Problema 1: Internet das Coisas

Disney Carvalho Andrade Filho

Departamento de Tecnologia – Universidade Estadual de Feira de Santana (UEFS) 44036–900 – Feira de Santana – Bahia

dcandrade@ecomp.uefs.br

*A IoT, concebida por Kevin Ashton, integra objetos físicos à Internet, permitindo a automação de sistemas através dos dados gerados por sensores, preenchendo lacunas entre o mundo físico e digital, A equipe da startup de IoT, adquirida por uma empresa de dispositivos industriais, está desenvolvendo um serviço broker para facilitar a comunicação entre dispositivos e aplicações, utilizando o TCP/IP como base, O desenvolvedor Disney criou um sistema de gerenciamento para uma piscina com três dispositivos (bomba, termômetro e medidor de volume), juntamente com um broker e uma aplicação, permitindo visualizar e controlar a temperatura e o volume da água, bem como ligar e desligar a bomba, Utilizando uma abordagem cliente-servidor, foi implementado um sistema com um broker que estabelece comunicação bidirecional entre os dispositivos por meio do protocolo TCP/IP, recebendo dados via UDP e os direcionando para a aplicação, possibilitando um eficiente monitoramento e controle da piscina, Foi desenvolvido um sistema com um broker que realiza comunicação bidirecional com os dispositivos via TCP/IP, recebendo informações através do protocolo UDP e encaminhando esses dados para a aplicação..*

**1. Introdução**

Uma empresa que desenvolve dispositivos industriais com capacidades de rede adquiriu sua startup de IoT para aproveitar sua expertise em middleware distribuído. Sua equipe deve criar um serviço broker baseado em TCP/IP que facilite a comunicação eficiente entre esses dispositivos e as aplicações, permitindo troca de mensagens e cooperação entre eles.

Em um esforço para melhorar a gestão de piscinas, um desenvolvedor Disney implementou um sistema avançado de monitoramento e controle utilizando IoT, integrando uma bomba, um termômetro e um medidor de volume de água. Para facilitar a comunicação, foi desenvolvido um broker que utiliza TCP/IP e recebe dados via UDP, repassando-os para a aplicação de controle. Isso permite visualização em tempo real e controle remoto da bomba, gerenciando múltiplos dispositivos simultaneamente e proporcionando uma gestão automatizada e eficiente da piscina.

A metodologia envolveu uma arquitetura cliente-servidor onde o broker atua como intermediário, garantindo comunicação estável via TCP/IP e recebendo informações de baixa latência via UDP. O broker processa e encaminha esses dados para a aplicação, apresentando-os de forma acessível para o usuário. Esta abordagem resultou em um sistema robusto e escalável, eficiente para monitoramento e controle da piscina.

Os resultados mostraram que o sistema conseguiu integrar os dispositivos da piscina com a aplicação de forma eficiente. Utilizando TCP/IP para comunicação bidirecional estável e UDP para recepção rápida de dados, o broker permitiu monitoramento preciso e controle remoto da bomba, gerenciando múltiplos dispositivos simultaneamente e melhorando a gestão automatizada da piscina.

*O restante deste trabalho está organizado da seguinte forma. A Seção 2 apresenta os fundamentos teóricos relacionados com o problema. A Seção 3 discute aspectos de implementação e testes da solução. A Seção 4 apresenta e avalia os resultados. No final da Seção 5, as conclusões e reflexões sobre os conhecimentos adquiridos*.

**2. Fundamentação Teórica ´**

Para implementar a comunicação TCP/IP em Python, os desenvolvedores utilizam a biblioteca padrão `socket`, que fornece uma interface de programação de sockets de baixo nível. Esta biblioteca oferece suporte para os protocolos TCP (Transmission Control Protocol) e UDP (User Datagram Protocol), permitindo que os desenvolvedores escolham o protocolo mais adequado para suas necessidades específicas.

Ao utilizar o `socket` em Python, os desenvolvedores podem criar e gerenciar conexões de rede de forma flexível e eficiente. Por exemplo, é possível criar sockets de cliente e servidor, estabelecer conexões com outros dispositivos em uma rede local ou na internet e configurar opções de comunicação, como timeout e reutilização de endereços.

O protocolo TCP é amplamente utilizado em situações onde a confiabilidade na entrega dos dados é crucial. Ele garante que os pacotes de dados sejam enviados e recebidos na ordem correta e pode retransmitir pacotes perdidos. Isso é especialmente útil em aplicações como transferência de arquivos, comunicação entre processos e acesso remoto a servidores.

Por outro lado, o protocolo UDP é preferido em cenários onde a velocidade e a eficiência na transmissão são mais importantes do que a garantia de entrega. Ele é comumente utilizado em aplicações como streaming de áudio e vídeo, jogos online e comunicações em tempo real, onde pequenas perdas de dados podem ser toleradas em troca de um desempenho mais rápido.

Além disso, o `socket` em Python permite que os desenvolvedores implementem uma ampla variedade de funcionalidades de rede, como transferência de arquivos, comunicação entre processos, streaming de mídia e comunicação em tempo real. Isso proporciona flexibilidade para desenvolver soluções personalizadas que atendam às necessidades específicas de cada projeto.

**3. Metodologia, Implementação e Testes**

Para solucionar o problema proposto, foi utilizado Python para implementar a comunicação TCP/IP e UDP. Inicialmente, o programa cria uma classe chamada `Broker`, responsável por gerenciar todo o funcionamento do broker. Esta classe possui parâmetros definidos após a sua instanciação, incluindo o IP do broker e as portas dos sockets TCP e UDP.

O broker utiliza três threads daemon para gerenciar as comunicações. A primeira thread é responsável por criar um servidor UDP. Ela inicializa um socket UDP e aguarda mensagens dos dispositivos. Quando uma mensagem é recebida, o broker responde ao dispositivo confirmando a recepção dos dados e adiciona a mensagem a uma lista para tratamento posterior.

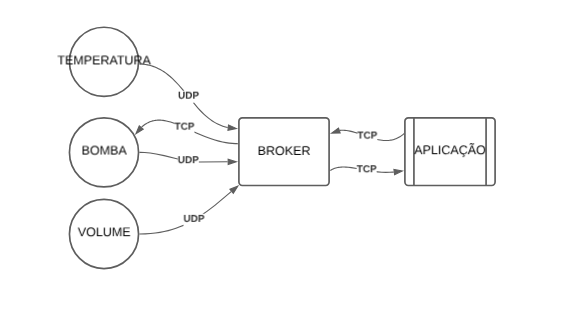
A segunda thread cria um servidor TCP que aceita conexões TCP. Para cada conexão TCP aceita, uma nova thread é criada para gerenciar a comunicação, ficando responsável por receber mensagens TCP.

A terceira thread percorre a lista de mensagens UDP, tratando-as e adicionando os dados a um dicionário de dispositivos. Este dicionário mantém um registro atualizado das mensagens e dos dispositivos conectados, facilitando a gestão e comunicação eficiente entre eles.

Os dispositivos comunicam-se com o broker via UDP, enviando seus dados. O broker, por sua vez, confirma a recepção dos dados e os encaminha para a aplicação utilizando TCP/IP, garantindo que a aplicação possua todos os dados de todos os dispositivos conectados ao broker.

Quando a aplicação envia um comando para um dispositivo, o broker processa o comando e o envia para o dispositivo via TCP/IP. Esta abordagem permite que a comunicação entre dispositivos e a aplicação seja gerenciada de maneira eficiente e organizada, utilizando as vantagens específicas de cada protocolo (UDP para a comunicação inicial rápida e TCP para a comunicação confiável e ordenada entre o broker e a aplicação).

Dessa forma, a implementação da comunicação TCP/IP e UDP em Python, por meio da classe `Broker` e o uso de threads, possibilita a solução do problema proposto, permitindo a comunicação confiável e eficiente entre dispositivos e a aplicação através do broker.

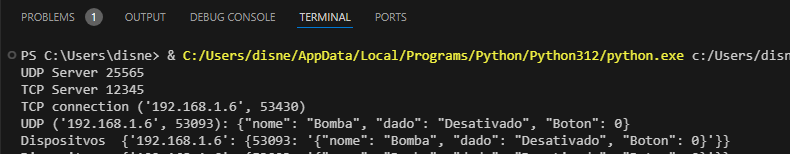


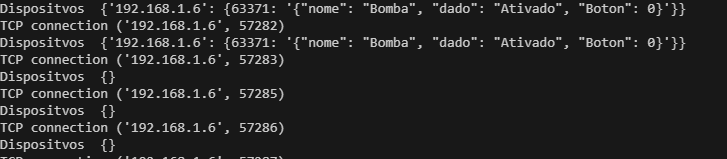
**Figura 1. Diagrama da comunicação do sistema.**

O broker tem uma série de tratamentos de erro de conexão para garantir que, quando a conexão com os dispositivos ou com a aplicação seja interrompida, o fluxo de execução do código prossiga sem problemas. Dessa forma, se alguma conexão com o broker for interrompida, ele continuará funcionando perfeitamente.

Os dispositivos comunicam-se com o broker via UDP, enviando seus dados. O broker responde aos dispositivos confirmando a recepção dos dados e, posteriormente, encaminha esses dados para a aplicação via TCP/IP, assegurando que a aplicação possua todas as informações dos dispositivos conectados ao broker. Quando a aplicação envia um comando para um dispositivo, o broker processa o comando e o envia para o dispositivo via TCP/IP.

Quando um dispositivo é conectado, seus dados são tratados e armazenados no dicionário de dispositivos. Se um dispositivo é desconectado, seus dados são removidos do dicionário. Essa abordagem permite que o sistema mantenha uma comunicação eficiente e resiliente, garantindo a continuidade operacional mesmo em casos de falhas de conexão.

**Figura 2. Conexões do broker.**

****

**Figura 3. Dispositivo desconectado do broker.**

**4. Resultados e Discussões**

Os testes realizados demonstraram que o programa funciona conforme o previsto, validando a eficácia da solução implementada. Durante os testes, os dispositivos comunicaram-se com o broker via UDP, enviando seus dados de forma eficiente. O broker, por sua vez, respondeu corretamente aos dispositivos, confirmando a recepção dos dados, e encaminhou essas informações para a aplicação via TCP/IP, garantindo que a aplicação tivesse acesso a todos os dados dos dispositivos conectados.

A capacidade do broker de manter a continuidade operacional mesmo em situações de falha de conexão foi um ponto crucial verificado durante os testes. Quando as conexões com os dispositivos ou com a aplicação foram deliberadamente interrompidas, o broker continuou a funcionar perfeitamente, demonstrando sua resiliência e robustez. Esse comportamento é essencial em sistemas de rede que dependem de comunicação constante e confiável.

Adicionalmente, o gerenciamento de dispositivos foi testado com sucesso. Quando um dispositivo era conectado, seus dados eram corretamente tratados e armazenados no dicionário de dispositivos. Da mesma forma, ao desconectar um dispositivo, seus dados eram prontamente removidos do dicionário, mantendo o registro atualizado e consistente.

Os resultados obtidos suportam a eficácia da abordagem adotada, onde o uso combinado de protocolos UDP e TCP/IP em Python, juntamente com o tratamento de erros de conexão, garantiu uma comunicação robusta e eficiente. A arquitetura baseada em threads permitiu a gestão simultânea de múltiplas conexões e a manipulação adequada das mensagens recebidas.

Esses resultados indicam que a solução proposta atendeu aos objetivos de garantir uma comunicação confiável e contínua entre dispositivos e aplicação. A escolha do protocolo UDP para a comunicação inicial rápida e do protocolo TCP para a transmissão confiável de dados críticos mostrou-se acertada. Além disso, a implementação do tratamento de erros assegurou que o sistema permanecesse operacional mesmo diante de falhas de conexão.

A abordagem adotada não apenas solucionou o problema inicial, mas também estabeleceu uma base sólida para futuros aprimoramentos e expansões. A arquitetura modular e a separação clara de responsabilidades entre as diferentes partes do sistema facilitam a manutenção e a adição de novas funcionalidades, se necessário.

Em conclusão, os testes confirmaram que o programa funciona como planejado, atingindo os objetivos de garantir uma comunicação eficiente, confiável e contínua entre dispositivos e aplicação. A solução implementada demonstrou ser robusta e resiliente, adequada para cenários de rede onde a comunicação constante e a capacidade de recuperação de falhas são essenciais.

**5. Conclusão**

O problema proposto exigia a implementação de uma solução robusta e eficiente para a comunicação entre dispositivos e uma aplicação, utilizando os protocolos TCP/IP e UDP em Python. Este desafio proporcionou um aprendizado significativo sobre os fundamentos de redes de computadores e a prática de programação em rede.

Através do desenvolvimento do broker, ficou claro como os protocolos TCP e UDP podem ser utilizados de forma complementar para alcançar uma comunicação rápida e confiável. O uso de UDP permitiu uma comunicação inicial rápida entre dispositivos e o broker, enquanto o TCP garantiu a transmissão confiável de dados críticos entre o broker e a aplicação. A implementação de tratamentos de erro de conexão demonstrou ser crucial para manter a continuidade operacional, destacando a importância de considerar a resiliência em sistemas de rede.

O aprendizado obtido com este projeto é valioso no contexto da disciplina de Redes de Computadores, pois aborda diretamente a aplicação prática dos conceitos teóricos estudados. A compreensão de como configurar e gerenciar sockets, lidar com diferentes protocolos de comunicação e implementar mecanismos de recuperação de falhas são habilidades fundamentais para qualquer profissional de TI que lida com sistemas distribuídos e comunicação de rede.

Além disso, a experiência de desenvolver uma solução que integra várias threads para gerenciar múltiplas conexões simultâneas é uma competência importante para a programação paralela e concorrente, áreas essenciais em sistemas modernos.

No contexto da formação profissional, este projeto destacou a importância da comunicação de rede eficiente e resiliente, que é uma habilidade altamente valorizada no mercado de trabalho. A capacidade de criar sistemas que garantam a continuidade operacional mesmo em face de falhas de conexão é crucial para o desenvolvimento de aplicações robustas e confiáveis, especialmente em áreas como IoT, sistemas distribuídos e serviços de internet.

Em conclusão, este projeto não só proporcionou um entendimento dos protocolos de comunicação TCP/IP e UDP, mas também destacou a importância de considerar a resiliência e a eficiência na comunicação de rede. O conhecimento e as habilidades adquiridas com esta experiência são fundamentais tanto para o desenvolvimento acadêmico quanto para a formação profissional, preparando-me para enfrentar desafios complexos em sistemas de rede e comunicação.