Tutorial OMF: Criação de tutoriais integrados a interface LS-WEB

1. Visão geral

Tutoriais integrados a interface do LS-WEB são tutoriais que podem ser executados pelos usuários do testbed de forma rápida e fácil onde o usuário apenas necessita ter acesso a interface WEB e o tutorial será executado utilizando o próprio OMF. Os tutoriais podem ter entradas configuradas de acordo com o experimento que o usuário deseja rodar e é mostrado ao usuário a saída do console da execução do OMF, além de ter a possibilidade de mostrar ao usuário a coleta dos dados do experimento em forma de gráficos e em tempo real. Os tutoriais podem ser acessados via portal LS-WEB pelo menu "**Tutorials**" que está disponível a todo usuário logado na ilha que possui suporte aos tutoriais.

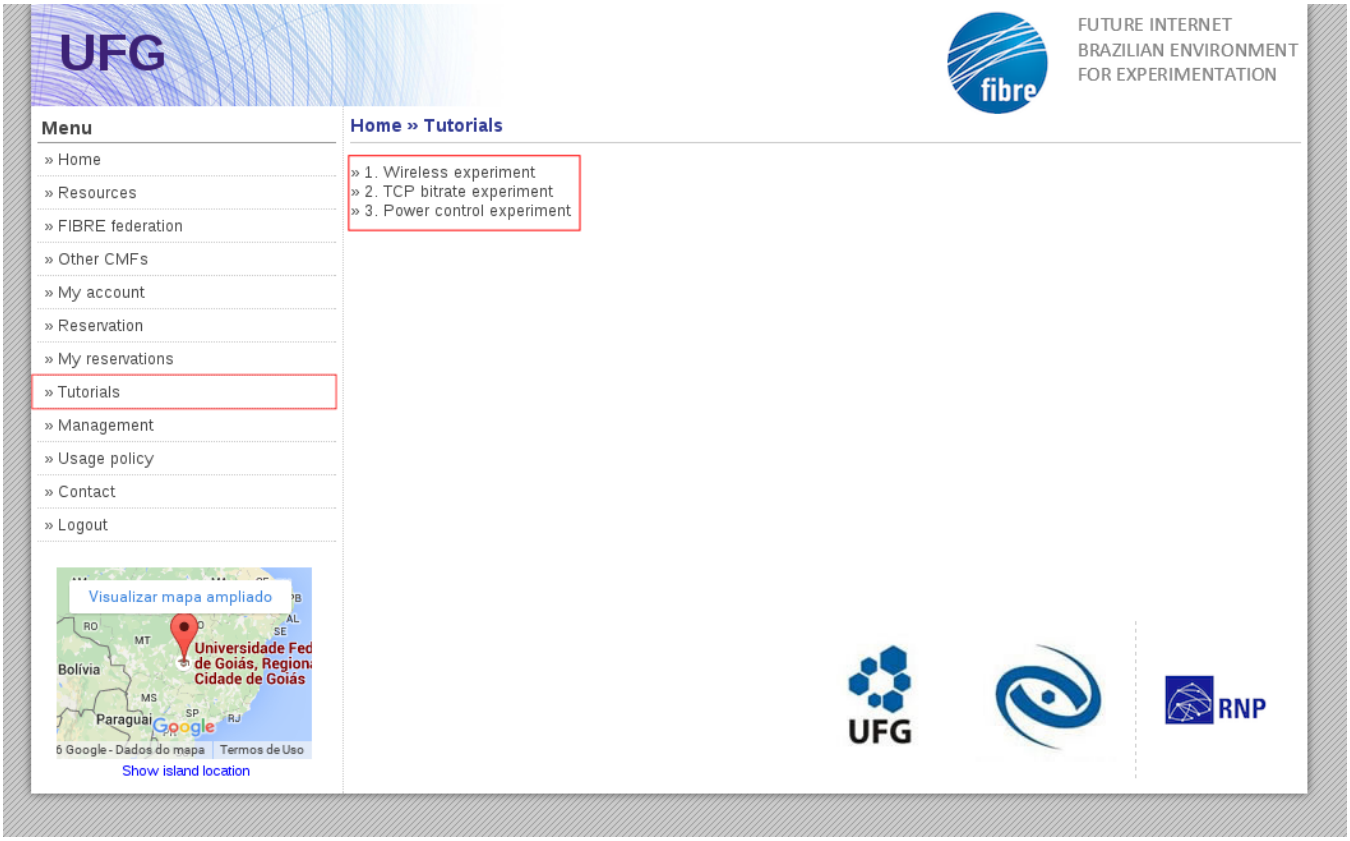

Figura 1: Acessando os tutoriais integrados ao LS-WEB

Os usuários poderão, caso desejem, rodar os experimentos dos tutoriais manualmente, utilizando o comando "omf exec" (que o próprio LS-WEB utiliza para executar o tutorial). Os scripts utilizados nos tutoriais estão todos disponíveis na home do usuário do testbed em "/home/**<Instituição>/<usuário sem o @instituição>**/tutorials/omf_experiments".

Como o LS-WEB roda por baixo o experimento do tutorial utilizando o próprio comando **exec** do OMF, neste tutorial não vamos ensinar como criar experimentos para o OMF, de modo que assumimos que o experimento OMF já está criado e se deseja apenas integrar e disponibilizar o experimento para que este seja executado utilizando o LS-WEB. Naturalmente, para prosseguir com a integração precisamos que o arquivo de descrição do experimento no formato **OEDL** já esteja criado e localizado no diretório "**/home/tutorials/omf_experiments/**" com a permissão 555. Abaixo um exemplo de um script **OEDL** utilizado no tutorial 1 "Wireless experiment", integrado ao LS-WEB.

Script OEDL do tutorial 1: Wireless Experiment

```
defProperty('server_node','omf.ufg.node8',"ID of server node")
defProperty('client_node','omf.ufg.node1',"ID of client node")
defProperty('runtime', 60, "Time in second for the experiment is to run")
defProperty('iperf_server_address', '192.168.137.2', "Iperf server IP address")
defProperty('iperf_client_address', '192.168.137.1', "Iperf client IP address")
defProperty('iperf_port', 2000, "Iperf port")
defProperty('iperf_interval', '1', "Iperf interval")
defProperty('iperf_isUDP', 'false', "Set Iperf UDP")
defProperty('iperf_bandwidth', '15M', 'Iperf bandwidth')
```

```
# Define the resources group 'Server'
defGroup('Server', property.server_node) do |node|
  node.addApplication("tutorials:iperf", :id => 'iperf_server') do |app|
    app.setProperty('server', true)
    app.setProperty('port', property.iperf_port)
    app.setProperty('interval', property.iperf_interval)
    app.setProperty('udp', (property.iperf_isUDP.value == 'true' ? true : false))
    app.setProperty('reportstyle', 'o')
    app.setProperty('oml-id', 'server')
    app.setProperty('oml-domain', 'wireless_experiment_iperf')
    app.setProperty('oml-collect', 'tcp:10.137.11.200:3004')
  end

  node.addApplication("tutorials:utils:wlanconfig", :id => 'server_wlanconfig') do |app|
    app.setProperty('wlan', 'wlan0')
    app.setProperty('mode', 'adhoc')
    app.setProperty('type', 'g')
    app.setProperty('channel', 6)
    app.setProperty('essid', 'omf_we')
    app.setProperty('ip_address', property.iperf_server_address)
    app.setProperty('duration', (property.runtime + 20))
  end
end

# Define the resources group 'Client'
defGroup('Client', property.client_node) do |node|
  node.addApplication("tutorials:iperf", :id => 'iperf_client') do |app|
    app.setProperty('client', property.iperf_server_address)
    app.setProperty('port', property.iperf_port)
    app.setProperty('interval', property.iperf_interval)
    app.setProperty('time', property.runtime)
    app.setProperty('udp', (property.iperf_isUDP.value == 'true' ? true : false))
    if property.iperf_isUDP.value == 'true'
      app.setProperty('bandwidth', property.iperf_bandwidth)
    end
    app.setProperty('reportstyle', 'o')
    app.setProperty('oml-id', 'client')
    app.setProperty('oml-domain', 'wireless_experiment_iperf')
    app.setProperty('oml-collect', 'tcp:10.137.11.200:3004')
  end

  node.addApplication("tutorials:utils:wlanconfig", :id => 'client_wlanconfig') do |app|
    app.setProperty('wlan', 'wlan0')
    app.setProperty('mode', 'adhoc')
    app.setProperty('type', 'g')
    app.setProperty('channel', 6)
    app.setProperty('essid', 'omf_we')
    app.setProperty('ip_address', property.iperf_client_address)
    app.setProperty('duration', (property.runtime + 20))
  end
end

onEvent(:ALL_UP_AND_INSTALLED) do |event|
  info "Running tutorial 1: Wireless experiment..."
  wait 10
  info "Configuring wireless network between nodes..."
  group("Server").startApplication('server_wlanconfig')
  group("Client").startApplication('client_wlanconfig')
  wait 5
  info "Starting iperf server..."
  group("Server").startApplication('iperf_server')
  wait 5
  info "Starting iperf client..."
  group("Client").startApplication('iperf_client')
  info "Iperf server and client started..."
  wait property.runtime
  info "Stopping iperf server and client..."
  wait 5
  allGroups.stopApplications
  info "Iperf server and client stopped..."
```

```
  Experiment.done
end
```

Observem que as variáveis que serão configuradas durante a execução do experimento devem estar todas definidas por "**defProperty**", conforme o exemplo acima, pois deste modo podemos substituir o valor dessas variáveis na execução do experimento via comando "omf exec".

A lista de aplicações instrumentadas e que podem ser utilizadas no experimento OMF está disponível em "**/usr/share/omf-expctl-5.4/repository/**". abaixo temos a lista das aplicações que foram instrumentadas para serem utilizadas nos tutoriais e que podem ser utilizadas em quaisquer experimentos OMF:

Nome	Identificação via OEDL	Descrição
iperf	tutorials:iperf	Aplicação do iperf instrumentada para coletar dados de saída do iperf, como pacotes de entrada e saída.
iwdata	tutorials:iwdata	Aplicação instrumentada para coletar dados do comando iw. **Atualmente a aplicação coleta apenas o RSSI**.
ss	tutorials:ss	Aplicação instrumentada para coletar dados do comando ss. **Atualmente a aplicação coleta apenas o RTT**.
hostapd	tutorials:utils:hostapd	Aplicação do hostapd instrumentada para criar uma rede em modo AP-Station.
wlanconfig	tutorials:utils:wlanconfig	Aplicação instrumentada para configurar a interface dos nós utilizando ifconfig, iw e iwconfig. Esta aplicação utiliza o hostapd e o wpa instrumentado caso necessário.
wpa	tutorials:utils:wpa	Aplicação do wpa supplicante instrumentada para estabelecer a conexão em uma rede AP-Station.

2. Criando o tutorial integrado no LS-WEB

O LS-WEB foi desenvolvido utilizando o modelo arquitetural ~~M~~VC, ou seja, ele possui 03 camadas bem divididas, são elas:

1. ~~Modelo (**M**VC)~~ - Camada responsável pelo acesso aos dados requisitados pela base de dados (No LS-WEB essa camada foi omitida por chamadas diretas à API do LS-Sched)
2. Controle (MV**C**) - Camada responsável pela execução e controle das páginas. Necessáriamente cada página deve possuir uma ação **(action)** em uma controladora **(controller)** responsável por um grupo de entidades ou páginas. No LS-WEB temos as seguintes controladoras:
    a. **ErrorsController**: Responsável pela renderização de páginas de erro no LS-WEB, como o 404 e 501.
    b. **PagesController**: Responsável por páginas mais genéricas, como por exemplo a página de contato.
    c. **MotherboardsController**: Responsável por páginas relativas a motherboards, como por exemplo páginas de visualização, criação, edição e até mesmo remoção de motherboards no sistema.
    d. **PxeImagesController**: Responsável por páginas relativas a pxe images, como por exemplo páginas de visualização, criação, edição e até mesmo remoção de pxe images do sistema.
    e. **ResourcesController**: Responsável por páginas relativas aos recursos do sistema. CRUD de recursos e reservas são de responsabilidade desta controladora.
    f. **UsersController**: Responsável por páginas relativas aos usuários do sistema, como por exemplo cadastro, conta do usuário, etc.
    g. **TutorialsController**: Responsável pelas páginas relativas aos tutoriais. **É AQUI QUE VAMOS ATUAR NO TÓPICO 2.1**.
3. Visualização (M**V**C) - Camada responsável pela renderização das páginas, basicamente nela temos os códigos HTML, CSS, jquery etc. Essas páginas recebem as variáveis que podem ser settadas pela controladora, como por exemplo a lista de usuários, etc.

2.1 Criando uma nova ação na controladora dos tutoriais

Para criar uma nova ação que será a controladora do novo tutorial que está sendo desenvolvido, basta criar um método **público** na classe **TutorialsController** localizada em "../LS-WEB/Controller/TutorialsController.php". Para explicar como a nova ação pode ser criada usaremos como exemplo a ação utilizada para o tutorial 1: Wireless Experiment, disponível no LS-WEB:

Controladora do tutorial 1: Wireless Experiment

```
/**
 * Wireless experiment Controller
 */
public function wireless_experiment() {
    if(count($_POST) > 0) {
        $experimentProperties = array(
            "iperf_isUDP" => ($_POST['iperf_type'] == 'udp') ? 'true' : 'false'
        );

        if($_POST['iperf_type'] == 'udp') {
            $iperfSize = (is_numeric($_POST['iperf_bandwidth_size']) && $_POST['iperf_bandwidth_size'] > 0 &&
$_POST['iperf_bandwidth_size'] < 100) ?
                $_POST['iperf_bandwidth_size'] : 15;
            $experimentProperties['iperf_bandwidth'] = $iperfSize . 'M';
        }

        $experimentParams = array(
            "username" => $_SESSION[SECURITY_SESSION]['username'],
            "experiment" => "wireless_experiment",
            "properties" => $experimentProperties
        );
        $executeExperiment = api_call("execute_experiment", $experimentParams);
        $this->setExperimentMessageStatus($executeExperiment['method_result']['status']);
    }

    $methodParams = array(
        "experiment" => "wireless_experiment"
    );
    $isExperimentExecuting = api_call("is_experiment_executing", $methodParams);
    $isUserExecutingExperiment = $isExperimentExecuting['method_result'] == $_SESSION[SECURITY_SESSION]
['username'];
    $isExperimentExecuting = $isExperimentExecuting['method_result'] != false;

    $experimentScript = api_call("getExperimentScript", $methodParams);
    $experimentScript = ($experimentScript["method_result"] !== false) ? $experimentScript["method_result"] :
"";

    $this->setVariable("is_executing", $isExperimentExecuting);
    $this->setVariable("is_executing_by_user", $isUserExecutingExperiment);
    $this->setVariable("experiment_script", $experimentScript);
}
```

- O if da linha 5 até a linha 23 {if(count($_POST) > 0)} se refere a execução do experimento, ou seja, o que deve ser executado quando o usuário efetivamente clicar no botão de Executar o experimento. Normalmente, se o experimento tiver dados de configuração, como o caso deste tutorial, esses dados chegarão na controladora por meio de $_GET ou $_POST. Neste exemplo os dados estão chegando por $_POST e dentro deste "if" nós verificamos e criamos os dados que serão passados na chamada da API do LS-Sched para execução do experimento. Os dados experados nessa chamada "execute_experiment" são:
    - **username**: Usuário que está solicitando a execução do experimento **(com @<instituição> incluso)**. Normalmente esse dado será obtido por "$_SESSION[SECURITY_SESSION]['username']", uma vez que essa variável retorna o usuário logado no LS-WEB, ao qual se destina a execução do tutorial.
    - **experiment**: Nome do experimento a ser executado **(mesmo nome do script OEDL incluso no diretório "/home/tutorials /omf_experiments" sem a extensão ".rb)**.
    - **properties**: Array contendo as propriedades do experimento, ou seja, as variáveis definidas no script OEDL que poderão ser modificadas e que, naturalmente, devem ter sido inseridas pelo usuário.
- A linha 21 envia a requisição de execução do experimento para a API LS-Sched e seu retorno é armazenado na variável $executeExperiment onde podemos analisar se houve algum erro no pedido da execução do experimento e que, na linha 22, criamos o balão de mensagem informando isto ao usuário.
- As linhas 25 à 27 definem os parâmetros comuns a serem utilizados em algumas outras chamadas à API LS-Sched, são elas:
    - **is_experiment_executing**: Usada na linha 28 e visa saber se o tutorial está sendo executado no momento.
    - **getExperimentScript**: Usada na linha 32 e visa buscar o script OEDL que é utilizado no tutorial.
- As linhas 35 à 37 utilizam o método **setVariable** para settar variáveis que poderão ser utilizadas na páginas (view) do tutorial, ou seja, neste tutorial a view terá acesso as variáveis "is_executing", "is_executing_by_user" e "experiment_script".

2.2 Criando uma página para a ação/tutorial criado

Uma vez com a ação criada na controladora, o nosso próximo passo para a criação do tutorial integrado ao LS-WEB é criar a página que será renderizada para o tutorial. Para isto, basta criar um arquivo de extensão ".vw" dentro do diretório "../LS-WEB/View/Tutorials/". O arquivo deve ter o nome da ação criada na controladora, por exemplo, para o tutorial 1: Wireless Experiment, a sua ação na controladora possui o nome "wireless_experiment", deste modo o arquivo da página deste tutorial deve ser o **"../LS-WEB/View/Tutorials/wireless_experiment.vw"**. Este arquivo irá possuir a view da página, contendo o código html e até mesmo a utilização das variáveis settadas pela ação da controladora dentro de tags do php. Abaixo temos como exemplo a view do tutorial 1: Wireless Experiment:

Página (view) do Tutorial 1: Wireless Experiment

```
<head>
   <!--[if lte IE 8]>
   <script language="javascript" type="text/javascript" src="js/flot/excanvas.min.js"></script>
   <![endif]-->

   <script language="javascript" type="text/javascript" src="js/jquery/jquery.syntaxhighlighter.js"></script>
   <script language="javascript" type="text/javascript" src="js/flot/jquery.flot.js"></script>
   <script language="javascript" type="text/javascript" src="js/flot/jquery.flot.time.js"></script>
   <script language="javascript" type="text/javascript" src="js/flot/jquery.flot.resize.min.js"></script>
   <script language="javascript" type="text/javascript" src="js/flot/jquery.flot.selection.js"></script>
    <script language="javascript" type="text/javascript" src="js/tutorials/flot/statistics.common.js"></script>
    <script language="javascript" type="text/javascript" src="js/tutorials/flot/statistics.wireless_experiment.
js"></script>
    <script language="javascript" type="text/javascript" src="js/tutorials/wireless_experiment.js"></script>
</head>

<?php
   $iperf_type = isset($_POST['iperf_type']) ? $_POST['iperf_type'] : "";
   $iperf_bandwidth_size = isset($_POST['iperf_bandwidth_size']) ? $_POST['iperf_bandwidth_size'] : "15";
?>

<div id='text'>
   <div class='title'>
      <a href='index.php'>Home</a> »
      <a href='index.php?controller=tutorials'>Tutorials</a> »
      Wireless experiment
   </div>
</div>

<form method='POST' action=''>
   <div class="box">
      <div class="box_header">
         Experiment description
      </div>

      <div class="box_body">
         <p>This experiment consists in an instrumented iperf (server/client) running on two icarus nodes
through a wireless network. The proccess of the experiment is below:</p>
         <p>1. Automated nodes reservation.</p>
         <p>2. Since the nodes are reserved, we can use the nodes as we want. Now we will run an iperf server
on the first node and an iperf client on the other one.</p>
         <p>3. The iperf is instrumented to collect and send the data periodically (every second) to the OML
server.</p>
         <p>4. Since we have the iperf output data, we can plot the results in the chart below.</p>
         <p>5. You can set the experiment configuration to use TCP or UDP.</p>
      </div>
   </div>

   <div class="box">
      <div class="box_header">
         Experiment architecture
      </div>

      <div class="box_body">
         <p>Describes the experiment architecture</p>
      </div>
   </div>

    <div class="box">
        <div class="box_header">
            Experiment script
```

```
            <input type="button" value="Show" name="experiment_script" />
        </div>

        <div class="box_body highlight-box hidden">
            <pre class="highlight">
                <?= (!empty($experiment_script)) ? $experiment_script : "Experiment script not found!" ?>
            </pre>
        </div>
    </div>

   <div class="box">
      <div class="box_header">
         Experiment configuration
      </div>

      <div class="box_body">
         <table>
            <tr>
               <td>Iperf type:</td>
               <td colspan="3">
                  <select name="iperf_type">
                     <option value="tcp" <?= ($iperf_type == 'tcp') ? 'selected' : '' ?>>TCP</option>
                     <option value="udp" <?= ($iperf_type == 'udp') ? 'selected' : '' ?>>UDP</option>
                  </select>
               </td>
            </tr>
            <tr class="udp_field" style="display: none;">
               <td>Iperf bandwidth size (Mbits):</td>
               <td>
                  <input type="number" name="iperf_bandwidth_size" value="<?= $iperf_bandwidth_size ?>" min="1"
/>
               </td>
            </tr>
         </table>
      </div>
   </div>

   <div class="box">
      <div class="box_header">
         Experiment execution
         <input type="submit" value="Run" <?php if($is_executing) { echo "disabled"; } ?> />
      </div>

      <div class="box_body">
         <?php if($is_executing): ?>
            <center class="hide-when-done"><b>This tutorial experiment is already running...</b></center>
         <?php endif; ?>

         <div class="box">
            <div class="box-header with-border">
               <h3 class="box-title">Experiment throughput monitoring</h3>
               <div class="box-tools pull-right">
                  Real time
                  <div class="btn-group realtime" data-toggle="btn-toggle">
                    <button type="button" class="btn btn-default btn-xs" data-toggle="on">On</button>
                    <button type="button" class="btn btn-default btn-xs" data-toggle="off">Off</button>
                  </div>
                  <button type="button" class="btn btn-default btn-xs btn-zoom-out">Zoom out</button>
               </div>
            </div>
            <div class="box-body">
               <div id="throughput-graph" style="height: 300px;"></div>
            </div>
            <div class="box-footer clearfix">
               <div id="throughput-legend" class="legend"></div>
            </div>
         </div>
      </div>
   </div>

   <?php if($is_executing_by_user): ?>
```

```
   <div class="box">
      <div class="box_header">
         Experiment console output
      </div>

      <div class="box_body">
         <div id="experimentOutput" style="color: white; background: black; width: calc(100% - 10px); height:
300px; overflow-y: scroll; padding: 5px;"></div>
      </div>
   </div>
   <?php endif; ?>
</form>
```

- Observe que na linha 101 utilizamos uma variável que foi settada pela ação da controladora.
- Os gráficos e a rederização da saída do console da execução do experimento via OMF são coletadas por chamadas assíncronas a um método de coleta criado na API LS-Sched. As chamadas a esse método e a renderização do gráfico e do console são feitas via jQuery/javascript nos arquivos "js/tutorials/flot/statistics.wireless_experiment.js" e "js/tutorials/wireless_experiment.js", conforme carregados nas linhas 12 e 13.

2.3 Criando e/ou ajustando as saídas do experimento

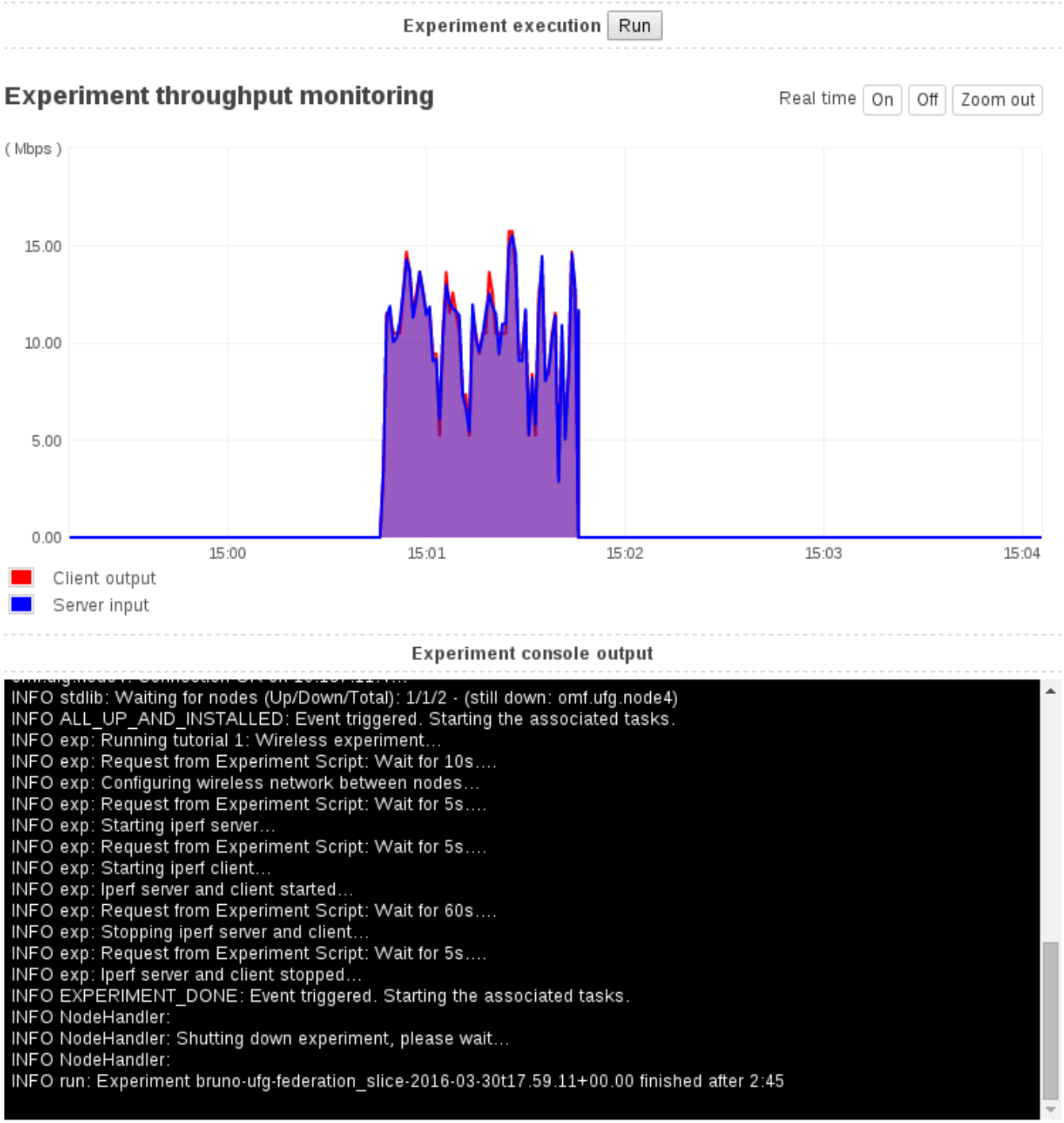

Figura 2: Execução e saída do experimento realizado pelo tutorial.

2.3.1 Criando gráficos utilizando o Flot Charts para visualização dos dados coletados no experimento do tutorial

Como comentado na seção 2.2, o gráfico é criado utilizando uma biblioteca em Jquery chamada Flot Charts. Essa biblioteca é capaz de criar gráficos de diversas formas (barras, estilo pizza, linha, etc) com opções para atualizações em tempo real. Para os tutoriais criados até o momento foi criado utilizando essa biblioteca um gráfico de área que tem opção de zoom e atualização em tempo real com opção desta ser desativada. Deste modo, caso o novo tutorial contenha um gráfico nessas condições, o código desenvolvido pode ser reaproveitado, assim como é feito nos tutoriais já disponíveis. Demais formas de gráficos será necessário consultar e criar utilizando a documentação da biblioteca.

A criação do gráfico consiste de 3 partes:

1. Código HTML que contém as tags que serão referenciadas na página para a criação do gráfico. **(Presente no arquivo .vw)**

2. Método na API do LS-Sched que captura os dados das bases de dados do OML e os retorna em formato para ser plotado no gráfico em formato JSON. **(Seção 3)**
3. Código JQuery/Javascript responsável pela chamada assíncrona que busca os dados na API do LS-Sched e cria/modifica o gráfico utilizando a biblioteca do Flot Charts.

Código HTML

```
<div class="box">
   <div class="box-header with-border">
      <h3 class="box-title">Experiment throughput monitoring</h3>
      <div class="box-tools pull-right">
         Real time
         <div class="btn-group realtime" data-toggle="btn-toggle">
           <button type="button" class="btn btn-default btn-xs" data-toggle="on">On</button>
           <button type="button" class="btn btn-default btn-xs" data-toggle="off">Off</button>
         </div>
         <button type="button" class="btn btn-default btn-xs btn-zoom-out">Zoom out</button>
      </div>
   </div>
   <div class="box-body">
      <div id="throughput-graph" style="height: 300px;"></div>
   </div>
   <div class="box-footer clearfix">
      <div id="throughput-legend" class="legend"></div>
   </div>
</div>
```

- Esse é o código criado para interagir com o JQuery/Javacript do gráfico criado utilizando o Flot Charts para os tutoriais. Vocês poderão observar no código do javascript abaixo que várias das tags utilizadas neste código HTML são referenciadas para que o gráfico seja criado corretamente na página do LS-WEB.

Código JQuery/Javascript

```
$(function() {
   function getData(beforeTime, graph, zoomMode, nowGetTime) {
      var dataUrl = '/LS-Sched/?method=get_wireless_experiment_data' +
         '&before_time_size=' + ( beforeTime + 1 ) + '&now_get_time=' + nowGetTime;

      $.ajax({
         url: dataUrl,
         dataType: 'json',
         async: true,
         success: function(statistics) {
            var distance = getTimmerDistance(graph);

            updateData(statistics.method_result, graph);
            updateGraph(graph);

            if(zoomMode == false) {
               setTimeout(function(){ getData(distance, graph, false, null); }, 1000);
            }

            // Update console box
            try {
               var expOutput = statistics.method_result.output.replace(/\n/g, "<br />").replace("<br />", "").
trim();
            } catch(e) {
               var expOutput = "";
            }

            try {
               if(expOutput) {
                  var experimentOutput = $("#experimentOutput");
```

```
                if(experimentOutput.text() != expOutput.replace(/<br \/>/g, "")) {
                   var oldContent = experimentOutput.text().trim();
                   var isInBottom = ((experimentOutput.scrollTop() + experimentOutput.height() + 10) >=
experimentOutput[0].scrollHeight);

                   experimentOutput.html(expOutput);

                   if(isInBottom || !oldContent) {
                      experimentOutput.scrollTop(experimentOutput[0].scrollHeight);
                   }
                }
             }
          } catch(e) {}

          // Enable test run when its finished.
          try {
             if (expOutput.indexOf("INFO run: Experiment") >= 0) {
                $("input[type=submit]").prop("disabled", false);
                $(".hide-when-done").hide();
             }
          } catch(e) {}
       }
    });
 }

 function updateData( statistics, graph ) {
    if( graph.data.length == 0 ) {
       graph.data.push({
          data: statistics.client,
          label: "Client output",
          name: "client"
       });

       graph.data.push({
          data: statistics.server,
          label: "Server input",
          name: "server"
       });
    } else {
       graph.data.forEach( function( graphData ) {
          switch( graphData.name ) {
             case "client":
                var newData = statistics.client;
                break;
             case "server":
                var newData = statistics.server;
                break;
          }
          graphData.data = graphData.data.concat(newData);
          graphData.data.sort( function(a, b){ return a[0] - b[0] } );
       });
    }
 }

 var pGraph = {
    type: "throughput",
    realTime: true,
    plot: {
       element: "#throughput-graph",
       graph: $.plot("#throughput-graph", [], getOptions("Mbps", [ "red", "blue" ], $("#throughput-legend")))
    },
    data: []
 };

 var zoomButton = $(pGraph.plot.element).parent().parent().find(".box-header .box-tools .btn-zoom-out");
 var realTimeButton = $(pGraph.plot.element).parent().parent().find(".box-header .box-tools .realtime .btn");

 // install select option
 $(pGraph.plot.element).bind("plotselected", function (event, ranges) {
    $(realTimeButton[1]).trigger("click");
```

```
   $.each(pGraph.plot.graph.getXAxes(), function(_, axis) {
      var opts = axis.options;
      opts.min = ranges.xaxis.from;
      opts.max = ranges.xaxis.to;
   });

   pGraph.plot.graph.setupGrid();
   pGraph.plot.graph.draw();
   pGraph.plot.graph.clearSelection();
});

// add zoom out button
zoomButton.click(function (event) {
   event.preventDefault();
   $(realTimeButton[1]).trigger("click");

   $.each(pGraph.plot.graph.getXAxes(), function(_, axis) {
      var opts = axis.options;
      var distance = 0;
      var now = new Date().getTime() - 3000;
      if( opts.min != null ) {
         if( opts.max != null ) {
            distance = opts.max - opts.min;
         } else {
            distance = now- opts.min;
         }

         distance = distance / 2;
      }

      opts.min = opts.min - distance;
      if( opts.max == null ) {
         opts.max = now;
      } else {
         opts.max = opts.max + distance;
      }

      if( opts.max > now ) {
         opts.max = now;
      }

      if(pGraph.data.length > 0) {
         var firstDataTime = Math.floor(pGraph.data[0].data[0][0] / 1000);
         var optsTime = Math.floor(opts.min / 1000 );

         if( optsTime < firstDataTime ) {
            var beforeData = Math.round( ( firstDataTime -  optsTime ) );
            getData(beforeData, pGraph, true, firstDataTime);
            updateGraph(pGraph);
         }
      }
   });

   pGraph.plot.graph.setupGrid();
   pGraph.plot.graph.draw();
   pGraph.plot.graph.clearSelection();
});

// Configure real time buttons
realTimeButton.click(function (event) {
   event.preventDefault();
   if ($(this).attr("data-toggle") === "on") {
      pGraph.plot.graph.clearSelection();
      updateRealTime( pGraph.plot.graph, 5 );

      pGraph.plot.graph.setupGrid();
      pGraph.plot.graph.draw();
      pGraph.realTime = true;
   } else {
      if(pGraph.realTime === true) {
         $.each( pGraph.plot.graph.getXAxes(), function(_, axis) {
```

```
            var opts = axis.options;
            opts.max = new Date().getTime() - 3000;
         });
      }
      pGraph.realTime = false;
   }
});

getData(1800, pGraph, false, null);
});
```

- A função **getData** presente nas linhas 2~52 é a responsável pelas chamadas que são realizadas de forma assíncrona para a API LS-Sched e que busca os dados do experimento executado no tutorial.
  - Observe que o método chamado la API do LS-Sched é o **get_wireless_experiment_data**, alguns outros parâmetros são passados neste método, mas isso veremos posteriormente na **seção 3**.
  - Uma vez de posse dos dados, o código presente nas linhas 10~50 é responsável por realizar a atualização do gráfico e do output da console na interface.
- A função **updateData** presente nas linhas 54~81 é a responsável por atualizar a estrutura de dados que contém os dados dos gráficos. Essa estutura de dados possui os padrões pedidos pela biblioteca do Flot Charts de modo que ela é diretamente utilizada na atualização do gráfico.
- O código das linhas 83~179 é executado assim que a página é carregada juntamente a este javascript e ele é responsável por iniciar a biblioteca do Flot Charts e configurar todo o gráfico.
- A linha 179 é a responsável por iniciar as requisições assíncronas à API LS-Sched, realizando a primeira chamada à função getData.

2.3.2 Inserção da saída da console do comando OMF exec

Os dados da saída da console do comando do OMF executado durante o tutorial é obtida através da API do LS-Sched. Para otimizar a quantidade de chamadas feitas à API para se obter os dados de saída do tutorial, é criado um único método no LS-Sched por tutorial, de modo que a chamada ao método do tutorial no LS-Sched retorna os dados coletados pelo OML, a saída do console e, caso necessário, demais dados. Deste modo, fica clara a motivação de a inserção e atualização da saída da console ser executada junta à função de atualização dos dados do gráfico, como exposto no tópico anterior no código das linhas 20~49, ou seja, caso deseje adicionar os dados da saída do console na página do tutorial, basta coloca-lo como dado na chamada do método da API LS-Sched, conforme explicado na seção 3 e adicionar os códigos HTML e javascript correspondentes na view da página:

Código HTML para a saída do console

```
<div class="box">
   <div class="box_header">
      Experiment console output
   </div>

   <div class="box_body">
      <div id="experimentOutput" style="color: white; background: black; width: calc(100% - 10px); height:
300px; overflow-y: scroll; padding: 5px;"></div>
   </div>
</div>
```

Código javascript para saída do console

```
// Update console box
try {
   var expOutput = statistics.method_result.output.replace(/\n/g, "<br />").replace("<br />", "").trim();
} catch(e) {
   var expOutput = "";
}

try {
   if(expOutput) {
      var experimentOutput = $("#experimentOutput");
      if(experimentOutput.text() != expOutput.replace(/<br \/>/g, "")) {
         var oldContent = experimentOutput.text().trim();
         var isInBottom = ((experimentOutput.scrollTop() + experimentOutput.height() + 10) >= experimentOutput
[0].scrollHeight);

         experimentOutput.html(expOutput);

         if(isInBottom || !oldContent) {
            experimentOutput.scrollTop(experimentOutput[0].scrollHeight);
         }
      }
   }
} catch(e) {}

// Enable test run when its finished.
try {
   if (expOutput.indexOf("INFO run: Experiment") >= 0) {
      $("input[type=submit]").prop("disabled", false);
      $(".hide-when-done").hide();
   }
} catch(e) {}
```

- Observe que a última parte se refere a habilitar o botão de execução do tutorial novamente quando o experimento do tutorial já tiver sido finalizado.

3. Criando um método na API do LS-Sched para buscar dados do experimento da base de dados do OML

O LS-Sched também foi desenvolvido seguindo o mesmo modelo MVC, entretanto a camada de visualização é estática para retornar apenas a saída em formato JSON. No caso dos experimentos temos uma peculiaridade também, que é uma camada de Serviços, respon?avel por executar tarefas auxiliares como chamadas de sistema durante a execução dos métodos de experimentos.

Para criar um método na API do LS-Sched com a finalidade de buscar dados a serem utilizados como saída e integrados na interface do LS-Web iremos precisar basicamente de realizar alterações nos seguintes arquivos:

- **"~/LS-Sched/Controller/ExperimentController.php"**: Classe controladora do que diz respeito aos experimentos no LS-Sched. Ela irá conter a definição do método que poderá ser utilizado pela API.
- **"~/LS-Sched/Model/ExperimentModel.php"**: Classe para manipulação de dados do que diz respeito aos experimentos, ou seja, é ela que é responsável por buscar os dados dos experimentos nas bases de dados do OML.
- **"~/LS-Sched/Service/ExperimentService.php"**: Classe de serviços auxiliares ao experimento. Nela, por exemplo, é que é realizada a leitura da saída do console do OMF para o experimento que está sendo executado.

3.1 Experiment Controller

Para criar um novo método na API LS-Sched basta criar um método público na classe ExperimentController. **O nome do método criado dentro da classe será o nome do método chamado pela API, logo se você criar um método de nome teste, ele poderá ser chamado passando o parâmetro "?method=teste" para à API**.

Como exemplo, abaixo temos o código do método para a coleta dos dados do tutorial 1: Wireless Experiment:

Código da controladora do método da API

```
/**
 * Gets the wireless experiment data.
 * @param array $parameters containing the treated call parameters.
 * @return array containing the wireless experiment data.
 */
public function getWirelessExperimentData($parameters)
{
    $beforeTimeSize = isset($parameters["before_time_size"]) ? $parameters["before_time_size"] : null;
    $nowGetTime = isset($parameters["now_get_time"]) ? $parameters["now_get_time"] : null;
    $experiment = "wireless_experiment";

    $return  = $this->Experiment->getIperfData($experiment . "_iperf", $nowGetTime, $beforeTimeSize);

    // Gets the experiment output
    $ownerExperiment = $this->Experiment->getExperimentSlice($experiment);
    if($ownerExperiment === false) {
        $return['output'] = "";
    } else {
        $return['output'] = $this->experimentService->getExperimentOutput($experiment,
            $ownerExperiment["username"], $ownerExperiment["slice"]);
    }

    return $return;
}
```

- Este método define o método da API "get_wireless_experiment_data" ou "getWirelessExperimentData" (aceita tanto padrão camel quanto underscore).
- Todos os métodos de API devem receber um parâmetro que irá conter os parâmetros de entrada da chamada.
    - O parâmetro será um array em forma de dicionário dos parâmetros passados durante a chamada.
    - Como pode ser visualizado nas linhas 8 e 9, utilizamos os parâmetros da chamada "before_time_size" e "now_get_time".
- O retorno do método deve ser convertível para um JSON, ou seja, se o retorno for um objeto, este deve implementar a interface **JsonSerializable** . Arrays, strings, booleanos e numéricos são convertíveis naturalmente.
- Na linha 12 utilizamos um método da **model** dos experimentos para buscar os dados do iperf coletados para este experimento.
- Nas linhas 14~21 utilizamos o serviço de experimentos e a model para buscar a saída da console do experimento OMF que está sendo executado durante o tutorial.

Após criar o método da API, você deverá configurar quais são os parâmetros de entrada **(caso não exista você pode pular essa parte)** que o seu método aceita e quais os tipos de dados eles devem ser. Essa configuração deve ser feita no método **getMethodParameters()** encontrado na classe ExperimentController:

Método para definição de parâmetros de entrada

```
/**
 * Gets the obligatory and optional parameters to execute the api call method.
 * @return array containing the obligatory and the optional parameters to execute the api call method.
 */
public function getMethodParameters()
{
    $params = parent::getMethodParameters();

    $obParams = [];
    $opParams = [];
    switch($this->getApiCall()) {
        case "executeExperiment":
            $obParams = [
                'username' => 'string',
                'experiment' => 'string',
                'properties' => 'array'
            ];
            break;
        case "isExperimentExecuting":
        case "getExperimentScript":
            $obParams = [
                'experiment' => 'string'
            ];
            break;
        case "getWirelessExperimentData":
        case "getTcpBitrateExperimentData":
        case "getPowerControlExperimentData":
            $obParams = [
                'before_time_size' => 'numeric'
            ];
            $opParams = [
                'now_get_time' => 'numeric'
            ];
            break;
    }

    $params["obligatory"] = array_merge($params["obligatory"], $obParams);
    $params["optional"] = array_merge($params["optional"], $opParams);

    return $params;
}
```

- Basta adicionar um "case" dentro do switch das linhas 11~35 com o nome de seu método definindo quais são os parâmetros obrigatórios ($obParams) e os parâmetros opcionais ($opParams).
    - Basta colocar dentro do array a chave contendo o nome do parâmetro e o valor contendo o tipo dele.
    - Os tipos aceitos são:
        - numeric
        - string
        - bool
        - array
        - double

Para finalizar com nosso desenvolvimento na controladora, precisamos definir o nível de acesso ao método. Por padrão, todos que possuem a key de acesso (passada via parâmetro) possuem acesso aos métodos da API, entretanto, para esses métodos de coleta dos dados para serem visualizados no portal, onde as requisições são feitas por quaisquer clientes, precisamos liberar o acesso sem a necessidade da Key. Para isto basta modificar o método **beforeFilter()** encontrado na classe ExperimentController:

Controle de acesso

```
/**
 * Method executed before execute the api call method
 * Verifies if is possible or not to execute the API call method.
 * @return bool True if the api call method can be executed or False if not.
 */
public function beforeFilter()
{
    $noAuthMethods = ['getWirelessExperimentData', 'getTcpBitrateExperimentData',
'getPowerControlExperimentData'];
    if(in_array($this->getApiCall(), $noAuthMethods)) {
        return true;
    }

    return parent::beforeFilter();
}
```

- Observe que nas linhas 8~11 já foi codificado uma forma de liberar o acesso para métodos que não precisam da key, ou seja, para você liberar seu método basta adiciona-lo no array contido na linha 8.
- Caso seja necessário, você pode programar neste método outras formas de autorização, lembrando que o "parent::beforeFilter()" possui a validação padrão que é verificar se foi passado como parâmetro a chave de acesso e se ela é válida.

3.2 Experiment Model

Apenas será necessário modificar essa classe nos seguintes caso:

- Nova aplicação instrumentada.
- Necessidade de modificar a forma de coleta dos dados do OML de alguma aplicação existente.
- Outros?

Caso você entre em algum dos casos listados acima, basta criar um método nas suas condições na classe e utiliza-lo na controladora ou na classe de serviços como desejar.

3.3 Experiment Service

Caso seja necessário realizar alguma atividade complementar durante a execução, configuração ou até mesmo coleta dos dados do experimento, como ler algum arquivo do sistema, você pode criar um método que o faça nessa classe e a partir daí utiliza-lo na controladora de modo similar ao que foi demonstrado na seção 3.1.