IoT messaging: Um Estudo Prático sobre Protocolos de Comunicação em um Contexto.*Internet of Things* para Casas

João Luís Veronezzi Pacheco
Universidade de São Paulo
Av. Arlindo Bettio, 1000
03828-000 São Paulo/SP
Email: joao.luis.pacheco@usp.br
NUSP: 9844853

Alexandre Farias
Universidade de São Paulo
Av. Arlindo Bettio, 1000
03828-000 São Paulo/SP
Email: alexandre.farias.santos@usp.br
NUSP: 9761826

SOBRE ESTE TRABALHO

Este relatório é parte integrante do Exercício Programa 1 da disciplina de Redes, do curso de sistemas de informação da Escola de Artes, Ciências e Humanidades da Universidade de São Paulo, ministrada pelo professor Dr. João Luiz Bernardes. O presente documento tem como objetivo apresentar a proposta do trabalho, especificação dos protocolos utilizados, um breve manual de instalação, observações para leitura do código, fluxos de execução e casos de teste, problemas entre as propostas e a aplicação e fontes.

I. PROPOSTA

Implementar um sistema híbrido de comunicação para automatizar tarefas costumeiras de uma casa. Como caso base utilizaremos o comando acender ou apagar luzes.

II. ARQUITETURA

Esta seção apresentará:

- Arquiteturas:
  - Hardware;
  - Software;
- Protocolos de Comunicação empregados;
- Módulo de Simulação;
- Tratamento de erros;
- Segurança;

A. Arquitetura

1) Arquitetura do Hardware: Como Hardware utilizaremos 1 Raspberry Pi como servidor (as funções serão explicadas na próxima seção) e 2 Arduinos como servidores-clientes. Como interface de conexão a rede utilizaremos a interface *wireless onboard* do RaspBerry Pi e o *shield* de interface *wireless* acoplado à cada arduino. Por fim utilizaremos 1 relé fotoelétrico para cada lampada que se pretende controlar.

2) Arquitetura do Software: Como arquitetura de software pensamos em um modelo híbrido de Cliente-Servidor e peer-to-peer. No modelo Cliente-Servidor temos o Raspberry se comportando como servidor registrando todos os Arduinos (que por sua vez seriam os clientes) que integram a rede, armazenando seus dados para roteamento, distribuindo os dados para outros Arduinos e solicitando a execução de algum comando a algum cliente. No modelo peer-to-peer temos todos os Arduinos se comportando como nós da mesma rede, cada arduino conhece todos os outros e mantém um sinal de *keep alive* para todos eles. Esse comportamento será utilizado em futuros aprimoramentos, na qual quando solicitado uma ação multipla como, por exemplo, desligar todas as luzes de um comodo, podemos solicitar que os arduinos logicamente vizinhos no controle desse comodo se comuniquem e executem a ação, exigindo que o Servidor solicite a ação a apenas um dos nós. Essa arquitetura *peer-to-peer* permite que o Servidor tenha mais certeza sobre a falha de algum arduino, pois, uma vez que o *keep alive* tenha parado de ser enviado ao servidor, este solicita aos nós da rede que informem o status do arduino em potencial falha no momento. As três principais ações-sequencia mapeadas foram:

- Ação Arduino - Raspberry;
- Ação Arduino - Arduino;
- Ação Cliente(Solicitante do Comando) - Raspberry.

A seguir apresentamos o diagrama de mensagens de cada interação principal do sistema:

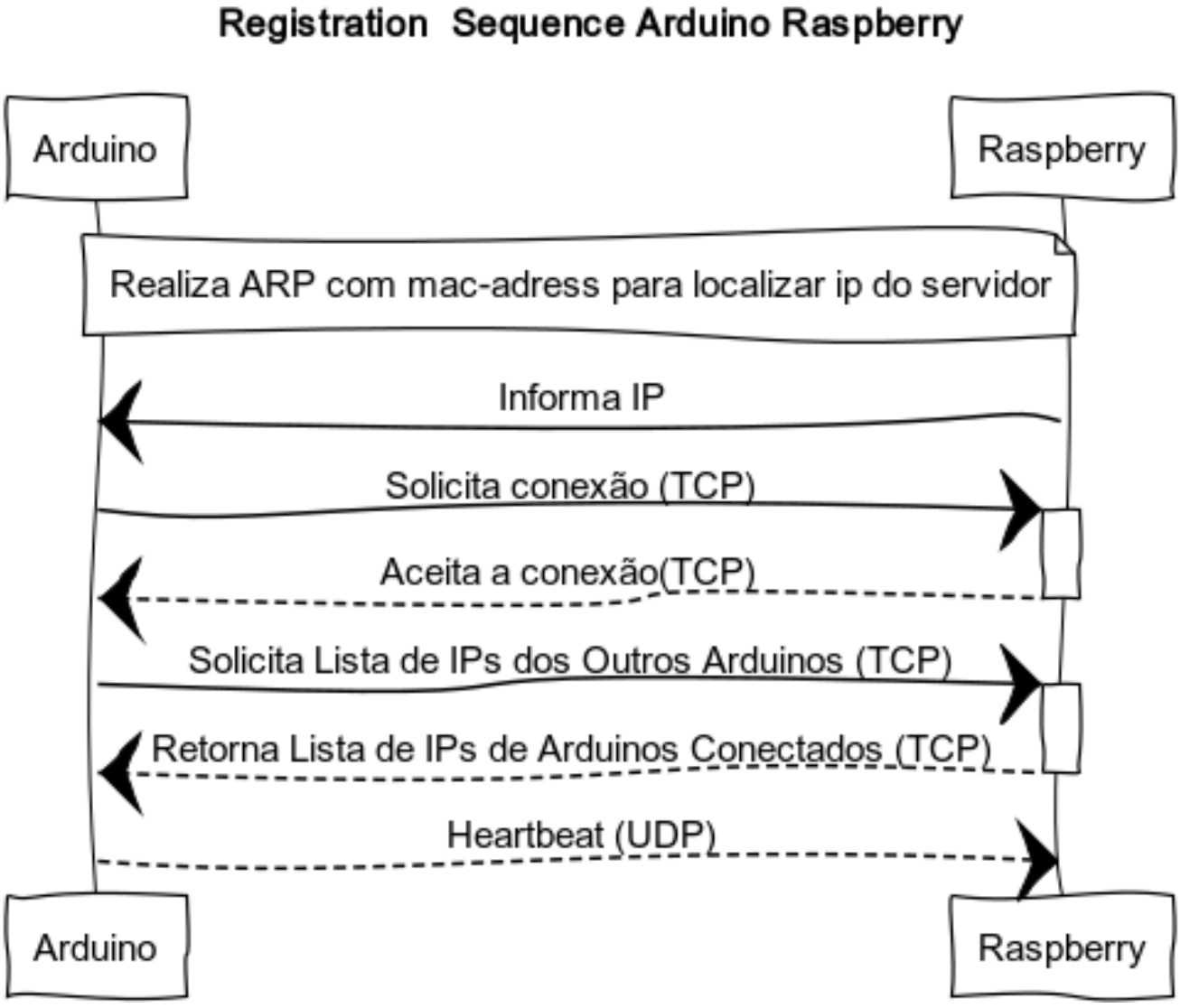

Figura 1: Diagrama de Mensagens: Ação de registro de um arduino

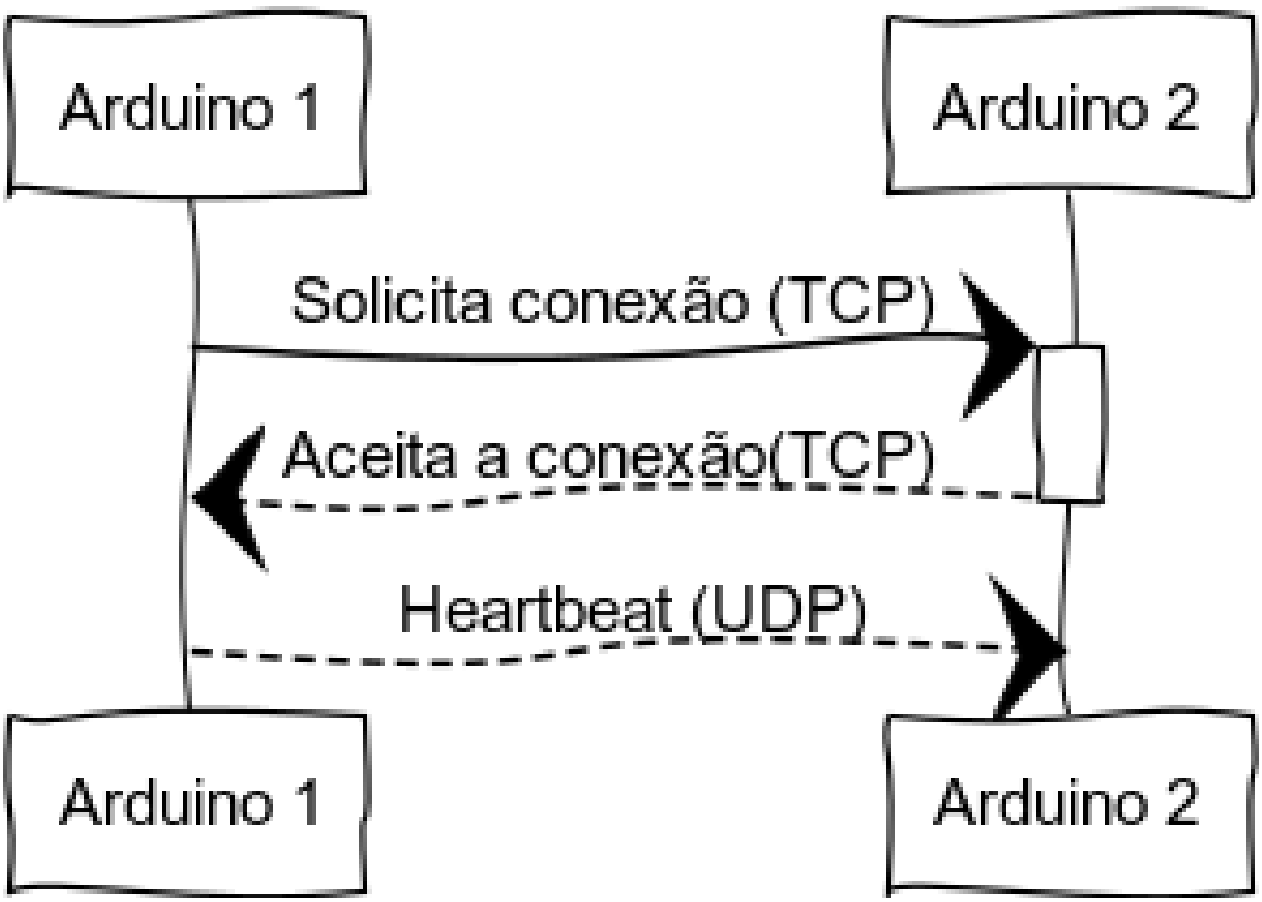

Figura 2: Diagrama de Mensagens: Ação de interação entre Arduinos

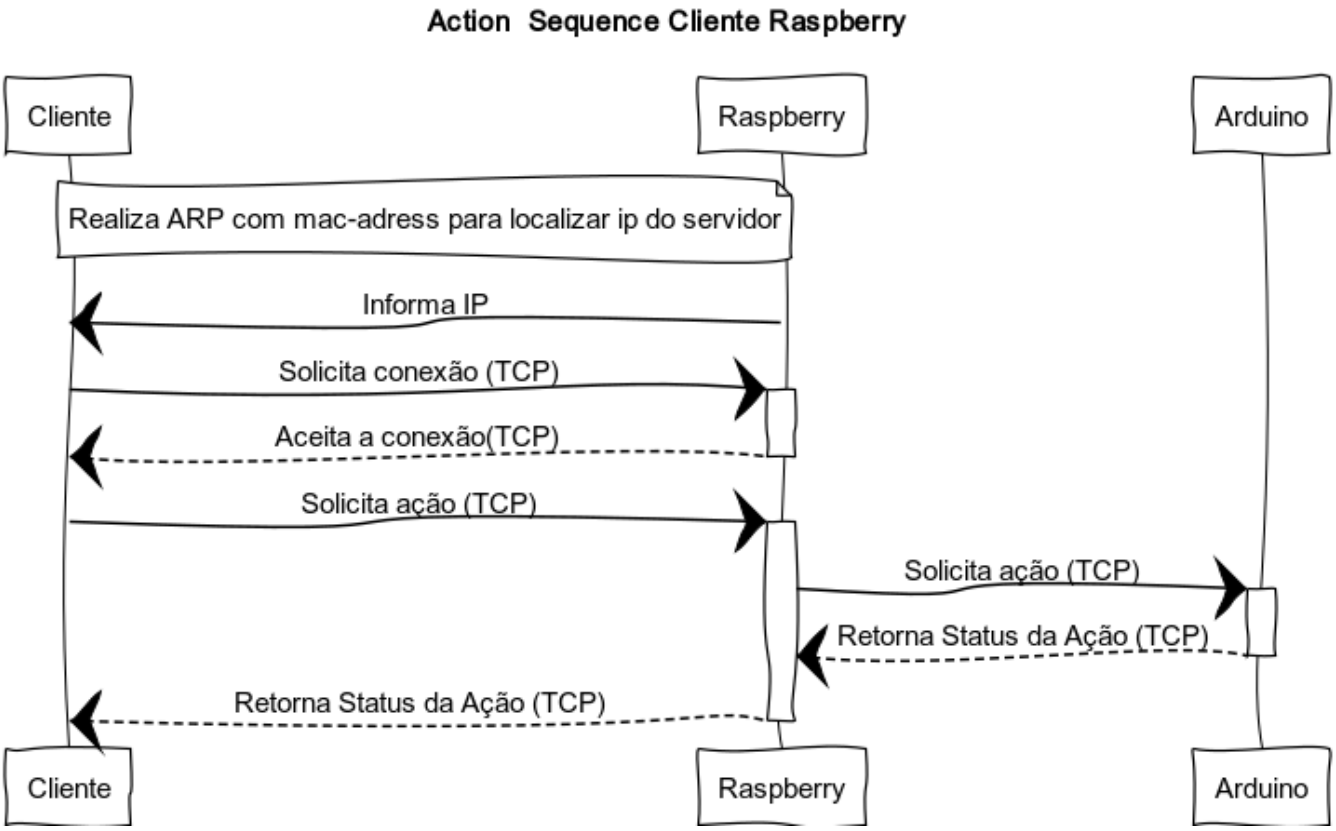

Figura 3: Diagrama de Mensagens: Ação de interação entre Arduinos

B. Protocolos de comunicação

C. Tratamento de erros

O programa assume que a porta indicada será a utilizada para comunicação TCP. O número da porta + 1 será utilizado para comunicação UDP, logo, a partir da porta 10000, para inserir novos dispositivos sempre coloque uma porta par como port, já que a porta impar seguinte será usada para UDP.

III. Manual de instalação

Para executar o programa é necessária a instalação de versão Python 3.6 ou maior. Segue o link para download das diferentes versões do Python www.python.org/downloads

Além disso, você precisará ter instalado o pip3 (a partir da versão 3.4 do python ele já vem por padrão nas instalações de python para Windows) para instalação dos requisitos necessários para executar o programa.

Com python e pip instalados, execute em um terminal ou no prompt de comando do Windows, na pasta raíz do projeto (icp-python/):

```
pip install -r requirements.txt
```

Feito isso a dependência PyQt5 já estará instalada.

Existem 2 modos de execução do programa, o primeiro é diretamente da linha de comando, para isso vá até a pasta simulation e no terminal execute

```
python Simulator.py
```

Para executar no modo interface, vá até a pasta interface e execute o comando:

```
python SimulatorInterface.py
```

A. Parametrização da simulação

Se quiser mudar os parâmetros da simulação, basta entrar no arquivo config da pasta que possui o modo de execução desejado e adicionar, remover ou editar os dispositivos já descritos no arquivo. A estrutura do arquivo é parecida com esta:

```
{
    "devices": {
        "0": {
            "name": "ARDUINO-01",
            "host": "localhost",
            "port": 10000,
            "failure_risk": 0.5
        },
        "1": {
            "name": "ARDUINO-02",
            "host": "localhost",
            "port": 10002,
            "failure_risk": 0.1
        }
    }
```

Importante ressaltar que o programa assume que a porta indicada será a utilizada para comunicação TCP o número da porta + 1 será utilizado para comunicação UDP, logo, a partir da porta 10000, para inserir novos dispositivos sempre coloque uma porta par como port, já que a porta impar seguinte será usada para UDP.

Além disso, o programa assume que sempre existiram dispositivos a serem configurados e não trata um arquivo de configuração vazia. O host também deverá ser definido como localhost para a realização de simulações locais do protocolo. O atributo failure_risk define a probabilidade do dispositivo ter uma falha e ficar 10 segundos sem responder a nenhuma requisição.

IV. Temporização

Os tempos de keep-alive entre os Arduínos são a cada 0.5 segundos, ou seja, a cada meio segundo, cada Arduíno envia para todos os seus conhecidos na rede um sinal de keep-alive. Dada a necessidade de simular uma falha no dispositivo quando a falha ocorre, o Arduíno fica 10 segundos sem realizar nenhuma ação, ou seja, não se comunica com ninguém. A cada 3 segundos o Raspberry pede para todos os dispositivos da rede a lista de timestamp dos dispositivos conhecidos, com base nesta lista captura o último timestamp de cada dispositivo e checa se a diferença do último timestamp até o momento da checagem é maior que 4 segundos, se for, declara-o como inativo, caso contrário, declara-o como ativo.

Ao inicializar uma simulação, se esta for via interface, a cada 1.3 segundo a classe Simulator faz uma chamada a classe SimulatorInterface para atualizar os dados de log e dispositivos ativos.

V. Como ler o código

Os dispositivos que compoem a arquitetura do sistema distribuido podem ser encontrados na pasta devices, os interpretadores das mensagens estão disponíveis na pasta interpreters, a implementação dos servidores e clientes TCP e UDP são encontradas na pasta net_utils ambas. A classe de simulação está dentro da pasta simulation enquanto a classe que renderiza a simulação em interface está dentro da pasta interface.

VI. Testes e resultados

Os testes realizados com o simulador mostraram resultados interessantes dado o objetivo do projeto, no primeiro testes, 3 dispositivos foram configurados com as probabilidades de falha de 50%, 10% e 0.01%, observou-se que a rede toda passou a conhecer todos os dipositivos conectados, pois o raspberry sempre envia o endereço de novos dispositivos para dispositivos que já estavam na rede, desse modo quando o segundo dispositivo entrou na rede, o dispositivo mais antigo não estava em sua lista de conhecidos, todavia, como o mais antigo recebeu seu endereço e o enviou seu keep-alive, o dispositivo mais recebte pode então registrar dispositivos mais antigos em sua lista de keep-alives, fazendo a rede de Arduínos

completamente conectadas até que uma parte dos mais antigos enviasse keep-alive para os mais novos.

Aumentou-se então a quantidade de dispositivos para 6 (com as mesmas probabilidades que se encontram no arquivo config da classe que realiza simulações e mostra em um interface).

Durante os testes detectou-se um problema com a biblioteca de interface, devido a suas limitações, quando o arquivo de log está muito cheio a interface pode travar pois não consegue renderizar a quantidade de texto gerada pela simulação.

Os testes foram realizados nos sistemas operacionais Windows 10 e Ubuntu 18, mas apenas no SO Windows 10 se observou um problema com os sockets TCP (especificamente os sockets clientes), o problema ocorria quando o servidor recusava a realização de uma conexão com o cliente, uma potencial explicação para esta condição pode ser o fato de que os servidores TCP não abrem novas Threads para novos clientes, desse modo, o servidor TCP atende apenas uma conexão por vez, como o Raspberry envia a requisição da lista de timestamps para todos os devices da rede em um curto espaço de tempo, as respostas precisariam então formar uma fila esperando a conexão ser liberada para o envio da mensagem. O que foi tradado de modo que se o servidor rejeitasse a conexão, o TCPClient esperaria 0.2 segundos e tentaria novamente a conexão até que esta fosse estabelecida.

De modo geral os testes foram satisfatórios, ou seja, o Raspberry conseguiu manter a lista de dispositivos da rede com seus status de atividade e inatividade corretos, apesar do delay devido a necessidade de calcular a diferença entre os timestamps para cada dispositivo na rede.

VII. Problemas entre proposta e aplicação

A. Encerrar programa por completo

Devido a natureza de simulação e necessidade de desacoplamento entre os dispositivos implementados, várias Threads tiveram que ser implementadas, no entanto ao executar a simulação no modo terminal o usuário precisa ou fechar o terminal para finalizar o programa, ou mandar um sinal de cancelamento de processo (Ctrl+C) para finalizar.

B. Portas dos sockets em utilização

Ao executar e fechar uma simulação o programa teve que esperar um período de tempo para iniciar novamente no sistema operacional Ubuntu 18, tentou-se corrigir este problema colocando no método destrutor __del__ das classes que possuem clientes e servidores TCP e UDP os comandos que fecham os sockets abertos. Todavia mesmo com este comando o programa ainda precisa esperar um certo tempo (em média 20 segundos) até iniciar novamente. Este problema não foi observado no sistema operacional Windows 10.

C. Travar durante a simulação

Um bloqueio devido a problemas de renderização da biblioteca de interface pode ocorrer devido ao tamanho excessivo do arquivo de LOG, se os bloqueios ocorrerem limpe o arquivo de LOG para reiniciar uma execução normal.