One implementation of API interface for RouterOS

Gencho Stoitsov[1], Vasil Rangelov[1]

[1]*Plovdiv University "Paisii Hilendarski", 236 Bulgaria Blvd., 4003 Plovdiv, Bulgaria*

Abstract **–The purpose of this publication is to present the implementation of API interface for RouterOS of the company MikroTik, which allows using PHP scripts for obtaining data for configuration and management of the routing devices that use this OS.**

Keywords **– API interface for RouterOS, MikroTik.**

1. Introduction

RouterOS is an operating system based on Linux V3.3.5 kernel[8][5][6]. It is used by MikroTik company in the implementation of specialized routing devices, and to convert a standard PC into a router. The system provides a rich set of networking features, making it a favorite in the implementation of network topologies.

One of the opportunities it provides is the integration with external systems through its API (Application Programmable Interface) interface. It allows the implementation of user software to communicate with RouterOS to obtain information, configuration and management of such devices. The service providing this opportunity is called api and answers to port 8728 [11], and the used protocol is referred to as RouterOS API. Its detailed description can be found on the page of MikroTik [7]. The structure of the command line is: API command [attributes]. It is designed for machine processing, so MikroTik is committed not to change the protocol in an incompatible way. Even if there are changes to the transition between the different versions, the format of the error message remains relatively constant, which guarantees (at least in theory) an adequate response to the client application when an error occurs.

2. Implementation of API

This work aims to provide a reliable, flexible and easy to use implementation of the API interface [12] for integration of external systems, pursuing different goals of their own, in relation to RouterOS. Another convenience is that the network administrator is able to provide applications for statistical data to the end clients.

The implementation involves significant, and in some cases unique opportunities for the use of the protocol in comparison with others of that kind [2],[3],[4]. It is marked as recommended on the page of MikroTik. It is carried out in PHP, due to its mass use. The standard codes PEAR2 are used[10], the tools PHPCS, PHPMD, PHPUnit and NetBeans editor, because of its excellent integration with PHPUnit, and the available supplements for PHPCS and PHPMD. The implemented versions[9] are as follows:

- 1.0.0b1 - first implementation;
- 1.0.0b2 - a parser is added to convert console commands into commands for the API protocol and the errors have been corrected;
- 1.0.0b3 – a support for "permanent links" is added, that allows to reduce the load on the router at the expense of slightly higher load on the web server;
- 1.0.0b4 – provides an opportunity for encryption with TLS connections for security. Util class is added [13], which facilitates more complex operations such as access and search by number, file operations and execution of temporary RouterOS scripts with parameters;

- 1.0.0b5 (in development) –will provide:
  - API console for quick tests and locating problems;
  - new Util methods - getall(), count() and prepareScript();
  - supporting the associative arrays introduced in RouterOS 6.2, flags as values with a numeric key;
  - searching and sorting on certain criteria into the answers from class ResponseCollection;
  - andsome another improvements.

3. Example applications

The examples suggest that the library has been installed with Pyrus, PEAR or Composer, and that PEAR2_Autoload has been installed. Also, the router is considered to be accessed via a private IP address in terms of the server. The library can work without these restrictions too, but we specify them now for clarity.

NOTE: The library is also available in the form of a „phar" file. In each of the examples you should be

able to replace „PEAR2/Autoload.php“ with the path to the „phar“ file.

- **Review of the system log**

The example shows the log of the RouterOS in a table. This type of information should not be available to public, in order to avoid unauthorized access.

```
<?php
use PEAR2\Net\RouterOS; //inclusion of the library
require_once 'PEAR2/Autoload.php'; //include
Autoload.php
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Strict//EN""http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type"
content="text/html;charset=UTF-8" />
<title>RouterOS log</title>
<style type="text/css">
table, td, th {border: 1px solid black;}
td span {outline: 1px dotted black;}
</style>
</head>
<body>
<?php
try {
        $client = new RouterOS\Client('192.168.0.1',
'admin', 'password');
    } catch (Exception $e) {
?><div> Failed to connect toRouterOS.</div>
<?php
    }
if (isset($client)) {
?><table>
<thead>
<tr>
<th>Time</th>
<th>Topic</th>
<th>Message</th>
</tr>
</thead>
<tbody><?php
        $logEntries = $client->sendSync(
newRouterOS\Request('/log print'))-
>getAllOfType(RouterOS\Response::TYPE_DATA);
foreach($logEntries as $entry) {?>
<tr>
<td><?php echo $entry('time');?></td>
<td><?php
$topics = explode(',', $entry('topics'));
foreach ($topics as $topic) {?>
<span><?php echo $topic;?></span><?php
}?>
</td>
<td><?php echo $entry('message');?></td>
</tr>
<?php }?>
</tbody>
</table>
<?php }?></body>
</html>
```

- **Ping from the router**

The example provides an opportunity for establishing a remote connection between the router and a client from the inside network.

```
<?php
use PEAR2\Net\RouterOS; //inclusion of the library
require_once 'PEAR2/Autoload.php'; //include
Autoload.php
header('Content-Type: text/html;charset=UTF-8');
if (isset($_GET['act'])) {
$client = new RouterOS\Client('192.168.0.1', 'admin',
'password');
if ($_GET['act'] === 'Ping' &&isset($_GET['address'])) {
$pingRequest = new RouterOS\Request('/ping
count=3');
$results = $client->sendSync($pingRequest-
>setArgument('address', $_GET['address']));
    }
}?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Strict//EN""http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Ping</title>
</head>
<body>
<div>
<form action="" method="get">
<ul>
<li>
<label for="address">IP address:</label>
<input type="text" id="address" name="address"
value="<?php
if (isset($_GET['address'])) {
echohtmlspecialchars($_GET['address'], ENT_COMPAT,
'UTF-8');
                    }?>" />
</li>
<li>
<input type="submit" id="act" name="act" value="Ping"
/>
</li>
</ul>
</form>
</div>
<?php
if (isset($_GET['act'])) {
echo '<div>Results:<ul>';
foreach ($results as $result) {
echo '<li>Time:', $result->getArgument('time'), '</li>';
    }
echo '</ul></div>';
}?>
</body>
</html>
```

- **Determining a client MAC address**

The example allows to determine the MAC address of the client device on the local network by specifying its IP address.

```
<?php
use PEAR2\Net\RouterOS; //inclusion of the library
require_once 'PEAR2/Autoload.php'; //include
Autoload.php
header('Content-Type: text/html;charset=UTF-8');?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Strict//EN""http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>YourMAC address</title>
</head>
<body>
<h1>
<?php
try {
   $client = new RouterOS\Client('192.168.0.1',
'admin', 'password');
   $printRequest = new RouterOS\Request('/iparp
print .proplist=mac-address');
   $printRequest-
>setQuery(RouterOS\Query::where('address',
$_SERVER['REMOTE_ADDR']));
   $mac = $client->sendSync($printRequest)-
>getArgument('mac-address');
if (null!== $mac) {
echo'YourMAC address: ', $mac;
   } else {
echo'Your IP address (',
$_SERVER['REMOTE_ADDR'],') is not part of our
network, so we can not determine MAC address';
   }
} catch(Exception $e) {
echo 'We can not determine the MAC address of your
time. We apologize for the inconvenience.';
}?>
</h1>
</body>
</html>
```

- **Traffic storage and review**

Traffic monitoring is something that is often used in practice. The instrument of MikroTik Winbox, and the web interface of RouterOS generate a graph that includes the period from launch to their suspension. With the help of the generated library, it is possible to run an application in PHP, storing the received data in the database, after which they can be analyzed. Figure 1. shows a graph based on stored traffic.

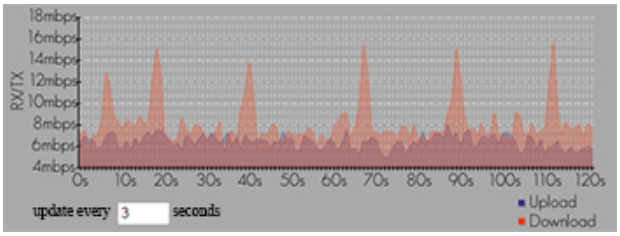

Figure 1. Graphics from data collected in the last two minutes

The application is composed of three parts:

1. A script for receiving and recording the current traffic in a database -monitor.php. It is necessary to be launched in advance from the command line. This is the only one of the three parts, which uses the library. The purpose of the others is to examine the gathered information.

```
<?php
use PEAR2\Net\RouterOS;
require_once 'PEAR2/Autoload.php';
try{
$client=new RouterOS\Client('192.168.0.1','admin',
'password');
$mysqli=new mysqli('localhost','root','password','db');
}catch(Exception$e){
die('Connectionerror:'.$e);
}
$insertQuery=$mysqli->prepare('INSERT INTO
`stats`(`rx-bits-per-second`,`tx-bits-per-second`) VALUES
(?,?)');
$insertQuery-
>bind_param('ii',$rx_bits_per_second,$tx_bits_per_se
cond);
$monitor=new RouterOS\Request('/interface monitor-
traffic interval=1s interface=LAN2.proplist=rx-bits-
per-second,tx-bits-per-second');
$monitor->setTag('m');
$client-
>sendAsync($monitor,function($response)use($insertQ
uery,$rx_bits_per_second,$tx_bits_per_second){
  $rx_bits_per_second=$response->getArgument('rx-
bits-per-second');
  $tx_bits_per_second=$response->getArgument('tx-
bits-per-second');
$insertQuery->execute();});
//traffic monitoring
$client->loop();
?>
```

The table, containing the traffic has the following structure:

```
CREATE TABLE IF NOT EXISTS `stats` (
 `time` TIMESTAMP NOT NULL DEFAULT
CURRENT_TIMESTAMP,
 `rx-bits-per-second` INT UNSIGNED NOT NULL,
 `tx-bits-per-second` INT UNSIGNED NOT NULL,
PRIMARY KEY (`time`) )
ENGINE = InnoDB;
```

2. A script for processing the traffic from the last two minutes and generates a graph for them - img.php. It requires the library pChart to draw the graph.

```
<?php
//Preparation of the necessary components pChart
require_once 'pChart/class/pDraw.class.php';
require_once 'pChart/class/pImage.class.php';
require_once 'pChart/class/pData.class.php';
//Connecting to databasefor reading data
try {
   $mysqli = new mysqli('localhost', 'root', 'password',
'db');
} catch (Exception $e) {
die('Error connecting to database: ' . $e);
}

//Reading data from a database
$time = $rx_bits_per_second = $tx_bits_per_second =
array();
$result = $mysqli->query(
   "SELECT `time`, `rx-bits-per-second`, `tx-bits-per-
second` FROM `stats`
   WHERE `time` >= '" .date('Y-m-d H:i:s', $startTime =
(time() - 121)) . "'"
);
while ($row = $result->fetch_assoc()) {
   $currentTime = 0 - ($startTime -
DateTime::createFromFormat('Y-m-d H:i:s', $row['time'])-
>getTimestamp() + 120);
if (0 === $currentTime % 10) {
      $time[] = $currentTime;
   } else {
      $time[] = VOID;
   }
   $rx_bits_per_second[] = $row['rx-bits-per-second'];
   $tx_bits_per_second[] = $row['tx-bits-per-second'];
}
//Appointment of retrieved data
$stats = new pData();
$stats->addPoints($time, 'Time');
$stats->addPoints($rx_bits_per_second, 'rx-bits-per-
second');
$stats->addPoints($tx_bits_per_second, 'tx-bits-per-
second');
//Settings of the appearance of the time axis
$stats->setSerieTicks('Time', 10);
$stats->setAbscissa('Time');
$stats->setXAxisName('Time');
$stats->setXAxisUnit('s');
$stats->setXAxisDisplay(AXIS_FORMAT_DEFAULT);
//Settings for the vertical axis
$stats->setAxisName(0, 'RX/TX');
$stats->setAxisUnit(0, 'bps');
$stats->setAxisDisplay(0, AXIS_FORMAT_METRIC);
$stats->setSerieTicks('RX/TX', 1024);

//Upload speed settings
$stats->setSerieOnAxis('rx-bits-per-second', 0);
$stats->setSerieDescription('rx-bits-per-second', 'Upload');
$stats->setPalette('rx-bits-per-second',array('R'=>50,
'G'=>50,'B'=>127));

//Download speed settings
$stats->setSerieOnAxis('tx-bits-per-second', 0);
$stats->setSerieDescription('tx-bits-per-second',
'Download');
$stats->setPalette('tx-bits-per-second',array('R'=>255,
'G'=>50,'B'=>1));
//Preparing the graphic
$chart = new pImage(700, 250, $stats, true);
$chart->FontName = 'pChart/fonts/GeosansLight.ttf';
$chart->setGraphArea(160, 40, 640, 180);
$chart->drawScale();
$chart->drawLegend(
   570, 210, array('Style'
=>LEGEND_NOBORDER,'Mode' =>
LEGEND_VERTICAL));
$chart->drawAreaChart(array('ForceTransparency' =>
60));
//Sending the result
$chart->Stroke();
?>
```

3. A script for visualization and update of the obtained data at regular intervals - index.php.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xml:lang="bg" lang="bg">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
<title>Statistics </title>
<style type="text/css">
html, body {background-color: #aaa;}
        #statHolder {width:800px;margin:0 auto;}
</style>
</head>
<body>
<div id="statHolder">
<imgsrc="img.php" alt="Statistics" id="stat"/>
</div>
<div>
<form action="">
<div>update every<input id="refreshRate" type="text"
size="2" value="3" />seconds</div>
</form>
</div>
<script type="text/javascript">
        //<![CDATA[
varloadingImage = new Image();
loadingImage.style.display = 'none';
loadingImage.src = 'loading.gif';
varstatImage = document.getElementById('stat');
statImage.parentNode.appendChild(loadingImage);
```

```
varrefreshRate = 3;
varrefreshRateElement =
document.getElementById('refreshRate');
refreshRateElement.onkeyup = function(e) {
varnewRefreshRate = new
Number(refreshRateElement.value);
if (!isNaN(newRefreshRate) &&newRefreshRate> 0) {
refreshRate = newRefreshRate;
            }
          };
var refresher = function() {
loadingImage.style.display = 'inline';
varnewImage = new Image();
            newImage.id = 'stat';
newImage.onload = function() {
statImage.parentNode.replaceChild(newImage, statImage);
statImage = newImage;
loadingImage.style.display = 'none';
setTimeout(refresher, refreshRate * 1000);
            };
newImage.src = "img.php?"+new Date();
          };
setTimeout(refresher, refreshRate * 1000);
          //]]>
</script>
</body>
</html>
```

- **Implementation of more complex systems**

The implementation can be used for the realization of more complex applications such as:

- Systems for client management (popular in the industry as „billing“ systems)[1], providing the opportunity to add, edit, remove, include and exclude subscribers, which is otherwise associated with manual execution of multiple commands to the router. We are informed about systems of this type, using a specific implementation, that provides features such as:
  - Adding /editing /removing clients, plans and IP addresses;
  - Exclusion of clients when the payment period has expired.

These operations cause the execution of multiple commands in RouterOS. For example, registration in the Queue list(the place where the speed is limited for each user), registration of the IP address in the NAT list and the filters of the firewall.

- Managing basic settings of the router. It can be realized as an opportunity provided by the internet service provider to end users to control the MikroTik routing devices given to the clients.

4. Conclusion

In practice, the opportunities offered by the console of the MikroTik system can be realized by a PHP script, using the discussed implementation of MikroTik API. The constant enrichment and improvement of the program code guarantees the reliable functioning of the applications that use it.

References

[1]. Adam Mohammed Saliu, Mohammed Idris Kolo, Mohammed Kudu Muhammad, Lukman Abiodun Nafiu, (2013).Internet Authentication and Billing (Hotspot) System Using MikroTik Router Operating System, International Journal of Wireless Communications and Mobile Computing,1(1),51-57.

[2]. API ActionScript 3 class, http://wiki.mikrotik.com/wiki/API_ActionScript_3_class , (last visited on 17.03.2014).

[3]. API Delphi, http://wiki.mikrotik.com/wiki/API_Delphi (last visited on 17.03.2014).

[4]. API in CPP, http://wiki.mikrotik.com/wiki/API_In_CPP

[5]. (last visited on 17.03.2014).

[6]. Burgess, D. (2009). Learn RouterOS, ISBN 978-0557092710.

[7]. Discher, St. (2011). RouterOS by Example, ISBN: 978-0615547046.

[8]. Manual:API, http://wiki.mikrotik.com/wiki/Manual:API (last visited on 21.02.2014).

[9]. MikroTikRouterOS, http://wiki.mikrotik.com/wiki/Manual:TOC (last visited on 21.02.2014).

[10]. Net_RouterOS Releases, https://github.com/pear2/Net_RouterOS/releases (last visited on 21.02.2014).

[11]. PEAR, http://pear2.php.net/ (last visited on 21.02.2014).

[12]. Stoitsov, G. (2010). Types of addresses and levels of use in the TCP/IP protocol stack, REMIA 2010, Plovdiv.

[13]. The implementation of the API Manual, https://github.com/pear2/Net_RouterOS/wiki (last visited on 21.02.2014).

[14]. Util extras, https://github.com/pear2/Net_RouterOS/wiki/Util-extras , (last visited on 17.03.2014).

Corresponding author: Gencho Stoitsov
Institution: Plovdiv University “Paisii Hilendarski“, Bulgaria
E-mail: stoitzov@uni-plovdiv.bg