

Relatório: Albano Leo Ehrenbrink, albano@ehrenbrink.com, 21 98176-7800

CPS 731 – Laboratório de Internet das Coisas, professor Paulo de Figueiredo

Aula de 03 de Abril de 2019, Exercício Aula 3 (<https://sites.google.com/cos.ufrj.br/lab-iot>)

Enunciado:

Projetar um sistema cliente servidor IoT que seja capaz de controlar de forma remota o acendimento de um conjunto de LEDs de acordo com a temperatura e umidade corrente do ambiente.

Um dos LEDs deve ser aceso sempre que a temperatura lida estiver **acima** de um valor definido pelo usuário e o outro LED deve ser aceso considerando o valor corrente da umidade.

Caso tanto a temperatura quanto a umidade estiverem acima/abaixo dos valores limites, os LEDs devem começar a piscar.

O estado dos LEDs também deve ser informado na **interface da aplicação cliente**. A comunicação entre o programa cliente e o programa servidor deve ser feita **usando o protocolo CoAP**.

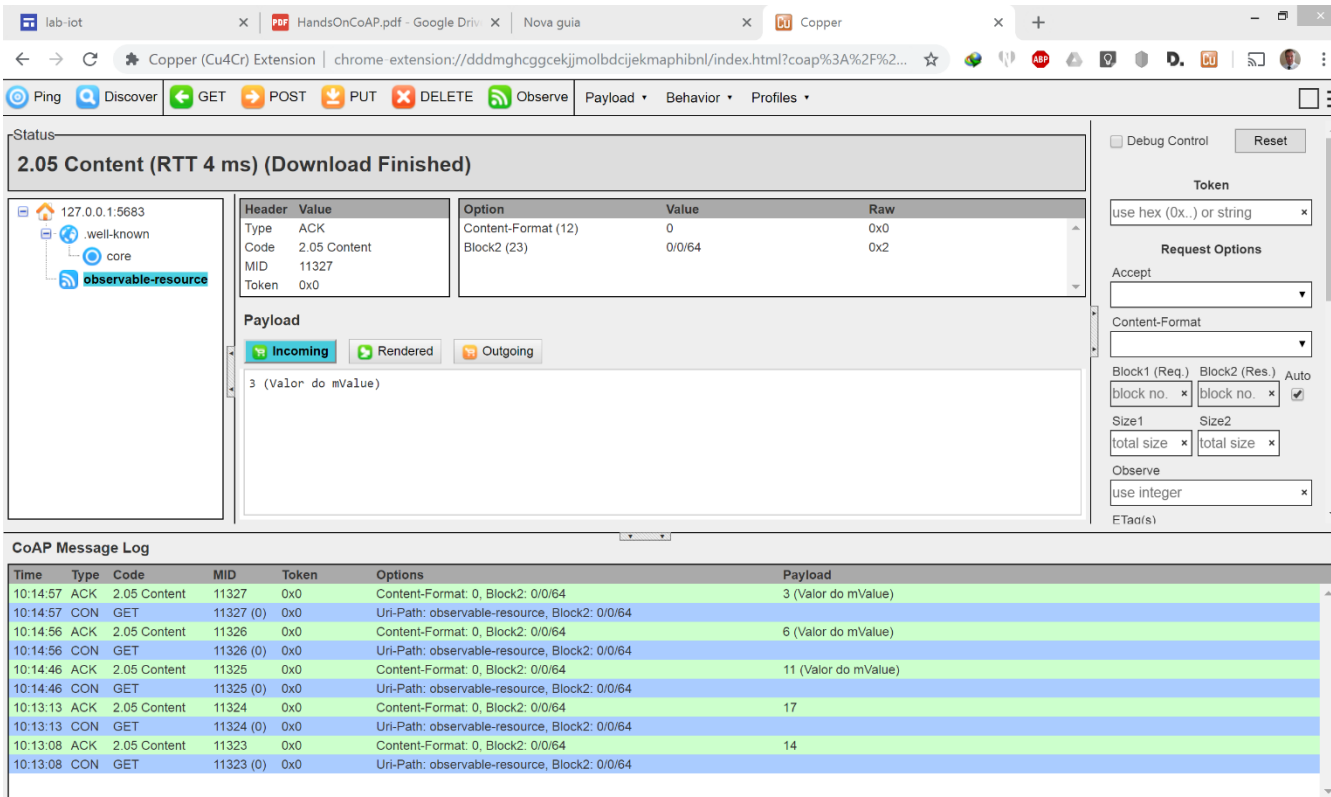
O programa servidor deve ser capaz de atender a diversos programas cliente simultaneamente.

Desenvolvimento:

Para usar o CoAP foi utilizado o Copper4Cr. Uma extensão que trabalha no Chrome e está disponível no endereço <https://github.com/mkovatsc/Copper4Cr>. Após fazer o download e descompactação do arquivo é necessário seguir o tutorial do mesmo endereço.

Para testar o CoAP em uma máquina local, foi carregado um serviço de servidor no Eclipse seguindo o tutorial disponível no endereço <https://drive.google.com/file/d/1Ydo-u80rnJB1z-ukx8tcZKfzFdmiu34n/view>. Em tempo, este tutorial utiliza o projeto disponível no link <https://drive.google.com/file/d/1UHGOofOKmzyqh1oqanae6gL9bJhwQ6VS/view> como máscara.

Após o teste obtém-se o seguinte resultado ao acessar o endereço `coap://127.0.0.1:5683/`:



The screenshot shows the Copper extension interface in a web browser. The main window displays a CoAP message log with the following entries:

Time	Type	Code	MID	Token	Options	Payload
10:14:57	ACK	2.05 Content	11327	0x0	Content-Format: 0, Block2: 0/0/64	3 (Valor do mValue)
10:14:57	CON	GET	11327 (0)	0x0	Uri-Path: observable-resource, Block2: 0/0/64	
10:14:56	ACK	2.05 Content	11326	0x0	Content-Format: 0, Block2: 0/0/64	6 (Valor do mValue)
10:14:56	CON	GET	11326 (0)	0x0	Uri-Path: observable-resource, Block2: 0/0/64	
10:14:46	ACK	2.05 Content	11325	0x0	Content-Format: 0, Block2: 0/0/64	11 (Valor do mValue)
10:14:46	CON	GET	11325 (0)	0x0	Uri-Path: observable-resource, Block2: 0/0/64	
10:13:13	ACK	2.05 Content	11324	0x0	Content-Format: 0, Block2: 0/0/64	17
10:13:13	CON	GET	11324 (0)	0x0	Uri-Path: observable-resource, Block2: 0/0/64	
10:13:08	ACK	2.05 Content	11323	0x0	Content-Format: 0, Block2: 0/0/64	14
10:13:08	CON	GET	11323 (0)	0x0	Uri-Path: observable-resource, Block2: 0/0/64	

The details panel on the right shows the message details for the selected entry (2.05 Content, RTT 4 ms, Download Finished). The header table is as follows:

Header	Value	Option	Value	Raw
Type	ACK	Content-Format (12)	0	0x0
Code	2.05 Content	Block2 (23)	0/0/64	0x2
MID	11327			
Token	0x0			

The payload is shown as "3 (Valor do mValue)".

Sugestão de errata do tutorial: O item 5.3 do tutorial criar uma classe “**ObservableResource**” mas sugere no código chamar a instancia “**ObsResource**” que deveria ter o mesmo nome da classe. Além deste, um erro de tipografia foi encontrado no último código sugerido na página 4 do tutorial que chama a classe com o “w” minúsculo, quando deveria ser maiúsculo. Sendo o correto `HelloWorldResource hello = new HelloWorldResource("hello-world");`

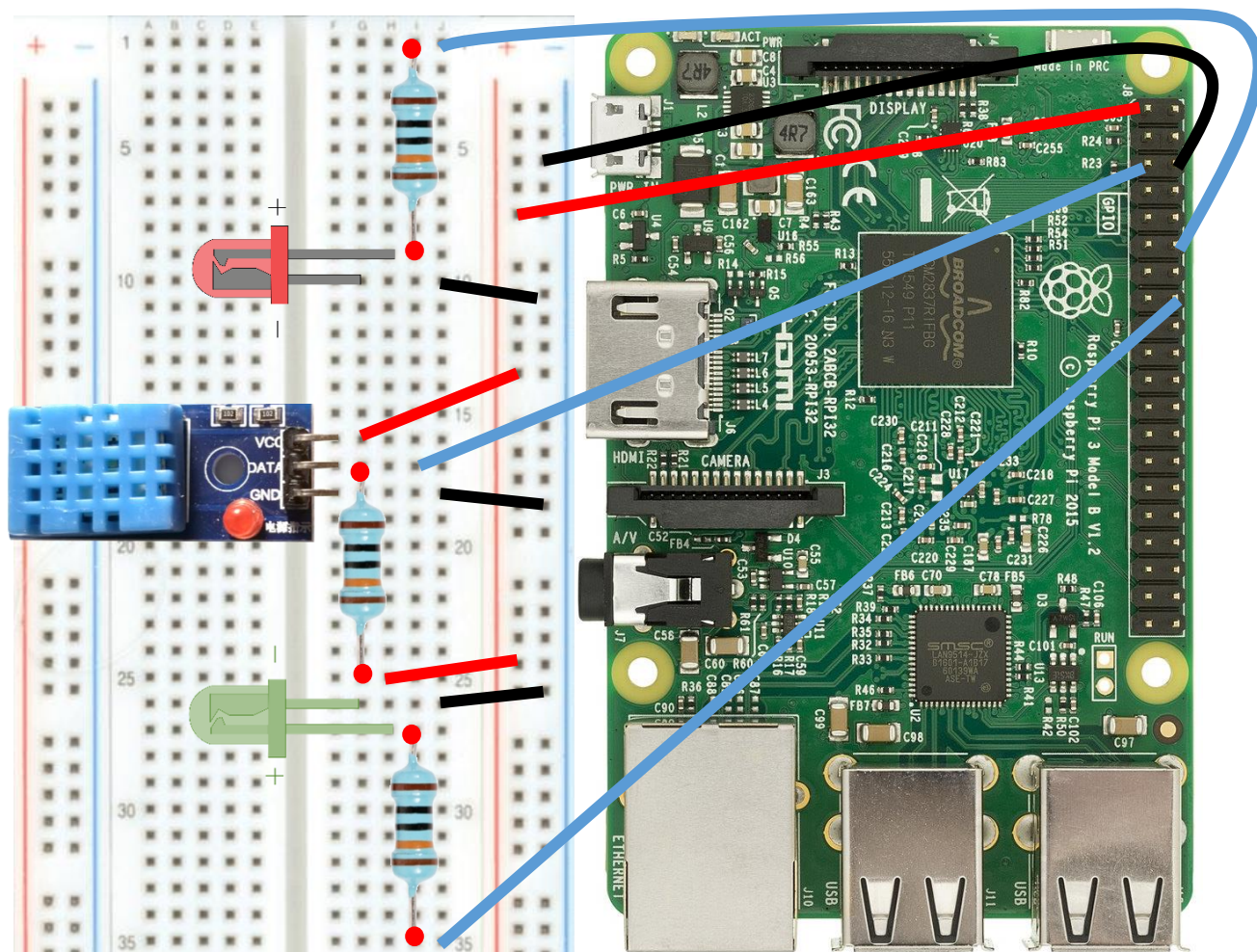
Sugestão de melhoria do tutorial: É utilizada uma task privada para gerar um número randômico baseada no relógio do computador dentro da classe ObservableResource (último código descrito no item 5.3). De forma a deixar o código estruturado, foi optado por criar uma instancia a parte para esta task.

Uma segunda melhoria sugerida é a inserção no código a seguir grifado em vermelho `public void handleGET(CoapExchange exchange) {exchange.respond(ResponseCode.CONTENT, mValue + " (Valor do mValue)", MediaTypeRegistry.TEXT_PLAIN); }` em substituição do simples `""` de forma a identificar no terminal CoAP de que se trata o valor informado.

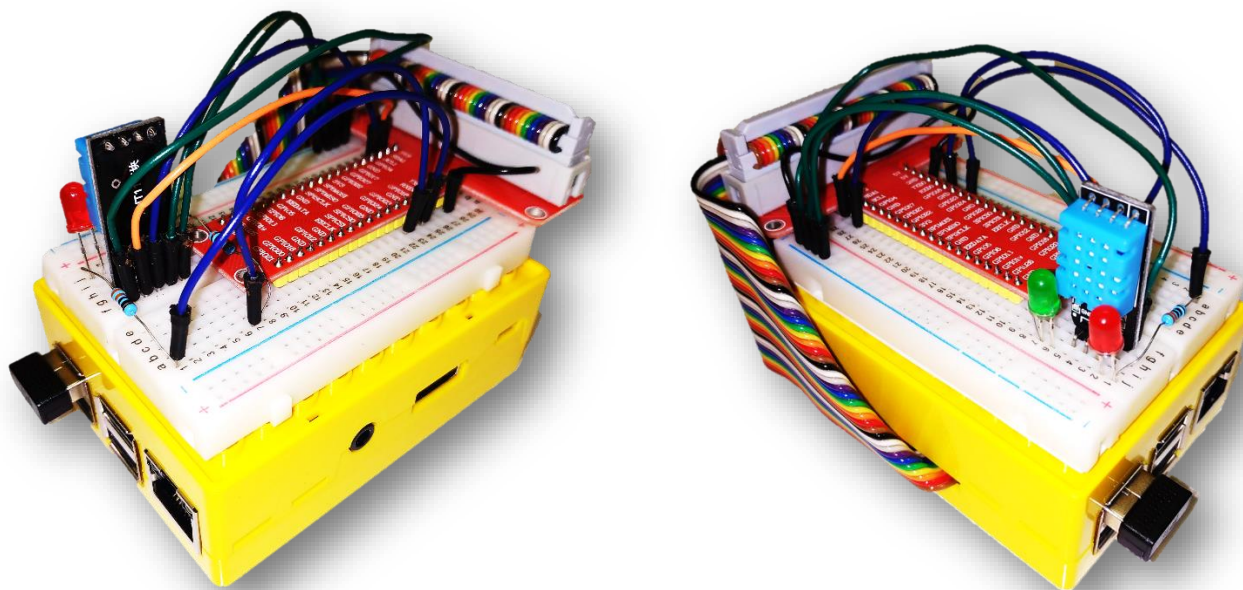
Instalando o sensor e os LEDs no Raspberry:

O esquema de ligação da placa utilizando 2 resistores de 330 Ohms, 1 resistor de 10k Ohm (pull up do sinal do sensor), 1 LED vermelho, 1 LED verde e um sensor DHT11. O esquema para ligação dos LEDs e do sensor seguem dois exemplos distintos encontrados em <https://pi4j.com/1.2/example/control.html> para o LED e

<https://tutorials-raspberrypi.com/raspberry-pi-measure-humidity-temperature-dht11-dht22/> para o sensor que juntos ficam como no seguinte diagrama.



Uma foto da instalação é mostrada a seguir.



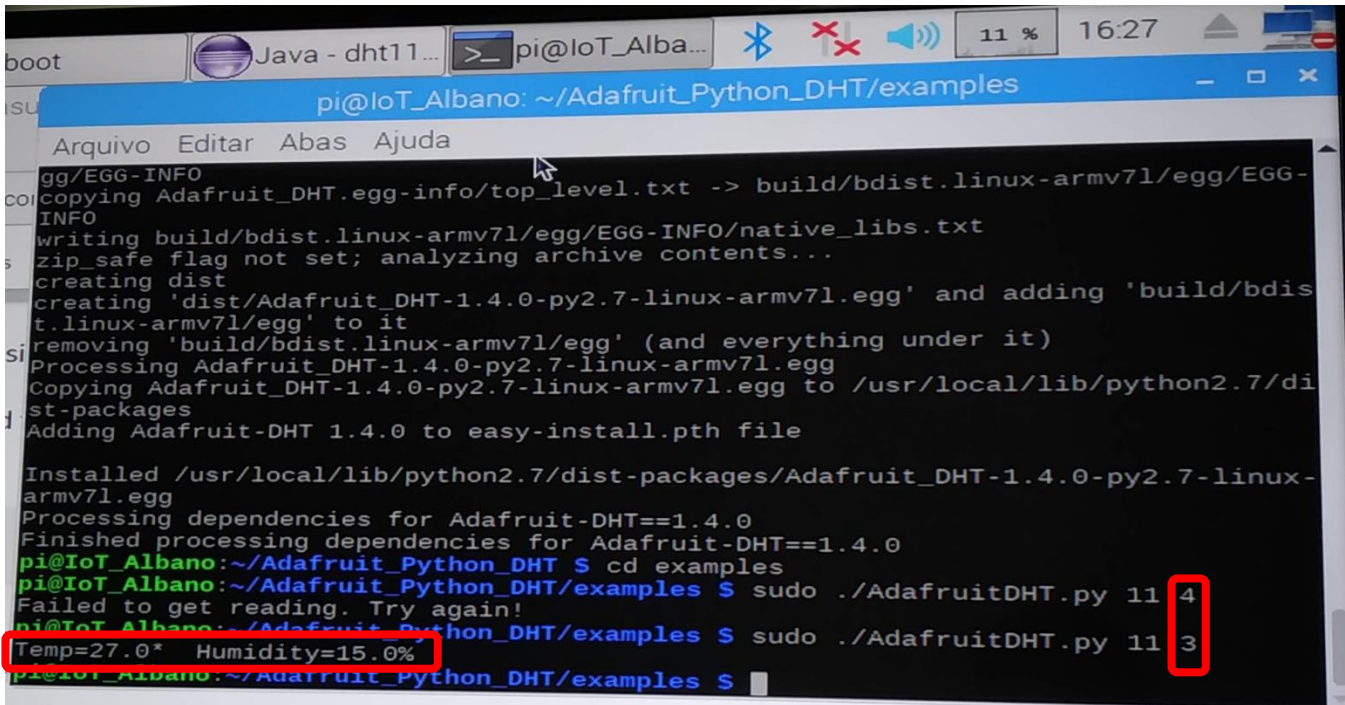
Para testar o funcionamento do sensor DHT11 foi utilizado o mesmo tutorial já citado que descreve o diagrama elétrico do sensor disponível no link <https://tutorials-raspberrypi.com/raspberry-pi-measure-humidity-temperature-dht11-dht22/>. Note que os pinos utilizados no esquema elétrico acima diferem um pouco do código do tutorial sendo:

DHT11 = pino 5 (Python GPIO_03). Nota: No Pi4J a mesma porta física é o GPIO_08

LED verde = pino 16 (Pi4J GPIO_04). Nota: No Python a mesma porta física é o GPIO_23

LED vermelho = pino 12 (Pi4J GPIO_01). Nota: No Python a mesma porta física é o GPIO_18

Portando o código correto é “sudo ./AdafruitDHT.py 11 **3**” porque usa o GPIO_03 para o sensor e não o 4 como no exemplo. Obtendo o resultado abaixo.



```
boot Java - dht11... pi@IoT_Alba... 11 % 16:27
pi@IoT_Alba: ~/Adafruit_Python_DHT/examples
Arquivo Editar Abas Ajuda
gg/EGG-INFO
copying Adafruit_DHT.egg-info/top_level.txt -> build/bdist.linux-armv7l/egg/EGG-
INFO
writing build/bdist.linux-armv7l/egg/EGG-INFO/native_libs.txt
zip_safe flag not set; analyzing archive contents...
creating dist
creating 'dist/Adafruit_DHT-1.4.0-py2.7-linux-armv7l.egg' and adding 'build/bdis
t.linux-armv7l/egg' to it
removing 'build/bdist.linux-armv7l/egg' (and everything under it)
Processing Adafruit_DHT-1.4.0-py2.7-linux-armv7l.egg
Copying Adafruit_DHT-1.4.0-py2.7-linux-armv7l.egg to /usr/local/lib/python2.7/di
st-packages
Adding Adafruit-DHT 1.4.0 to easy-install.pth file

Installed /usr/local/lib/python2.7/dist-packages/Adafruit_DHT-1.4.0-py2.7-linux-
armv7l.egg
Processing dependencies for Adafruit-DHT==1.4.0
Finished processing dependencies for Adafruit-DHT==1.4.0
pi@IoT_Alba:~/Adafruit_Python_DHT $ cd examples
pi@IoT_Alba:~/Adafruit_Python_DHT/examples $ sudo ./AdafruitDHT.py 11 4
Failed to get reading. Try again!
pi@IoT_Alba:~/Adafruit_Python_DHT/examples $ sudo ./AdafruitDHT.py 11 3
Temp=27.0* Humidity=15.0%
pi@IoT_Alba:~/Adafruit_Python_DHT/examples $
```

NOTA: Após o teste e registro acima foi verificado que o tutorial de aula usava o GPIO_04 e não o 3 conforme mostrado no esquemático, na montagem e no registro do teste. Esta discrepância não interfere no resultado sendo compensada apenas com a substituição do pino conforme relatado no parágrafo anterior.

Implementação do serviço remoto IoT

Comunicação

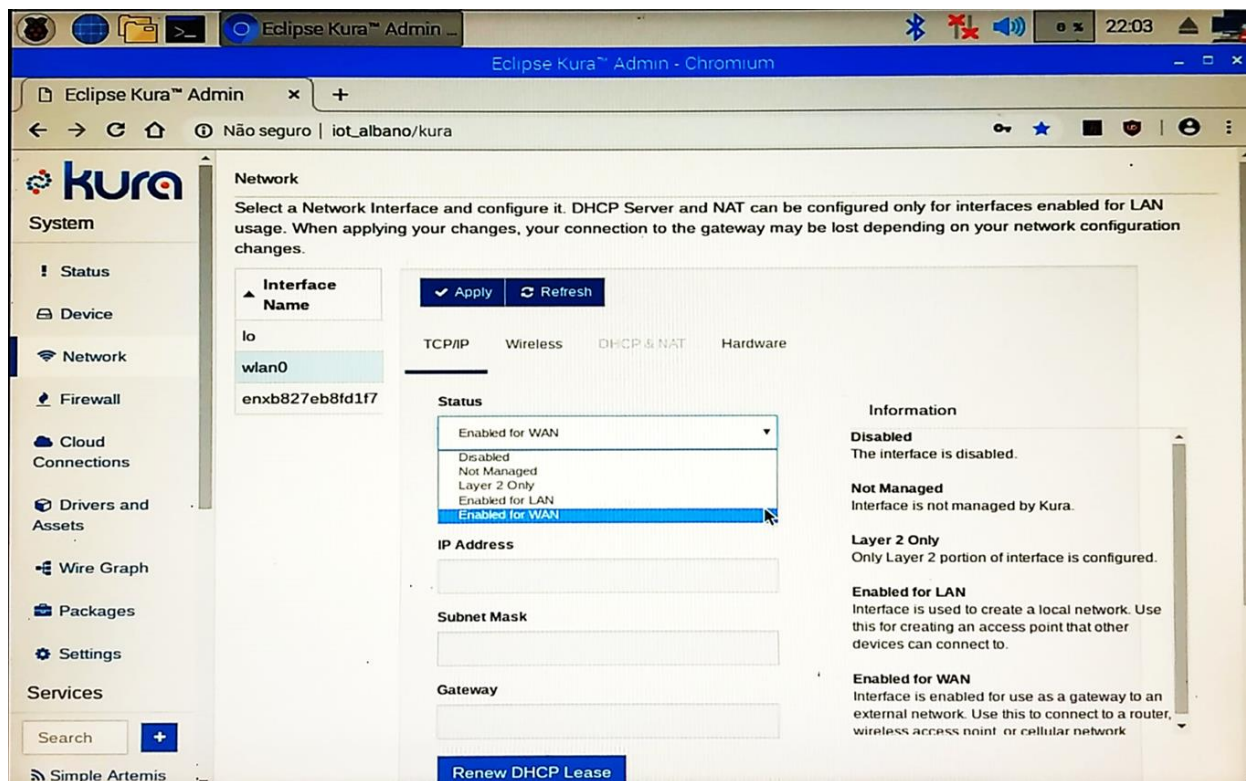
O gerenciamento da conexão no servidor que roda a aplicação no Raspberry Pi é feito pelo serviço Kura descrito no tutorial disponível no link <https://iot.eclipse.org/java/tutorial/>.

ADVERTÊNCIA: O tutorial acima indica a instalação do Kura versão 1.3, lançada em 2015, para o Raspberry Pi 2. Quando instalado no Raspberry Pi 3, lançado em 2016 (A+ ou B+ em 2018), ele não permite o acesso a página de configuração e acaba por eliminar a conexão WiFi e cabeada do Raspberry. Para instalar a versão mais atual do Kura ou mesmo a versão 1.3 para o Raspberry Pi 3, que foi portada posteriormente ao lançamento de 2015, visite o link <https://www.eclipse.org/kura/downloads.php>.

O serviço Kura gerencia as conexões do Raspberry Pi e por padrão torna o Raspberry Pi um Access Point. Neste modo, o Raspberry Pi não depende de uma conexão direta com a internet através de roteador por exemplo.

Outra característica importante é que ele é um container que possui diversos serviços declarativos e configuráveis do OSGi facilitando a aquisição de informações por meio de um serviço de Gateway e protocolo MQTT que gerencia o paradigma de publicadores e inscritos.

Ao instalar a versão correta, é possível acessar a página de configuração pelo navegador do Raspberry. Conforme demonstra a tela a seguir.

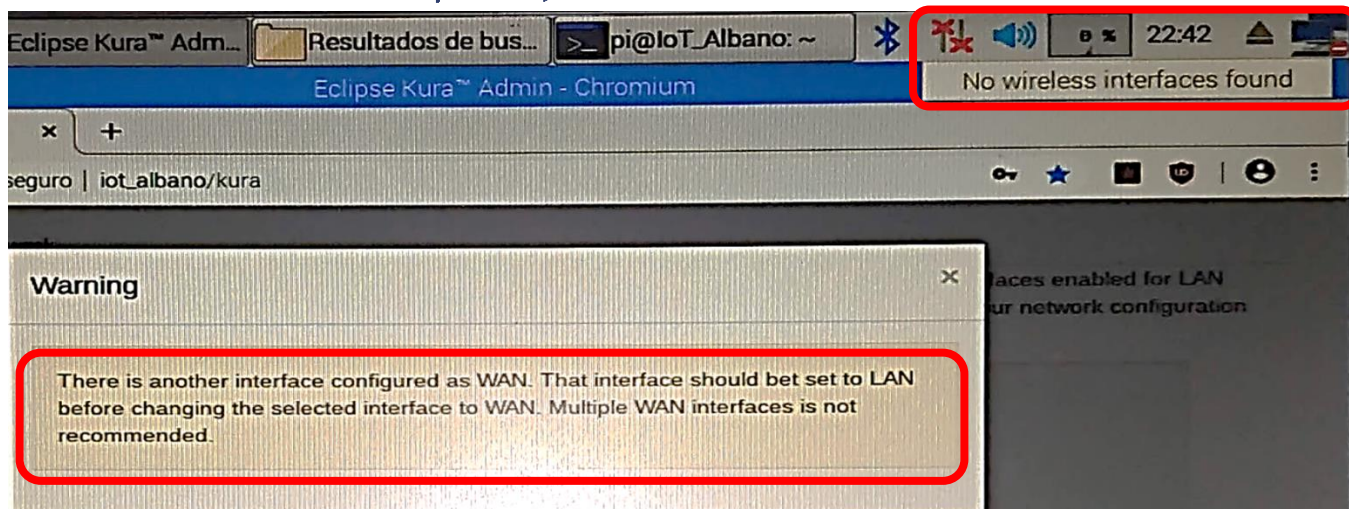


A página de configuração aceita o endereço 0.0.0.0 ou o nome da máquina conforme mostrado acima e o login e senha padrão são admin.

Para continuar a usar e configurar o Raspberry em sua rede e permitir a conexão a Internet por meio de um Roteador clique no serviço “Network” presente no lado esquerdo, clique na “Interface Name” wlan0, selecione o status “Enable for WAN” na guia TCP/IP e clique em “Apply”, após isso clique na guia Wireless, clique na lupa logo abaixo do campo “Network Name” para procurar redes disponíveis e selecione a rede de sua preferência. Digite a senha no campo “Wireless Password” e clique no botão logo abaixo deste campo para confirmar que a senha esteja correta. Na sequência clique novamente em “Apply”.

Além desta configuração, é necessário liberar a porta 5863 que CoAP / Copper usam. Vá até ao serviço “Firewall” no lado esquerdo, clique no botão “New”, digite 5863 no campo “Porto r Port Range*” e clique no botão “Submit”.

Se sua rede é cabeada, poderá fazer a configuração acima na “Interface Name” lo. Mas somente uma das duas interfaces poderão ser WAN conforme mostra o aviso a seguir.



Note também que ao clicar sobre o ícone de rede WiFi o gerenciador nativo de redes do Raspberry indica que não existem interfaces sem fio uma vez que agora ela é gerenciada pelo Kura.

NOTA: As configurações acima seriam implementadas de forma semi automática com o script descrito no tutorial apontado para o endereço <http://iot.eclipse.org/java/demo/org.eclipse.greenhouse.iot-0.1.0.dp> mas este infelizmente não está mais disponível, mesmo nos sites de Archive da Internet ou no Marketplace do Eclipse/Kura.

Com dito o serviço Kura é importante ao dar suporte ao protocolo MQTT que possui um payload menor que o HTTP e transforma o Raspberry Pi em um Broker de conexão atendendo diversos clientes simultâneos.

Topologia



Para implementação do sensor como dispositivo IoT proposto no exercício, foi definido a topologia acima. Nela o Raspberry Pi Recebe os valores dos limites do cliente CoAP disponível no notebook compara com os valores provenientes do sensor DHT 11 e aciona os LEDs vermelho e verde seguindo uma lógica descrita na próxima seção.



Além destas tarefas o Raspberry também disponibiliza as informações lidas nos sensores e o estado dos LEDs ao cliente CoAP do notebook.

Além da possibilidade de usar um cliente CoAP como o Copper descrito no tutorial de sala de aula, foi desenvolvido um cliente dentro do código fonte que pode ser executado diretamente como aplicação java no console do Eclipse.

Tanto a utilização do Copper como o cliente Java precisam do IP onde em que se encontra o RPi 3 que pode ser facilmente obtido pelo comando ifconfig no prompt de comando do RPi.

Lógica do serviço Implementada

Como dito no enunciado do exercício dois estados dos LEDs são desejados:

	Ligado	Piscando
 Temperatura	Dentro dos limites	Fora dos limites
 Umidade	Dentro dos limites	Fora dos limites

Para tal o cliente deve informar os mínimos e máximos desejados para umidade e temperatura. O estado com LEDs desligados é obtido na inicialização do serviço quando ainda não foi informado os limites.

Instalação

O código fonte, o código compilado, a biblioteca em Python do sensor DHT11 e este relatório podem ser obtidos no GitHub endereço https://github.com/AlbanoL/CoAP_DHT11.

Para fazê-lo funcionar, você precisa colocar o código compilado IoT_Albano.jar e a biblioteca dht2.py em um mesmo diretório do RPi.

Abra a janela de comando, vá até o diretório onde se encontram os arquivos e digite “ifconfig” para descobrir o IP que será usado mais a frente.

Na sequência, para iniciar o serviço, digite o comando “java -jar IoT_Albano.jar” na janela de comando dentro do diretório onde se encontram os arquivos descritos no parágrafo anterior. O resultado é esperado é mostrado na figura abaixo onde os comandos e resposta do IP estão destacados por caixas vermelhas.



PESC

Programa de Engenharia
de Sistemas e Computação

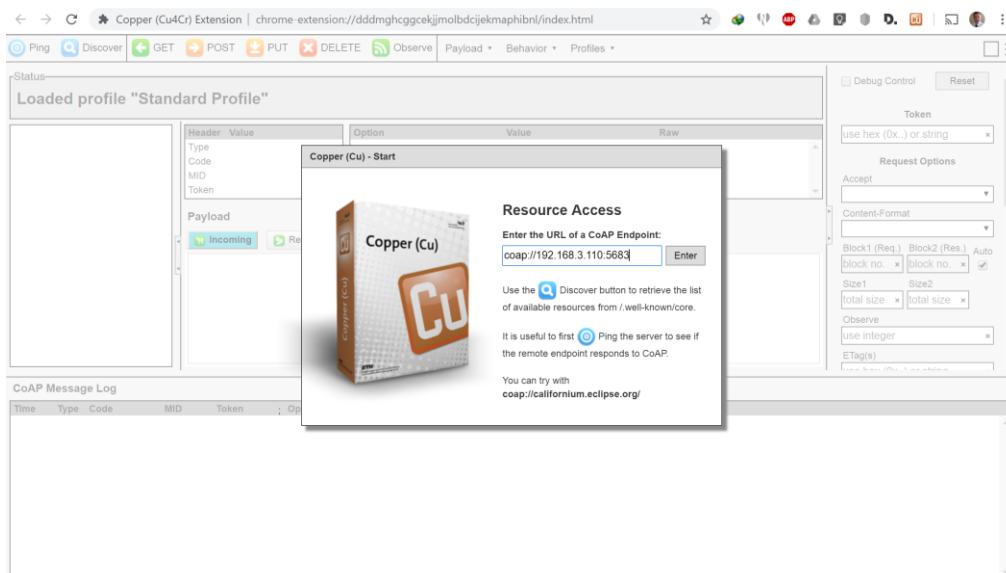
```
pi@IoT_Alban: ~/workspace
Arquivo Editar Abas Ajuda
pi@IoT_Alban:~/workspace $ ifconfig
enxb827eb8fd1f7: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    ether b8:27:eb:8f:d1:f7 txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    loop txqueuelen 1000 (Loopback Local)
    RX packets 9 bytes 387 (387.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 9 bytes 387 (387.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

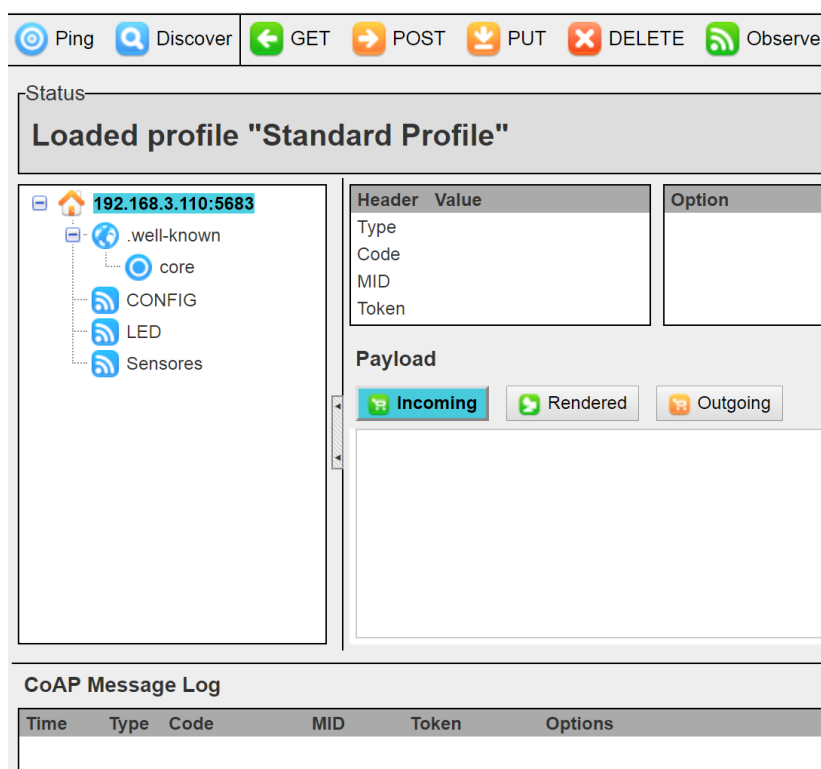
wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.3.110 netmask 255.255.255.0 broadcast 192.168.3.255
    ether b8:27:eb:da:84:a2 txqueuelen 1000 (Ethernet)
    RX packets 3534 bytes 757467 (739.7 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 76 bytes 15646 (15.2 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

pi@IoT_Alban:~/workspace $ java -jar IoT_Alban.jar
mai 20, 2019 9:10:08 PM org.eclipse.californium.core.network.config.NetworkConfig createStandardWithFile
INFORMAÇÕES: Loading standard properties from file Californium.properties
ENTROU no StatusLed CONSTRUTOR
mai 20, 2019 9:10:09 PM org.eclipse.californium.core.CoapServer start
INFORMAÇÕES: Starting server
mai 20, 2019 9:10:09 PM org.eclipse.californium.core.CoapServer start
INFORMAÇÕES: No endpoints have been defined for server, setting up default endpoint at port 5683
mai 20, 2019 9:10:09 PM org.eclipse.californium.core.network.CoAPEndpoint start
INFORMAÇÕES: Starting Endpoint bound to 0.0.0.0/0.0.0.0:5683
mai 20, 2019 9:10:09 PM org.eclipse.californium.core.network.EndpointManager setDefaultEndpoint
INFORMAÇÕES: Started new default endpoint 0.0.0.0/0.0.0.0:5683
```

Como dito, o cliente pode ser usado o Copper por meio de um plugin dentro de algum navegador no computador do usuário onde sua instalação foi comentada na seção “Desenvolvimento” deste relatório. A tela de entrada é mostrada na figura abaixo já com o IP e a porta 5683 inseridas.



Ao clicar entrar aparecerá os recursos disponíveis no serviço rodando no RPi.



O recurso Sensores pode ser usado com o GET ou o OBSERVE para obter a leitura do sensor.

Payload

Incoming Rendered Outgoing

Temperature: 27°C Humidity:15%

Como os limites ainda não foram configurados, os LEDs devem estar desligados. Além da visualização física dos LEDs, onde o significado é obtido pela tabela descrita seção “Lógica do serviço implementado” seu status pode ser consultado no recurso LED com o comando GET. Onde podem ser obtidas estas 3 respostas dependendo dos limites configurados.

Incoming Rendered Outgoing

RED: n/a GREEN: n/a

Incoming Rendered Outgoing

RED: temp in GREEN: humid in

Incoming Rendered Outgoing

RED: temp out GREEN: humid out

O recurso CONFIG é usado com PUT ou POST para configurar os limites. Para publicar os limites é utilizada a string “{“TempSup”:30,“TempInf”:10,“HumSup”:30,“HumInf”:10}” no campo “Outgoing” do Recurso “Config”.

Ping Discover GET **POST** PUT DELETE Observe Payload ▾

Status

Loaded profile "Standard Profile"

192.168.3.110:5683

- .well-known
- core
- CONFIG**
- LED
- Sensores

Header	Value	Option	Value
Type			
Code			
MID			
Token			

Payload

Incoming Rendered **Outgoing**

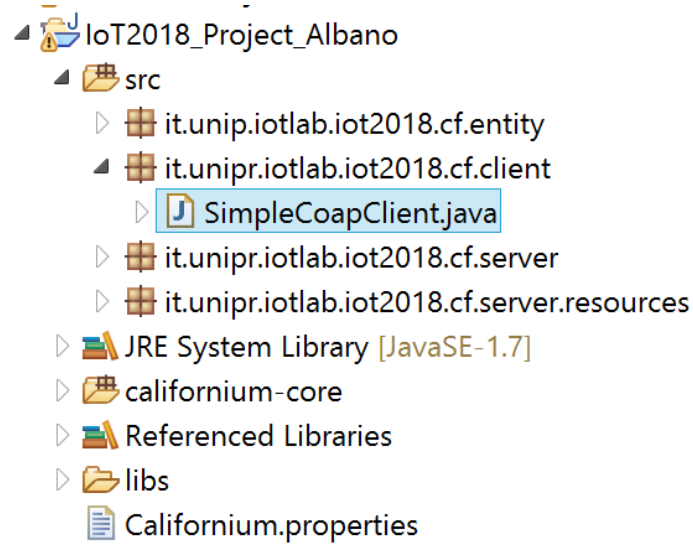
{ "TempSup":12, "TempInf":10, "HumSup":12, "HumInf":10 }

Outra maneira de usar o serviço é pelo cliente desenvolvido em Java e disponível dentro do código fonte. Para executá-lo clique duas vezes em “SimpleCoapClient.java” como mostra a figura a seguir.




PESC

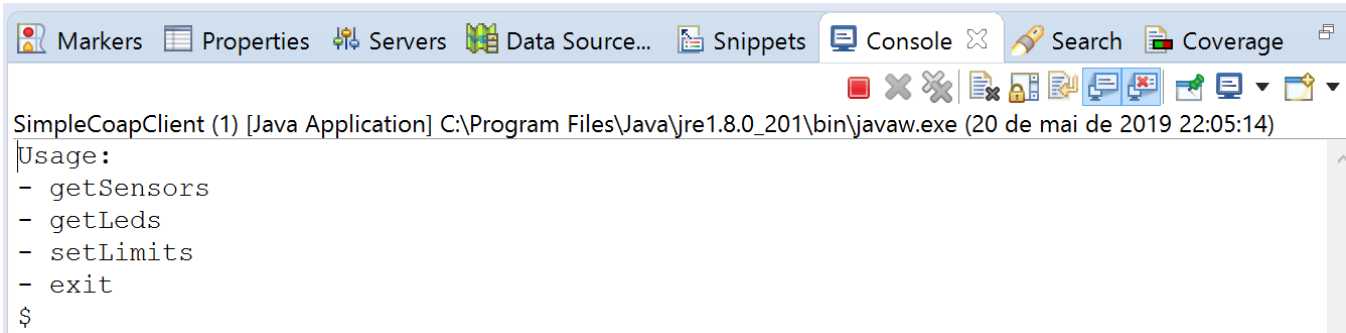
Programa de Engenharia
de Sistemas e Computação



Dentro do código modifique o IP para o utilizado pelo RPi onde mostra a caixa vermelha da próxima figura.

```
1 package it.unipr.iotlab.iot2018.cf.client;
2
3 import java.io.BufferedReader;
4
5
6
7
8
9
10
11
12
13
14 public class SimpleCoapClient {
15
16     private String baseUrl = "coap://192.168.3.110:5683/%s";
17
18     public static void main(String[] args) throws IOException {
19         // Create the client
20         SimpleCoapClient instance = new SimpleCoapClient();
21
22         // Create the reader
23         BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
24
25         // Show usage
26         System.out.println("Usage: ");
27         System.out.println("- getSensors");
28         System.out.println("- getLeds");
29         System.out.println("- setLimits");
30         System.out.println("- exit");
31
32         // Read the cmd
33         String cmd = "";
34         while (!cmd.equals("exit")) {
35             System.out.print("$ ");
36             cmd = reader.readLine();
37             if (cmd.equals("getSensors")) {
```

Para executar este código como uma aplicação Java diretamente na tela de console do Eclipse basta clicar no botão  que o console parecerá abaixo da seguinte forma:



```
SimpleCoapClient (1) [Java Application] C:\Program Files\Java\jre1.8.0_201\bin\javaw.exe (20 de mai de 2019 22:05:14)
Usage:
- getSensors
- getLeds
- setLimits
- exit
$
```

Digite “getSensors” para saber a leitura. Aviso: O Java é diferencia letras maiúsculas.

```
$ getSensors
Reading sensors:
mai 20, 2019 10:07:35 PM org.eclipse.californium.core.network.config.NetworkConfig createStandardWithFile
INFORMAÇÕES: Loading standard properties from file Californium.properties
mai 20, 2019 10:07:35 PM org.eclipse.californium.core.network.CoAPEndpoint start
INFORMAÇÕES: Starting Endpoint bound to 0.0.0.0/0.0.0.0:0
mai 20, 2019 10:07:35 PM org.eclipse.californium.core.network.EndpointManager createDefaultEndpoint
INFORMAÇÕES: Created implicit default endpoint 0.0.0.0/0.0.0.0:58037
Temperature: 28°C Humidity:15%
```

Ao se usar o comando “getLeds” temos a resposta (para os LEDs desligados).

```
$ getLeds
Reading leds:
RED: n/a GREEN: n/a
```

O comando “setLimits” perguntará os 4 valores de limites necessários e enviará a sentença correta descrita na utilização do Copper.

```
$ setLimits
Enter the temperature [min]: 10
Enter the temperature [max]: 30
Enter the humidity [min]: 10
Enter the humidity [max]: 30
Setting limits:
{"TempSup":30,"TempInf":10,"HumSup":30,"HumInf":10}
```