OpenWrt LuCI Support for the LMAP Daemon

by

Lyubomir Tsvetkov

Bachelor Thesis in Computer Science

Submission: May 15, 2020 Supervisor: Prof. Jürgen Schönwälder

Jacobs University Bremen | Department of Computer Science and Electrical Engineering

English: Declaration of Authorship

I hereby declare that the thesis submitted was created and written solely by myself without any external support. Any sources, direct or indirect, are marked as such. I am aware of the fact that the contents of the thesis in digital form may be revised with regard to usage of unauthorized aid as well as whether the whole or parts of it may be identified as plagiarism. I do agree my work to be entered into a database for it to be compared with existing sources, where it will remain in order to enable further comparisons with future theses. This does not grant any rights of reproduction and usage, however.

This document was neither presented to any other examination board nor has it been published.

German: Erklärung der Autorenschaft (Urheberschaft)

Ich erkläre hiermit, dass die vorliegende Arbeit ohne fremde Hilfe ausschließlich von mir erstellt und geschrieben worden ist. Jedwede verwendeten Quellen, direkter oder indirekter Art, sind als solche kenntlich gemacht worden. Mir ist die Tatsache bewusst, dass der Inhalt der Thesis in digitaler Form geprüft werden kann im Hinblick darauf, ob es sich ganz oder in Teilen um ein Plagiat handelt. Ich bin damit einverstanden, dass meine Arbeit in einer Datenbank eingegeben werden kann, um mit bereits bestehenden Quellen verglichen zu werden und dort auch verbleibt, um mit zukünftigen Arbeiten verglichen werden zu können. Dies berechtigt jedoch nicht zur Verwendung oder Vervielfältigung.

Diese Arbeit wurde noch keiner anderen Prüfungsbehörde vorgelegt noch wurde sie bisher veröffentlicht.

May 15, 2020
Lyubomir Tsvetkov

[illegible]

Contents

1 Introduction **1**

2 Large-scale measurement system **1**
2.1 Motivation and Goals 1
2.2 Components of the LMAP Framework 2

3 LMAP Information Model **3**
3.1 Control Protocol 3
3.1.1 Pre-configuration Information 4
3.1.2 Configuration Information 4
3.1.3 Instruction Information 4
3.1.4 Capability, Failure and Logging Information 6
3.2 Report Protocol 7
3.3 Common Information Objects 7
3.4 Data Model and Configuration Example 8

4 OpenWrt **10**
4.1 UCI System 11
4.2 LuCI 11
4.3 Evolution of OpenWrt UIs 13
4.4 LuCI2 vs LuCI 14
4.5 Ubus 15
4.5.1 ACLs 15
4.5.2 Access and modify UCI files via Ubus 17

5 OpenWrt LuCI interface supporting the LMAP Daemon **19**
5.1 Former LuCI interface for the LMAP Daemon 19
5.1.1 Possible improvements 20
5.2 Current LuCI interface for the LMAP Daemon 21
5.2.1 Design decisions 21
5.2.2 Robustness 26
5.2.3 Implementation details 27
5.2.4 Security - ACL 32

6 Conclusion and future work **33**

7 Acknowledgements **34**

1 Introduction

As the Internet grows, the need for large-scale measurement systems increases. Parties who have a benefit from these measurement systems are Internet Service Providers, regulators and end users. Current solutions like SamKnows, RIPE Atlas, and Netalyzr [3] are of proprietary nature and one cannot compare measurement results obtained from the different platforms.
For this reason, the LMAP working group of the Internet Engineering Task Force (IETF) developed an information and data model for a standardized large-scale measurement system. The LMAP Daemon (LMAPD) is the proof-of-concept implementation for the LMAP information and data model.
OpenWrt is a Linux distribution for embedded devices, which provides a fully writable file system with package management. OpenWrt provides a command-line interface via SSH and a web-based user interface for configuration of services. The LMAP Daemon, run on the OpenWrt system can be configured through this command-line interface. However, enabling the option to configure the LMAP Daemon through the OpenWRT web user interface (UI) would make it more accessible to more users since it would be easier for less-experienced users to change LMAPD configurations. Therefore, the design of an OpenWrt user interface for LMAPD is the topic of research of this paper.
The rest of the paper is structured as follows. Section 2 provides an outline of an LMAP-Based Measurement System, Section 3 discusses the LMAP information and data model, Section 4 introduces the OpenWrt operating system and its user interface. Afterwards, Section 5 makes an overview of the former and current LuCI interface supporting LMAPD. Section 6 is the conclusion and Section 7 are the acknowledgements.

2 Large-scale measurement system

Building a large-scale measurement system implies building a large-scale set of measurement agents (Mas) which perform broadband measurements and collect measurement data to assess the performance of the Internet [6]. MAs are a piece of software that can be executed on specialized hardware like hardware probes or on general-purpose devices like mobile phones and PCs. [2] Current measurement platforms use proprietary protocols and metrics, which has negative impacts. Results obtained by different platforms cannot be compared, different platforms cannot use each other's probes to aggregate more data and also it is not possible to buy components from different vendors. The IETF LMAP working group aims to achieve a large-scale measurement platform, which uses standard protocols, standard metrics, standard architecture and information model. In the following sections I am outlining the LMAP Framework according to [6] [4] .

2.1 Motivation and Goals

There are two main ideas behind Internet measurement platforms - topology discovery and performance measurement. Performance measurement platforms can be further classified into fixed-line access measurements, mobile access measurements and operational support, measuring the performance of fixed-line networks, mobile access networks and the access network of network operators respectively. Topology discovery measurement platforms have been utilized to understand the macroscopic network-level topology of the Internet. Lately, a shift towards deployment of performance measurement platforms has been observed. Motivation for this is the need to assess the broadband quality and to verify service offers against contractual agreements, which helps regulate the broadband industry. There are multiple parties which can benefit from such a network performance measurement system, such as Internet-service providers (ISPs), regulators and end users. It can help ISPs plan their network, identify, isolate and fix problems in their access network and also evaluate the quality of service experienced by their users. Regulators can use the measurement results to compare multiple broadband provider offerings and support public policy development to regulate the broadband industry. Consumers would like to run diagnostic tests at their end to verify if the ISP is providing the service promised in the contract. Consumers can also use the measurement results to diagnose network problems in their home network [6] [3] .

The design goals of the LMAP Framework [6] are the following features:

- Standardized - The components, information models and communication protocols should be standardized so one can compare data measured at different times, in different places or by components from different vendors.
- Large-scale - Building a network of millions of Measurement Agents to collect more profound data and make more meaningful network diagnostics.
- Diversity - The measurement system should be able to support measurement agents produced by different vendors that are in wired or wireless networks, installed on devices with IPv4 or IPv6 addresses and have different capabilities in terms of measurement tasks they can perform.
- Privacy respecting - Sensitive information of all people involved in measurements should be respected.

2.2 Components of the LMAP Framework

The LMAP framework consists of three main components – Measurement Agents, Controllers and Collectors [4].

- Measurement agent (MA) - Software, which is installed on specialized hardware like a probe, or on general purpose devices, such as PCs and mobile phones. Its purpose is to execute measurements, which are called measurement tasks. It can observe existing traffic or generate traffic for the sake of the measurement. These are called passive and active network measurements respectively.
- Controller - They have the task to instruct MAs. They can instruct the MA what measurement task to perform and when, which is referred to as Measurement Schedule. Controllers can also instruct the MA which collector to report the measurement results to and when, which is referred to as Report Schedule. One MA can take instructions only from one Controller and Controllers can instruct a set of one or more MAs to perform a set of one or more Measurement Tasks at a given time.
- Collector - It receives the measurement results collected by the Measurement Agents and provides the results to a repository. MAs report the Measurement Results to one or more Collectors.

There are three protocols involved, Control Protocol for the communication between Controller and MA, Report Protocol for the communication between MA and Collector and several Measurement Protocols used between Measurement agents (MA) and Measurement Peers (MP). Measurement Peers collaborate in measurement tasks by playing the role of a responder, which means that they are not directly managed by the Controller. The Control Protocol delivers Instruction messages from the Controller to the MA and Capabilities, Logging and Failure information in the other direction. The Report Protocol delivers Reports to the Collector [6].

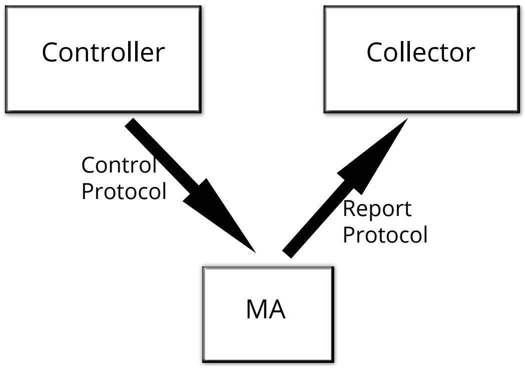


Figure 1: LMAP Framework architecture. MA receives instruction from the Controller what Network Measurement to perform and delivers the Measurement results to the Collector [3].

Measurement methods define how to measure a metric of interest. It is important to standardize measurement methods so results collected at different times and places can be compared. There are two types of measurement methods: active and passive measurements. In both cases the Measurement Agent measures the traffic flow. In the case of passive measurement, the MA observes an existing traffic flow and calculates for instance the average loss for a particular flow. In the case of active measurement, the MA generates traffic for the purpose of measurement. For instance, to measure the round-trip delay the MA is performing an active measurement by sending an ICMP ECHO request ("ping") to a given destination device (MP) on the Internet [6].

3 LMAP Information Model

The information model [4] is an abstract, protocol-neutral and independent of specific implementation description of the information stored, received and transmitted by a Measurement Agent as outlined in the LMAP Framework. The information described in the Information Model will be transmitted by protocols according to a data model. The following sections will outline multiple aspects of the Information Model including the Common Information Objects that support them and how they map to the Control and Report Protocol.

3.1 Control Protocol

The Control Protocol is used by the Controller to send instructions to the MA about what Measurement Tasks to do, when to do them and how to report the results. Also, the Controller can send the MA Configuration data. The MA, on the other hand, utilizes the Control Protocol to send to the Controller Capabilities, Logging and Failure Information.

3.1.1 Pre-configuration Information

The purpose of the Pre-configuration information is to integrate a Measurement Agent into a Measurement System so that it can successfully communicate with a Controller. As a result of this process, the MA learns from the Controller the identifier (MA-ID), Group-ID, which is shared by multiple MAs and the Control Channel, more specifically URL of the Controller from where Configuration Information will be sent and security information required for the communication. In case the MA pulls Information from the Controller, the Pre-configuration Information also needs to provide timing of the communication and what data will be transferred. Timing is represented as an Event that triggers a Schedule which performs a Task responsible for communication with the Controller. In case information is pushed to the MA by the Controller, a Schedule is provided which executes a Controller Listener Task when the MA is started.

3.1.2 Configuration Information

Its purpose is to assign an identifier MA-ID to the MA if one was not provided during Pre-configuration or update the MA-ID, also Group-ID. The Control Channel, hence Choice of Controller, timing of the communication with the Controller or parameters for the Communication Tasks can also be updated.

```
+-----------------+                                  +-------------+
|                 |                                  | Measurement |
|  Controller     |==================================|  Agent      |
+-----------------+                                  +-------------+

Configuration information:              ->
(MA-ID),
(Group-ID),
(Control Channel)
                                        <-        Response(details)
```

Figure 2: Configuration Information, transferred between Controller and Measurement Agent. Controller can update MA's MA-ID and Group-ID and Control Channel [6].

3.1.3 Instruction Information

The Instruction Information defines the Measurement tasks the MA should execute. More specifically the Instruction Information Model has four sub-elements. The purpose to define four sub-elements is to enable each part of the Information Model to change without affecting the other three, since Report Channels and Task Configurations stay relatively static and Schedules are more dynamic.

- Measurement Task configurations
  This element of the Instruction Information Model contains the metric to be measured and the specification of the measurement method, input parameters that the measurement method needs, like the address of the Measurement Peer that is collaborating in the measurement task for example, and interface to use if the device hosting the MA has multiple interfaces.

- Schedule Configuration
  It defines the actions to be executed, which extend configured Tasks with additional parameters, Events which trigger the execution of the Actions and specifications how to link output data from Tasks to other Schedules.

- Configuration of the Report Channel
  It contains address of the Collector and Security for this Report Channel.

- Suppression Information - goal is to enable the Measurement System to stop Measurement traffic in case of some unexpected network issue. One can specify a set of Measurement Tasks to suppress, a set of Measurement Schedules to suppress, a set of Reporting Schedules to suppress, start time at which Suppression begins and end time at which Suppression ends.

```
+------------------+                                    +-------------+
|                  |                                    | Measurement |
|  Controller      |====================================|  Agent      |
+------------------+                                    +-------------+

Instruction:                             ->
[(Measurement Task configuration
    URI of Metric(
   [Input Parameter],
   (role)
   (interface),
   (Cycle-ID)
   (measurement point)),
 (Report Channel),
 (Schedule),
 (Suppression information)]
                                         <-         Response(details)
```

Figure 3: Instruction Information transferred between Controller and Measurement Agent. Controller can specify configurations about the Measurement Tasks, Schedules to be executed by the MA, Suppression information about Tasks or Schedules which will not be executed by the MA. Moreover, information about the Report Channel, which is used to report the measurement results is also configured [6].

```
+-----------------+                             +-------------+
|                 |                             | Measurement |
|  Controller     |=============================|  Agent      |
+-----------------+                             +-------------+

Suppress:
[(Measurement Task),                 ->
 (Measurement Schedule),
 (start time),
 (end time),
 (on-going suppressed?)]

Un-suppress                          ->
```

Figure 4: Suppression Information, which is part of the Instruction Information and is transferred between Controller and MA. It specifies Tasks and Schedules that should not be executed by the MA, start and end time of the Suppression [6].

3.1.4 Capability, Failure and Logging Information

Not only the Controller can send Information to the MA, but Information also flows in the reverse direction.

- Capability and Status Information
  It contains the Measurement Method that the MA supports, the Measurement Protocol types the MA supports, interfaces that the MA has, version of the MA, version of the Hardware, Software or Firmware of the device with the MA etc.
- Failure Information - Information concerning why an MA has failed to execute a Measurement or Reporting Task. Possible reason for failure could be spare memory, no spare CPU cycles or a Collector is not responding.
- Logging Information
  Success/Failure Information in response to Information received from the Controller, Operational updates from the MA and Status updates from the MA.

```
+------------------+                                  +--------------+
|                  |                                  | Measurement  |
|  Controller      |==================================|  Agent       |
+------------------+                                  +--------------+

    (Request Capabilities),
    (Request Failure Information),
    (Request Logging Information),
    (Request Instruction)                ->
                                         <-           (Capabilities),
                                               (Failure Information),
                                               (Logging Information),
                                                       (Instruction)
```

Figure 5: Capability, Failure and Logging Information, sent by the MA to the Controller. It includes Measurement Methods and Protocol types the MA supports, interface that the MA has, Hardware and Software version of the device hosting the MA, any failure information on why a Measurement or Reporting Task failed to execute, as well as Operational and Status updates from the MA [6].

3.2 Report Protocol

Utilizing the Report Protocol, the Measurement Agent can send its Measurement Results to a Collector when instructed by the Controller. The Report contains the following information.

- MA-ID or Group-ID
- Measurement results and the time they were obtained
- Details about the Measurement Task

```
+------------------+                                  +--------------+
|                  |                                  | Measurement  |
|   Collector      |==================================|  Agent       |
+------------------+                                  +--------------+


                                  <-    Report:
                                               [MA-ID &/or Group-ID],
                                                [Measurement Result],
                                      [details of Measurement Task],
                                                          (Cycle-ID)
ACK                               ->
```

Figure 6: Report, sent by an MA to a Collector. The information transferred includes the MA-ID or Group-ID, Measurement Results and time they were obtained, details about the Measurement Task [6].

3.3 Common Information Objects

As stated above, the Common Information Objects support the various aspects of the LMAP Information Model. They are used within the information, transmitted between the different components of the LMAP Framework.

- Schedules - A Schedule comprises of a set of Actions, which lead to the execution of a Task. Schedules are Information Objects used within the Instruction to specify what tasks should be performed, when and how to report their results. They are also used within the Pre-configuration and Configuration Information to execute tasks required to communicate with the Controller.
- Channels - Channels are used to communicate with endpoints like Controllers and Collectors. A Channel Object contains the information required to communicate with a single endpoint, such as Endpoint location and security details.
- Task Configurations - Information used to configure the Tasks that will be run by the Measurement Agent, such as registry entry for the task and any parameters. Task Configurations are referenced from a Schedule such that the MA knows what tasks to perform.
- Events - An Event Object specifies a single or a series of Events that indicate when the Tasks within a Schedule should be executed. Event Objects are referenced by a Schedule and each Schedule can reference only one Event Object.

```
Schedule
        |-- triggered by --> Event
        |
        |-- executes --> Action 1
        |                  |-- using --> Task Configuration
        |                  |
        |                  `-- feeding to --> Destination Schedule
        :
        :
        `-- executes --> Action N
                           |-- using --> Task Configuration
                           |
                           `-- feeding to --> Destination Schedule
```

Figure 7: Schedule schema. A Schedule consists of a set of Actions which are executing Tasks. A Schedule references an Event object which determines when the Actions should be executed. The results from the Measurement Tasks are then fed to Destination Schedules [4].

As shown in Figure 7, a Schedule is executed by an MA. The Schedule references an Event which triggers the execution of the Actions specified within the Schedule, which are executing the tasks. The Actions refer to Task Configurations. The Results are then fed to a Destination Schedule. For instance, Actions implementing Measurement Tasks can feed the results into a Schedule that triggers Actions implementing Reporting Tasks. Actions and Task Configurations can have parameters, which are called Action Options and Task Options respectively. [4]

3.4 Data Model and Configuration Example

Below we can see an example XML Configuration of LMAPD, which complies with the Data Model of the LMAP Framework. LMAPD is the proof-of-concept daemon implemen-

tation for the Large-Scale Measurement Platforms (LMAP) information and data model developed by the LMAP working group of the Internet Engineering Task Force (IETF) [10]. This Configuration shows how to create a Measurement Schedule, which contains one Action with one Measurement Task, Action and Task options respectively, Destination Schedule and Event which triggers the execution of the Schedule.
At line 4, Configuration Information about the Measurement Agent like MA-ID and Group-ID is set. Then, Instruction Information on what Measurement the MA has to execute is specified. At line 12, a Schedule is created with name 'demo', which references an Event with name 'every-minute'. This event is defined at line 56, it is periodic and triggers the execution of the Schedule every 60 seconds. Execution mode of the Schedule is set to sequential, which means that if there are multiple Actions in this Schedule, they are executed in a consecutive manner. At line 16, an Action is specified, which has the name 'mtr' and refers to a Task with the same name. The action has two options which specify the host names that the 'mtr' task is run against. In this case, 'www.ietf.org' and 'www.ieee.org'. This Action also has a destination Schedule with name 'report-primary' where the results of the Measurement Task will be sent to. In this example the configuration of 'report-primary' schedule is omitted for brevity.
The task with name 'mtr' is defined at line 33. It refers to a program located in '/usr/bin/mtr'. This program is a network diagnostic tool [19]. It combines the functionality of 'ping' and 'traceroute' in a single tool. This tool investigates the network connection between the host 'mtr' runs on and another device in the network, by sending packets and noting the response percentage and response time of the routers on the way. An increase in packet loss or response time indicates a bad or overloaded link. In this configuration, there are multiple options specified for this Task, which will be passed to the 'mtr' tool when started.

```
1  <config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
2    <lmap xmlns="urn:ietf:params:xml:ns:yang:ietf-lmap-control">
3
4        <agent>
5            <agent-id>550e8400-e29b-41d4-a716-446655440000</agent-id>
6            <group-id>network measurement at the north-pole</group-id>
7            <report-agent-id>true</report-agent-id>
8            <report-group-id>false</report-group-id>
9        </agent>
10
11       <schedules>
12           <schedule>
13               <name>demo</name>
14               <start>every-minute</start>
15               <execution-mode>sequential</execution-mode>
16               <action>
17                   <name>mtr</name>
18                   <task>mtr</task>
19                   <option>
20                       <id>www.ietf.org</id>
21                       <value>www.ietf.org</value>
22                   </option>
23                   <option>
24                       <id>www.ieee.org</id>
```

```
                                <value>www.ieee.org</value>
                            </option>
                            <destination>report-primary</destination>
                    </action>
                </schedule>
            </schedules>

            <tasks>
                <task>
                    <name>mtr</name>
                    <program>/usr/bin/mtr</program>
                        <option>
                            <id>numeric</id>
                            <name>--no-dns</name>
                        </option>
                        <option>
                            <id>csv</id>
                            <name>--csv</name>
                        </option>
                        <option>
                            <id>lookup AS numbers</id>
                            <name>-z</name>
                        </option>
                        <option>
                            <id>one cycle</id>
                            <name>--report-cycles=3</name>
                        </option>
                </task>
            </tasks>

            <events>
                <event>
                    <name>every-minute</name>
                    <periodic>
                        <interval>60</interval>
                    </periodic>
                </event>
            </events>

        </lmap>
      </config>
```

4 OpenWrt

The OpenWrt project as described in [22] is a Linux distribution targeting embedded devices. It provides a fully writable file system with package management. This gives users the freedom to customize the device using packages to suit any application and not only the applications provided by the vendor. Developers can also build new applications without building a whole firmware around it. OpenWrt is preferred because it can be superior to the stock firmware of a router or embedded device. OpenWrt provides command-line interface via SSH and a web-based user interface for configuration.

4.1 UCI System

UCI is an abbreviation of Unified Configuration Interface and it is a system to centralize the configuration on OpenWrt devices [28]. Applications can be made UCI-compatible by transferring the settings chosen in the UCI file to the original configuration file which is read by the program. This transfer is done when running the initialization scripts in '/etc/init.d/'. As stated in [7], initialization scripts are run to start required processes as part of the boot process. The init scripts are called by the init process at boot time. When the initialization script is started, the program's original configuration file is overwritten with the settings from the corresponding UCI file. When a UCI configuration file is changed, the services which are affected must be reloaded by an init.d call so the changes can be applied. The example given at [28] is with Samba/CIFS, the original configuration file '/etc/samba/smb.conf' is overwritten with the UCI settings specified in the UCI configuration file '/etc/config/samba' when '/etc/init.d/samba start' is run. There is a way to disable UCI configuration and change the settings in the original configuration file. UCI files are located in the '/etc/config/' directory and can be edited via a text editor or with the command-line utility program 'uci'. There are also programming APIs for instance for Lua language, which is how the Luci web interface modifies the UCI files.

UCI configuration files [28] consist of multiple sections denoted by 'config' statements. Every section defines options which hold the actual configuration values. The following snippet shows an example UCI configuration:

```
package 'example'
config 'example' 'test'

        option 'string' 'some value'
        option 'boolean' '1'
        list 'collection' 'first item'
        list 'collection' 'second item'
```

Config 'example' 'test' defines the start of a section with type 'example' and name 'test'. The name can be omitted and in this case the section is anonymous. The option statements define the option name and value within the section. The lines starting with list define an option with multiple values. List statements that share the same name, in this case 'collection', are grouped into a single list of values preserving the order of the list statements in the UCI file. Options which are required but not defined may trigger an error, if they are not required and not defined in the UCI file, the default value is assumed.

4.2 LuCI

LuCI, as defined in [12] is a web user interface for embedded devices. It is an alternative to managing OpenWrt through SSH and the terminal. LuCI splits the interface into logical parts like Models, Views, uses object-oriented libraries and templating. LuCI provides an MVC (Model-View-Controller) Framework [9], which is what the UI is written on top of. As stated in [11], LuCI installation comes with a uHTTPd web server, configured to use with LuCI. There is a little configuration necessary as uHTTPd is configured with CGI to support the Lua interpreter. The standard document root is ’/www’ and making a request to device’s IP address + ’/www’ searches for a index.html page. The file ’/www/index.html’ redirects you to ’/cgi-bin/luci’, which is the default CGI gateway for LuCI.

According to [14], LuCI uses a dispatching tree that is built by executing the index function of every available Controller. To register a function on the dispatching tree, one uses the entry-function provided by luci.dispatcher:

```
entry(path, target, title=nil, order=nil)
```

- Path describes the position in the dispatching tree. For instance, a path of “foo”, “bar”, “baz” inserts the node at ’foo.bar.baz’ which is accessible by ’/cgi-bin/luci/foo/bar/baz’
- Target describes the action that will be taken when a user requests the node
- Title describes the title of the node which the user will see in his/her menu
- Order is a number which defines the order at which nodes at the same level will be shown in the user menu

In order to create a new module ‘lmapd’, a file called ’lmapd.lua’ is created in directory ’/usr/lib/lua/luci/controller/’, which for example has the following content:

```
module("luci.controller.lmapd", package.seeall)

function index()
    root_level_node = entry({"admin", "lmapd"}, firstchild(), "lmapd",
       60)
    root_level_node.dependent=false
    root_level_node.sysauth="root"
    root_level_node.sysauth_authenticator="htmlauth"

    overview = entry({"admin", "lmapd", "overview"}, template("lmapd/
       overview"), "Overview", 1)
end
```

The first line of the code snippet declares a new module with name ’lmapd’. At the first line of the index function a top-level tab is added to the LuCI menu. The new tab appears as ’lmapd’. There are attributes assigned to the table returned by the entry call, which define this tab as not dependent on a parent node, authorized user who can acceess this tab is the ’root’ user and the authorization method is ’htmlauth’. Then a sub-tab of the ’lmapd’ tab is added. It appears as ’Overview’ in the user’s menu and corresponds to the following route: ’/admin/lmapd/overview’. When requested, this route renders the ’lmapd/overview.htm’ template, which is located at ’/usr/lib/lua/luci/view/lmapd/overview.htm’.
Now if we click on ’Overview’ in the LuCI menu, we can see the ’overview.htm’ page.
The target parameter in the entry function can also be a function or a CBI model. Functions are called with call(“function_name”) and CBI models with cbi(“filepath”).

- Templates
  As described in [13], LuCI has a simple regex-based template processor which

parses HTML files to Lua functions. The simplest form of a template is an HTML file. The following template prints 'Hello World' as a header on the screen:

```
<%+header%>
<h1><%:Hello World%></h1>
<%+footer%>
```

However, one can also include Lua code, write variables and function values. The Lua code can add some logic to retrieve data from UCI configuration files for instance, and render this data as part of the HTML page at the end. As opposed to static HTML pages, templates are first processed by the template engine, so the logic inside them can be executed and at the end, a static HTML page is produced and returned to the user.

- CBI models
  According to [5], CBI (Configuration Bind Interface) models map UCI configuration files to HTML forms. They describe the structure of the UCI config files and the resulting HTML forms, which are evaluated by the CBI parser. CBI models return an object of type 'luci.cbi.Map'. Describing the structure of the configuration file in a CBI Model allows LuCI to generate, parse and validate XHTML forms, read and write to the UCI file.

4.3 Evolution of OpenWrt UIs

In this sub-section I will go over the different OpenWRT UI ideas, developed over the years.

- OUI
  OUI is a web user interface implemented in vue.js and element-ui. It is an open-source project and its first commit is on May 19, 2019. Its latest commit is on February 14, 2020. OUI is inspired by LuCI2 and uses json-rpc to communicate with OpenWRT subsystems. To access any kind of system data, one calls Ubus via json-rpc with the help of uhttpd-mod-ubus to provide HTTP based API [23].

- JUCI
  JUCI is a modern web interface for embedded devices running OpenWRT. It is also open-source and its first commit is on May 31, 2015. Its latest commit is on August 8, 2018. The JUCI web interface is implemented using HTML5, Bootstrap and Angular.js and uses web-sockets for communicating with a Lua based backend RPC server, which is running on the embedded device. JUCI offers full mobile support by reusing the same code for building native mobile applications. Latest version of JUCI is 2.16.05. Since version 2.16.03 the backend was moved to a specialized juci server and the dependency on Ubus tool was removed. The motivation for removing the Ubus dependency is that with Ubus everything is a service and even the simplest tasks requires rpc message passing, which comes with packing and unpacking messages, instead of making simple method calls. This brings an overhead, according to the JUCI creators [8].

- xLuCI2
  xLuCI2 is a JavaScript Webgui for embedded devices running OpenWRT. XLucI2 is based on the original LuCI2 UI, fixed bugs from the original one and added new features, such as dynamic skin, dynamic switching language, modularization. XLuCI2

is an open-source project and its latest commit is on July 16, 2019. The Front-end uses Bootstrap framework for styling the pages and the Front-end communicates with the Back-end through Ubus, as in the original LuCI2 [29].

- LuCI - OpenWrt 19.07.2 Release
  There is a new version of LuCI, part of the OpenWRT 19.07.2 release, presented on March 6, 2020. The new version integrates client-side rendering of views. This improves performance, since some of the work previously done on the server using Lua code is moved to the client browser in the form of JavaScript code. With this update, LuCI is coming closer to the idea of the experimental LuCI2 interface [16].

4.4 LuCI2 vs LuCI

LuCI, as already described, is a web user interface written in Lua. According to [18], it turned out that LuCI is very resource consuming and performs bad on devices with slow CPU and little amount of RAM. This is the reason for the development of LuCI2. LuCI2 is a new user interface which has a different architecture and is still experimental. It doesn't make use of the Lua programming language. Instead, it uses static HTML pages and JavaScript XMLHttpRequests [26] to get the required data. Filling the HTML pages with content is done on the client side, which means less load on the server hardware (MA device). To access system data, the Ubus tool is used with the help of uhttpd-mod-ubus to provide HTTP-based API.

- Menu
  LuCI2 has a two-level menu limited to entries the current user has rights to and ordered according to an index value. The following is a definition of a top-level entry in the LuCI2 menu with title ”Foo”:

  ```
  "foo": {
    "title": "Foo",
    "index": 12
  }
  ```

  Definition of a second-level entry in the LuCI2 menu with title ”Bar”:

  ```
  "foo/bar": {
    "title": "Bar",
    "acls": [ "baz", "qux" ],
    "view": "foo/bar",
    "index": 5
  }
  ```

  The files containing the definitions of the menu entries are located in '/usr/share/r-pcd/menu.d' directory. Second-level entries may be located in separate files. This allows easy addition of new entries without modifying existing files [18].

- Templates
  Every sub-page has a template located in '/www/luci2/template/' . Templates are simple HTML files and don't contain any reference to variables or lua logic, as opposed to LuCI [18].

- Views
  Every sub-page also needs to have a view located in '/www/luci2/view/'.
  Views [18] are JavaScript files extending 'L.ui.view' using a sub-page specific object and must provide an execute method which is called after a template is loaded:

  ```
  L.ui.view.extend({
    title: L.tr('Foo'),
    description: 'Bar',

    execute: function() {
        var deferred = $.Deferred();
        deferred.resolve();
        return deferred.promise();
    }
  });
  ```

4.5 Ubus

As described in [27], Ubus provides inter-process communication between daemons and applications in an OpenWrt system. The Ubus daemon provides an interface for other daemons to register themselves. Daemons register a set of paths under a specific namespace and every path has multiple procedures which can reply with messages. Ubus has a command line tool which enables interaction with the ubusd server, and all services registered on it. There is also an uhttpd plugin called uhttpd-mod-ubus which enables Ubus call using the HTTP protocol. This is what LuCI2 uses. Requests are sent to /ubus URL using POST method. The interface uses jsonrpc v2.0 . The RPC-JSON container format looks like in Figure 8.

```
{ "jsonrpc": "2.0",
  "id": <unique-id-to-identify-request>,
  "method": "call",
  "params": [
      <ubus_rpc_session>, <ubus_object>, <ubus_method>,
      { <ubus_arguments> }
  ]
}
```

Figure 8: RPC-JSON Request to the Ubus service format [27].

4.5.1 ACLs

When logged in using ssh, one has full access to Ubus. However, when using Ubus through HTTP, uhttpd first queries whether our call is allowed. rpcd [24] is used to implement the ACLs. ACLs (Access Control List) are configured in '/usr/share/rpcd/acl.d/*.json'. The names of the files are not important but the top-level keys inside them define roles. An ACL example of a user which has access only to specific Ubus modules is shown in Figure 9. This file will be placed in '/usr/share/rpcd/acl.d/lesssuperuser.json'.

```
{
  "lesssuperuser": {
      "description": "not quite as super user",
      "read": {
          "ubus": {
              "file": [ "*" ],
              "log": [ "*" ],
              "service": [ "*" ]
          }
      },
      "write": {
          "ubus": {
              "file": [ "*" ],
              "log": [ "*" ],
              "service": [ "*" ]
          },
      }
  }
}
```

Figure 9: Example of an ACL definition that only allows read and write access to some specific Ubus modules, rather than unrestricted access to everything. The '*' symbol means that the user has access to all procedures registered under the given path in the Ubus daemon. One can substitute this symbol with a comma-separated list of procedure names. In this case, the user will be able to access only the procedures, which are present in the list [27].

We also need an entry in '/etc/config/rpcd', which is shown in Figure 10. $p means to look in /etc/shadow file and root means to use the password for the root user from that file. The list read and write statements map ACL roles to user accounts. For instance, user 'blaer' has the rights of a 'lesssuperuser'.

Now we need to make a session.login Ubus call with our username and password to 'https://your.server.ip/ubus', which returns a JSON object containing the ubus_rpc_session key, session timeout and the ACLs for this user. This ubus_rpc_session key can now be used to make actual calls based on the ACLs and the available Ubus services using the JSON format specified in Figure 8.

```
config login
  option username 'root'
  option password '$p$root'
  list read '*'
  list write '*'

config login
  option username 'blaer'
  option password '$p$blaer'
  list read lesssuperuser
  list write lesssuperuser
```

Figure 10: Entry in rpcd, assigning users access levels, which are defined in the ACLs [27].

Figure 11 shows the Ubus communication as a sequence diagram. It includes logging in, obtaining a session and then actually making Ubus calls using the existing session.

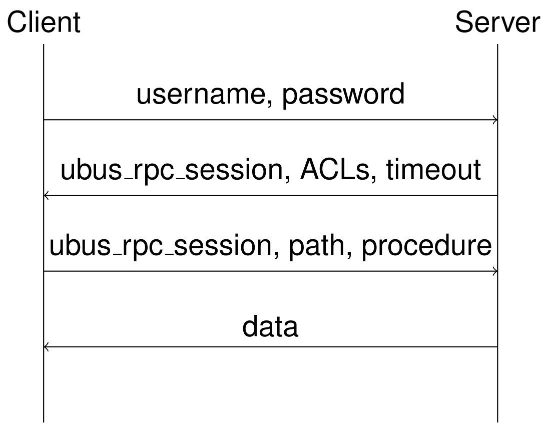

Client
Server
username, password
ubus_rpc_session, ACLs, timeout
ubus_rpc_session, path, procedure
data

Figure 11: Ubus communication sequence diagram. The user is first making a login Ubus call by providing username and password. The server is returning the session ID and the ACLs, which define which Ubus services the user can access. Afterwards, the user sends a request with path and procedure, together with the session ID in order to authenticate himself. Finally, the server returns the data requested by the user. The session timeout can be specified when the user logs in, otherwise it defaults to 5 minutes. It is automatically reset on every use [27].

4.5.2 Access and modify UCI files via Ubus

As already explained, some services have daemons, which are running all the time and can register a set of paths and methods using Ubus. UCI is an example of a command-line tool without any background process running all the time. That is why one makes use of rpcd, which is a tiny daemon registered on Ubus and has support for plugins. There is a built-in uci plugin for the rpcd daemon [24].
In the following snippet, one can see the JSON object, used to access UCI data via the

Ubus tool over HTTP:

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "call",
  "params":
  [
      "<ubus_rpc_session>",
      "uci",
      "get",
      {
          "config": "network",
          "section": "wan",
          "option": "ifname"
      }
  ]
}
```

To get a specific option of a named section, we specify a 'get' method, the config file and the section name we want to access. In this case we are retrieving option 'ifname' from section 'wan', which is in config file 'network'. We should first retrieve the ubus_rpc_session as explained in the previous section.

In the snippet shown below, one can see the JSON object, used to modify UCI data via the Ubus tool over HTTP:

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "call",
  "params":
  [
      "<ubus_rpc_session>",
      "uci",
      "set",
      {
          "config": "network",
          "name": "wan",
          "values": {
              "proto": "static",
              "ip6addr": "fdca:1234::1/64"
          }
      }
  ]
}
```

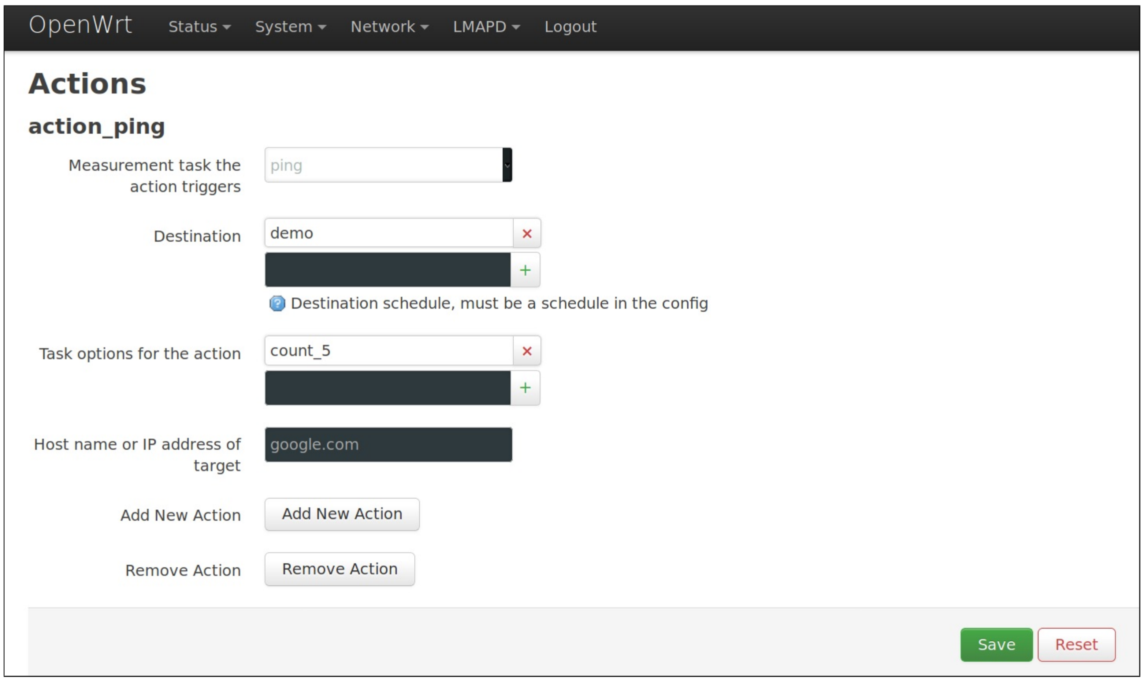
To set a value in a named section, we specify a 'set' method, the config file, the section name and the values we want to set. In this case we are setting option 'proto' to be equal to 'static', option 'ip6addr' to be equal to 'fdca:1234::1/64' in section 'wan', which is in config file 'network' [1].

5 OpenWrt LuCI interface supporting the LMAP Daemon

In this section I will make an overview of the different generations of OpenWrt UI, which support LMAPD configurations.

5.1 Former LuCI interface for the LMAP Daemon

This project was developed by Mohit Shrestha as a Bachelor Thesis at Jacobs University last year [25]. His version of the UI which supports LMAPD has a basic and an advanced menu [17]. In the advanced menu, one can add, delete, edit Schedules, Actions, Events and Task Options. The advanced mode assumes that the user is familiar with LMAP and more specifically the Common Information Objects to be able to correctly configure the LMAP Daemon. In Figure 12 one can see the screen for adding an action.



Actions

action_ping

Measurement task the action triggers: ping

Destination: demo

Destination schedule, must be a schedule in the config

Task options for the action: count_5

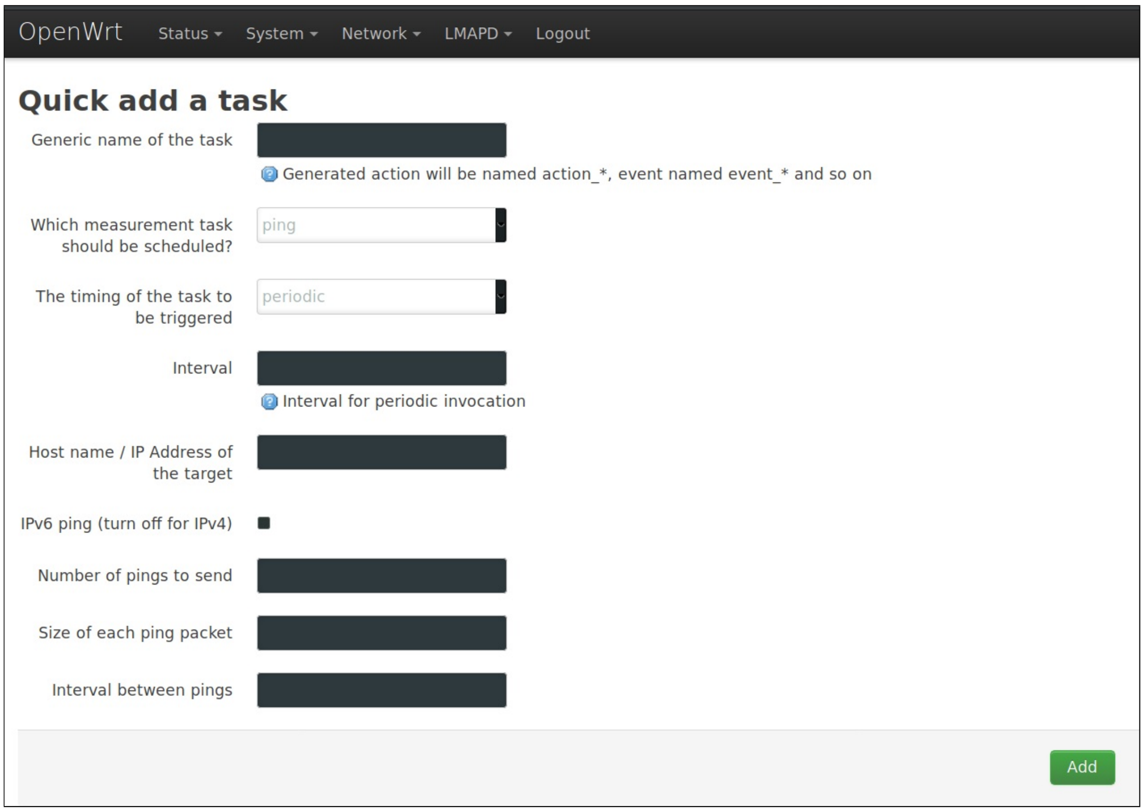
Host name or IP address of target: google.com

Add New Action

Remove Action

Figure 12: LMAPD UI screen for adding an action to the config file [17].

In Figure 13, one can see the Quick Add option, which is part of the basic menu. Via Quick Add, one specifies the information needed to execute a Measurement or Report Task and this information is automatically transformed in the form of a Schedule, Event, Action and Task Options and added to the config file. This saves the user the need to understand the Common Information Objects and the relations between them. Besides, in the Quick Add option, the available tasks are given in the form of a drop-down list, as well as the types of timing for the task, so the risk of a user mistake is minimized. In Quick Remove option, one has a drop-down list of the tasks added with the Quick Add functionality. By selecting a quick task and pressing the remove button, the Schedule, Event, Action and Task Options associated with this quick task are removed from the config file.



Quick add a task

Generic name of the task	
	Generated action will be named action_*, event named event_* and so on
Which measurement task should be scheduled?	ping
The timing of the task to be triggered	periodic
Interval	
	Interval for periodic invocation
Host name / IP Address of the target	
IPv6 ping (turn off for IPv4)	
Number of pings to send	
Size of each ping packet	
Interval between pings	

Figure 13: LMAPD UI screen for quick adding a task, which adds a schedule, action, task options and event to the config file automatically [17].

There is also an Overview page and a Results page. On the Overview page, one can see status information about LMAPD and on the Results page, one can see the results from the Measurement Tasks. The Results page has a drop-down list with Actions and the user can choose which Action to view results for. After the user selects an Action, another drop-down list is filled with the results for this Action. The user can then select a Result and see the Measurement Result data, as well as meta information about the task.
To represent LMAP config in UCI format, we need to do a conversion from either JSON or XML to UCI, since LMAPD accepts only JSON or XML configurations. LuCI has a json-c API for Lua language, which is used to convert from and to JSON format. Since JSON files have keys with hyphens and UCI files do not allow hyphens, when converting from JSON to UCI format the hyphens are substituted with underscores and vice versa.

5.1.1 Possible improvements

In my observations, the Quick Add can be made even simpler for users, since it is intended for people who are not familiar with the LMAP Daemon. For instance, the task options fields that expect a number can be made with an input number field, which has an up and down arrow for increasing and decreasing the number respectively. User will not be allowed to write in the field to avoid mistakes.

I also observed that for the interval field there is no specification on what data should be inserted, which causes confusion. By attempting to insert a string in that field I triggered a server error, which means that there is no front-end validation for that field if it is a number or not.
The Results page as I already explained has a drop-down list to choose an Action to view results for. Selecting an Action fills the second drop-down list with the available Measurement Results. However, in my opinion, the second filed should show up once an Action from the first List is chosen. The reason is that there is a confusion when both fields are there and their Choose buttons respectively. The user has the possibility to press the Choose button of the empty second field before choosing an Action from the first field. Moreover, the Choose buttons can be completely removed and just selecting a value from the list could trigger the next event.
These are some of the observations I have made. Another improvement could be to use more JavaScript to render the pages on the front-end instead of using Lua to render them on the server. This is a trend in OpenWRT UIs as we already observed. Not only LuCI2 UI but also the newest release of the LuCI UI are going in that direction.

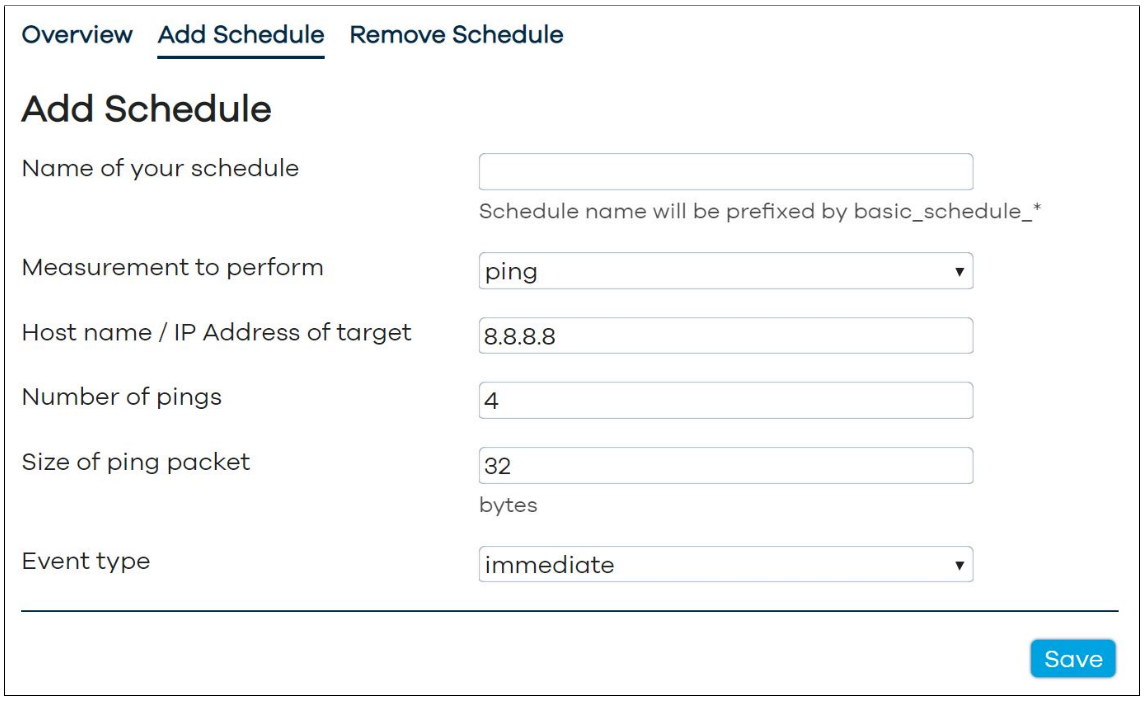
5.2 Current LuCI interface for the LMAP Daemon

Based on the former LuCI interface of the LMAP Daemon described in Section 5.1 and the possible improvements I outlined in Sub-section 5.1.1, I came up with a new interface, which I will describe thoroughly in the following sub-sections.

5.2.1 Design decisions

The goal of the UI is to make it easier for a novice user to configure LMAPD using a graphical interface. As already explained, an expert user can connect via SSH to the server and change the entries in the LMAPD configuration file directly. However, a novice user is not familiar with the structure of the LMAPD configuration file and all the details should be abstracted away.
I added 'LMAPD' tab to the main menu of the OpenWrt user interface. It has three sub-tabs: 'Add Schedule', 'Remove Schedule' and 'Overview'.

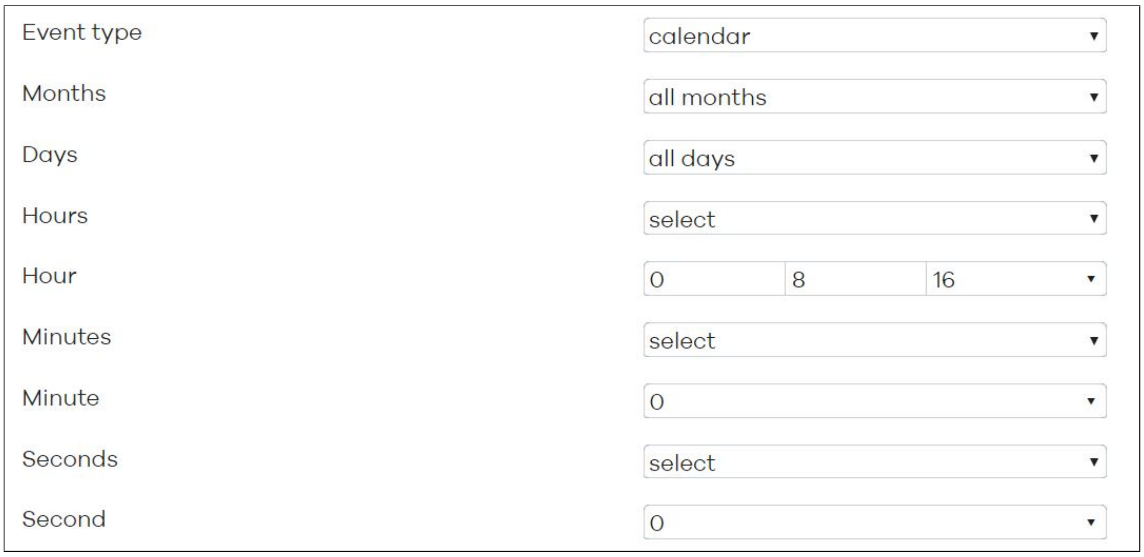
On the 'Add Schedule' page, the user can specify information about the name of his/her Schedule, what Measurement Task to execute, what command-line options to pass to the Measurement Task, IP address of target and when to execute the measurement. In the background all the information is split into Schedule, Action, Event and Task Options objects, so that LMAPD can understand the new configuration.
The Measurement Tasks currently supported by the interface are 'ping' and 'traceroute'. For 'ping', the user can specify the number of pings to send and the size of the ping packet. For 'traceroute', one can specify the hop limit and check the option to prevent resolving IP addresses to host names.



Add Schedule

Name of your schedule	Schedule name will be prefixed by basic_schedule_*
Measurement to perform	ping
Host name / IP Address of target	8.8.8.8
Number of pings	4
Size of ping packet	32 bytes
Event type	immediate

Figure 14: LMAPD UI screen for adding a Schedule, which adds a Schedule, Action, Task Options and Event to the configuration file automatically.

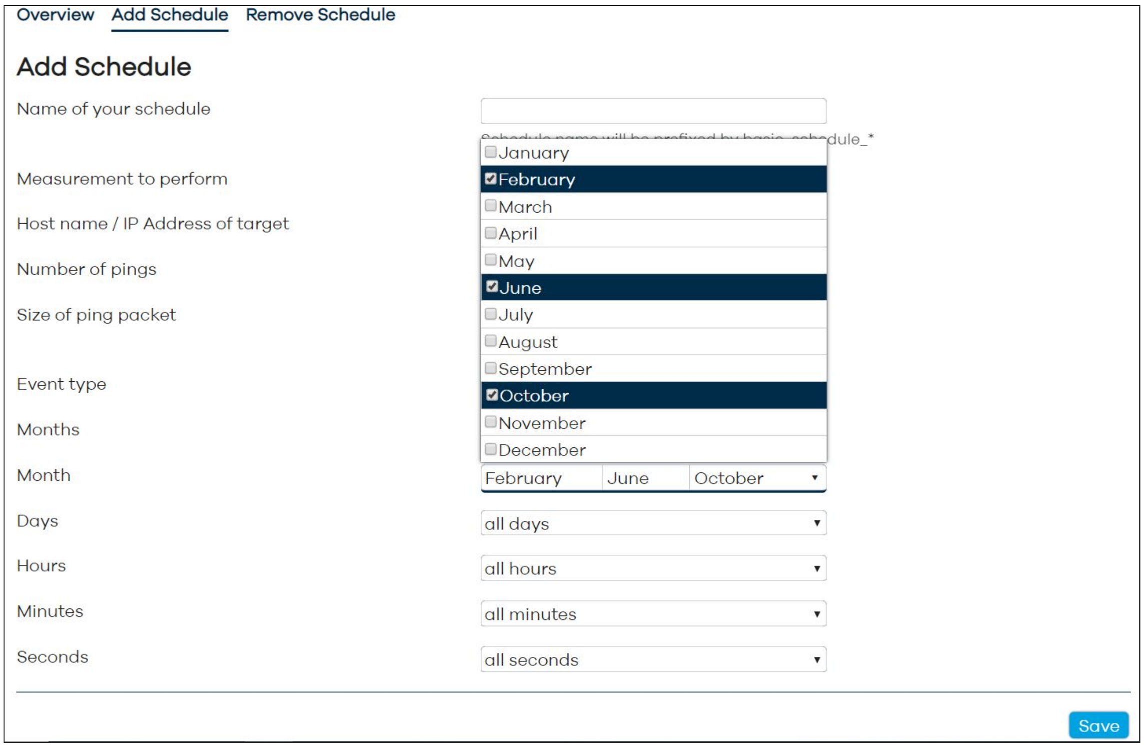


Event type	calendar
Months	all months
Days	all days
Hours	select
Hour	0 8 16
Minutes	select
Minute	0
Seconds	select
Second	0

Figure 15: Calendar Event setting which will trigger the execution of the Schedule at 0, 8 and 16 o'clock sharp every month and every day of the month.

There are three types of Events, which determine when the Schedule will be executed: immediate, periodic and calendar. They correspond to the LMAP data model respectively. Immediate Events trigger the schedule to be executed once at the moment of creation. Periodic Events have a interval field, where the user can set the number of seconds between each invocation of the Schedule. Calendar Events have multiple fields where the user can select in which month, day, hour, minute and second the Schedule should

be executed.
To make it easier for the user and also to give him/her a hint of what type of value is expected, I put some default values for IP address of target, Measurement Task options and options related to the Event type. I also put drop-down lists in the fields where that is applicable to reduce the work of the user and also limit him/her within a group of valid values to lower the chances of a mistake. For instance, the Measurement Task is selected from a drop-down list, populated with the available Tasks from the configuration file. The Event type is also selected from a pre-populated list, as well as the fields required for the calendar Event, as shown in Figure 16.


Overview | Add Schedule | Remove Schedule

Add Schedule

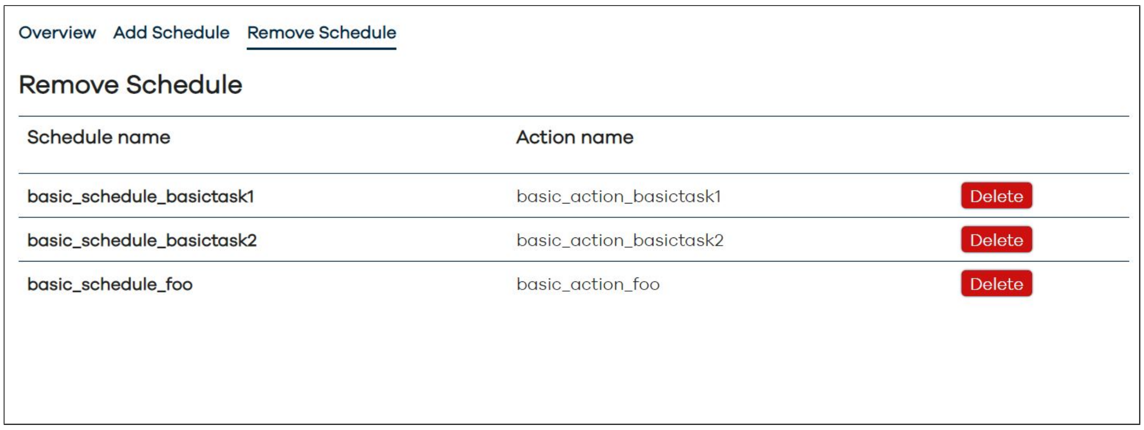
Name of your schedule	
	Schedule name will be prefixed by basic-schedule_*
Measurement to perform	
Host name / IP Address of target	
Number of pings	
Size of ping packet	
Event type	
Months	
Month	February, June, October
Days	all days
Hours	all hours
Minutes	all minutes
Seconds	all seconds

- ☐ January
- ☑ February
- ☐ March
- ☐ April
- ☐ May
- ☑ June
- ☐ July
- ☐ August
- ☐ September
- ☑ October
- ☐ November
- ☐ December

Save

Figure 16: Drop-down list for selecting an exact month or multiple months, in which the Schedule will be executed.

On the 'Remove Schedule' page, all the Schedules are listed with their respective Action and there is a 'delete' button, which will delete the Schedule, Action, Event and Task Options associated with this Schedule from the LMAPD configuration file and also delete the result files associated with this Schedule.

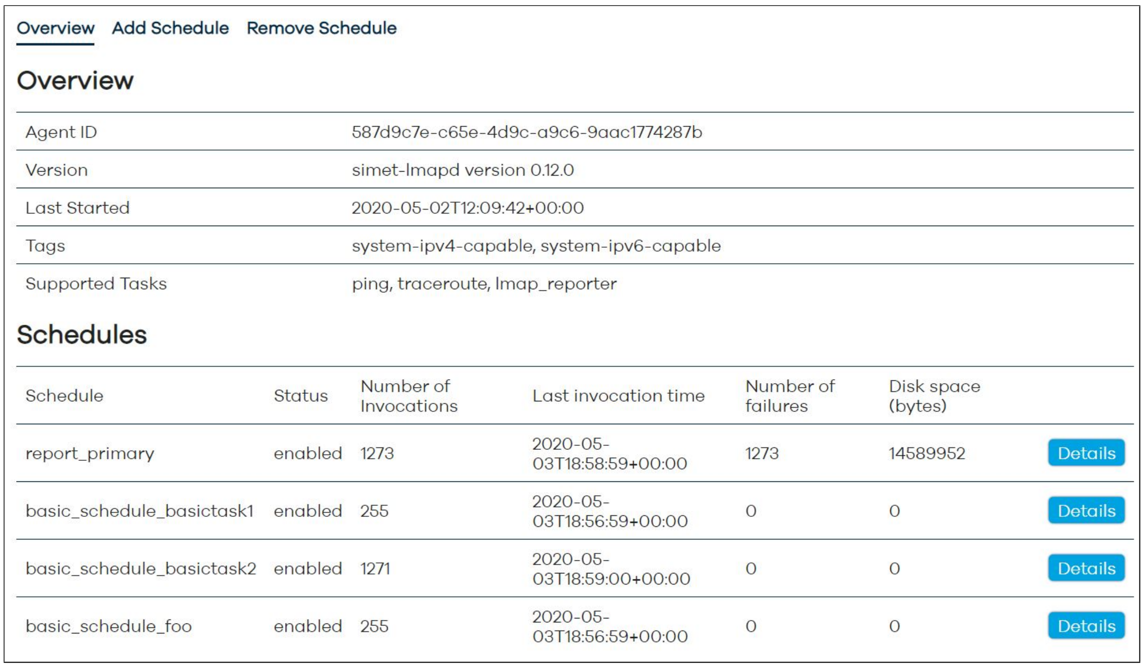


Remove Schedule

Schedule name	Action name	
basic_schedule_basictask1	basic_action_basictask1	Delete
basic_schedule_basictask2	basic_action_basictask2	Delete
basic_schedule_foo	basic_action_foo	Delete

Figure 17: Remove Schedule page where one can see all existing Schedules with their respective Actions and an option to delete them

On the 'Overview' page, the user can see status information about the Measurement Agent and information about the existing Schedules.



Overview

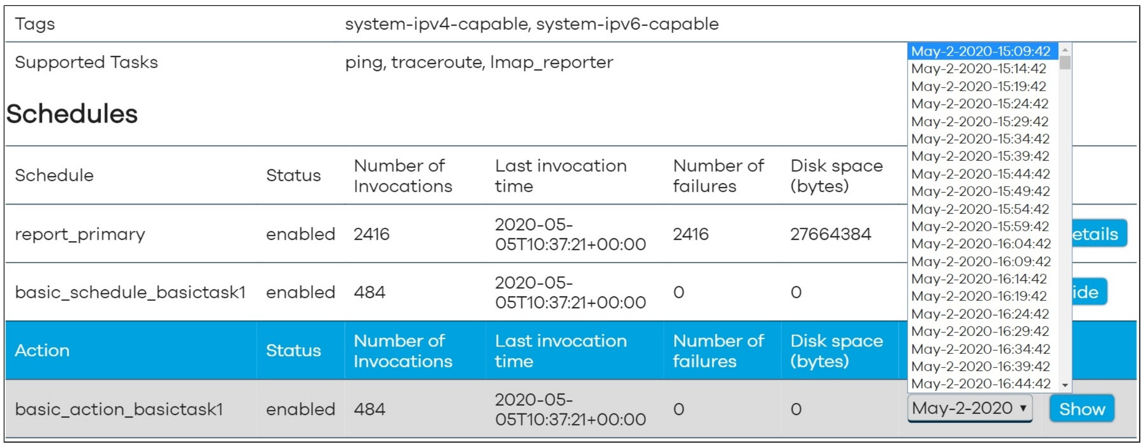
Agent ID	587d9c7e-c65e-4d9c-a9c6-9aac1774287b
Version	simet-lmapd version 0.12.0
Last Started	2020-05-02T12:09:42+00:00
Tags	system-ipv4-capable, system-ipv6-capable
Supported Tasks	ping, traceroute, lmap_reporter

Schedules

Schedule	Status	Number of Invocations	Last invocation time	Number of failures	Disk space (bytes)	
report_primary	enabled	1273	2020-05-03T18:58:59+00:00	1273	14589952	Details
basic_schedule_basictask1	enabled	255	2020-05-03T18:56:59+00:00	0	0	Details
basic_schedule_basictask2	enabled	1271	2020-05-03T18:59:00+00:00	0	0	Details
basic_schedule_foo	enabled	255	2020-05-03T18:56:59+00:00	0	0	Details

Figure 18: Overview page, where the user can see general information about the Measurement Agent and information about the existing Schedules.

The different Schedules are listed in the form of a table with a 'details' button, which adds a new line to the table with more detailed information about the Action this particular Schedule performs and the results produced by this Action. The results are distinguished by the date and time at which they were obtained and are displayed in the form of a drop-down list. Selecting a result to view and clicking on the 'show' button opens a modal at the center of the screen where the result data and meta data are presented.



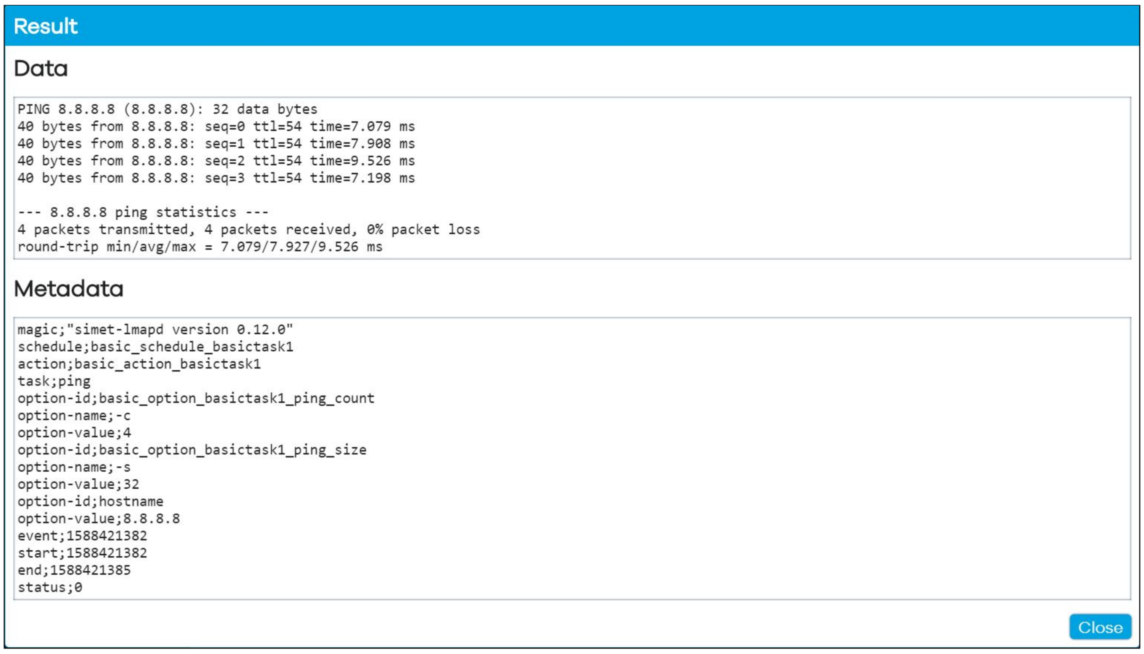
Tags	system-ipv4-capable, system-ipv6-capable
Supported Tasks	ping, traceroute, lmap_reporter

Schedules

Schedule	Status	Number of Invocations	Last invocation time	Number of failures	Disk space (bytes)	
report_primary	enabled	2416	2020-05-05T10:37:21+00:00	2416	27664384	etails
basic_schedule_basictask1	enabled	484	2020-05-05T10:37:21+00:00	0	0	ide
Action	**Status**	**Number of Invocations**	**Last invocation time**	**Number of failures**	**Disk space (bytes)**	
basic_action_basictask1	enabled	484	2020-05-05T10:37:21+00:00	0	0	May-2-2020 ▾ Show

```
May-2-2020-15:09:42
May-2-2020-15:14:42
May-2-2020-15:19:42
May-2-2020-15:24:42
May-2-2020-15:29:42
May-2-2020-15:34:42
May-2-2020-15:39:42
May-2-2020-15:44:42
May-2-2020-15:49:42
May-2-2020-15:54:42
May-2-2020-15:59:42
May-2-2020-16:04:42
May-2-2020-16:09:42
May-2-2020-16:14:42
May-2-2020-16:19:42
May-2-2020-16:24:42
May-2-2020-16:29:42
May-2-2020-16:34:42
May-2-2020-16:39:42
May-2-2020-16:44:42
```

Figure 19: Overview page, where a user can select to view a result for a particular Schedule from a list of results, distinguished by date and time of creation.



Result

Data

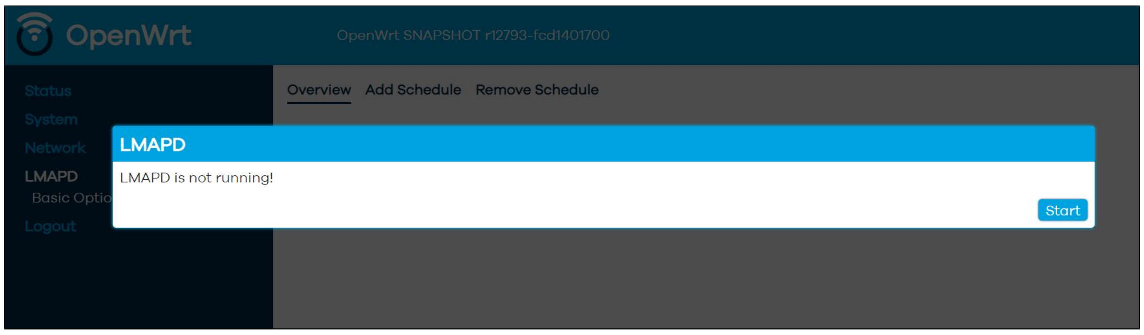
```
PING 8.8.8.8 (8.8.8.8): 32 data bytes
40 bytes from 8.8.8.8: seq=0 ttl=54 time=7.079 ms
40 bytes from 8.8.8.8: seq=1 ttl=54 time=7.908 ms
40 bytes from 8.8.8.8: seq=2 ttl=54 time=9.526 ms
40 bytes from 8.8.8.8: seq=3 ttl=54 time=7.198 ms

--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 7.079/7.927/9.526 ms
```

Metadata

```
magic;"simet-lmapd version 0.12.0"
schedule;basic_schedule_basictask1
action;basic_action_basictask1
task;ping
option-id;basic_option_basictask1_ping_count
option-name;-c
option-value;4
option-id;basic_option_basictask1_ping_size
option-name;-s
option-value;32
option-id;hostname
option-value;8.8.8.8
event;1588421382
start;1588421382
end;1588421385
status;0
```

Figure 20: Modal to display result data and metadata, shown in the center of the screen.

When LMAPD is not running, the interface prevents any interaction with the daemon. A modal is shown on the screen every time a user tries to access one of the LMAPD tab pages and LMAPD is shut down. The modal diplays a warning that LMAPD is not running and there is a 'start' button at the bottom right, which launches the LMAP Daemon and grants access to the page initially requested by the user.



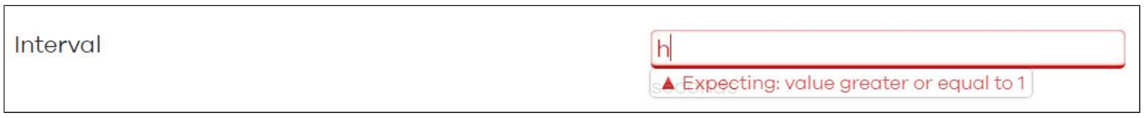
LMAPD is not running!

Figure 21: Modal, shown in the center of the screen when LMAPD is not running. It is shown on ’Overview’, ’Add Schedule’ and ’Remove Schedule’ screens and blocks any interaction with the page before the start button is clicked and LMAPD is launched.

5.2.2 Robustness

Besides the drop-down lists where possible, there are other means to prevent a user mistake and avoid a wrong configuration file.

- In the fields where the user should enter a value, there is a data-type check if the provided value corresponds to the required type. When the provided value is wrong, the field is colored red and a message is displayed stating what type of value is expected.

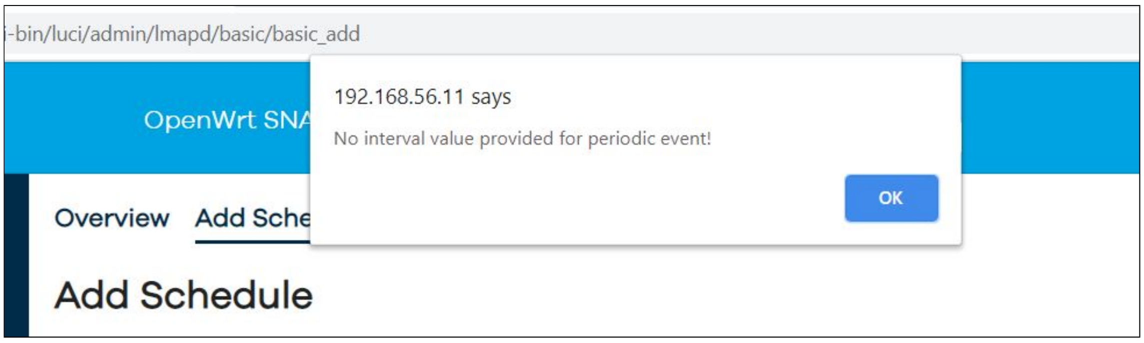


Interval: h

Expecting: value greater or equal to 1

Figure 22: Data-type validation in the interval field for periodic Events on the ’Add Schedule’ page.

- When an empty value is provided, an alert message is displayed on the screen, stating the name of the field, which needs to be filled before the new Schedule is added.

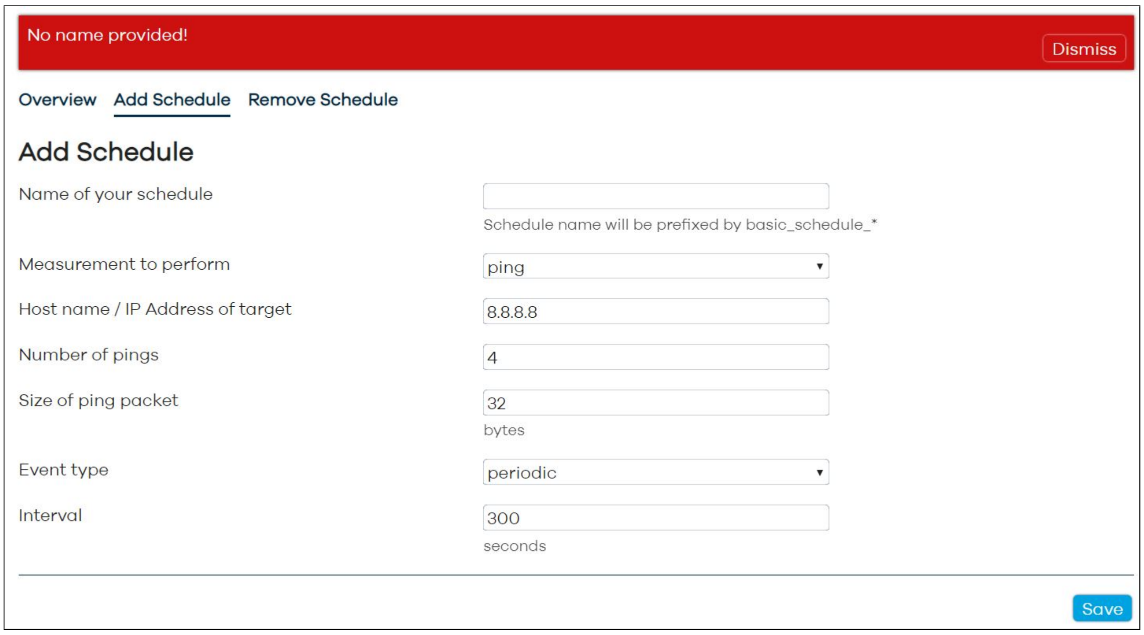


192.168.56.11 says

No interval value provided for periodic event!

Figure 23: Alert message displayed when the user is trying to add a Schedule but left a required field blank.

- In the case when the Schedule name field is left blank, a warning notification, colored in red, is shown at the top of the screen. This warning is made so obvious because this is the most important field of the form and the only field without a default value.


No name provided!

Dismiss

Overview Add Schedule Remove Schedule

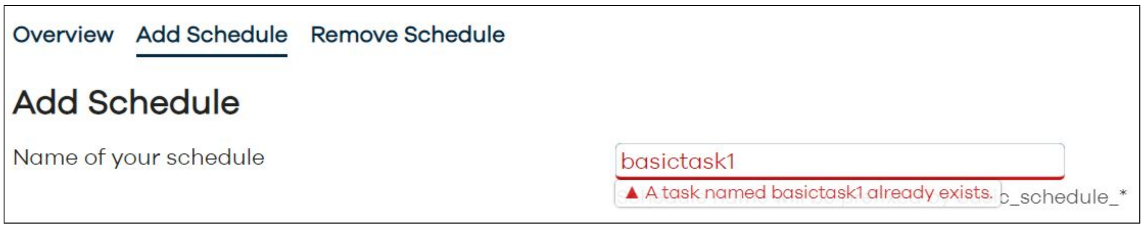
Add Schedule

Name of your schedule	Schedule name will be prefixed by basic_schedule_*
Measurement to perform	ping
Host name / IP Address of target	8.8.8.8
Number of pings	4
Size of ping packet	32 bytes
Event type	periodic
Interval	300 seconds

Save

Figure 24: 'No Schedule name' error message, displayed as a notification when the user tries to save a new Schedule without providing a name for it.

- In the case when the Schedule already exists in the configuration file, the interface makes a check and warns the user to change the name by coloring the field red and displaying a message, stating that the name already exists.


Overview Add Schedule Remove Schedule

Add Schedule

Name of your schedule: basictask1

▲ A task named basictask1 already exists. c_schedule_*

Figure 25: Check if the Schedule name already exists in the UCI configuration file and if that is the case, a warning is displayed.

5.2.3 Implementation details

For the purpose of this project I am using LuCI master branch, which provides menu.d support. The 19.07 LuCI branch also supports client-side rendering. However, it still relies on Lua controllers instead of declarative JSON menu descriptions.
In order to use the LuCI master branch, I had to create a custom OpenWrt Build by following a set of procedures [20]. First, I cloned the OpenWrt master branch and in the feeds.conf.default file I changed the src-git luCI feed [21] to point to the LuCI master branch: https://github.com/openwrt/luci.git. Then, I ran the following commands: ./scripts/feeds update -a and ./scripts/feeds install -a with option 'a', which means 'all'. After that,

running 'make menuconfig' opens a UI menu where I selected the target system to be x86 64-bit, the compilation result to be a vdi image and the LuCI theme to be the newest up to this point: luci-theme-openwrt-2020. The final step is to run the make file, which produces a OpenWrt vdi image including the newest LuCI features.

The newest LuCI version resembles very much to the LuCI2 framework, which was discussed in the previous sections. It uses Ubus calls to retrieve and set data and also to execute Lua scripts on the server where necessary. The rest of the logic is in the Javascript code, executed on the browser, where the developer also builds the HTML page, instead of retrieving a ready-made HTML file from the server. The LuCI client-side Javascript API provides wrapper classes and methods, which make the development way more user-friendly [15].

LuCI master branch uses declarative JSON menu descriptions, split between multiple files, which are located in '/usr/share/luci/menu.d/'. In order to add LMAPD as a new tab to the menu, I created a JSON file in the given location with the name 'luci-app-lmapd.json'. The following is a fragment from the file:

```
{
    "admin/lmapd": {
        "title": "LMAPD",
        "order": 60
    },
    "admin/lmapd/basic": {
        "title": "Basic Options",
        "order": 20,
        "action": {
            "type": "firstchild"
        }
    },
    "admin/lmapd/basic/overview": {
        "title": "Overview",
        "order": 10,
        "action": {
            "type": "view",
            "path": "lmapd/overview"
        }
    }
}
```

As shown in the code snippet, we have a JSON object, which contains multiple JSON objects inside it. Every object represents a tab in the menu and the name of the object represents the path to this particular tab. This means that the 'LMAPD' tab is shown in the main menu, 'Basic Options' is displayed under it when 'LMAPD' is clicked. The 'Overview' tab becomes visible when 'Basic Options' is clicked. The action defines what should be displayed on the screen. For instance, the action type 'firstchild' defines that the view of the first child tab should be displayed. In this case, the 'Overview' page. The action type 'view' defines a path to the view, which will be displayed. Views are Javascript files, located at '/www/luci-static/resources/view/'.

In order to see something displayed on the screen when the view is requested, we have to extend the 'L.view class' and return a new sub-classed Class instance. In our sub-class we need to override the render method to define what will be rendered on the screen. In this code snippet, we are creating a page with an 'Overview' header:

```
return L.view.extend({
    title: "Overview",
    render: function() {
        return E('h2', [ 'Overview' ]);
    }
});
```

As you already saw in the previous sub-section, HTML pages are not retrieved from the server, but built on the browser with the help of E() invocations. E() invocations can also be nested, as we can see on the example below. In case there are multiple top-level elements, one can also construct a document fragment without an extraneous wrapper element.
In the following example we observe how we can build the first row with column titles from the table on the 'Overview' page:

```
E('div', { 'class: 'tr' }, [
    E('div', { 'class: 'td' }, [ 'Schedule' ]),
    E('div', { 'class: 'td' }, [ 'Status' ]),
    E('div', { 'class: 'td' }, [ 'Number of Invocations' ]),
    E('div', { 'class: 'td' }, [ 'Last invocation time' ]),
    E('div', { 'class: 'td' }, [ 'Number of failures' ]),
    E('div', { 'class: 'td' }, [ 'Disk space (bytes)' ])
]);
```

- The 'Map' class is a member of LuCI.form. To draw an analogy between the former LuCI and the new one, the 'Map' class resembles to the 'CBIModel' used by the former LuCI. It represents a form, which maps one UCI configuration file and is divided into multiple sections containing multiple fields each.

- The 'JSONMap' class functions similar to 'LuCI.form.Map' but uses a multidimensional JavaScript object instead of UCI configuration as data source. The data entered into the fields by the user is then saved into the JSON object and then it can be examined and transformed according to one's needs.

I made use of the 'JSONMap' class to implement the adding of Schedules because I am not doing a direct mapping to UCI sections but first transforming the user's data into the right format and then saving it in the UCI file. This is done in order to abstract the user away from the actual UCI sections.

The following example is a fragment from the 'Add Schedule' page, showing how to create a simple form using the 'JSONMap' class:

```
let formData = {
    basic_add: {
        host_address: null
    }
};

return L.view.extend({
    render: function() {
        var m, s, o;

        m = new form.JSONMap(formData, 'Add Schedule');
        s = m.section(form.NamedSection, 'basic_add');
        o = s.option(form.Value, 'host_address', 'IP Address of the
            target');
        o.datatype = 'ipaddr';
        o.default = '8.8.8.8';

        return m.render();
    }
});
```

We can see on the example that a 'JSONMap' instance is created using 'new' with the object 'formData' as data storage. One section named 'basic_add' with one option named 'host_address' is added to the form. The option is an object, which has a property 'datatype' where one can set the datatype validation and a property 'default' where the default value displayed in the form can be set. At the end, the 'render' method is invoked on the instance to assemble the HTML markup and insert it into the DOM.

In order to access data from the server, we are using a wrapping library for Ubus calls, more specifically, LuCI.fs to access the file system and LuCI.uci to access the UCI configuration files.

- LuCI.fs
  It provides high level utilities to wrap file system related RPC calls [15].
  - The 'exec_direct' method is used to execute a specified command on the server. I am using it to execute scripts on the server in the cases where that makes sense. For instance, I am executing shell-scripts to start the LMAP Daemon and to invoke the 'lmapctl status' command. I am also executing Lua scripts to convert the UCI configuration file to a JSON configuration format and also to fetch the paths to the result files of a given Action, or to delete the results files of a given Action. The Lua scripts are located at '/usr/lib/lua/lmapd'. I will give an example with calling the command to start the LMAP Daemon:

    ```
    let p = fs.exec_direct('/etc/init.d/lmapd', [ 'start' ]);
    ```

    The result is a promise resolving with the command stdout output interpreted according to the specified type or rejecting with an error stating the failure reason [15]. We can see in the example how the 'lmapd' script is executed with a 'start' command-line argument.
  - The 'read_direct' method reads the contents of the given file and returns them. I am using it to read the results files of a given Action from the server or to read the 'lmapd-state' file.

This is an example of how the 'lmapd-state' file is read:

```
let p = fs.exec_direct('/usr/share/lmapd/lmapctl.sh', [ 'status'
    ]).then(function(result) { return fs.read_direct('/var/run/
    lmapd-state.json', 'json');
});
```

The result is a promise resolving with the file contents interpreted according to the specified type or rejecting with an error stating the failure reason [15]. In this example we can see how the 'lmapctl' script is executed with a 'status' command-line argument and then the resulting 'lmapd-state.json' file is read and parsed in JSON format.

- LuCI.uci
  The LuCI.uci class provides methods for making remote UCI Ubus calls and change the configuration files. I am mainly using the 'get', 'set' and 'add' methods.

  - The 'get' method has the following declaration: get(config, sid, option). 'config' is the name of the configuration file, 'sid' is the name of the section and 'option' is the name of the option to retrieve. The result is the value of the given option within the specified section of the given configuration. The option can be ommited, in which case the entire section is returned as an object.
    Here is an example of how to retrieve the execution_mode of a specific Schedule:

    ```
    uci.get('lmapd', 'basic_schedule_foo', 'execution_mode');
    ```

    The Schedule is represented as a section with name 'basic_schedule_foo' in the LMAPD configuration file, which has an option 'execution_mode'.
  - The 'set' method sets the value of the given option within the specified section of the given configuration. It has the following declaration: set(config, sid, option, value). It is the same as the 'get' method parameters, except that in this case we also provide the value that is going to be set for the given option.
    Here is an example of how to set the execution_mode of a given Schedule:

    ```
    uci.set('lmapd', 'basic_schedule_foo', 'execution_mode', '
        sequential');
    ```
  - The 'add' method adds a new section of the given type and name to the given configuration.
    It has the following declaration: add(config, type, name). 'config' is the name of the configuration file as already discussed, 'type' is the section type and 'name' is the section name. Before setting the option value in a specific section, the section first has to be added to the configuration file.
    Here is an example of how to add a section of type 'schedule':

    ```
    uci.add('lmapd', 'schedule', 'basic_schedule_foo');
    ```

5.2.4 Security - ACL

To secure the data on the server, we should restrict the user only to specific files that he/she can see and modify via Ubus. That is done via an 'Access Control List' (ACL). The ACL I created for LMAPD is located at '/usr/share/rpcd/acl.d/luci-app-lmapd.json'. This ACL describes the permissions for the LMAPD interface:

```
{
   "luci-app-lmapd": {
       "description": "Grant access to LMAPD configuration",
       "read": {
           "cgi-io": [ "exec", "download" ],
           "file": {
               "/usr/share/lmapd/lmapctl.sh status": [ "exec" ],
               "/etc/init.d/lmapd start": [ "exec" ],
               "/var/run/lmapd-state.json": [ "read" ],
               "/tmp/lmapd/report_primary/*": [ "read" ],
               "/var/run/lmapd.pid": [ "read" ],
               "/usr/bin/lua /usr/lib/lua/lmapd/generateJson.lua": [ "exec
                  " ],
               "/usr/bin/lua /usr/lib/lua/lmapd/get_results_filenames.lua
                  *": [ "exec" ],
               "/usr/bin/lua /usr/lib/lua/lmapd/delete_results.lua *": [ "
                  exec" ]
           },
           "uci": [
               "lmapd"
           ]
       },
       "write": {
           "uci": [
               "lmapd"
           ]
       }
   }
}
```

In order to define what permissions a user has, there is login-to-acl-group relation, declared in '/etc/config/rpcd'. By default there is a 'root' login with 'read *' and 'write *', which grants access to all defined ACL groups. Therefore, the 'root' user has access to the LMAPD ACL.

In the code snippet shown above, we can see that the allowed functionality is 'exec' and 'download', which means that we can read files and execute commands on the server. The inner-object with name 'file' lists all the paths to the files that can be read and all the commands that can be executed. In the cases where there is a '*' symbol, it means that all the files in the given directory can be read or the command can be executed with any command-line arguments.
In the case of deleting results or getting the results' filenames, the user can execute the

command with any argument. However, The script is expecting a Schedule name and an Action name. It is checking if the passed parameters are valid Schedule and Action names by searching for them in the directory where LMAPD stores its measurement results: '/tmp/lmapd'. If the passed arguments are invalid, no such file is found and no results are returned or deleted.
The 'uci' object gives access only to the LMAPD configuration file so the user cannot change any other configuration file according to this ACL.
Now, we are not only securing that no other user except the root can access the LMAPD interface but also restricting the root user within a set of files that he/she can access. If he/she tries to access something out of the given scope, an error message 'Request denied by ACL' is displayed on the screen.

6 Conclusion and future work

The aim of my project is to create an interface for configuring the Measurement Agent without needing a Controller and a Collector. The interface that I made is web-based and it is making the configuration of the Measurement Agent more intuitive, since users don't have to go to the configuration file and change the entries there directly.

There was an LMAPD interface, which was developed as a Bachelor Thesis by Mohit Shrestha at Jacobs University last year. It was written using the MVC (Model-View-Controller) Framework provided by LuCI. This framework uses a template engine and the views are rendered on the server. The interface was written mainly in Lua programming language and the logic was executed mainly on the server.
The interface that I created uses a client-side rendering approach. It is written mainly in Javascript and uses the LuCI client-side Javascript API. Most of the logic is executed on the client's browser and the Ubus tool is used to communicate with the server and access system data. Lua scripts are used scarcely, for instance, to convert the LMAPD configuration from UCI to JSON format. The reason I chose this approach is the evolution of LuCI towards client-side rendering. LuCI is very resource consuming and performs bad on devices with slow CPU and little amount of RAM. OpenWrt devices are usually wireless routers, which don't have strong parameters. Therefore, moving the logic to the client's browser will take the load off the OpenWrt devices.
The interface I created is also focusing on user-friendliness. I am providing drop-down lists with available values for the fields where a set of predefined values is expected. Also, I am providing the metric for the fields where a number is expected, as well as default values to give a hint of what type of value should be inserted. There are also appropriate error messages when an invalid value is provided or a value is missing.

Future work could focus on the following points:

- A menu, designed for advanced users could be made, targeting people who are familiar with the structure of the configuration file of LMAPD and would like to have more options for configuration.
- Support for further Measurement Tasks could be added to the 'Add Schedule' page. At the moment only 'ping' and 'traceroute' Measurement Tasks are supported.
- The option to add multiple Actions in a Schedule could also be added.

- A survey could be made with users from different backgrounds in order to improve the user-friendliness of the interface.

7 Acknowledgements

I would like to thank my thesis supervisor, Prof. Jürgen Schönwälder for his advice and guidance throughout the whole process. I'd like to thank Jo-Philipp Wich, who is one of the main contributors to the LuCI Framework and who helped me a lot to gain the initial knowledge I needed to make use of the framework. Last but not least, I want to thank Mohit Shrestha for his advice and also for the work he did on this topic last year, which served as a basis for my thesis project.

References

[1] *Accessing uci procedures with ubus over HTTP; visited on: 08.03.2020*. URL: https://forum.archive.openwrt.org/viewtopic.php?id=52868.

[2] M. Bagnulo, T. Burbridge, S. Crawford, P. Eardley, J. Schoenwaelder, and B. Trammell. "Building a standard measurement platform". In: *IEEE Communications Magazine* 52.5 (May 2014), pp. 165–173. ISSN: 1558-1896. DOI: 10.1109/MCOM.2014.6815908.

[3] V. Bajpai and J. Schönwälder. "A Survey on Internet Performance Measurement Platforms and Related Standardization Efforts". In: *IEEE Communications Surveys Tutorials* 17.3 (thirdquarter 2015), pp. 1313–1341. ISSN: 2373-745X. DOI: 10.1109/COMST.2015.2418435.

[4] T. Burbridge, P. Eardley, M. Bagnulo, and J. Schoenwaelder. *Information Model for Large-Scale Measurement Platforms (LMAPs)*. RFC 8193. Aug. 2017. DOI: 10.17487/RFC8193.

[5] *CBI; visited on: 22.02.2020*. URL: https://github.com/openwrt/luci/wiki/CBI.

[6] P. Eardley, A. Morton, M. Bagnulo, T. Burbridge, P. Aitken, and A. Akhter. *A Framework for Large-Scale Measurement of Broadband Performance (LMAP)*. RFC 7594. Sept. 2015. DOI: 10.17487/RFC7594.

[7] *Init Scripts; visited on: 22.02.2020*. URL: https://openwrt.org/docs/techref/initscripts.

[8] *JUCI web interface for OpenWRT; visited on: 09.03.2020*. URL: https://github.com/mkschreder/juci.

[9] A. Leff and J. T. Rayfield. "Web-application development using the Model/View/-Controller design pattern". In: *Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference*. Sept. 2001, pp. 118–127. DOI: 10.1109/EDOC.2001.950428.

[10] *lmapd - xml configuration; visited on: 07.03.2020*. URL: https://github.com/schoenw/lmapd/blob/master/doc/lmapd-config.xml.

[11] *LuCI - Essentials; visited on: 22.02.2020*. URL: https://openwrt.org/docs/guide-user/luci/luci.essentials.

[12] *LuCI - Technical Reference; visited on: 22.02.2020*. URL: https://openwrt.org/docs/techref/luci.

[13] *LuCI - Templates; visited on: 22.02.2020*. URL: https://github.com/openwrt/luci/wiki/Templates.

[14] *LuCI - Writing Modules; visited on: 22.02.2020*. URL: https://github.com/openwrt/luci/wiki/ModulesHowTo.

[15] *LuCI client-side Javascript API; visited on: 08.04.2020*. URL: http://openwrt.github.io/luci/jsapi/LuCI.fs.html.

[16] *LuCI in OpenWrt 29.07.2 Release; visited on: 09.03.2020*. URL: https://openwrt.org/releases/19.07/notes-19.07.2.

[17] *LuCI support for LMAP daemon; visited on: 09.03.2020*. URL: https://github.com/mmshress/lmapd-openwrt.

[18] *LuCI2; visited on: 23.02.2020*. URL: https://openwrt.org/docs/techref/luci2.

[19] *mtr - Linux man page; visited on: 07.03.2020*. URL: https://linux.die.net/man/8/mtr.

[20] *OpenWrt build system – Installation; visited on: 14.04.2020*. URL: https://oldwiki.archive.openwrt.org/doc/howto/buildroot.exigence.

[21] *OpenWrt Feeds; visited on: 14.04.2020*. URL: https://oldwiki.archive.openwrt.org/doc/devel/feeds.

[22] *OpenWrt Project; visited on: 22.02.2020*. URL: https://openwrt.org/.

[23] *OUI web interface for OpenWRT; visited on: 09.03.2020*. URL: https://github.com/zhaojh329/oui.

[24] *RPC daemon; visited on: 08.03.2020*. URL: https://openwrt.org/docs/guide-developer/rpcd.

[25] M. Shrestha. “OpenWrt LuCI Support for the LMAP Daemon”. In: (2019).

[26] Jungkee Song, Julian Aubourg, Hallvord Steen, and Anne van Kesteren. *XML-HttpRequest Level 1*. W3C Note. https://www.w3.org/TR/2016/NOTE-XMLHttpRequest-20161006/. W3C, Oct. 2016.

[27] *Ubus; visited on: 23.02.2020*. URL: https://openwrt.org/docs/techref/ubus.

[28] *UCI System; visited on: 22.02.2020*. URL: https://openwrt.org/docs/guide-user/base-system/uci.

[29] *xLuCI2 web interface for OpenWRT; visited on: 09.03.2020*. URL: https://github.com/zhaojh329/xluci2.