit

```

y=40000 fs=4000
-----
y=30000 fs=3000
-----
y=20000 fs=2000
-----
y=10000 fs=1000
-----
y=red fs=green
  
```

Figure 9 The Add/Edit tab result

### Pattern Name

In this text area you can choose the name of the pattern you want to edit  
 If you don't change anything you will modify the pattern displayed onscreen  
 If the name doesn't exist, a new one will be created. The name HAS TO start with a letter!

t	y	FS
<input type="text"/>	<input type="text"/>	<input type="text"/>

Add row

Edit

However, for the implementation of the delete button, the page had to be refreshed, and for the edit button, the button has to redirect to the Add/Edit page corresponding to the pattern the user wants to edit. For the delete feature, I use a JavaScript function *delete\_data(id)* which responds when the button is pressed, then the function gathers the name of the pattern the user wants to delete, sends it towards the “controller/admin/system.lua” file and refresh the current page. There another function called *delete\_data(x)* too (I gave both functions the same name because both are part of the same function) that calls the Ubus function that deletes the pattern from the Flowtester filesystem.

As for the Edit button, I used a get method in the HTML code in order to get the name of the pattern the client wishes to edit, and display it on the web address of the Editing page. Thus if you click on the Edit button corresponding to the “basic” pattern, the web address of the editing page will be: “http://192.168.56.101/cgi-bin/luci/admin/system/pattern/add\_pattern?name=basic”. The rest of the function is implemented on the page displaying the editing feature and will be developed in the section describing it.

### [The Add/Edit pattern page](#)

On this page the user needed to be able to see all the relevant details about the pattern he wants to edit (The name, the pattern data, which is a table defining the dataflow, as well as the frame size the pattern uses at a given time, and a graph representing this pattern data).

In order to do so I created the file “/view/admin\_system/add\_pattern”, to create the wanted framework. In this file lies the end of the edit button implemented in the Pattern Status page. To get the name of the pattern we want to edit the program will automatically read the web address via JavaScript instructions, as the “get” method was used in the edit form programming the button, the name of the corresponding

pattern is displayed in the web address. Then this address is filtered to get only the name of the pattern. (This is actually a two steps process, in order to determine if the user has clicked the edit button on the pattern status page, or the Add/Edit tab.) Afterwards, the name of the pattern to display (or just a blank if the user pressed Add/Edit) is sent towards "controller/admin/system.lua" once again, to trigger the function `render_data(x)`. This function gets the name of the pattern and print it into a text file. This text file is then read by the `render_data()` function from `"/view/admin_system/add_pattern"` . This function calls the Ubus pattern reading function and getting the corresponding pattern content.

OpenWrt Main -

### Pattern Name

In this text area you can choose the name of the pattern you want to edit  
 If you don't change anything you will modify the pattern displayed onscreen  
 If the name dosen't exist, a new one will be created. The name HAS TO start with a letter!

t	y	FS	
<input type="text" value="60"/>	<input type="text" value="50"/>	<input type="text" value="0"/>	<input type="button" value="Delete row"/>
<input type="text" value="120"/>	<input type="text" value="7000"/>	<input type="text" value="64"/>	<input type="button" value="Delete row"/>
<input type="text" value="180"/>	<input type="text" value="0"/>	<input type="text" value="256"/>	<input type="button" value="Delete row"/>
<input type="text" value="240"/>	<input type="text" value="20000"/>	<input type="text" value="256"/>	<input type="button" value="Delete row"/>
<input type="text" value="300"/>	<input type="text" value="0"/>	<input type="text" value="512"/>	<input type="button" value="Delete row"/>
<input type="text" value="360"/>	<input type="text" value="20000"/>	<input type="text" value="512"/>	<input type="button" value="Delete row"/>
<input type="text" value="420"/>	<input type="text" value="0"/>	<input type="text" value="1024"/>	<input type="button" value="Delete row"/>
<input type="text" value="480"/>	<input type="text" value="20000"/>	<input type="text" value="1024"/>	<input type="button" value="Delete row"/>
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="button" value="Add row"/>

---

Powered by LuCI Master (git-16.217.35130-8a11f71) / OpenWrt Designated Driver r49395

Figure 10 Displaying of the pattern data for the test pattern

Then the table is refined in order to get the values inside, and finally displaying it in the appropriate fields as well as the name gathered from the web address. This whole operation requires the page to be reloaded once in order to get the correct name inside the text file mentioned earlier, a JavaScript instruction which reload the page, but add a String at the end of the web address allows this solution to be possible (Indeed at first the page would reload each time it is called, resulting in an infinite loop) This answer to this problem might not be the best or the most elegant, but is the only viable one so far.

Regarding the editing part of the page, the user had to be able to remove and add rows as he wished. However, it was not realistically possible to allow the user to add or remove several rows at the same time, so I have settled for a solution allowing the client to add or remove only one row at a time. Both functions and the function which only modify data from a row a very similar to each other. Indeed, all three of them uses the Ubus function which create/overwrite the pattern to operate. The only difference resides on the amount of data each function sends. For the *create\_data()* function, it reads the values on the fields containing it, pack it into a table, gets the name of the pattern to edit, and then sends it to the “controller/admin/system.lua” file where, the Ubus function is called. Then the Editing page is refreshed, allowing the user to see the changes. For the *add\_row()* function, same principle, but the last set of values is added to the table of values gathered by the function, and then is sent to the “system.lua” file. And for the *delete\_row()*, same idea again but here the values of the deleted row are set to “null” and then sent to the system.lua file.



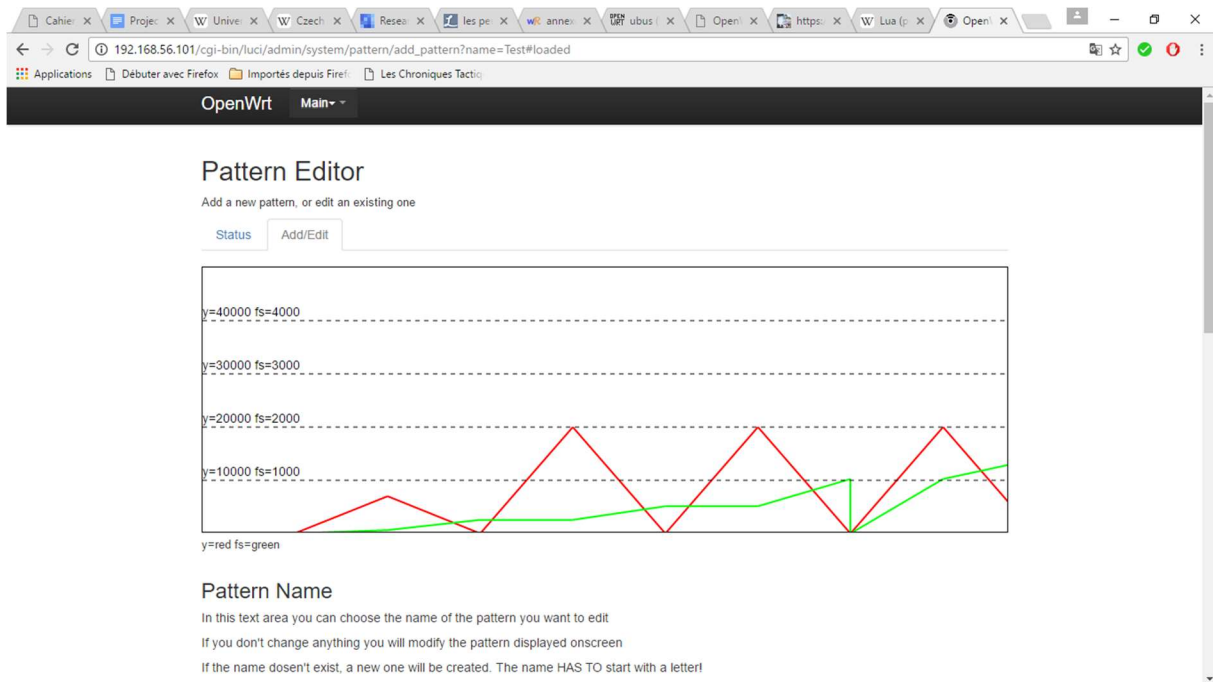


Figure 11 Graph representation of the pattern data

Finally, for the graph representation, I have used the SVG library from the HTML language. The time values, as well as the dataflow, and the frame size, are gathered from the HTML code, and put into tables regrouping each type of values, time values are sorted to remain consistent when drawn, and, thanks to a loop, the dataflow, and the frame size are drawn at a given time.

### III Measurements

Last part of this step of the project was the creation of three pages in order to manage the measurements profiles of the Flowtester:

- One to list those profiles and give access to the two other pages.
- One to create a profile thanks to the Ubus function for the creation of such profile.
- One to see the status of a running profile

## Profile list page

For this page I had to display any profile available on the Flowtester filesystem, creating solutions to start, stop, delete, and access the status of a given profile. Plus, the page had to contain a link towards the profile creation page.

Here, the solution to display the profile is nearly identical to the one used to display the available patterns (both requirements are similar after all). First get the names from the Ubus function listing the profiles, and then putting them into a table to print them with a for loop.

Same thing with the buttons Start, Stop, Delete. One JavaScript function to assess whether or not the button has been pressed or not, then reading the name of the appropriate pattern from the HTML code, and sending it towards “controller/admin/system.lua”, calling the proper Ubus function.

And for the Status function, the method is similar to the Edit button from the Pattern Status page (Get form, then the other page handles the rest with the web address transmitting the proper name)

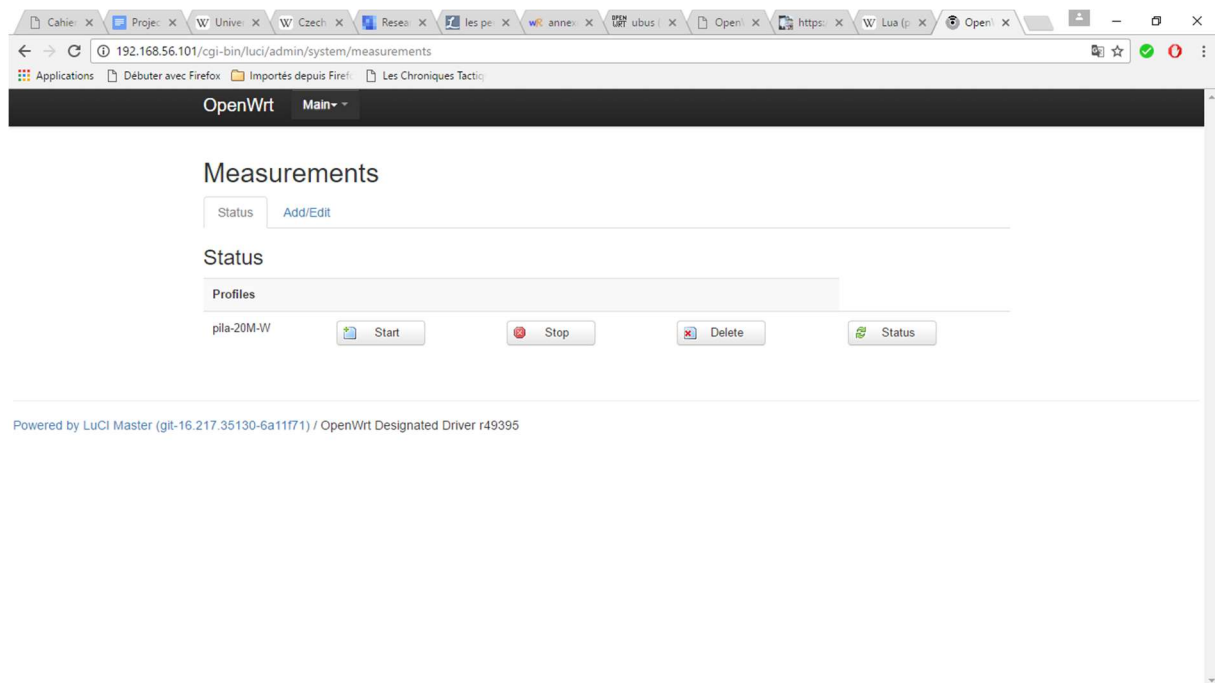


Figure 12 Final result of the Measurement page

## Profile Status page

For this page, I was requested to display any data about a running measurement profile. And as the previous page I used the same method as the Pattern Editing page to implement it:

- First getting the name of the profile the user wants to monitor from the web address
- Then sending it to “controller/admin/system.lua” and writing it into a temporary text file
- Read this text file from the Profile status page, and calling the Ubus tasks status function
- Filter and print the appropriate data on the page

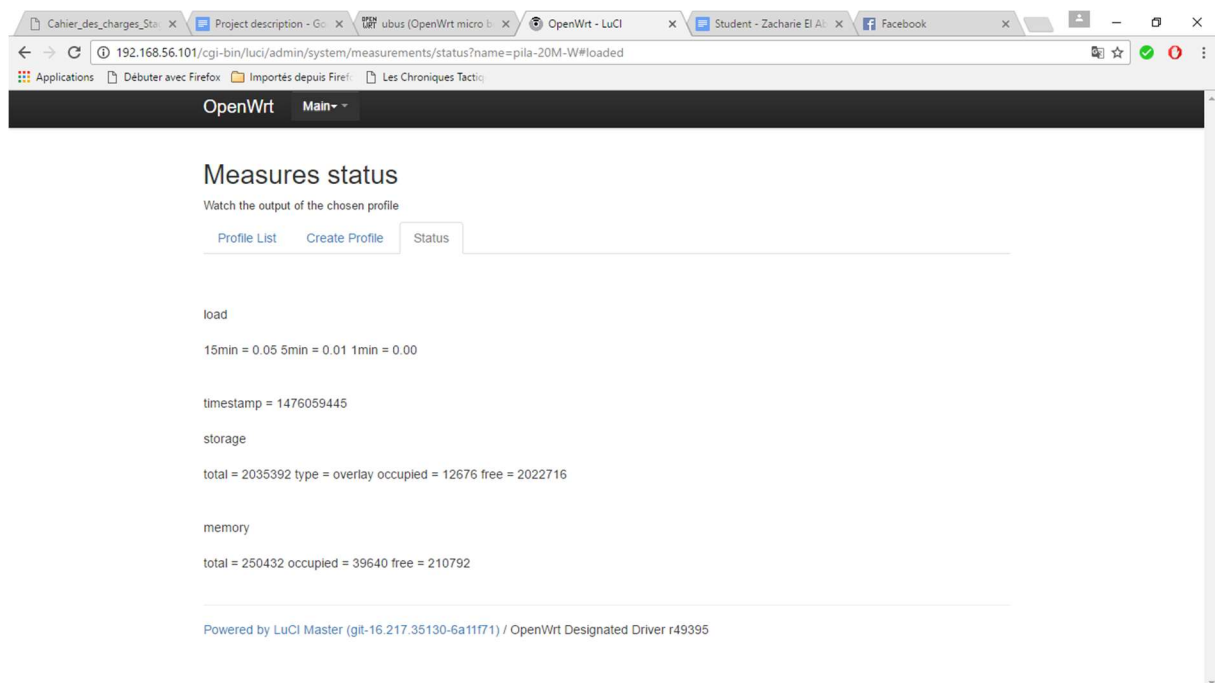


Figure 13 Final look of the Profile status page

## Profile Creation page

For this final part of the project, the page had to allow the user to create a measurement profile of his choice, with all the parameters he wants. As the two previous implemented pages, the task was similar to the Pattern Creation/Edition page, the main problem here was that the Ubus function allowing a user to create a profile contained up to 32 parameters. Apart from that the method is the same as the *create\_data()* function:

- Gathering all required data from the HTML code
- Creating a table, and filling it with all gathered data
- Sending the data to “controller/admin/system.lua”
- Calling the Ubus function that creates the pattern

Unfortunately, the method used here did not produced the expected results. However, since this method is nearly identical to the method used in the Pattern Creating/Editing page, I can see only three reasons why it does not work yet:

- The Ubus function which creates the profiles might contain a bug preventing it to work correctly.
- This Ubus function requires Tables of tables as parameters, and I have seen no guarantee that this is accepted by the Ubus library in Lua.
- Since I had to use many loops in order to cover all the cases posed by the 32 parameters, a slight error might still be in the code (However this is very unlikely. Indeed, the function is implemented in the “controller/admin/system.lua” file, and if an error is detected in any file in the controller, the LuCi does not even launch and points out the error)

Here are all the parameters of the Create function

```
{
  "name": "test1",
  "description": "Test 1 description",
  "duration": "20",
  "repetition": "5",
  "delay": "1",
  "sleep": "1",
  "schedule": {
    "disable": "true",
    "year": "string",
    "month": "string",
    "day": "string",
    "hour": "string",
    "minute": "string"
  },
  "load": {
    "disabled": "false",
```

```

        "loop": "5"
    },
    "flowping": {
        "disabled": "false",
        "dataset": {
            "filename": "/opt/flowtester/data.dat",
            "content": [ [ 0, 10, 64 ], [60, 1024, 64], [60, 10, 150],
[120, 1024, 150], [120, 10, 515] ]
        },
        "target_IP": "147.32.211.110",
        "target_PORT": "2424",
        "opts": "-X"
    },
    "iperf": {
        "disabled": "true",
        "target_IP": "string",
        "target_PORT": "integer",
        "opts": "string"
    },
    "analyze": {
        "disabled": "true",
        "mode": "measurement",
        "opts": "string",
        "csv": "bool",
        "graphs":{
            "type":
["full","all","loss","lossra","loss_hist","spd","spdl","spd_loss","spdl_los
s","spd_lossra","spdl_lossra","spd_delay","spdl_delay","delay","delay_hist"
,"delay_loss","delay_lossra"],
            "output": "pdf/png"
        }
    }
}

```

## Profile Creator

Fill the parameters you want for your profile

Status Add/Edit

### Profile Description

Write down the description of your profile

Name	Duration	Repetition	Delay	Sleep
<small>(WITHOUT SPACES)</small>				
<input type="text" value="Name of the profile"/>	<input type="text" value="Duration"/>	<input type="text" value="Repetitions"/>	<input type="text" value="Delay"/>	<input type="text" value="Sleep"/>

Flowtester On/Off	Filename	Content	Target_IP	Target_Port	Options
<input type="radio"/> Enable	<input type="text" value="String"/>	<input type="text" value="String"/>	<input type="text" value="String"/>	<input type="text" value="String"/>	<input type="text" value="String"/>

Schedule On/Off	Year	Month	Day	Hour	Minute
<input type="radio"/> Enable	<input type="text" value="String"/>	<input type="text" value="String"/>	<input type="text" value="String"/>	<input type="text" value="String"/>	<input type="text" value="String"/>


Analyze On/Off	Mode	Options	CIV	Graph type	Graph output
<input type="radio"/> Enable	<input type="text" value="Measurement"/>	<input type="text" value="String"/>	<input type="text" value="Bool"/>	<input type="text" value="full,all,loss,losses,lor"/>	<input type="text" value="pdf/png"/>

iPerf On/Off	Target_IP	Target_Port	Options
<input type="radio"/> Enable	<input type="text" value="String"/>	<input type="text" value="String"/>	<input type="text" value="String"/>

Load On/Off	Loops
<input type="radio"/> Enable	<input type="text" value="String"/>

 Edit

Powered by LuCI Master (git-16.217.35130-6a11f71) / OpenWrt Designated Driver #49395

Figure 14 Measurement creation page

Apart from these three reasons, since the reasoning is the same as the Pattern Editing page which works just fine, I cannot think of any other why the solution should not work as well.

## Conclusion

During this internship, I have been confronted to several problematics entirely new to me. First of all, working in an international context. Indeed, this causes much more misunderstandings than I thought and I sometimes misunderstood what did my training supervisor (Mr Kozak), exactly expected from me (Especially in the second part of the project). In the other way around I found out that, still during the second part of the project, I was not provided with the proper virtual machine (Containing some features I needed in order to advance). This means I still have to work on my communication skills in order to limit those problems in the future. The second new problem I faced during this internship was the confrontation with several programming languages I had no prior knowledge. This forced me to learn and adapt quickly to all of these new environments, and will definitely be of use in my professional career, since I will surely face this issue once again in a future project.

Finally, the majority of the tasks required, and that I was realistically able to carry out for this project have been achieved and will help the CTU to launch the test phase of the Flowtester project before, letting it into the hands of the customers. With that being said, I think some of the solutions proposed still have room for improvement, to be more efficient, and more easily customized.

I would like to thank Mr. Bestak for the opportunity he has given me to work as an intern at the CTU in Prague, and Mr. Kozak, and Mr. Kocur, for having me working on this project, and for their help throughout my entire stay in Czech Republic.

# Web user-interface for FlowTester

## Main goal

In cooperation with the team at the CTU in Prague, design and develop a new web interface (UI) of FlowTester a network benchmarking tool. Currently, the FlowTester runs on GNU/Linux, but it undergoes a redesign and was ported to OpenWRT.

Project tasks on the UI are divided among a few connected projects where you will learn how to work with OpenWRT's UBUS interface for configuring tests and reading status information of FlowTester. Based on that information read from UBUS interface, you will create a web interface for visualization where you will show defined measurements and their status. This web interface will allow several actions, e.g., a definition of the traffic pattern, measurement, and postponed measurement. Each feature is detailed in the following text.

## FlowTester

FlowTester is a set of software modules which are designed to measure parameters of communication networks based on TCP/IP protocols. These measurements rely on traffic definitions based on predefined traffic patterns. Obtained results of such measurements are given in time-sequence parameters such as a response time of the communication network, round-trip time, and error rate. Results are then visualized in time-sequence diagrams using PDF format. Additionally, FlowTester stores all measured results in the CSV format in the internal memory. Analysis can be conducted by FlowTester itself or on a master unit which can interconnect a number of FlowTesters, and, subsequently, analyze data and control measurements from them in a 24/7 mode of operation.

FlowTester allows network testing in

- Short-term manner
  - brief verification of a network
  - bottleneck detection
- Long-term manner
  - detailed analyses scaled to hours, days, and weeks
  - communication stability tests

Testing is carried out using:

- Predefined traffic pattern measurements (traffic patterns of time-varying loads and packet length): constant, steps, saw traffic pattern or other comprehensive traffic patterns



- Measurement of network throughput over time (using TCP flows)
- Multiple flows between various FlowTesters in the network
- Simulation of traffic patterns based on capture traffic, e.g., industrial protocols, http, VoIP, IPTV, etc.
- DoS and DDoS attacks

Based on those tests FlowTester provides:

- Time charts (combination of multiple charts in one graph)
- Histograms (distribution of a measured parameter's frequency)
- Statistics and box-plots (mean, variance, median, min-max value)
- Threshold detection

Typical deployment of FlowTesters for a measurement is depicted in the Fig 1.

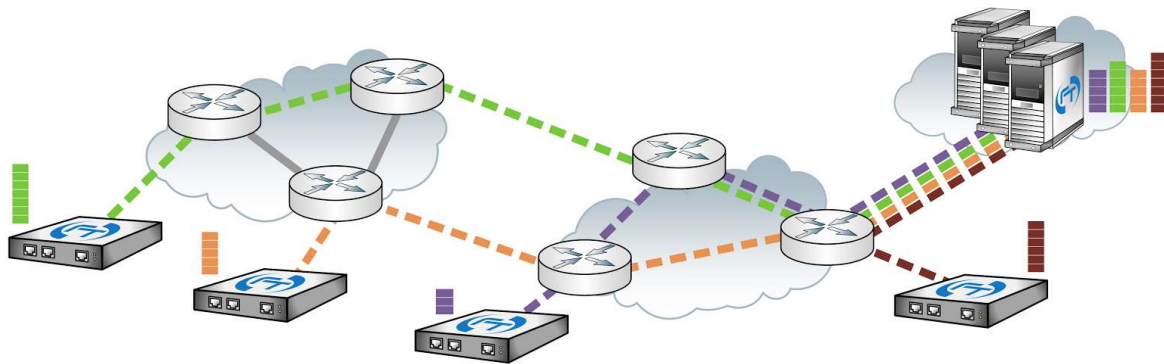


Fig. 1: An example of FlowTester deployment in a more comprehensive scenario in a point-to-multipoint case (P2MP or MESH). Performance analysis is carried out between two FlowTesters or between FlowTesters and a VM in a datacenter.

One FlowTester can coordinate multiple FlowTesters as long as they can communicate using TCP/IP. This coordination is carried out by distribution of profile and data files among devices. When those files are distributed, tests are initiated through remote command over SSH.

When tests are finished, postprocessing is carried out in the device, and this way each FlowTester can provide visualisation of measured network performance. Such examples are captured in Fig. 2 and 3.

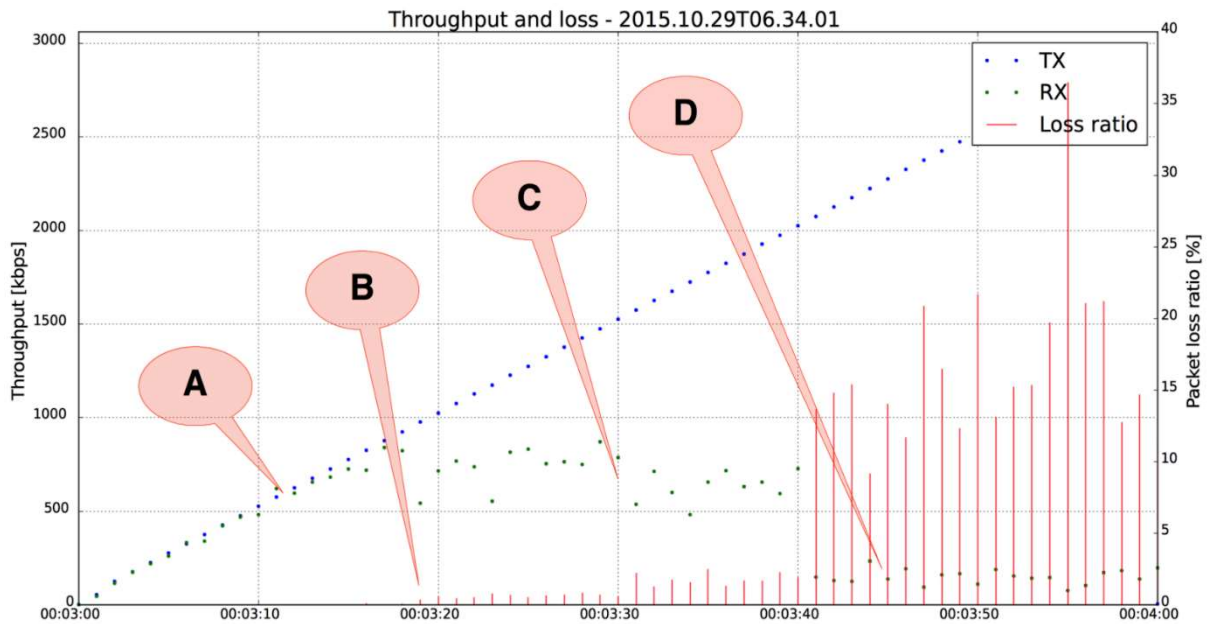


Fig. 2: Graphical visualisation of maximum throughput detection for a measured network connection. This measurement is based on the incremental increase of offered load of traffic and the monitoring of real network throughput. Letters A - D denote important areas of the graphical visualisation. Point **A** indicates the saturation threshold. Point **B** shows the moment when the first packets begin to loss. Point **C** shows the maximal network throughput just before another threshold and causing an increase of packet loss. A highly overloaded connection is underpinned by area **D**.

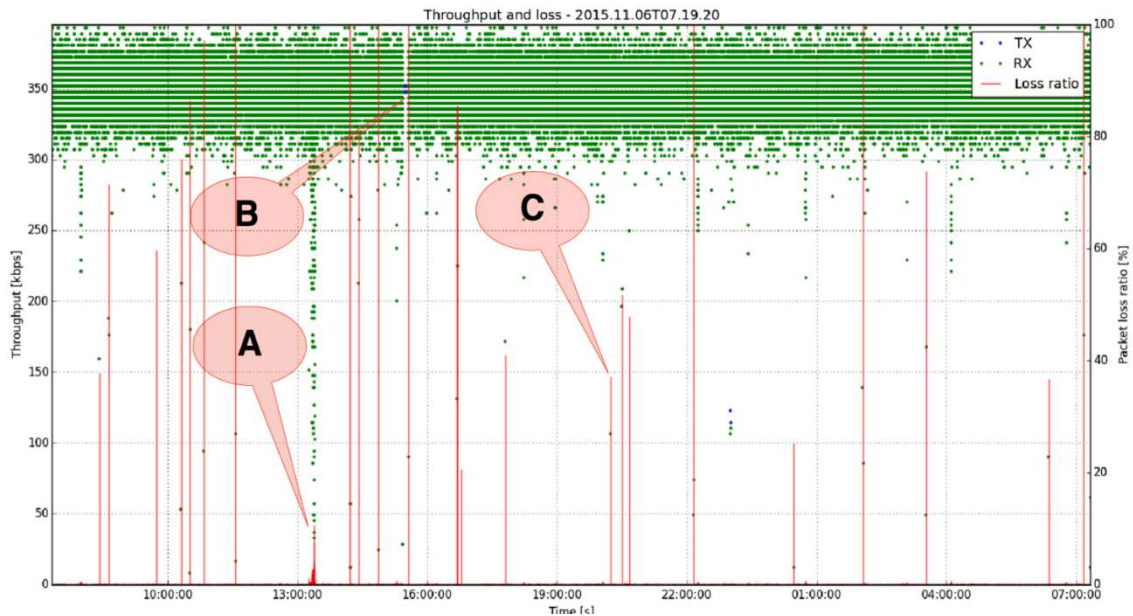


Fig. 3: This graph captures the long-term measurement (24h) of packet loss with a data connection continually loaded by an offered load of 350 kbps. Over time, packet loss is monitored. Points A - C show typical events. Point **A** shows the obvious degradation of transmission parameters of the data connection leading to a decrease of throughput and an increase of packet loss. Point **B** indicates connection disconnect in length in tens of minutes. Point **C** shows the short-term degradation of data connection parameters in tens of seconds.

## State of the art

The FlowTester redesign into OpenWRT platform allows multiple features like concurrent measurements provided by one device, etc. FlowTester itself relies on FlowPing and Iperf applications which are designed for performance testing of data networks

## OpenWRT UBUS

UBUS provides communication between various daemons and applications in OpenWrt, and UBUS is deeply described at Wiki page: <https://wiki.openwrt.org/doc/techref/ubus>

## FlowTester API

All necessary software, e.g., Iperf and FlowPing, is installed in the provided system, and FlowTester API is accessible through ubus interface:

```
root@OpenWrt:/# ubus -v list
'flowtester' @7926f7ca
  "tasks": {"action": "String", "task_id": "String", "value": "String"}
  "pattern": {"action": "String", "name": "String", "content": "String"}
  "results": {"action": "String", "value": "String"}
  "servers": {"server": "String", "action": "String"}
```

Each module is described in more details in the following chapters. The FlowTester API is used by command on command line of FlowTester called *flowtester-cli*. This command can be used as a reference for development of web interface.

## Module Pattern

The point of this module is to manage traffic patterns that carry information about link load during time. Each traffic pattern can represent specific communication protocol, or just any performance test.

Patterns can be managed using 4 actions.

**Actions:** list, read, create, delete

### list

This method provides list of patterns stored on the filesystem of FlowTester device. The list of patterns can be accessed using JSON or through flowtester-cli:

Input JSON:

```
{"action": "list", "name": "pattern1/all"}
```

Output JSON:

```
[  "pattern1": {
    "name": "pattern1",
    "filename": "/opt/flowtester/dataset/pattern1.dat",
```

```

        "modified": "DATE-TIME"
    },
    "pattern2": {
        "name": "pattern2",
        "filename": "/opt/flowtester/dataset/pattern2.dat",
        "modified": "DATE-TIME"
    },
]

```

## read

Action read provides all necessary information about one specific traffic pattern.

Input JSON:

```
{"action": "list", "name": "pattern1"}
```

Output JSON:

```

{
    "name": "pattern2",
    "filename": "/opt/flowtester/dataset/pattern2.dat",
    "content": [ [ 0, 10, 64 ], [60, 1024, 64], [60, 10, 150], [120, 1024,
150], [120, 10, 515] ]
}

```

## create

Write action stores defined pattern onto filesystem of FlowTester. This does not check existence of other patterns of the same name. The overwriting tests must be carried out programatically.

Input JSON:

```
{"action": "list", "name": "pattern1", "content": [ [ 0, 10, 64 ], [60, 1024, 64], [60, 10, 150], [120, 1024, 150], [120, 10, 515] ]}
```

## delete

This action delete profile out of filesystem, so list wont show such a traffic pattern when listing is performed for the next time.

Input JSON:

```
{"action": "delete", "name": "pattern1"}
```

## Module Tasks

Module tasks is responsible for all actions related to tasks. Task can be understand as an combination of a traffic pattern and extra information which create an instruction for a network measurement or performance test.

**Actions:** list, create, delete, start, stop, status

## list

List is used in order to show all prepared measurements. The response of this actions gives information about all defined measurements and their state, as whether measurement is running or not.

Input JSON:

```
{"action": "list"}
```

Output JSON:

```
{
  "test1": {
    "name": "test1",
    "description": "Test 1 description",
    "filename": "\\mnt\\data\\profiles\\test1.profile",
    "running": false
  },
  "test2": {
    "name": "test1",
    "description": "Test 1 description",
    "filename": "\\mnt\\data\\profiles\\test2.profile",
    "running": false
  },
  "test3": {
    "name": "test1",
    "description": "Test 1 description",
    "filename": "\\mnt\\data\\profiles\\test3.profile",
    "running": false
  }
}
```

## delete

FlowTester API allows deleting measurement profiles.

Input JSON:

```
{"action": "delete", "task_id": "test3"}
```

Output JSON:

```
{
  "status": "OK",
  "message": "Task test3 (\\mnt\\data\\profiles\\test3.profile) was deleted"
}
```

Status gives feedback about the process. In case of failure message attribute provides information about the failure.

## create

Create action requires a complex JSON description of operation thus this description is going to be described in more details. Each profile defines properties of a measurement. An example of profile is as follows:

```
{
  "name": "test1",
  "description": "Test 1 description",
  "duration": "20",
  "repetition": "5",
  "delay": "1",
  "sleep": "1",
  "schedule": {
    "disable": "true",
    "year": "string",
    "month": "string",
    "day": "string",
    "hour": "string",
    "minute": "string"
  },
  "load": {
    "disabled": "false",
    "loop": "5"
  },
  "flowping": {
    "disabled": "false",
    "dataset": {
      "filename": "/opt/flowtester/data.dat",
      "content": [ [ 0, 10, 64 ], [60, 1024, 64], [60, 10, 150],
[120, 1024, 150], [120, 10, 515] ]
    },
    "target_IP": "147.32.211.110",
    "target_PORT": "2424",
    "opts": "-X"
  },
  "iperf": {
    "disabled": "true",
    "target_IP": "string",
    "target_PORT": "integer",
    "opts": "string"
  },
  "analyze": {
    "disabled": "true",
    "mode": "measurement",
    "opts": "string",
    "csv": "bool",
    "graphs": {
      "type":
["full", "all", "loss", "lossra", "loss_hist", "spd", "spd1", "spd_loss", "spd1_los
s", "spd_lossra", "spd1_lossra", "spd_delay", "spd1_delay", "delay", "delay_hist"
, "delay_loss", "delay_lossra"],
      "output": "pdf/png"
    }
  }
}
```

One can see the first part contains description of measurement specifying name, repetition, and trigger for measurement starting:

```
"name": "test1",
"description": "Test 1 description",
"duration": "20",
"repetition": "5",
"delay": "1",
"schedule": {
  "disable": "true",
  "year": "string",
  "month": "string",
  "day": "string",
  "hour": "string",
  "minute": "string"
},
```

In detail:

- **name** - Name of the test.
- **description** - Detail description of the test.
- **duration** - Duration of the measurement being performed in seconds.
- **repetition** - Number of repetition desired test
- **delay** - Delay in seconds before running the test
- **schedule** - Date end time definition for running the test. (Not working now)

Following, optional part, contains flowping measurement description:

```
"flowping": {
  "disabled": "false",
  "dataset": {
    "filename": "/opt/flowtester/data.dat",
    "content": [ [ 0, 10, 64 ], [60, 1024, 64], [60, 10, 150],
[120, 1024, 150], [120, 10, 515] ]
  },
  "target_IP": "147.32.211.110",
  "target_PORT": "2424",
  "opts": "-X"
},
```

Dataset describes traffic pattern used for measurement, and dataset can be included in two ways: filepath or by list of tuples describing time, throughput, and packet size. Following are parameters for flowping application such as destination address, port, and other parameters.

Similarly, Iperf can be configured. However, Iperf allows only to specify destination address, port, and extra attributes. This configuration comes from the Iperf measurement method.

```
"iperf": {
  "disabled": "true",
  "target_IP": "string",
  "target_PORT": "integer",
  "opts": "string"
},
```

Eventually, measurement profile specifies what statistics should be carried out:

```
"analyze": {
  "disabled": "true",
  "mode": "measurement",
  "opts": "string",
  "csv": "bool",
  "graphs":{
    "type":
["full","all","loss","lossra","loss_hist","spd","spd1","spd_loss","spd1_los
s","spd_lossra","spd1_lossra","spd_delay","spd1_delay","delay","delay_hist"
,"delay_loss","delay_lossra"],
    "output": "pdf/png"
  }
}
```

This JSON measurement description is send through FlowTester API then this measurement profile can be seen in the list of profiles.

### **start**

Unless the delayed started in defined with create action JSON measurement profile is in stop state. In order to start such a profile action start is triggered by following JSON.

Input JSON:

```
{"action":"start","task_id":"test2"}
```

### **stop**

Any running measurement profile can be stop by the stop action:

Input JSON:

```
{"action":"stop","task_id":"test2"}
```

Output JSON:

```
{
  "test2": {
    "status": "OK",
    "message": "test2 with PID: 17216 was killed"
  }
}
```

### **status**

Status provides information about running measurement profiles. In order to initiate profile verification action status and identification of task must be triggered.

Input JSON:

```
{"action":"status","task_id":"test2"}
```

If the status action is performed on not running profile.

Output JSON:

```
{
```



```

    "status": "Info",
    "message": "No active tasks"
}

```

On the other hand, the running profile can provide status information about running processes related to the profile.

Output JSON:

```

{
  "test2": {
    "pid": 17216,
    "start_time": 1466670966,
    "progress": 0,
    "current_task": 2,
    "max_tasks": 5,
    "task_time": 20,
    "project_time": 88,
    "data_file": "\mnt\data/test2-1466670966/test2-002/test2-002-2016.06.23T08.36.28.gz",
    "data_file_size": 0
  },
  "system": {
    "timestamp": 1466670990,
    "memory": {
      "free": 214876,
      "occupied": 35524,
      "total": 250400
    },
    "storage": {
      "type": "overlay",
      "free": 1026312,
      "occupied": 6968,
      "total": 1033280
    },
    "load": {
      "1min": "0.08",
      "5min": "0.03",
      "15min": "0.04"
    }
  }
}

```

Status provides information about FlowTester itself as well as information about measurement supporting processes.

## Module Results

Access to each measurement is provided through FlowTester API which provides

- Measurement listing in order to show results of finished measurements
- Results delete
- JSON measurement descriptor
- File download from the results directory

**Actions:** list, summary, detail, delete

## summary

Basic statistics of stored measurements.

- measurements - number of stored measurements
- tasks - number of stored tasks
- size - size of all measurements in bytes

Input JSON:

```
{"action": "summary"}
```

Output JSON:

```
{
  "measurements": 2,
  "tasks": 7,
  "size": 3
}
```

## list

Input JSON:

```
{"action": "list"}
```

Output JSON:

```
{
  "test1-1466663949": {
    "timestamp": "Thu Jun 23 06:39:09 UTC 2016",
    "size": 67,
    "report": false,
    "instances": {
      "1": "test1-001",
      "2": "test1-002"
    }
  },
  "test2-1466670966": {
    "timestamp": "Thu Jun 23 08:36:06 UTC 2016",
    "size": 222,
    "report": false,
    "instances": {
      "1": "test2-001",
      "2": "test2-002",
      "3": "test2-003"
    }
  }
}
```

## detail

```
{"action": "detail", "measurement": "test1-1466663949", "instance": "1"}
```

```
{
  "files": [
    {
      "filename": "test1-001-2016.06.23T06.39.09.gz",
      "size": 219,
      "modified": "Thu Jun 23 06:39:31 UTC 2016",

```

```

        "mimetype": "application/x-gzip"
    },
    {
        "filename": "test1-001-
2016.06.23T06.39.09_load.csv",
        "size": 1,
        "modified": "Thu Jun 23 06:39:29 UTC 2016",
        "mimetype": "text/csv"
    },
    {
        "filename": "test1.dat",
        "size": 1,
        "modified": "Thu Jun 23 06:39:09 UTC 2016",
        "mimetype": "text/csv"
    },
    {
        "filename": "test1.profile",
        "size": 2,
        "modified": "Thu Jun 23 06:39:09 UTC 2016",
        "mimetype": "application/json"
    }
],
"results": [
    {
        "mimetype": "text/csv",
        "graphtype": "parsed",
        "size": 42693,
        "modified": "2016-08-11T14:25:03.922020",
        "filename": "test3.2016.06.08T09.13.17_parsed.csv"
    },
    {
        "mimetype": "image/png",
        "graphtype": "all",
        "size": 183564,
        "modified": "2016-08-11T14:25:06.217375",
        "filename": "test3.2016.06.08T09.13.17_all_avg.png"
    }
]
}

```

## delete

### Input JSON:

```
{ "action": "delete", "measurement": "test1-1466663949" }
```

### Output JSON:

```
{
  "test1-1466663949": {
    "status": "OK",
    "message": "Result test1-1466663949 was sucessfully deleted"
  }
}
```

## Module Servers

## Projects

### API příkazy

`ubus call flowtester task '{"action":"start","value":"test1"}'`

- *start, stop, delete, create, list, status*

`ubus call flowtester results '{"action":"list","value":"test1"}'`

- *list, detail, delete*

`ubus call flowtester servers '{"server":"flowping","action":"start"}'`

- *action: start, stop, enable, disable, status*
- *server: flowping, iperf*

### Správce systému

- Systémový soubor `/etc/config/flowtester`
  - V souboru jsou uloženy základní parametry nutné pro běh systému FlowTester a aplikací FlowPing a Iperf v režimu server
  - V rámci integrace s GUI se očekává možnost editace konfiguračního souboru
- Flowping/Iperf server
  - Pro obě aplikace v serverovém režimu by měla být možnost jejich ovládání přes webové rozhraní, tak jako je tomu u běžných služeb OS OpenWRT viz položka Startup.
  - Oba servery jsou ovládány přes ubus rozhraní
- Systémové prostředí
  - Vytažení konfigurace LAN, WAN, Switche, WiFi, systému (hostname, passwords, datum a čas etc.) do vlastního rozhraní
- Šablona vzhledu systému
  - Nalezení a aplikace vhodné šablony prostředí, které bude spjaté se systémem FlowTester a bude používán napříč celým systémem.

### Správce paternů

- Aplikace umožňuje definovat patterny pomocí JavaScriptového rozhraní, takovýto pattern je automaticky zobrazen
- Na zařízení je možné uložit obecně N paternů, ty je následně možné spravovat
  - Mazat
  - Editovat
- Vkládání paternů z externího souboru

### Správce měřících úloh

- Založení nové měřící úlohy, kde je možné nastavit
  - Pattern
    - Je požadována volba patternu provozu
    - Po zvolení je tento patern začleněn do JSON definujícím měření

- Cílová adresa
- Port
- Čas spuštění
  - Ruční spuštění
  - Načasované spuštění
- Správce úloh
  - Vypadá jako seznam, kde se indikuje, zda se jedná o naplánovanou úlohu nebo úlohu k ručnímu spuštění
  - V případě, že je úloha spuštěna, je zobrazen její aktuální stav
    - Měření x/N
    - Progresbar
    - Odhadovaný čas dokončení měření
    - Tlačítko pro zastavení měření
  - V případě, že se jedná o úlohu k ručnímu spuštění je zde tlačítko pro spuštění a následně zastavení měření
  - Možnost editovat nastavení měřící úlohy
    - Typicky vypnout načasování úlohy nebo naopak nastavení časování úlohy.

#### *Správce výsledků*

- Zobrazení hotových měření
  - Každý řádek výpisu bude obsahovat informace
    - Kdy měření probíhalo
    - Jména profilu, kterým výsledek vznikl
- Detailní pohled na měření
  - Zobrazeno po rozkliknutí odpovídajícího řádku v seznamu měření
  - Zobrazí se náhled na výsledky měření
    - Pomocí tabulek budou zobrazeny základní pohledy na data
    - S využitím obratu pro získání souborů v base64 budou zobrazeny závěry z vyhodnocení
    - Seznam souborů ke stažení

#### *Dashboard*

- Zobrazení průběhu probíhajících měření:
  - Aktuální data na In/Out rozhraních
  - Aktuální data stavu systému
  - Aktuální data/statistiky probíhajících testů
  - Monitoring diskového úložiště
- Činnost k dashboardu je nutné optimalizovat s ohledem na požadavek minimální zátěže systému.
- 

## Abstracts

Keywords: OpenWRT, Linux, Lua, LuCi, HTML, Shell, JSON

### 1. FlowTester system configuration

Familiarize yourself with LuCI a web user interface of OpenWRT. Customize graphical style of the web interface using some modern framework in order to be long-term sustainable and

responsive. Using basic widgets, implement basic functionality to control FlowPing, Iperf, FlowTester configuration files, and system properties like LAN, WAN, WiFi, etc.

Instructions:

- Project for 1 or 2 persons
- Duration: 1 month
- Required skills: HTML, Lua, JSON, Shell, CSS, JS
- Expected outputs: SW module, documentation

## 2. Measurement configurator

Familiarize yourself with LuCI a web user interface of OpenWRT. Use and extend LuCI in order to create a graphical pattern generator such that it enables easily manage and describe patterns used by the FlowTester. Additionally, these patterns can be used for definition of measurements which implementation is a part of this project as well.

Instructions:

- Project for 1 or 2 person
- Duration: 1 month
- Required skills: HTML, Lua, JSON, JS
- Expected outputs: SW module, documentation