Projeto 2 - Constrained Application Protocol (CoAP)

Daniel Cabral Correa
Luísa Machado
Professor: Marcelo Maia Sobral.
Disciplina: Projeto de Protocolos.
Engenharia de Telecomunicações

18 de dezembro de 2018

1 Introdução

Este relatório refere-se ao segundo projeto da disciplina de Projetos de Protocolos. Neste documento será descrito o desenvolvimento de uma versão simplificada do protocolo *CoAP*. O protocolo deverá comunicar-se com um servidor *CoAP*. Para a implementação deste projeto foi utilizada a linguagem *Python*[1].

O documento será apresentado da seguinte forma: a parte inicial do trabalho trata da especificação do protocolo, em seguida é feita a análise do protocolo e apresentado o serviço, o ambiente de execução, o vocabulário e a codificação do mesmo. Após isso é abordada a descrição do protocolo e seu comportamento. Por fim é descrito o manual do usuário.

2 Especificação

Nesta atividade foi desenvolvido de forma simplificada o Protocolo de Aplicação Restrita, CoAP (*Constrained Application Protocol*). O protocolo CoAP é um protocolo de comunicação ponto-a-ponto utilizado em rede limitadas, principalmente para Internet das Coisas (sigla em inglês IoT).

Para facilitar a comunicação e diminuir o tamanho da mensagem foram criados quatro tipos de mensagem para simplificar seu funcionamento.

- ***Confirmable*:** Quando no quadro da mensagem o tipo é identificado como confirmável, ele exige a confirmação do recebimento da mensagem.
- ***Non-confirmable*:** Quando no quadro da mensagem o tipo é identificado como não-confirmável, não existe a necessidade de confirmação do recebimento.
- ***Acknowledgement*:** Quando no quadro da mensagem o tipo é identificado como ACK, significa que esta mensagem é uma confirmação do recebimento da mensagem anterior.
- ***Reset*:** Quando no quadro da mensagem o tipo é identificado como Reset, significa que em algum momento existiu um problema na comunicação e está sendo solicitado a retransmissão de toda a informação.

[1] Python é uma linguagem de programação.

3 Análise do protocolo

- **Serviço:** Protocolo ponto-a-ponto para camada de aplicação utilizando *socket* UDP.
- **Ambiente de execução:** Redes IoT.
- **Vocabulário:** GET, PUT, POST, DELETE.
- **Codificação:** Bytes.

4 Descrição do protocolo

O protocolo de aplicação *CoAP* é utilizado para a comunicação de forma assíncrona baseado no protocolo UDP. Um dos objetivos principais do *CoAP* é ser uma alternativa de protocolo de cliente/servidor para redes *IoT*. Por ser compatível com o protocolo HTTP, facilita a integração e o reuso de aplicações. O *CoAP* é um conjunto *REST* otimizado para *M2M* (*Machine to Machine*), com suporte a descoberta de recursos, *multicast* e troca de mensagens assíncronas com simplicidade e baixo *overhead*. Conforme descrito na seção 2, protocolo possui quatro tipos de mensagens: CON (*Confirmable*), NON (*Non-confirmable*), ACK (*Acknowledgement*) e RST (*Reset*). Na figura 1 é mostrado o formato do pacote.

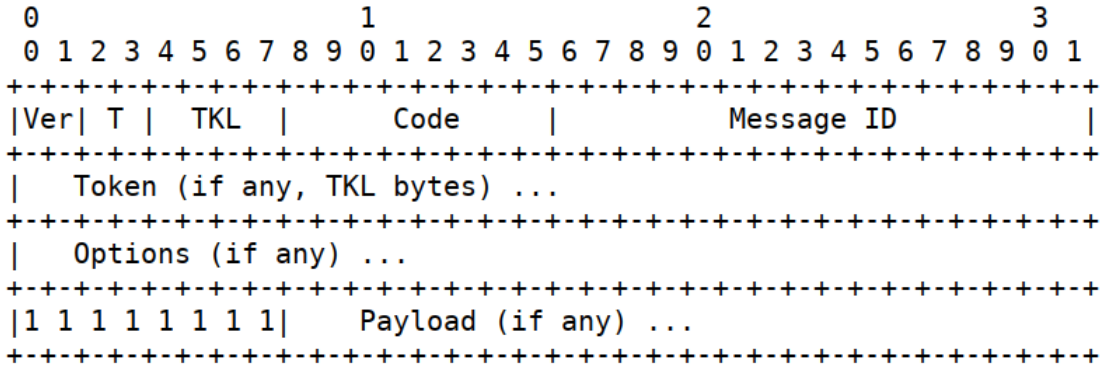

Figura 1 - Quadro do protocolo CoAP.

4.1 Cabeçalho

- ***Version*:** Campo de 2 bits responsável por informar a versão do protocolo. Atualmente o protocolo possui apenas uma versão, portanto esse campo tem valor fixo em '01'.
- ***Type*:** Campo de 2 bits responsável por identificar qual o tipo de mensagem. Para este campo são usadas as quatro opções citadas na seção 2.
- **TKL (*Token Length*):** Campo de 4 bits, responsável por informar o tamanho do *Token*.
- ***Code*:** Campo de 1 byte, responsável pela sinalização da mensagem. Essa sinalização foi dividida em quatro grupos de informação (requisição, resposta com sucesso, erro no cliente, erro no servidor). Esses grupos são identificados nos 3 primeiros bits, os últimos 5 bits são responsáveis por informar a opção dentro do grupo.

- ***MessageID*:** Campo de 2 bytes responsável por detectar duplicatas e de identificar respostas de requisição.
- ***Token*:** Campo opcional de tamanho variável de 0 a 8 bytes dependendo do valor de TKL. É responsável por correlacionar as solicitações com as respostas.
- ***Option Delta*:** Campo de 4 bits, responsável por informar a opção do recurso. Existem três valores reservados, os valores 13 e 14 indicam que o valor da opção é maior que 12, sendo assim é adicionado 1 ou 2 bytes, respectivamente, no *Option Delta (extended)*. O valor 15 indica ERRO.
- ***Options Length*:** Campo de 4 bits, responsável pelo comprimento do valor da opção (recurso). Assim como no *Option Delta*, existem três valores reservados, os valores 13 e 14 indicam que o valor da opção é maior que 12, sendo assim é adicionado 1 ou 2 bytes, respectivamente, no *Option Length (extended)*. O valor 15 indica ERRO.
- ***Options*:** Campo de valor variável com o tamanho dependente do valor do *Options Length*. Armazena a opção do recurso.
- ***Payload Marker*:** Campo de 1 byte com valor fixo em '0xFF'. Responsável por identificar o fim do cabeçalho e início do Payload.
- ***Payload*:** Campo que contém a informação (carga útil).

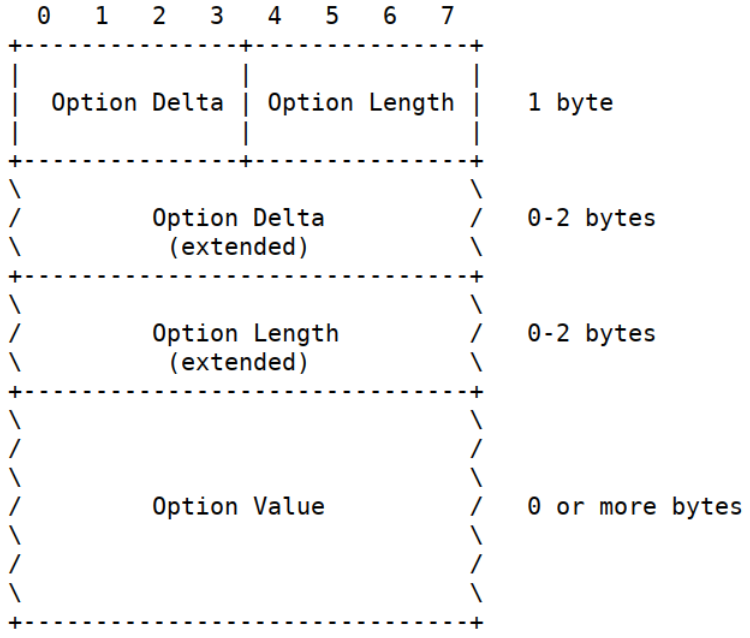

Figura 2 - Cabeçalho *Options*.

4.2 Métodos

- **GET:** Solicita ao servidor o valor da informação desejada partindo do recurso informado pelo usuário (Método totalmente implementado e testado).
- **PUT:** Solicita ao servidor alteração do recurso especificado, caso o recurso não exista ele é criado com o dado informado no payload (Método parcialmente implementado e testado).
- **POST:** Solicita ao servidor que processe uma informação. Essa função cria ou atualiza um recurso do servidor. (Método parcialmente implementado e não testado).
- **DELETE:** Solicita ao servidor que seja deletado o recurso informado (Método parcialmente implementado e não testado).

5 Manual de utilização da API

O uso deste protocolo inicia-se importando a API da seguinte forma:

– import coapLD

Em seguida, é necessário instanciar a classe, conforme o exemplo a seguir:

– c = coapLD.coap()

Feito isso pode-se utilizar as funções descritas abaixo.

5.1 Função GET

Para utilizar a função GET, o usuário deverá passar uma string, contendo obrigatoriamente o IP ou DNS do servidor e o recurso a ser acessado. Também é possível informar o número da porta, essa informação é opcional, caso não for informada a porta utilizada será a 5683. O usuário pode também informar um valor para o token, com tamanho máximo de 8 caracteres, separado da string que contém o recurso.

Exemplo de uso:

– resposta, lista_opções = c.GET(coap://ip_dns:porta/recurso, **token=**valor_token)

Essa função retorna dois valores, o primeiro é o código da resposta do servidor junto com o payload, e o segundo valor de retorno é uma lista com as opções respondidas pelo servidor.

5.2 Função PUT

Para utilizar a função PUT, o usuário precisa informar uma string, contendo obrigatoriamente o IP ou DNS do servidor e o recurso a ser solicitado. Assim como na função GET, também é opcional passar o valor da porta, caso não seja informada será utilizada a porta padrão 5683. Nessa função não é possível passar um valor para o token.

Exemplo de uso:

– resposta = c.PUT(coap://ip_dns:porta/recurso)

Essa função retorna apenas o código da resposta do servidor junto com o payload.

6 Manual de utilização dos programas de teste

Para iniciar a execução deste protocolo é necessário ter pré-instalado a versão 3 ou superior do *Python*. Além disso é necessário ter um servidor CoAP rodando. Neste projeto foram utilizados dois servidores CoAP para validar o funcionamento. Para realizar os testes com a função GET foi utilizado a biblioteca LibCoAP disponibilizada pelo professor. Enquanto os testes da função PUT foram realizados usando a biblioteca AioCoAP disponibilizada na página do protocolo (Chrysn).

Atendidos os requisitos acima, pode-se iniciar a execução, para isto, deve-se ter o arquivo coapLD.py e pelo menos um dos arquivos a seguir no mesmo diretório para realizar os testes:

- get_direto.py
- get_indireto.py
- put_direto.py
- put_indireto.py

Para executar a função GET deste projeto deve-se executar o servidor CoAP da biblioteca LibCoAP conforme apresentado a seguir:

  - coap-server

O teste pode ser realizado de duas formas diferentes, o método direto recebe os parâmetros no formato padrão mais usado de acesso ao servidor, e o método indireto onde os parâmetros de acesso são passados separadamente.

- **Exemplos método direto:**
  (Formato: *python3 get_indireto.py recurso token*)
  - python3 get_direto.py coap://127.0.0.1/.well-known/core 123
  - python3 get_direto.py coap://127.0.0.1:6583/.well-known/core
  - python3 get_direto.py coap://localhost/.well-known/core
- **Exemplos método indireto:**
  (Formato: *python3 get_indireto.py recurso endereço_servidor porta token*)
  - python3 get_indireto.py .well-known/core 127.0.0.1 5683 123
  - python3 get_indireto.py .well-known/core 127.0.0.1 5683
  - python3 get_indireto.py .well-known/core 127.0.0.1

Para executar a função PUT deste projeto deve-se executar o servidor CoAP da biblioteca AioCoAP, para isso é necessário estar no diretório aiocoap, pasta que contém a biblioteca, e executar o seguinte comando:

  - ./server.py

Assim como o teste da função GET, há duas formas diferentes de realizar o teste, o método direto e o método indireto.

- **Exemplos método direto:**
  (Formato: *python3 put_indireto.py recurso token*)
  - python3 put_direto.py <coap://127.0.0.1/other/block> 123
  - python3 put_direto.py <coap://127.0.0.1:6583/other/block>
  - python3 put_direto.py <coap://localhost/other/block>
- **Exemplos método indireto:**
  (Formato: *python3 put_indireto.py recurso endereço_servidor porta token*)
  - python3 put_direto.py other/block 127.0.0.1 5683 123
  - python3 put_direto.py other/block 127.0.0.1 5683
  - python3 put_direto.py other/block 127.0.0.1

Ao executar quaisquer comandos citados acima, será solicitado ao usuário que digite a informação que deseja-se enviar ao servidor.

Observação: Nos programas de teste do método indireto, tanto do GET quanto do PUT, não é possível utilizar um valor para o token sem um valor para a porta, devido ao formato do código de teste.

7 Conclusão

O protocolo obteve sucesso no método de acesso GET e parcialmente no PUT. Sendo assim, o protocolo desenvolvido pela equipe atende grande parte das especificações, exceto pelo funcionamento das funções POST e DELETE que não houve tempo hábil para realizar sua implementação. O principal desafio dessa atividade, foi o entendimento dos requisitos descritos na RFC 7252[2], que descreve o funcionamento do protocolo CoAP.

Referências

A Internet das Coisas: CoAP. Disponível em: <https://www.gta.ufrj.br/ensino/eel878/redes1-2016-1/16_1/iot/iot.html?coap>. Acesso em: 15 dez. 2018.

CHRYSN, C. *The Python CoAP library*. Disponível em: <https://github.com/chrysn/aiocoap>. Acesso em: 15 dez. 2018.

SOBRAL, M. *PTC29008*. Disponível em: <https://wiki.sj.ifsc.edu.br/wiki/index.php?title=PTC-EngTel_(página)>. Acesso em: 15 dez. 2018.

THE Constrained Application Protocol (CoAP). Disponível em: <https://tools.ietf.org/html/rfc7252>. Acesso em: 15 dez. 2018.

[2]https://tools.ietf.org/html/rfc7252