

UNIVERSIDADE ESTADUAL DE FEIRA DE SANTANA

DEPARTAMENTO DE TECNOLOGIA

TEC498 PROJETO DE CIRCUITOS DIGITAIS

PROBLEMA 1: INTERNET DAS COISAS

Cláudia Inês Sales

**Tutor:** Antonio Augusto

**Resumo.** Com a internet das coisas é possível identificar e conectar objetos do meio físico na internet, permitindo troca de informações entre eles. Para que os objetos reais, como dispositivos, se comuniquem com a aplicação é preciso um tratamento entre a comunicação. Este projeto visa a solução entre a comunicação de dispositivos reais, que se comunicam com a aplicação através de um serviço broker. Utilizando recursos nativos e também frameworks da linguagem java e python o sistema utiliza paradigma cliente-servidor. Através da aplicação, o usuário consegue enviar comandos para o dispositivo, além de verificar o estado atual do mesmo.

## 1 INTRODUÇÃO

A comunicação entre objetos físicos, conhecida como Internet das Coisas (IoT), permite a troca de informações entre eles por meio da Internet. A IoT é aplicável em diversas áreas, como saúde e energia, e, neste projeto, em dispositivos eletrônicos, como um ventilador.

A capacidade de rede dos dispositivos permite que eles se conectem a aplicativos usados por usuários. Este projeto foca em um ventilador que pode ser ligado, desligado ou ajustado em velocidade remotamente. Com a capacidade de comunicação pela rede, é possível controlar o ventilador por meio de uma aplicação específica. Assim, a comunicação entre objetos físicos torna a utilização mais prática e eficiente, possibilitando modificar o estado do dispositivo sem a necessidade de proximidade física.

Para viabilizar este projeto, foi criado um serviço broker que atua como intermediário na comunicação entre o usuário, por meio do aplicativo, e o ventilador, que é o dispositivo com capacidade de rede. O broker usa a rede TCP/IP para se conectar ao ventilador por comunicação nativa via socket, enquanto se comunica com o aplicativo por meio de requisições HTTP.

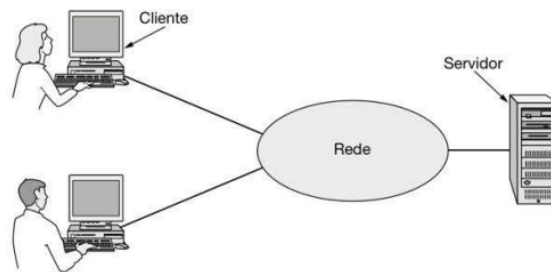
A comunicação entre o broker e o dispositivo ocorre de forma nativa, por meio de sockets configurados para troca de dados via UDP e TCP. Para isso, foi utilizada a linguagem Python, que possui suporte nativo para operações de socket. A aplicação, por sua vez, se conecta ao broker usando requisições HTTP, evitando a necessidade de lidar diretamente com sockets, utilizando frameworks para simplificar a integração.

Através da aplicação o usuário consegue verificar o estado atual do ventilador, como também altera-lo para desligado, ligado, e mudar, também, a velocidade de ventilação. Dessa forma, não é preciso que o usuário esteja no mesmo ambiente que o dispositivo para conseguir

visualizar as informações e alterá-las.

## 2 FUNDAMENTAÇÃO TEÓRICA

Para este projeto, o modelo cliente-servidor foi fundamental. Segundo Tanenbaum, grande parte das informações na internet é acessada por meio do modelo cliente-servidor. Nesse modelo, o cliente faz requisições a um servidor que possui as informações desejadas (TANENBAUM, 2003). O cliente inicia a comunicação solicitando recursos do servidor, enquanto o servidor responde fornecendo os dados requisitados pelo cliente, como na figura x. Embora exista essa definição, é possível que no mundo real em um momento o cliente acabe se comportando como servidor, ou seja, disponibilizando informações que foram solicitadas através de uma requisição.



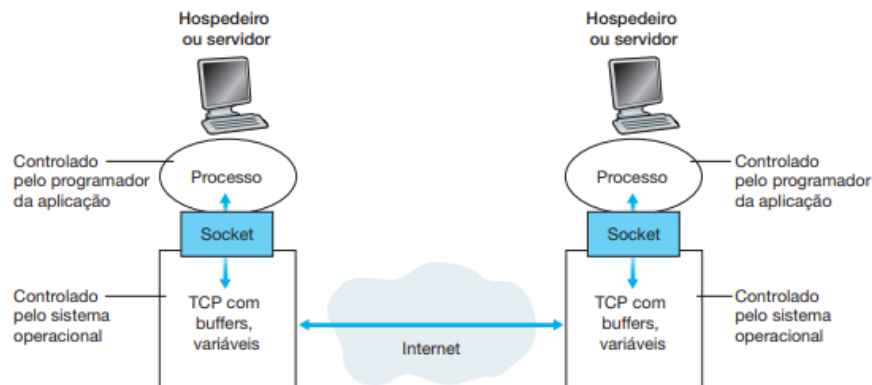
**Figura 1:** Cliente-Servidor

Autor:(TANENBAUM, 2003)

Assim, quando um cliente faz uma requisição a um servidor, ele precisa fornecer o endereço do servidor. Esse endereço é definido pelo IP, que é um protocolo de endereçamento na internet. Cada dispositivo tem seu próprio IP, usado para identificá-lo na rede. Quando a informação chega ao servidor, é necessário identificar para qual processo ela está destinada, o que torna crucial a combinação do IP e da Porta. O IP identifica o dispositivo, enquanto a Porta aponta para o processo ou aplicação específicos a que os dados se destinam.

A comunicação entre cliente e servidor pode ocorrer por meio de sockets, que são interfaces de software responsáveis pelo envio e recebimento de mensagens na rede (KUROSE; ROSS, 2006). Sockets fornecem um mecanismo de baixo nível para a transmissão de dados, permitindo que clientes e servidores se conectem, troquem informações e mantenham a comunicação ao longo do tempo.

No livro “Redes de Computadores e a Internet: Uma Abordagem Top-Down“, James Kurose faz uma analogia para ilustrar o conceito de sockets. Ele compara processos a casas e sockets a portas. Quando um processo deseja enviar uma mensagem, é como se estivesse usando uma porta para a comunicação. Para que a mensagem seja recebida, deve haver um socket correspondente no destino, como uma porta na casa do destinatário. Essa analogia ajuda a entender como os sockets funcionam, conectando processos e permitindo a comunicação entre eles na rede.



**Figura 2:** Socket

Autor: (KUROSE; ROSS, 2006)

Outro aspecto fundamental é como a mensagem será enviada, ou seja, qual protocolo será utilizado na camada de transporte. Protocolos são padrões que definem como a comunicação é estabelecida, como os dados são transferidos e como as conexões são mantidas. Por exemplo, o protocolo TCP é orientado a conexões, o que significa que, antes de enviar uma mensagem, ele estabelece uma conexão entre o remetente e o destinatário, garantindo a entrega confiável dos dados.

Por outro lado, o protocolo UDP é mais simples e não orientado a conexões. Diferentemente do TCP, ele não envolve um processo de estabelecimento prévio de conexão. Com o UDP, os dados são enviados sem se preocupar se chegarão ao destino ou como serão recebidos, tornando-o ideal para aplicações onde a velocidade é mais importante que a confiabilidade (KUROSE; ROSS, 2006).

Outro protocolo importante, amplamente utilizado na web, é o HTTP (HyperText Transfer Protocol). Ele é um protocolo de solicitação-resposta baseado em texto que funciona sobre TCP (TANENBAUM, 2003). No HTTP, existem dois tipos principais de mensagens: requisições e respostas.

As requisições do HTTP exigem a definição de um método usando verbos predefinidos como “GET”, “POST”, “DELETE”, entre outros. Geralmente, “GET” é utilizado quando o cliente quer obter um recurso, enquanto “POST” é usado para enviar ou publicar informações. Nas requisições, também é necessário especificar o caminho do recurso a ser acessado, incluindo o protocolo (por exemplo, http://), o domínio (que pode ser um endereço IP) e a porta para a qual a requisição deve ser enviada (como 0.0.0.0:1111). Dependendo do tipo de requisição, pode ser necessário incluir dados no corpo (body), que podem ser em diversos formatos, como JSON ou XML.

Nas respostas do HTTP, é importante fornecer a versão do protocolo, um código de status que indica se a requisição foi bem-sucedida (200, 201, etc.) ou falhou (400, 500, etc.), e uma mensagem de status que explica o motivo do código. Opcionalmente, as respostas podem conter um corpo com os dados requisitados, que podem ser em formatos como JSON ou XML. Esses componentes formam a base do protocolo HTTP, que é amplamente usado para comunicação na

web.

### 3 METODOLOGIA

A implementação do dispositivo, responsável por receber e responder às requisições, foi realizada utilizando a linguagem Python sem a inclusão de frameworks ou bibliotecas externas, apenas recursos nativos. A comunicação entre o dispositivo e o broker ocorre por meio de UDP e TCP, estabelecida através da configuração de sockets nativos da linguagem Python. A troca de dados entre as duas partes foi implementada utilizando métodos padrão de configuração de socket, garantindo a flexibilidade necessária para o projeto sem a dependência de soluções de terceiros.

O broker, embora use a mesma configuração de comunicação via TCP e UDP que o dispositivo, também utiliza o framework Flask para estabelecer a comunicação HTTP com a aplicação. Portanto, o broker é implementado em Python, com a ajuda do Flask para lidar com requisições HTTP. Dessa forma, ele pode atuar como um intermediário entre a aplicação e o dispositivo, garantindo flexibilidade e robustez na comunicação.

A aplicação foi implementada em Java, com o uso do framework Spring Boot para simplificar a configuração e inicialização das classes necessárias ao sistema. O Feign Client, uma biblioteca para clientes HTTP declarativa, também foi utilizado para facilitar as requisições HTTP ao broker.

Além das linguagens Java e Python, o Docker foi utilizado no desenvolvimento para simplificar a inicialização da aplicação. Com ele, cada componente do sistema é isolado em containers, garantindo maior consistência e facilitando a implantação.

O GitHub, que é uma plataforma de hospedagem de código-fonte e arquivos com controle de versão usando o Git, foi usado para versionar o código, permitindo reverter a versões anteriores, bem como para disponibilizar a aplicação e a documentação do sistema.

### 4 DISCUSSÃO E RESULTADOS

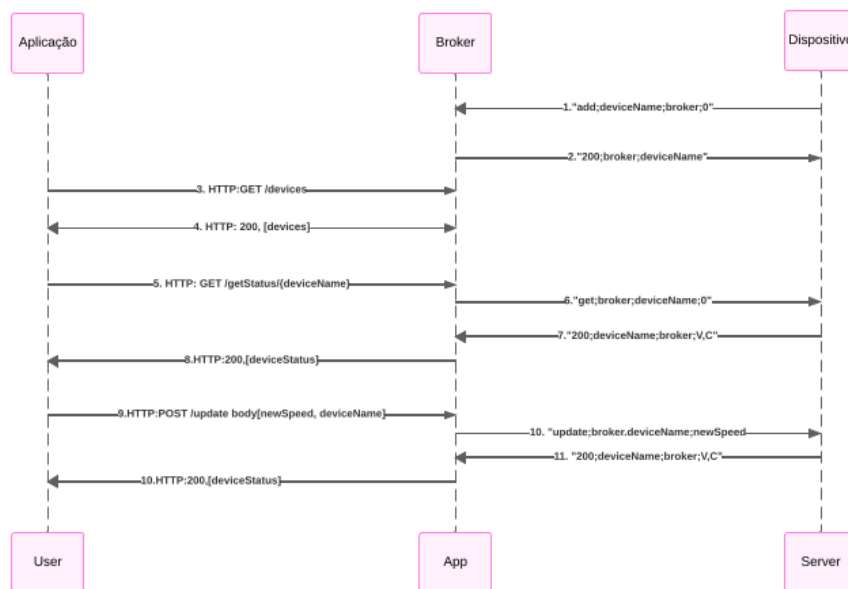
O sistema é composto por três componentes principais: a aplicação, o serviço broker e o dispositivo. A comunicação entre eles é essencial para o funcionamento do sistema. A aplicação envia solicitações HTTP ao serviço broker, atuando como intermediária entre o usuário e o broker. Ela é responsável por processar as solicitações do usuário e traduzi-las para um formato que o broker compreenda, além de interpretar as respostas para o usuário.

O broker, por sua vez, recebe as requisições da aplicação e as encaminha ao dispositivo correto. Ele garante que as solicitações sejam direcionadas para o componente apropriado. Uma vez recebida a resposta do dispositivo, o broker a envia de volta para a aplicação, que a apresenta de forma compreensível ao usuário. Dessa maneira, a comunicação entre a aplicação, o broker e o dispositivo permite uma interação fluida e eficaz.

Como mostra o Diagrama de Sequência 1, em um cenário ideal, o dispositivo primeiro

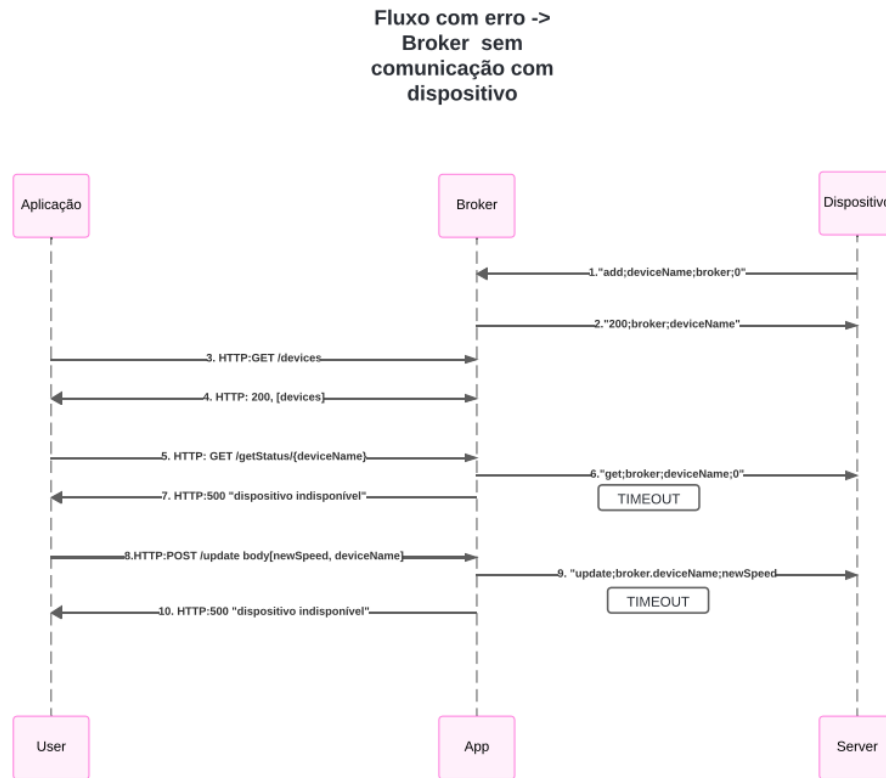
envia uma solicitação para que o broker o identifique, registrando seu endereço (IP e Porta) e seu identificador em uma estrutura de dicionário. A aplicação pode então solicitar ao broker uma lista dos identificadores dos dispositivos registrados, e o broker responde fornecendo todos os identificadores armazenados em seu dicionário. Além disso, a aplicação pode solicitar informações sobre o status atual de um dispositivo específico, bem como atualizar seu estado conforme necessário.

#### Fluxo com sucesso



#### Diagrama de sequência 1: Fluxo de sucesso

Podem ocorrer situações em que o dispositivo ou o broker fiquem indisponíveis. No entanto, há um mecanismo de tratamento de erro para esses casos, que será detalhado posteriormente. No Diagrama de Sequência 2, é possível ver, por exemplo, como o sistema lida com uma situação em que o dispositivo não responde ao broker dentro de um tempo limite de 5 segundos.



**Diagrama de sequência 2:** Fluxo de erro no dispositivo

#### 4.1 O DISPOSITIVO

O dispositivo, desenvolvido em Python, inicialmente se comporta como um cliente, enviando uma solicitação de inclusão ao broker e aguardando uma resposta de confirmação. Para padronizar a comunicação, foi estabelecido que todas as requisições entre o dispositivo e o broker seguiriam um formato específico: “método; remetente; destinatário; informação\_adicional“. As respostas também seguem um padrão semelhante: “status\_da\_solicitação; remetente; destinatário; informação\_adicional“.

Os métodos reconhecidos pelo dispositivo são:

1. add: Solicita ao broker que seja incluído em sua estrutura de armazenamento de dispositivos.
  - a) Exemplo request: “add;device\_name;broker;0“
2. get: É requisitado ao dispositivo o seu estado atual, como resposta devolve no informação adicional o seu estado, sendo “V” referente a velocidade e “C” referente ao consumo:
  - a) Exemplo de response de sucesso: “200;device\_name;broker;V:0,C:0“
  - b) Exemplo de response de erro: “400;device\_name;broker;erro\_mensagem“
3. update: É requisitado a alteração do seu estado e, após fazer as alterações é devolvida a mesma mensagem que é devolvida no método get.

Os códigos de respostas são:

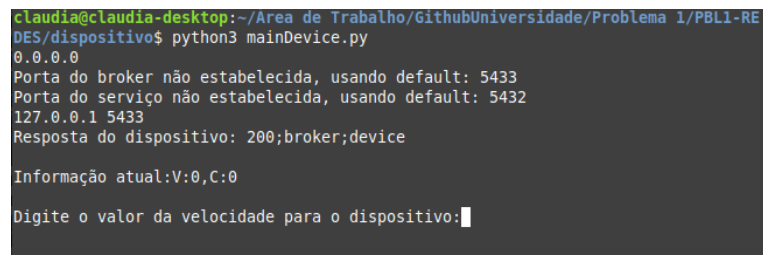
1. 200: Indica sucesso na requisição, seguindo o mesmo princípio do HTTP.
2. 400: Indica um erro na requisição, geralmente devido a informações incorretas. A explicação para o erro será fornecida no campo de informação adicional.

O envio de informações do dispositivo para o broker é feito via socket configurado para usar o protocolo UDP, o que significa que não há garantia de entrega das mensagens. Por isso, ao adicionar um dispositivo ao broker, é solicitada uma resposta de confirmação para assegurar que o broker recebeu a solicitação. No entanto, para outras mensagens de resposta, não há garantia de que elas cheguem ao destino, pois o UDP não possui mecanismos integrados de confirmação de entrega.

Para receber requisições do broker, o dispositivo é configurado para usar sockets com o protocolo TCP. Isso garante uma comunicação confiável, pois o TCP oferece mecanismos para confirmar que o dispositivo recebeu efetivamente as requisições do broker. Assim, há um maior nível de segurança e confiabilidade no processo de comunicação.

É importante ressaltar que, ao receber uma requisição do broker, o dispositivo precisa desserializar a mensagem. As mensagens recebidas são apenas bits, então a desserialização, realizada por um método do Python, converte a mensagem em uma string legível.

Das funções presentes no dispositivo, se inclui a mudança da sua velocidade de ventilação, por se tratar de um ventilador, como também a contabilidade do consumo de acordo com o tempo em que o dispositivo estava ligado. Embora possa ter seu estado alterado remotamente, também permite entradas via terminal. Dessa forma, o estado pode ser modificado não apenas através da aplicação, mas também diretamente pela interface do dispositivo.



```
claudia@claudia-desktop:~/Area de Trabalho/GithubUniversidade/Problema 1/PBL1-RE
DES/dispositivo$ python3 mainDevice.py
0.0.0.0
Porta do broker não estabelecida, usando default: 5433
Porta do serviço não estabelecida, usando default: 5433
127.0.0.1 5433
Resposta do dispositivo: 200;broker;device

Informação atual:V:0,C:0

Digite o valor da velocidade para o dispositivo:█
```

**Figura 3:** Menu do dispositivo

## 4.2 O BROKER

O serviço broker atua como um intermediário entre a aplicação e o dispositivo, se comportando como um cliente-servidor. Para garantir uma comunicação confiável, o broker utiliza o protocolo TCP para enviar mensagens ao dispositivo, assegurando o recebimento. Ao mesmo tempo, ele recebe mensagens do dispositivo via protocolo UDP.

A comunicação entre o broker e o dispositivo segue um formato padronizado para garantir que as mensagens sejam compreendidas por ambas as partes. Os métodos de comunicação utilizados são os seguintes:

Os métodos na interface broker/dispositivo:

1. add:Solicitação do dispositivo para ser incluído no sistema de armazenamento do broker.
  - a) Exemplo de resposta com sucesso: "200;broker;device\_name;0"
  - b) Exemplo de resposta com erro: "400;broker;device\_name;mensagem\_de\_erro"
2. get: O broker realiza a requisição para o dispositivo informar o seu estado atual:
  - a) Exemplo de requisição: "get;broker;device\_name;0"
3. update: O broker pede ao dispositivo que modifique seu estado atual, especificando a nova velocidade de ventilação como um número inteiro.
  - a) Exemplo de requisição: "update;broker;device\_name;nova\_velocidade"

No entanto, as requisições feitas pela aplicação utilizam o protocolo HTTP. Para gerenciar essas requisições, o framework Flask foi empregado, permitindo ao broker receber e processar as solicitações da aplicação. Os métodos disponíveis no broker para essas requisições são os seguintes:

1. Endpoint `"/getState/<deviceId>"` usando o método `"GET"`: Recebe o ID do dispositivo como um parâmetro de caminho e retorna um JSON contendo informações sobre o nome, velocidade e consumo do dispositivo.
2. Endpoint `"/devices"` usando o método `"GET"`: Retorna uma lista de dispositivos registrados.
3. Endpoint `"/update"` usando o método `"POST"`: Recebe no corpo da requisição um JSON contendo o nome do dispositivo e a nova velocidade que deve ser definida. Como resposta, o broker devolve um JSON com informações sobre o nome, velocidade e consumo do dispositivo.

Para assegurar um desempenho eficiente e lidar com o volume de requisições ao broker, foi implementado threads para processar as solicitações da aplicação e do dispositivo de forma independente. Com essa abordagem, cada thread pode tratar sua própria requisição sem causar bloqueios no serviço broker, evitando possíveis gargalos e melhorando o tempo de resposta. O uso de threads permite que o sistema suporte uma alta carga de trabalho, garantindo desempenho consistente mesmo sob alta demanda.

#### 4.3 A APLICAÇÃO

A aplicação foi desenvolvida em Java usando o framework Spring Boot, e sua função é lidar com as solicitações do usuário por meio de um terminal. Ao iniciar a aplicação, é exibida uma lista de dispositivos disponíveis, permitindo que o usuário visualize e escolha um dispositivo, como mostrado na Figura 4.



```
1
Buscando dispositivos...
[device1]
Dispositivos:
1. device1
Opção:
1. Buscar dispositivo
2. Selecionar dispositivo
0. Encerrar aplicação
```

**Figura 4:** Tela inicial

Depois de selecionar um dispositivo, a aplicação mostra suas informações atuais e oferece opções para interagir com ele, como atualizar para obter seu estado mais recente, ajustar a velocidade ou retornar ao menu anterior para escolher outro dispositivo na lista.

```
Estado atual do device1  Ultima atualizacao:01/05/2024 10:45:49 PM
Velocidade: V:1
Consumo: C:2.7e-06
Opções:
1. Atualizar
2.Alterar a velocidade
3.Voltar
1
Estado atual do device1  Ultima atualizacao:01/05/2024 10:46:19 PM
Velocidade: V:1
Consumo: C:3.0725186000000004
Opções:
1. Atualizar
2.Alterar a velocidade
3.Voltar
```

**Figura 5:** Estado atual

Logo no início, a aplicação faz uma requisição ao broker para obter a lista de dispositivos, usando o endpoint “/devices“. Da mesma forma, quando o usuário solicita a busca de novos dispositivos, a aplicação envia uma nova requisição ao broker. Quando um dispositivo é selecionado, a aplicação solicita ao broker o estado atual do dispositivo por meio do endpoint “/getState/<deviceId>“, e usa o mesmo processo para atualizar o dispositivo ao pedido do usuário. Para alterar a velocidade, o usuário informa o novo valor, e a aplicação passa essa informação ao broker usando o endpoint “/update“, incluindo os detalhes do dispositivo no corpo da requisição.

Após a realização de testes, foi confirmado que o sistema possui mecanismos robustos para lidar com a perda de conexão com o broker e/ou o dispositivo. O sistema é capaz de se recuperar mesmo quando há desconexões, como a remoção de cabos, restabelecendo a conexão automaticamente quando o cabo é reconectado, sem afetar a experiência do usuário. A aplicação não é interrompida abruptamente ou de forma não planejada durante essas situações.

## 5 CONCLUSÃO

A comunicação via protocolo UDP não é confiável porque não garante que a mensagem enviada será recebida pelo destinatário. No entanto, o UDP é útil por oferecer velocidade na transmissão, especialmente quando a perda ocasional de mensagens não é um problema crítico. Já o TCP garante a entrega das mensagens, pois estabelece uma conexão segura antes de enviar qualquer dado, proporcionando maior certeza de que a informação chegou ao destino.

Para o desenvolvimento do sistema, o uso de frameworks foi essencial para abstrair várias configurações necessárias para a comunicação via protocolo HTTP. O Flask e o Feign Client desempenharam papéis fundamentais, facilitando a configuração e permitindo uma comunicação fluida entre o broker e a aplicação.

De acordo com a lógica cliente-servidor, a aplicação é o único elemento que se comporta inteiramente como cliente, já que apenas faz requisições e não disponibiliza recursos ou informações. Por outro lado, embora o dispositivo funcione principalmente como servidor, ele também age como cliente ao fazer uma solicitação inicial ao broker. Já o broker desempenha ambos os papéis: atua como cliente ao solicitar informações do dispositivo e como servidor ao responder às solicitações da aplicação.

Para garantir o bom funcionamento do sistema, o uso de threads foi necessário para evitar gargalos durante as requisições. Sem threads, uma solicitação precisaria esperar que outra fosse concluída, prejudicando o desempenho. A solução adotada foi criar uma nova thread para cada requisição, mas há outras abordagens, como limitar o número de threads e colocar as requisições em uma fila, processando-as por ordem de chegada.

Embora a abordagem escolhida tenha suas desvantagens, como o risco de criar um número indeterminado de threads, levando ao possível travamento do sistema, ela oferece uma resposta rápida e direta, já que cada solicitação é tratada individualmente. Para minimizar os riscos, é essencial monitorar e gerenciar a criação de threads para evitar sobrecarga no sistema.

A criação de um protocolo de comunicação personalizado na camada de aplicação foi necessária para que tanto o dispositivo quanto o broker compreendam as requisições e sigam um padrão comum de comunicação, tanto para solicitações quanto para respostas. Esse protocolo permite uma compreensão mútua entre as partes, garantindo consistência e clareza na transmissão de dados.

O sistema pode ser aprimorado com uma interface mais amigável para a aplicação, proporcionando uma experiência mais intuitiva para o usuário. Outra melhoria seria a atualização automática do estado do dispositivo, eliminando a necessidade de atualizações manuais. Além disso, seria benéfico ajustar o broker para usar um número fixo de threads, evitando o risco de sobrecarga e garantindo a estabilidade do sistema. Essas melhorias podem aumentar a usabilidade e a confiabilidade do sistema.

## REFERÊNCIAS

KUROSE, J. F.; ROSS, K. W. Redes de computadores e a internet. *São Paulo: Person*, v. 28, 2006. Citado 2 vezes nas páginas 2 e 3.

TANENBAUM, A. S. *Redes de computadoras*. [S.l.]: Pearson educação, 2003. Citado 2 vezes nas páginas 2 e 3.