INSTITUTO FEDERAL
Santa Catarina

Projeto 02

Protocolo de aplicação

Curso: Engenharia de Telecomunicações
Disciplina: PTC29008 – Projeto de Protocolos
Professor: Marcelo Maia Sobral

Aluno
Renan Rodolfo da Silva

20 de junho de 2018

1 Introdução

Projeto 2 consiste no desenvolvimento de um protocolo de aplicação CoAP, tendo como base a RFC 7252 em que foram seguidas suas especificações para implementar um protótipo de sistema de aquisição de dados no qual as mensagens são codificadas com Protocol Buffers e transmitidas por links sem-fio ponto-a-aponto . O sistema proposto está demonstrado na Figura 1, em que as unidades de aquisição são representados por um kit baseado em RaspberryPi 3,

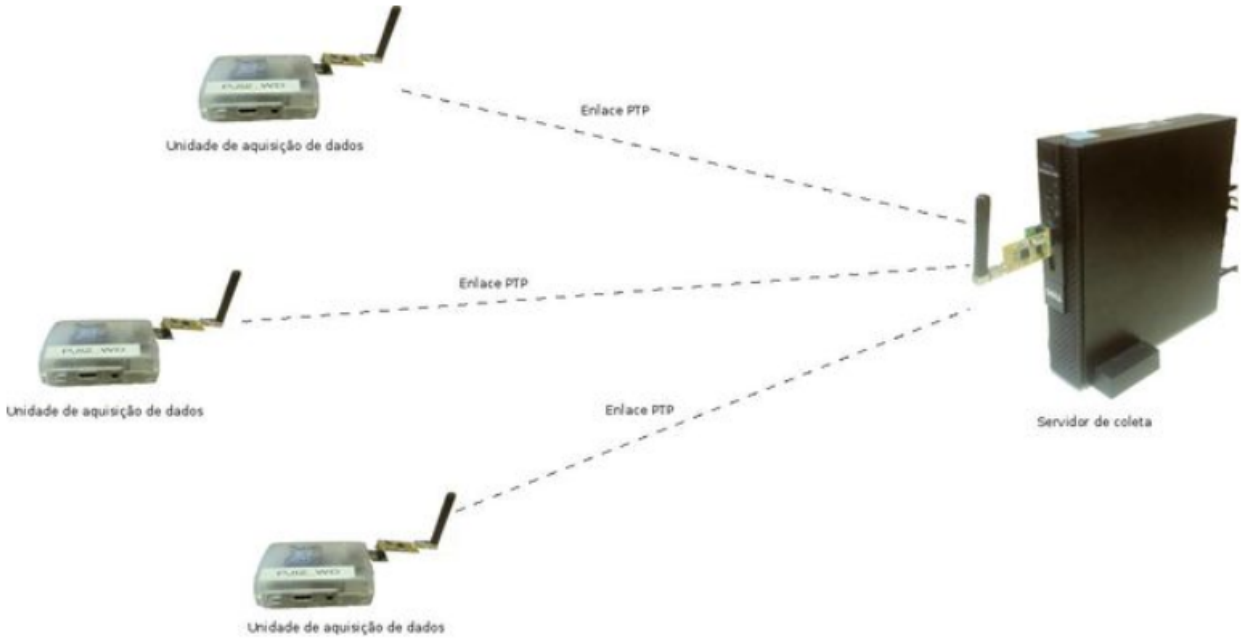

Figura 1: Sistema proposto

2 Especificações

O formato de mensagens utilizado por esse protocolo pode ser verificado na Figura 2. O valor de cada campo foi definido logo abaixo:

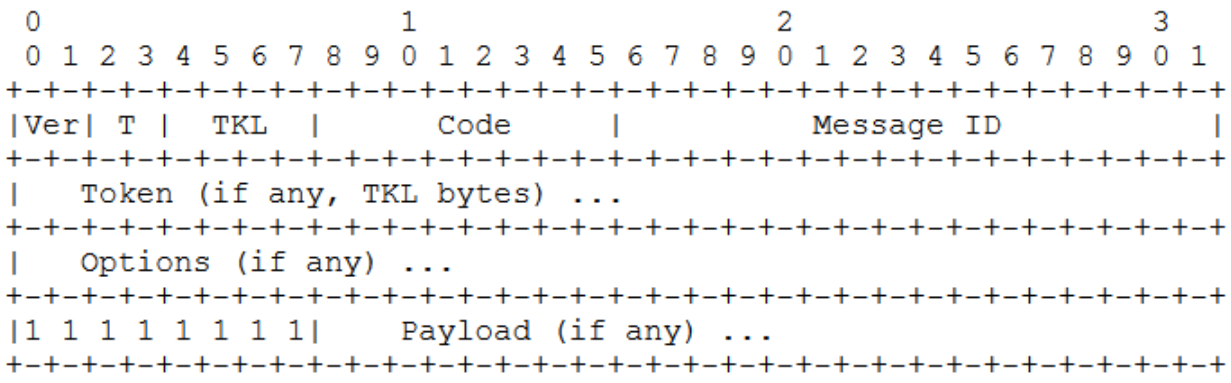

Figura 2: Formato da mensagem

- Ver: 2-bit unsigned interger, sendo esses bits 01, representando a versão.
- T: 2-bit unsigned interger, nesse caso as mensagens enviadas são sempre confirmáveis, então foi assumido o valor 00. Para respostas assume-se o valor 10(Confirmação)
- Token Length(TKL): 4-bit unsigned integer, o tamanho do Token utilizado foi de apenas 1 byte(0001).
- Code: A RFC possui quatro métodos possíveis além do Ack para serem utilizados, mas nesse protocolo foi usado somente os abaixo Get e Post. O Ack é usado apenas para confirmação de uma resposta recebida pelo protocolo.

Code	Name	Reference	Hexa	Class Detail
0.01	GET	[RFC7252]	0x01	000 00001
0.02	POST	[RFC7252]	0x02	000 00010
0.00	ACK	[RFC7252]	0x00	000 00000

Figura 3: Formato das opções

- Message ID: 16-bit unsigned integer, no protocolo está sendo representado pelas variáveis MID1 e MID2 que são gerados randomicamente a cada envio de mensagem com valor entre 0 e 255.
- Token: Como TKL é 1 byte, o valor do Token utilizado é um inteiro randômico entre 0 a 255 alterando a cada envio assim como Message ID.
- Options: Foram assumidos dois tipos de options: Uri-Path e Content-Format. A Uri-Path é representada pelo valor 11(4-bit MSB) mais o tamanho da URI(4-bit LSB). Já o Content-Format é representado pelo valor 12(4-bit MSB) mais o tamanho da opção(4-bit LSB) que nesse caso é 1 byte, pois o valor correspondente ao tipo de mídia transportado é representando por 1 byte de valor 42.

- Payload: O Payload que esse protocolo suporta, são mensagens codificadas por meio do Protocol Buffers que resultam em um tipo de mída chamado octet-stream, que é representado pelo valor 42, como mencionado na Options.

3 Modelo de Software

O funcionamento do protocolo CoAP desenvolvido pode ser verificado na Figura 4

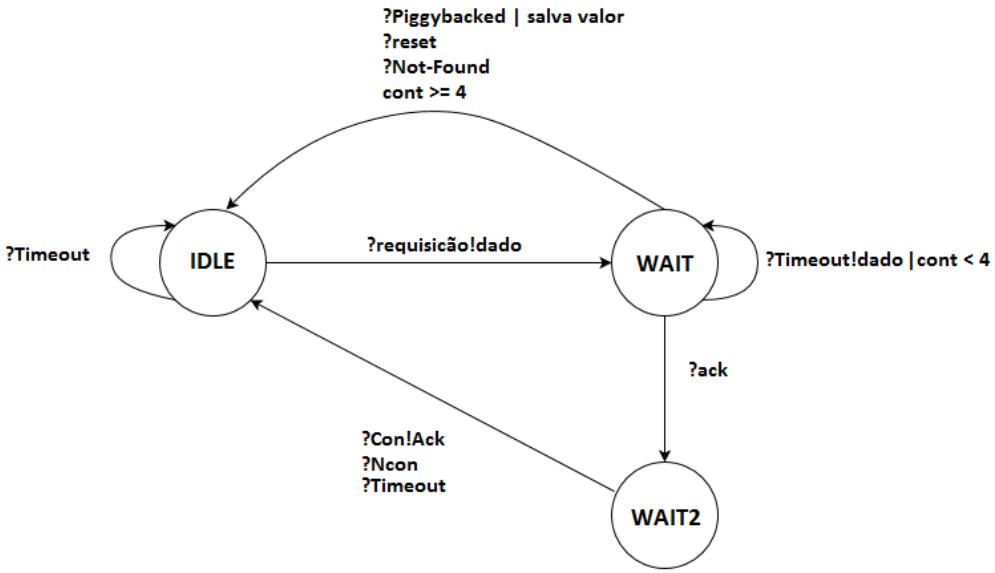

Figura 4: Máquina de estado CoAP Client

4 Descrição da API

Ack_Response(self):
** Função interna do protocolo, não deve ser chamada por uma aplicação, **
Monta uma mensagem de resposta apenas para ser usado como comparação para quando receber algum datagrama.

Do_Get(self, Uri, Payload):
Recebe uma Uri e um payload. Monta uma mensagem com Code GET e envia para o handle_fsm fazer a tratativa. Logo em seguida faz o despache para o poller monitorar os eventos futuros.

Do_Post(self, Uri, Payload)
Recebe uma Uri e um payload. Monta uma mensagem com Code POST e envia para o handle_fsm fazer a tratativa. Logo em seguida faz o despache para o poller monitorar os eventos futuros.

Push_Ack(self, MID1, MID2, TOKEN)
** Função interna do protocolo, não deve ser chamada por uma aplicação, **
Recebe como parâmetro os dois bytes contidos na Message ID e o Token de 1 byte. Monta uma mensagem de confirmação com payload vazio

Reload_Timeout_Ack(self, value)
** Função interna do protocolo, não deve ser chamada por uma aplicação, **
O tempo da retransmissão é calculado como (2+4+8+16)*ACK_TIMEOUT_FACTOR, onde a primeira transmissão é 2*ACK_TIMEOUT_FACTOR, segunda 4*ACK_TIMEOUT_FACTOR e assim

por diante.Resultando em um total de 4 tentativas de retransmissão, não ultrapassando o tempo máximo de 45 segundos estipulado pela RFC.

Return_Base_Timeout(self)
Retorna o valor do timeout para o valor que foi definido como padrão. No caso foi 2*ACK_TIMEOUT_FACTOR

Set_Periodo(self, periodo)
Recebe uma valor em milisegundos para ser utilizado como tempo de envio de dados. No caso, irá ficar enviando dados coletados a cada periodo recebido.

__init__(self, ip, Uri) Construtor do CoAP:
Recebe como parâmetro um IP(IVP6) mais a Uri que será utilizada como destino ao enviar mensagens para o servidor. Cria um socket e vincula-o com o IP fornecido.

handle(self)
Fica monitorando o socket criado para caso receber alguma mensagem, enviar para máquina de estado tratar. Recebe mensagem de até 2048 bytes.

handle_fsm(self, data)
Máquina de estado responsável por enviar as mensagens assim que a aplicação faz uma requisição, assim como trata um datagrama recebido do servidor. Nesse protocolo desenvoldido, o estado idle é o responsável por enviar o datagrama contendo as configurações da API assim que é iniciada. O Estado wait fica aguardando uma resposta do servidor, caso envie uma mensagem piggybacked salva o valor recebido para aplicação coletar o payload. Wait2 seria caso o servidor enviasse apenas um Ack estando no Wait, ele fica esperando o recebimento da resposta.

handle_timeout(self) handle_timeout é o responsável por fazer a retransmissão caso alguma mensagem COAP seja enviada e não receba nenhuma resposta do servidor. Tempo utilizado para isso esta descrito em Reload_Timeout_ack()

Data and other attributes defined here:

ACK_TIMEOUT = 2

ACK_TIMEOUT_FACTOR = random.uniform(1,1.5)

Delta1_Path = 176

Delta2_Path = 0

MAX_RETRANSMIT = 4

Teste_Get = None

end = 255

idle = 0

wait = 1

wait2 = 2

Methods inherited from poller.Callback:

disable(self)
Desativa o monitoramento do descritor neste callback

disable_timeout(self)
Desativa o timeout

enable(self)
Reativa o monitoramento do descritor neste callback

enable_timeout(self)
Reativa o timeout

reload_timeout(self)
Recarrega o valor de timeout

update(self, dt)
Atualiza o tempo restante de timeout

Data descriptors inherited from poller.Callback:

isEnabled
true se monitoramento do descritor estiver ativado neste callback

isTimer
true se este callback for um timer

5 Manual da aplicação

Para demonstrar o funcionamento do sistema de aquisição de dados deve-se executar o arquivo CoAP_APP.py com python3. Por padrão ele está setado para enviar dados de dois sensores. Os valores da medição são calculados pela própria aplicação, não sendo necessário passar como parâmetro. Os campos possíveis de serem alterados são: Nome da placa, lista de sensores(máximo dois) e a URI de destino, que nesse caso é /ptc.Um exemplo de como criar um objeto CoapAPP é:

Payload_App = CoapAPP("Renan",["Velocidade","Temperadura"],"/ptc")

Com o objeto criado é necessário chamar o método Protobuffer_Payload() que fará a codificação da mensagem utilizando o Protocol Buffers e envia o payload para o Coap_Client.py através do método Do_Post() disponível no mesmo, e logo em seguida, faz o despache utilizando o poller.

Figura 5 mostra o monitoramento feito na transmissão da aplicação-servidor. Inicialmente a aplicação envia um Post com Config, contendo o nome da placa e os nomes dos sensores a serem monitorados. O servidor responde com created com payload contendo o valor do período que a aplicação deve ficar enviando os dados. Como pode se observar, o tempo foi de 5 segundos. Por fim, mostra a retransmissão quando o servidor está indisponível.

No.	Time	Source	Destination	Protocol	Info	Length
55	9.883063287	::1	::1	CoAP	CON, MID:16735, POST, TKN:6f, /ptc	112
56	9.887953563	::1	::1	CoAP	ACK, MID:16735, 2.01 Created, TKN:6f, /ptc	107
58	11.891134459	::1	::1	CoAP	CON, MID:38532, POST, TKN:c6, /ptc	130
60	12.209718594	::1	::1	CoAP	ACK, MID:38532, 2.03 Valid, TKN:c6, /ptc	69
72	17.213588611	::1	::1	CoAP	CON, MID:9506, POST, TKN:68, /ptc	139
75	18.105020820	::1	::1	CoAP	ACK, MID:9506, 2.03 Valid, TKN:68, /ptc	69
82	23.111240940	::1	::1	CoAP	CON, MID:45917, POST, TKN:55, /ptc	130
83	24.128306605	::1	::1	CoAP	ACK, MID:45917, 2.03 Valid, TKN:55, /ptc	69
84	29.130059337	::1	::1	CoAP	CON, MID:3881, POST, TKN:e5, /ptc	130
85	29.625386200	::1	::1	CoAP	ACK, MID:3881, 2.03 Valid, TKN:e5, /ptc	69
90	34.631427660	::1	::1	CoAP	CON, MID:56243, POST, TKN:06, /ptc	130
91	34.631457269	::1	::1	ICMPv6	Destination Unreachable (Port unreachable)	178
95	36.845091096	::1	::1	CoAP	CON, MID:56243, POST, TKN:06, /ptc	130
96	36.845124717	::1	::1	ICMPv6	Destination Unreachable (Port unreachable)	178
98	41.271990367	::1	::1	CoAP	CON, MID:56243, POST, TKN:06, /ptc	130
99	41.272022344	::1	::1	ICMPv6	Destination Unreachable (Port unreachable)	178
119	50.125349156	::1	::1	CoAP	CON, MID:56243, POST, TKN:06, /ptc	130
120	50.125386662	::1	::1	ICMPv6	Destination Unreachable (Port unreachable)	178
146	67.813128185	::1	::1	CoAP	CON, MID:56243, POST, TKN:06, /ptc	130
147	67.813157404	::1	::1	ICMPv6	Destination Unreachable (Port unreachable)	178

Figura 5: Monitoramento de mensagens CoAP

Figura 6 mostra alguns sensores que foram enviados ao servidor, e na Figura 7 mostra alguns valores de amostras coletadas pela aplicação e enviado ao servidor. Essas informações estão contidas no bando de dados do servidor.

```
CREATE TABLE Sensor (id integer primary key,nome text,placa integer);
INSERT INTO "Sensor" VALUES(1,'luz',1);
INSERT INTO "Sensor" VALUES(2,'temperatura',1);
INSERT INTO "Sensor" VALUES(3,'Temperatura',2);
INSERT INTO "Sensor" VALUES(4,'Velocidade',2);
INSERT INTO "Sensor" VALUES(5,'Temperatura',3);
INSERT INTO "Sensor" VALUES(6,'Velocidade',3);
INSERT INTO "Sensor" VALUES(7,'Temperadura',2);
CREATE INDEX ind_Sensor_nome on Sensor (nome);
CREATE INDEX ind_Sensor_placa on Sensor (placa);
```

Figura 6: Sensores medidos

```
INSERT INTO "Amostra" VALUES(3578,'50',1560898332,4);
INSERT INTO "Amostra" VALUES(3579,'15',1560898332,7);
INSERT INTO "Amostra" VALUES(3580,'19',1560899338,4);
INSERT INTO "Amostra" VALUES(3581,'87',1560899338,7);
INSERT INTO "Amostra" VALUES(3582,'-1',1560899343,4);
INSERT INTO "Amostra" VALUES(3583,'30',1560899343,7);
INSERT INTO "Amostra" VALUES(3584,'34',1560899349,4);
INSERT INTO "Amostra" VALUES(3585,'63',1560899349,7);
INSERT INTO "Amostra" VALUES(3586,'7',1560899355,4);
INSERT INTO "Amostra" VALUES(3587,'95',1560899355,7);
CREATE INDEX ind_Amostra_sensor on Amostra (sensor);
CREATE INDEX ind_Amostra_timestamp on Amostra (timestamp);
COMMIT;
```

Figura 7: Valores e Timesamp enviados

6 Conclusão

Esse projeto demonstrou como poderia ser projetado um sistema de aquisição de dados para lugares que não é possível a utilização de internet por exemplo. Além de elucidar os desafios propostos para quem atua na área de Projeto de Protocolos, tendo que projetar/modificar algum protocolo existente para atender a um cenário/projeto em específico, respeitando sua RFC.