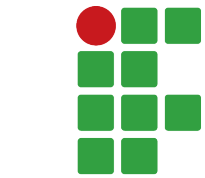


INSTITUTO
FEDERAL
Santa Catarina

# Relatório 2

**Protocolo de aplicação**

**Maria Fernanda Silva Tutui**
**Paulo Fylippe Sell**

Junho de 2019

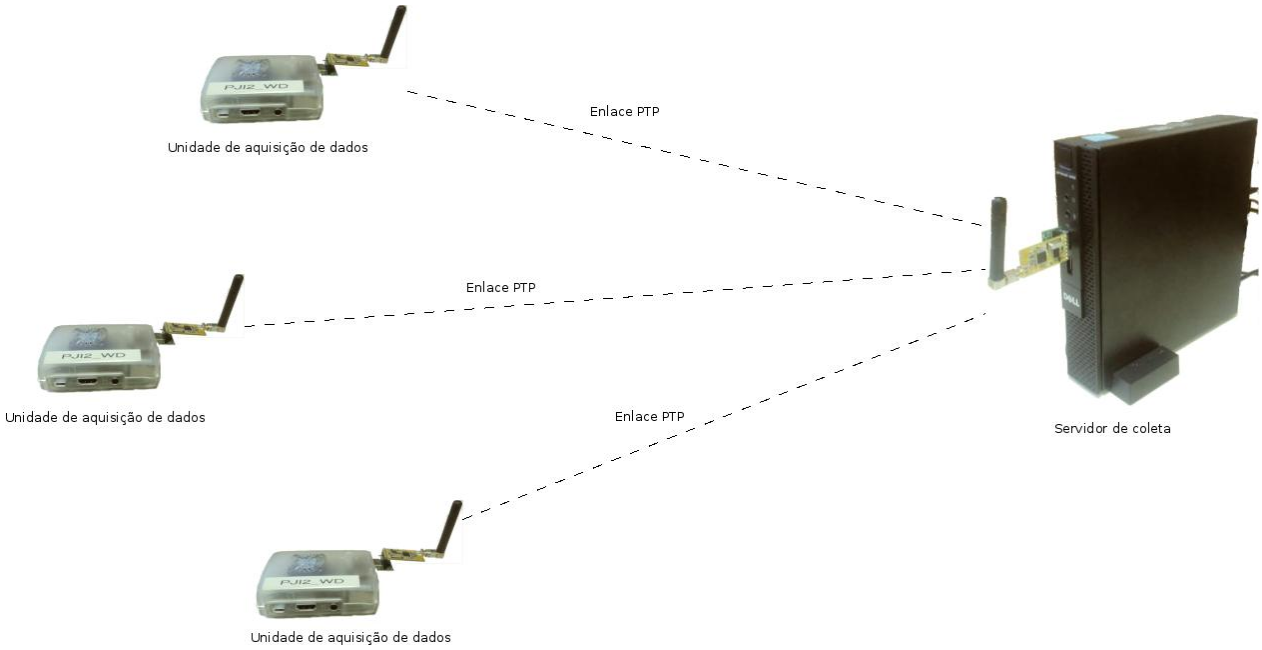
# Sumário

# 1 Introdução

O seguinte relatório apresenta o desenvolvimento do segundo projeto feito no decorrer da disciplina de Projeto de Protocolos, ministrada pelo professor Marcelo Sobral durante o primeiro semestre de 2019.

O protocolo foi implementado a partir da necessidade do envio de mensagens de pequenos *bytes*, com o foco em Internet das Coisas. Seu uso pode ser justificado a partir de um cenário como o da figura 1, onde vários dispositivos necessitam enviar dados de sensores para um servidor de coleta.

Figura 1: Sistema de aquisição de dados remotos.




O protocolo *CoAP* foi projetado para uso em situações como o da figura acima, onde nodos (unidades de aquisição de dados) com características de baixo consumo energético, baixa taxa de transmissão e sem necessidade de confiabilidade na transmissão dos dados são necessários (IETF, 2014).

## 1.1 Objetivos do projeto

- Especificar um protocolo de aplicação, que deve estar fundado no protocolo *CoAP*. Esse protocolo de aplicação deve ser capaz de:
  - Configurar a unidade de aquisição de dados
  - Transmitir mensagens contendo dados monitorados
- Implementar um cliente *CoAP* para o envio das mensagens de aquisição de dados
- Usar uma técnica de codificação de mensagens que possibilite o intercâmbio de dados entre sistemas heterogêneos

# 2 Desenvolvimento

## 2.1 Serviço oferecido

O protocolo *CoAP* funciona a partir do consumo de *URI's* existentes no lado servidor da aplicação. Desta forma, o lado cliente prove funções que implementem o consumo dos métodos do protocolo: GET, PUT, POST E DELETE. No cenário apresentado, o servidor de testes implementa apenas os métodos GET e POST. Logo, no lado cliente foram desenvolvidos métodos para consumo apenas destes recursos. Desta forma, nosso cliente *CoAP* oferece a possibilidade de o usuário consumir os métodos GET e POST de um servidor *CoAP* e tratar as respostas recebidas de forma isolada, isto é, cada requisição tem uma resposta única. O *payload* da mensagem deve estar codificado a partir da especificação apresentada na próxima sessão, com o auxílio do mecanismo *Protocol Buffers*.

## 2.2 Especificação das mensagens do *payload*

O *payload* da mensagem a ser trocada deve estar codificado a partir da especificação ditada pelo *Protocol Buffers*. O uso do *Protocol Buffers* foi feito afim de que a implementação do cliente *CoAP* pudesse ser feita em qualquer linguagem de programação, desde que o *payload* das mensagens continuasse respeitando a especificação abaixo.

```
message Mensagem {
  required string placa = 1;
  oneof msg {
    Config config = 2;
    Dados dados = 3;
  }
}
```

Como é possível verificar, dentro da mensagem genérica *"Mensagem"*, pode existir uma outra mensagem especificada pelo *Protocol Buffers*: mensagem *"Config"* ou mensagem *"Dados"*.

Estas mensagens, por sua vez, são regidas através da seguinte especificação:

```
message Config {
  // periodo dado em milissegundos
  required int32 periodo = 1;
  // lista de nomes de sensores
  repeated string sensores = 2;
}

message Dados {
  // lista de sensores com seus valores amostrados e timestamp
  repeated Sensor amostras = 1;
}
```

Onde, por fim, a mensagem *"Sensor"* se dá através da especificação:

```
19 message Sensor {
20   // nome do sensor
21   required string nome = 1;
22   // valor amostrado do sensor
23   required int32 valor = 2;
24   // timestamp da amostragem, em milissegundos desde 1/1/2019 0:0:0
25   optional int32 timestamp = 3;
26 }
```

## 2.3 Formato das mensagens

A tabela 1 demonstra o quadro do protocolo *CoAP*. Onde nele deve-se especificar a versão do protocolo, o tipo da mensagem (confirmável, não confirmável ou *ack*), tamanho do *token* (número que identifica uma requisição), código da requisição (GET, POST, PUT ou DELETE) e a identificação da mensagem. Deve-se também indicar o *token* e alguns parâmetros da requisição no campo opções.

Para nosso cenário, casa haja um *payload* na mensagem, o mesmo deve estar codificado conforme a sessão anterior.

Tabela 1: Quadro do protocolo CoAP

<table>
<tr><td>versão</td><td>tipo</td><td>tamanho token</td><td>código</td><td>mensagem id</td></tr>
<tr><td colspan="5">token (se tiver)</td></tr>
<tr><td colspan="5">opções (se tiver)</td></tr>
<tr><td colspan="3">11111111</td><td colspan="2">payload (se tiver)</td></tr>
</table>

Onde:

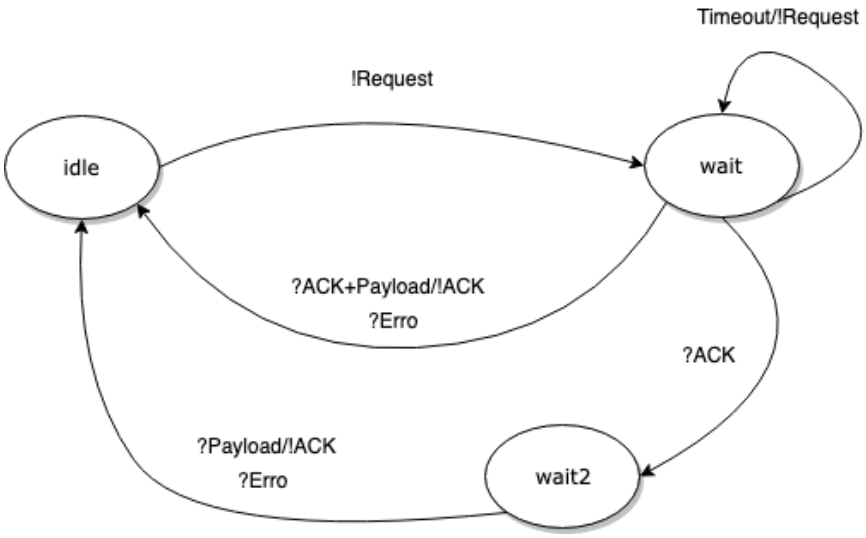
Tabela 2: Número de bits em cada campo - CoAP

| | **Número de bits** |
|---|---|
| **Versão** | 2 |
| **Tipo** | 2 |
| **Tamanho token** | 4 |
| **Código** | 8 |
| **Mensagem ID** | 16 |

## 2.4 Comportamento do lado cliente do protocolo *CoAP*

O lado cliente do protocolo foi desenvolvido conforme as especificações adotadas pelo *CoAP* juntamente com as necessidades expostas pelo professor. Dessa maneira, uma máquina de estados finita foi modelada para a compreensão do comportamento do protocolo. A máquina de estados pode ser observada na figura abaixo.

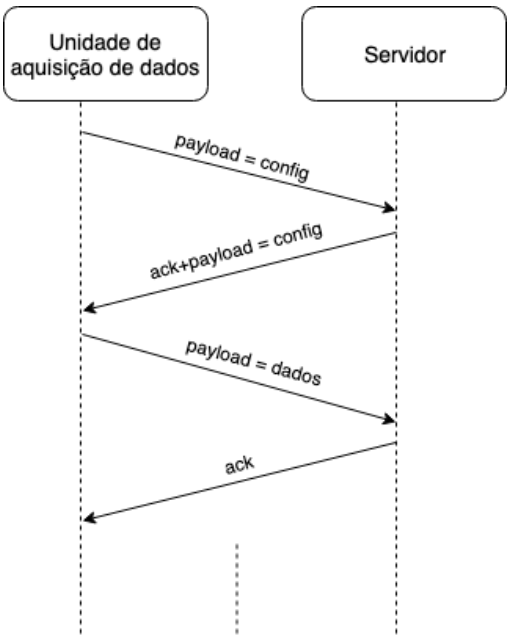
Figura 2: Máquina de estados finito.




O cliente realizará uma requisição para o servidor (GET, POST, PUT ou DELETE). Caso a resposta seja apenas um *ACK*, o cliente vai para o segundo estado de espera. Sempre que o cliente recebe um *ACK* juntamente com um *payload*, o mesmo volta ao estado inicial.

Em cima desta máquina de estados é possivel modelar a troca de mensagens (figura 3) do protocolo da aplicação. O cliente envia uma mensagem onde em seu *payload* está encapsulada a configuração da placa de unidade de dados. O servidor por sua vez irá responder enviando a configuração desejada. Uma vez configurada, a unidade de aquisição de dados pode então enviar mensagens de dados no tempo definido pelo servidor.

Figura 3: Troca de mensagens entre UAD e Servidor.



# 3 API coap

O cliente *CoAP* foi desenvolvido na forma da uma API (*Application Programming Interface*), onde o usuário pode realizar os métodos GET e POST do protocolo *CoAP*. O usuário deve se preocupar em montar o *payload* (no método POST) de forma que o servidor aceite a requisição, com base na sessão 2.2 deste relatório. Abaixo um exemplo de uso do método *do_post()* desta API.

```
27 from coapc import coap
28 from response import Response
29
30 if __name__ == '__main__':
31     r = coap.('::1').do_post(coap.CON, payload, 'ptc')
```

O construtor da classe *coap* recebe como parâmetro o IP do servidor o qual a aplicação irá trocar dados (deve ser IPv6). Os métodos *do_get()* e *do_post()* recebem o tipo da mensagem (*coap.CON*, *coap.NON* ou *coap.ACK*) e uma lista de *URI's*. Além disso, o método *do_post()* recebe também o *payload* da mensagem, codificado de acordo com a sessão 2.2 deste relatório.

```
33 class coap(poller.Calback):
34
35  def __init__(self, ip):
36     pass
37
38  def do_get(self, type, *uris):
39     pass
40
41  def do_post(self, type, payload, *uris):
42     pass
```

Criou-se também uma classe para obtenção das respostas do servidor, chamada *Response*.

```
43 class Response():
44
45     def __init__(self, type, tkllen, code, mid, token, payload):
46         pass
47
48     def getPayload(self):
49         pass
50
51     def getCode(self):
52         pass
53
54     def getMid(self):
55         pass
56
57     def getToken(self):
58         pass
59
60     def getType(self):
61         pass
```

Os métodos *do_get()* e *do_post()* da classe *coap* retornam objetos da classe *Response*. Definiu-se que, caso ocorra um *timeout* durante durante uma requisição ao servidor, o atributo *payload* da classe *Response* irá retornar o valor **-1**. Além disso, caso haja má formação *payload* por parte do usuário, o atributo *payload* da classe *Response* irá retornar o valor **-2**.

# 4 Demonstração

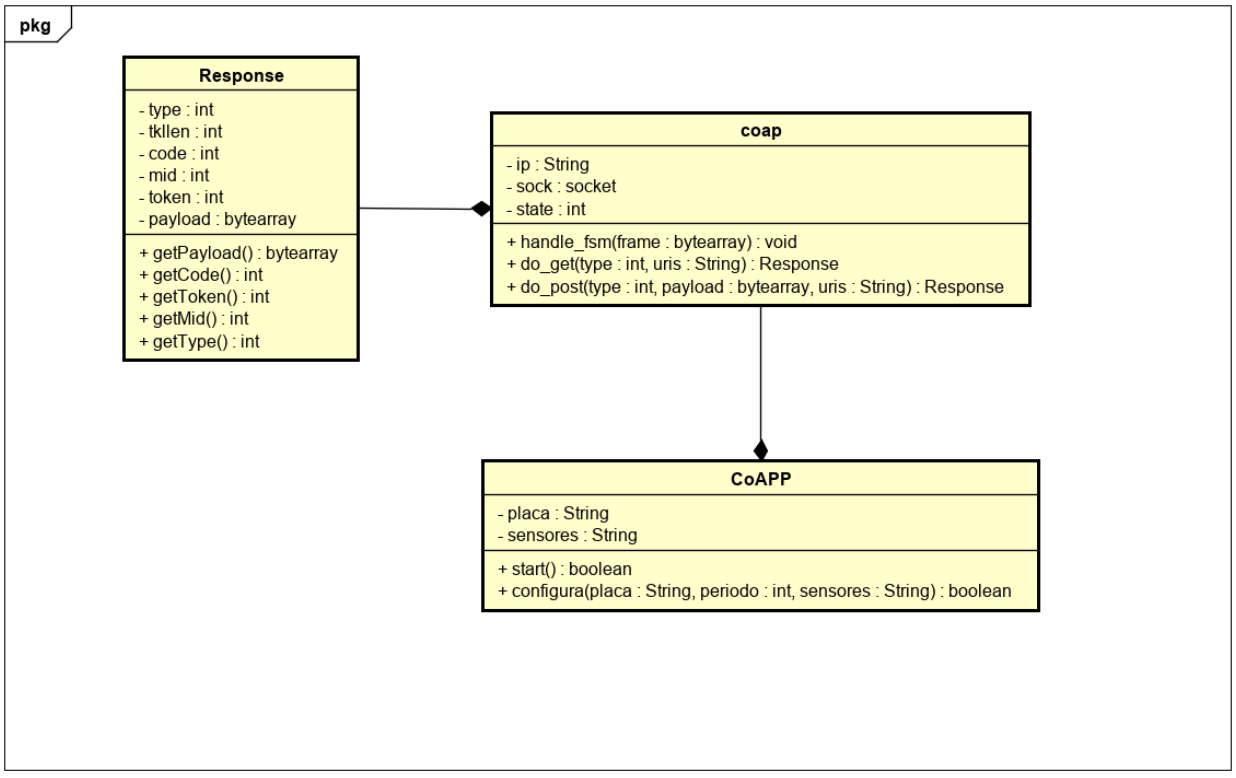
Afim de demonstração da API desenvolvida, criou-se uma classe chamada *CoAPP* (*coap application*), que utiliza os métodos da API para implementação do sistema de coleta de dados demonstrado na introdução deste relatório.

Esta classe possui os métodos *configura()*, que faz o registro da unidade de aquisição de dados no servidor *CoAP*, e *start()*, que manda inicia a troca de dados das coletas dos sensores para o servidor.

```
63 from coapc import coap
64 from response import Response
65
66 class CoAPP(Callback):
67     def __init__(self):
68         pass:
69     def configura(self, placa, periodo, *sensores):
70         pass
71
72     def start(self):
73         pass
74
75
76 if __name__ == '__main__':
77 a = CoAPP()
78 if (a.configura('placa2',10,'sensor A', 'sensor B')):
79     a.start()
```

A relação entre as classes pode ser vista a partir do diagrama de classes simplificado na figura abaixo:

Figura 4: Diagrama de classes da implementação



# 5 Conclusão

Foi desenvolvida uma API que implementa o lado cliente de uma versão simplificada do protocolo *CoAP*, com base na RFC 7252. A API permite que o usuário consuma os métodos GET e POST de um servidor *CoAP*. Afim de demonstração, foi desenvolvida uma classe chamada *CoAPP*, que se aproveita da API do cliente *CoAP* e se comunica com um servidor implementado pelo professor da disciplina.

# Referências

INTERNET ENGINEERING TASK FORCE. *RFC 7252*: The constrained application protocol (coap). [S.l.], 2014. 112 p.