INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE MINAS GERAIS - *CAMPUS* IBIRITÉ
ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Matheus Yago Silva Fonseca

PROTÓTIPO DE SOFTWARE SIMULADOR DE REDE MULTI-CLIENTE

Ibirité - MG

MATHEUS YAGO SILVA FONSECA

PROTÓTIPO DE SOFTWARE SIMULADOR DE REDE MULTI-CLIENTE

Trabalho de conclusão de curso apresentado ao Curso de Engenharia de Controle e Automação do Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais - *Campus* Ibirité para a obtenção do título de Bacharel em Engenharia de Controle e Automação.

Orientador: Prof.Dr.Carlos Dias Da Silva Junior

Coorientador: Prof.Dr.Thiago Henrique Barbosa de Carvalho Tavares

Ibirité - MG

F676p Fonseca, Matheus Yago Silva.
Protótipo de software simulador de rede multi-cliente. [recurso eletrônico] / Matheus Yago Silva Fonseca. – Ibirité, MG, 2024.
66 p. : il. color.

Orientador: Prof. Carlos Dias da Silva Junior.
Coorientador: Prof. Thiago Henrique Barbosa de Carvalho Tavares.
Trabalho de Conclusão de Curso (Graduação) – Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais, *Campus* Ibirité, Bacharelado em Engenharia de Controle e Automação, 2024.

1. Rede de computador - Protocolos 2. Python (Linguagem de programação de computador). 3. Simulação (Computadores). I. Silva Junior, Carlos Dias da. II. Tavares, Thiago Henrique Barbosa de Carvalho. III. Instituto Federal de Minas Gerais. *Campus* Ibirité. IV. Título.

CDD 004.6

Catalogação: Viviane Barbosa Andrade - Bibliotecária - CRB-6/2819

Matheus Yago Silva Fonseca

PROTÓTIPO DE SOFTWARE SIMULADOR DE REDE MULTI-CLIENTE

Trabalho de conclusão de curso apresentado ao Curso de Engenharia de Controle e Automação do Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais - *Campus* Ibirité para a obtenção do título de Bacharel em Engenharia de Controle e Automação.

Aprovado em 25/setembro/2024 pela banca examinadora:

Documento assinado digitalmente
gov.br
CARLOS DIAS DA SILVA JUNIOR
Data: 12/11/2024 19:50:45-0300
Verifique em https://validar.iti.gov.br

Prof.Dr.Carlos Dias Da Silva Junior - IFMG (Orientador)

Documento assinado digitalmente
gov.br
THIAGO HENRIQUE BARBOSA DE CARVALHO TA
Data: 14/11/2024 14:10:27-0300
Verifique em https://validar.iti.gov.br

Prof.Dr.Thiago Henrique Barbosa de Carvalho Tavares - IFMG (Coorientador)

Documento assinado digitalmente
gov.br
DIEGO HENRIQUE DE SOUZA CHAVES
Data: 12/11/2024 20:32:53-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Diego Henrique de Souza Chaves - IFMG (avaliador)

Dedico este trabalho primeiramente a Deus, por me conceder as condições, a inteligência e a sabedoria necessárias para sua realização.

Aos meus pais, Maurício Fonseca Silva e Eliane Geralda da Silva Fonseca, expresso minha eterna gratidão pelo direcionamento, motivação e disciplina que me proporcionaram, fundamentais para a conclusão deste projeto.

À minha namorada, Ana Letícia Amaral de Oliveira e ao meu irmão Pedro Augusto Silva Fonseca dedico meu agradecimento por seu suporte constante e por estar ao meu lado durante todo o processo, oferecendo encorajamento e compreensão.

Sem o apoio e a presença de cada um de vocês, este trabalho não teria sido possível.

AGRADECIMENTOS

Agradeço primeiramente a Deus, por me conceder força, sabedoria e fé ao longo de toda a minha jornada acadêmica.

Agradeço a toda a minha família, em especial aos meus pais, Maurício e Eliane, e ao meu irmão, Pedro Augusto, por acreditarem em mim e pelo incentivo constante na realização deste trabalho.

Expresso minha profunda gratidão à minha namorada, Ana Letícia, pelo apoio incondicional e por estar ao meu lado durante todo o processo.

Agradeço imensamente ao meu orientador, Prof. Carlos Dias da Silva Júnior, pela orientação, paciência e dedicação ao longo deste projeto. Ao meu coorientador, Prof.Thiago Henrique Barbosa de Carvalho Tavares, sou grato pelo suporte e pelos valiosas percepções que enriqueceram este trabalho.

Por fim, agradeço a todos que contribuíram de alguma forma para a realização deste trabalho.

“Foi o tempo que dedicaste à tua rosa que a fez tão importante"(Antoine de Saint-Exupér).

RESUMO

Este trabalho tem como objetivo o desenvolvimento de um simulador de rede focado no protocolo DHCP, utilizando a linguagem de programação Python, conhecida por sua simplicidade, clareza e pela vasta coleção de bibliotecas disponíveis. A escolha do Python se justifica pela sua versatilidade e facilidade de uso, características que o tornam uma excelente ferramenta tanto para iniciantes quanto para profissionais mais experientes. O simulador foi projetado para proporcionar um ambiente controlado e seguro, ideal para testar e validar diferentes configurações de rede, realizar testes de estresse e permitir a interação direta de usuários sem experiência prévia, sem comprometer a integridade de uma infraestrutura física real. Além disso, o simulador oferece uma grande flexibilidade, possibilitando a criação e experimentação de diversos cenários de rede, tornando-se uma ferramenta valiosa para o aprendizado prático e a análise técnica avançada de redes de computadores. O ambiente de desenvolvimento escolhido para este projeto é o Visual Studio Code (VS Code), devido à sua extensibilidade, suporte ao linting e integração robusta com Python, fatores que facilitam o processo de desenvolvimento, depuração e análise do sistema, assegurando maior eficiência e controle na implementação do simulador.

Palavras-chave: DHCP. Redes de computadores. Python.

ABSTRACT

This work aims to develop a network simulator focused on the DHCP protocol, using the Python programming language, known for its simplicity, clarity, and vast collection of available libraries. The choice of Python is justified by its versatility and ease of use, characteristics that make it an excellent tool for both beginners and experienced professionals. The simulator is designed to provide a controlled and secure environment, ideal for testing and validating different network configurations, performing stress tests, and allowing direct interaction of users without prior experience, without compromising the integrity of a real physical infrastructure. Additionally, the simulator offers great flexibility, enabling the creation and experimentation of various network scenarios, making it a valuable tool for practical learning and advanced technical analysis of computer networks. The development environment chosen for this project is Visual Studio Code (VS Code), due to its extensibility, linting support and robust integration with Python, factors that facilitate the development, debugging and analysis process of the system, ensuring greater efficiency and control in the implementation of the simulator.

Keywords: DHCP.Computer networks. Python.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

OSI	International Standards Organization
RM-OSI	Reference Model for Open Systems Interconnection
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
UDP	User Datagram Protocol
IP	Internet Protocol
DHCP	Dynamic Host Configuration Protocol
MAC	Media Access Control.
LAN	Local Area Network
Wi-Fi	Wireless Fidelity
WANs	Wide Area Network
IFMG	Instituto Federal de Minas Gerais

SUMÁRIO

1 INTRODUÇÃO

A rápida expansão da tecnologia tem remodelado profundamente a maneira como nos comunicamos e interagimos no mundo contemporâneo. Assim como a correspondência postal uma vez foi a principal forma de comunicação, hoje os computadores e a internet são os pilares fundamentais da comunicação humana. Neste cenário em constante evolução, a conexão IP e o switch desempenham papéis vitais na facilitação da comunicação de dados e na construção de redes de computadores robustas e eficientes.

Com a crescente complexidade das redes de computadores, a necessidade de automatização e gerenciamento eficiente de recursos se tornou imprescindível. Neste contexto, o Dynamic Host Configuration Protocol (DHCP) emerge como uma solução indispensável para a atribuição dinâmica de endereços IP e outros parâmetros de configuração de rede, permitindo que dispositivos conectados a uma rede operem de forma eficiente sem a necessidade de configuração manual. O DHCP simplifica o gerenciamento de redes, particularmente em ambientes que demandam a conexão de múltiplos dispositivos, como em corporações, instituições de ensino e data centers.

A implementação e o estudo de um simulador de rede com foco no DHCP proporcionam benefícios significativos, especialmente em termos de redução de custos e segurança. O uso de um simulador elimina a necessidade de uma infraestrutura física, evitando os custos associados à aquisição e à manutenção de equipamentos de rede. Além disso, permite que testes de estresse e cenários complexos sejam conduzidos sem o risco de danificar componentes físicos, garantindo que as operações em ambientes de produção não sejam interrompidas.

Os simuladores de rede têm se mostrado ferramentas fundamentais para o desenvolvimento e análise de redes em ambientes seguros e controlados. De acordo com os estudos de (Cavin Yoav Sasson, 2002), os simuladores permitem a avaliação de protocolos de rede em cenários variados, proporcionando uma visão detalhada de como diferentes configurações podem afetar a performance e a segurança da rede. Dessa forma, simuladores são amplamente utilizados tanto no mercado quanto na academia para treinamento, teste de novas tecnologias e otimização de redes antes de sua implementação em ambientes reais.

Neste contexto, o desenvolvimento de um simulador de rede com foco no protocolo DHCP é uma ferramenta essencial para profissionais de TI e engenheiros de redes. Ele permite não apenas o estudo teórico, mas também a prática segura e econômica do gerenciamento de redes. Este trabalho tem como objetivo a construir um simulador de rede em Python, com ênfase na funcionalidade do DHCP, oferecendo uma ferramenta didática para a análise do comportamento desse protocolo em diferentes cenários e a aplicação segura em redes de computadores.

1.1 Objetivos

Este capítulo apresenta uma introdução aos objetivos relacionados a este trabalho, bem como a abordagem adotada e a necessidade identificada no contexto das redes de computadores, que influenciaram diretamente a sua realização para criação do simulador.

1.1.1 *Objetivo geral*

Construir um simulador de rede para abordagem DHCP.

1.1.2 *Objetivos específicos*

- Criar um servidor e clientes simulados;
- Abordar TCP para transmissão de dados;
- Verificar se o servidor simulado aceita a conexão DHCP;
- Demonstrar as conexões efetivadas pelos clientes no servidor simulado.

1.2 Justificativa

No ambiente corporativo moderno, as redes de computadores desempenham um papel central nas operações diárias das empresas, sendo fundamentais para a comunicação interna, transações comerciais e acesso a serviços essenciais. Dada à crescente complexidade dessas redes e a dependência das organizações em uma infraestrutura de TI robusta, torna-se imperativo que as soluções implementadas sejam testadas e validadas de forma exaustiva antes de serem aplicadas em ambientes de produção.

Nesse contexto de acordo com (Gallo; Hancock, 2002), ferramentas de simulação são essenciais para testar cenários de rede e evitar impactos reais nas operações, sendo o DHCP um dos componentes críticos que precisam de testes rigorosos, principalmente em redes de grande escala onde falhas podem comprometer a conectividade.O protocolo DHCP, por sua função crítica de atribuição dinâmica de endereços IP, é um componente essencial a ser testado, especialmente em redes de grande escala, onde erros na configuração podem levar a falhas generalizadas de conectividade.

Diante de tais abordagens este trabalho, busca suprir essa necessidade por meio do desenvolvimento de um simulador de rede focado na interação com o protocolo DHCP. A proposta é criar uma ferramenta prática, acessível e de baixo custo,devido à necessidade de uma ferramenta para rodar o código proposto, que possa ser utilizada por empresas para testar a implementação e o comportamento do DHCP em diferentes cenários, incluindo testes em larga escala. Além

disso, o simulador proposto poderá ser expandido para suportar outros protocolos de rede como DNS(Domain Name System) e NAT(Network Address Translation), oferecendo uma plataforma versátil para o teste e a validação de infraestruturas de TI.

A construção desse simulador, embora inicialmente voltada para a aplicação prática em ambientes corporativos, também tem um forte potencial didático. Ele pode ser utilizado como uma ferramenta de ensino, proporcionando aos alunos um ambiente interativo onde podem observar e experimentar o comportamento dos protocolos em uma rede simulada sem causar danos a uma rede real e através das simulações testa erros que podem acontecer no uso de sistemas de redes. Isso facilita a visualização de conceitos teóricos de forma prática, permitindo que os estudantes compreendam melhor a dinâmica de alocação de endereços IP pelo DHCP, o estabelecimento de conexões via TCP, e a função do IP na comunicação entre dispositivos.

2 REVISÃO BIBLIOGRÁFICA

É de conhecimento que muitas pessoas têm uma compreensão limitada em relação ao sistemas de redes e tal fato deve ser considerado como apontado por (Macedo *et al.*, 2018). Embora o uso diário dos serviços oferecidos pelas redes de computadores seja comum e termos técnicos sejam frequentemente empregados, muitos usuários desconhecem o significado real desses termos. Essa lacuna de conhecimento pode impactar a eficácia na utilização e gerenciamento de redes.

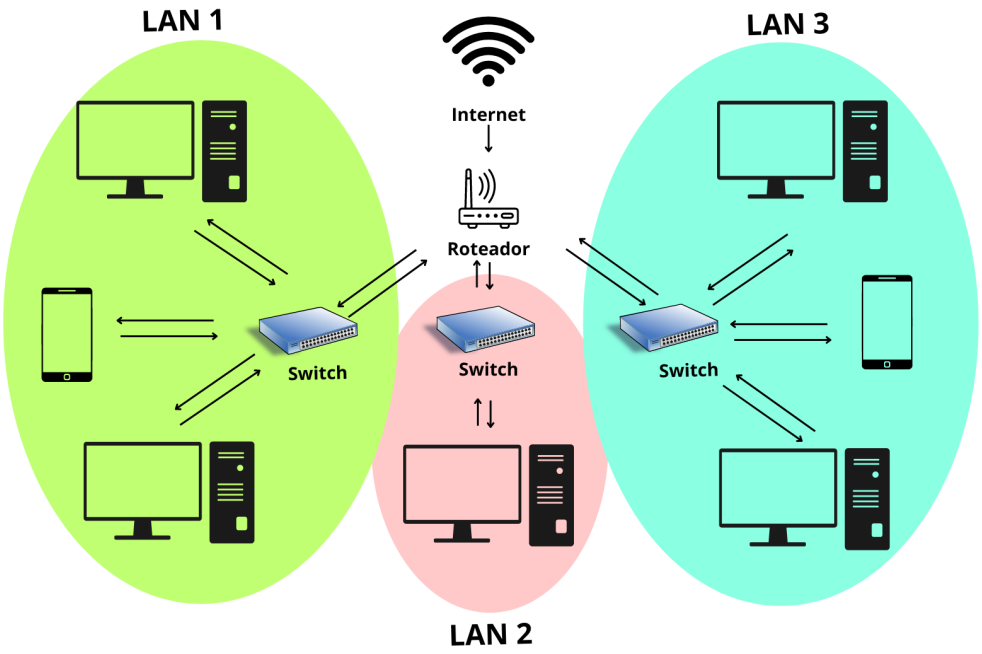
Além disso, (Comer, 2015) destaca a existência de uma distinção significativa na percepção da Internet entre programadores e usuários comuns. Para que a Internet funcione de maneira eficiente são necessários dispositivos especializados, como os roteadores, que desempenham a função crucial de interligar redes e transferir pacotes de dados entre elas. Essa distinção é fundamental para o entendimento das operações internas das redes de computadores e sua estrutura básica.

Considerando esses conceitos e a necessidade de um ambiente de teste aplicável tanto no contexto corporativo quanto em ambientes privados, conforme discutido anteriormente, nesta seção será explorado o conceito básico de alguns componentes essenciais que integram uma rede de computadores, vale ressaltar que esse capitulo será usado para explicação e as imagens não refletem o simulador e quantidade de conexões. A compreensão desses componentes é vital para o desenvolvimento de soluções práticas e para o aprimoramento do ensino e do gerenciamento das redes.

2.1 Rede de computadores

Conforme abordado anteriormente, o termo "Redes de Computadores"refere-se a um sistema de computadores interligados por uma tecnologia comum, como pode ser verificado na figura 1.

Figura 1 – Estrutura de uma rede de computadores interconectada por roteadores e switches.



Fonte: Elaborado pelo autor (2024). .

Na Figura 1 é apresentada uma configuração típica de uma rede de computadores composta por três LANs (Local Area Networks) onde cada uma conectada através de switches, com comunicação entre elas mediada por um roteador central que também oferece conectividade à Internet. As setas presentes na figura representam o fluxo de informações entre os diferentes dispositivos, destacando a troca de dados que ocorre tanto dentro das redes locais (LANs) quanto entre as redes distintas, mediada pelo roteador.

Os conceitos de LAN, switches e roteadores, fundamentais para a estrutura de redes de computadores, serão explicados de forma detalhada posteriormente neste trabalho. Esses componentes desempenham papéis essenciais na organização e operação das redes, permitindo que dispositivos conectados possam comunicar-se de maneira eficiente e segura.

Através desses componentes temos um sistema que integra múltiplas LANs, permitindo que redes locais interajam não apenas entre si, mas também com redes externas, como a Internet. A comunicação pode ser realizada tanto por conexões cabeadas quanto sem fio (como o Wi-Fi), utilizando diferentes meios físicos, como cabos de cobre ou fibras ópticas, e tecnologias de transmissão de dados.

Conforme destacado por (Tanenbaum; Wetherall, 2011),as redes de computadores podem variar em tamanho, forma e funcionalidade. Comumente, redes locais (LANs) são interconectadas para formar redes maiores e mais complexas, conhecidas como WANs (Wide Area Networks). Essas conexões são estabelecidas por meio de diferentes tecnologias, incluindo cabos, fibras ópticas, e até mesmo satélites, com a Internet sendo o exemplo mais proeminente de

uma rede global composta por múltiplas redes interconectadas.

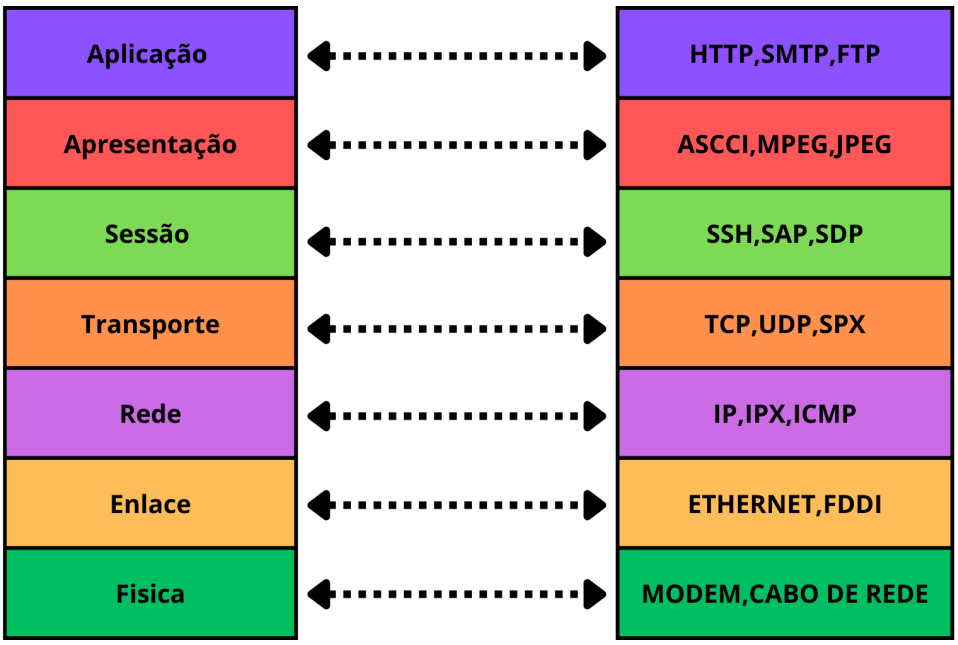
De maneira geral, as redes de computadores consistem em componentes físicos (hardware), como switches, roteadores,cabos e componentes lógicos (software e protocolos), que controlam a comunicação e a troca de dados. Quando um computador está conectado a uma rede, ele se torna parte de um sistema maior que facilita a troca e o compartilhamento de informações, proporcionando uma interação robusta e eficiente entre os dispositivos conectados. (Tanenbaum; Wetherall, 2011).

2.1.1 Camada OSI

Com o avanço das redes de comunicação e a crescente necessidade de interconexão entre computadores, tornou-se evidente a importância da padronização dessas redes, como descrito em (Feres, 2006).

A ISO (International Standards Organization) reconheceu essa necessidade e iniciou um esforço global para criar uma arquitetura padronizada que pudesse ser aplicada em diversas redes de comunicação. O resultado desse esforço foi o desenvolvimento do Modelo de Referência para a Interconexão de Sistemas Abertos, conhecido como RM-OSI (Reference Model for Open Systems Interconnection) (Stemmer, 2001),conforme apresentado na figura 2.

Figura 2 – Camada OSI.



Fonte: Elaborado pelo autor (2024). .

Esse modelo, ilustrado na Figura 2, serve como um guia para a construção de redes de comunicação que possam interagir de forma eficiente, independentemente das diferenças entre

os equipamentos envolvidos. O RM-OSI permite que diferentes sistemas, fabricados por diversos fornecedores, possam se comunicar de maneira transparente para o usuário final.

O modelo OSI é composto por sete camadas distintas, cada uma desempenhando uma função específica para garantir que os dados sejam transmitidos de maneira eficiente e sem erros entre dispositivos. Essas camadas, começando da mais básica até a mais complexa, são: Física, Enlace, Rede, Transporte, Sessão, Apresentação e Aplicação.

A Camada Física é responsável pela transmissão dos dados brutos sob a forma de sinais elétricos, ópticos ou de rádio, através de meios físicos como cabos ou ondas de rádio. Em seguida, a Camada de Enlace assegura que os dados transmitidos pela camada física cheguem ao seu destino sem erros, gerenciando as conexões entre dispositivos diretamente conectados.

A Camada de Rede cuida do roteamento dos dados entre diferentes redes, determinando a melhor rota para que as informações cheguem ao destino final. A Camada de Transporte, por sua vez, garante que os dados sejam entregues de forma confiável e em ordem correta, estabelecendo uma comunicação de ponta a ponta. Subindo no modelo, a Camada de Sessão é responsável por estabelecer, gerenciar e finalizar as sessões de comunicação entre aplicações em dispositivos diferentes. A Camada de Apresentação traduz os dados entre o formato usado pela aplicação e o formato necessário para a transmissão, realizando funções como criptografia e compressão de dados.

Por fim, a Camada de Aplicação oferece serviços de rede diretamente aos usuários, suportando aplicações como navegação na web, envio de e-mails e transferência de arquivos. Cada uma dessas camadas interage com as camadas adjacentes, garantindo que a comunicação seja realizada de maneira coesa e eficiente.

Portanto, o modelo OSI não apenas padroniza a comunicação em redes de computadores, mas também assegura que sistemas heterogêneos possam se interconectar e trocar informações de forma harmoniosa, independentemente das diferenças em seus fabricantes ou tecnologias assim como descrito por (Zimmermann, 1980). Nesse projeto, serão trabalhadas as camadas de Rede e Transporte, que servirão como base para o desenvolvimento das demais camadas subsequentes, reforçando a importância dessas camadas na construção de uma comunicação eficiente e confiável em redes de computadores.

2.1.2 Topologia de rede

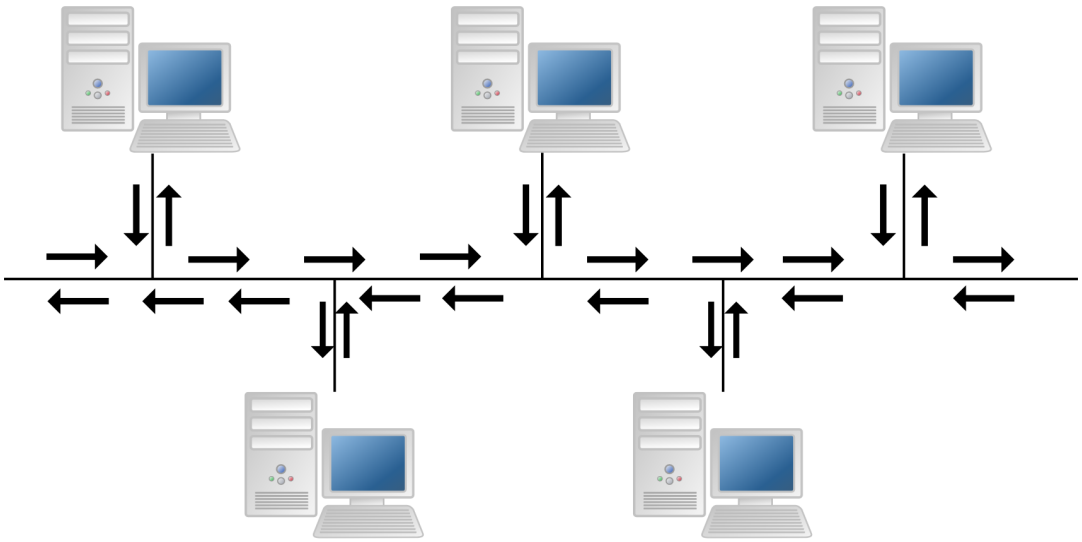
Mesmo a camada OSI tendo sua definição no início de 1980 é imprescindível a discursão abordada nesse trabalho contendo redes a topologia de redes, que como referido por (Fernando *et al.*, 1995) o sistema de comunicação é constituído de um arranjo topológico interligando as máquinas por via de enlaces físicos (meios de transmissão) juntamente com um conjunto de regras que organizam a comunicação. A topologia de rede é a estrutura física ou lógica que define como os dispositivos em uma rede de comunicação estão conectados uns aos outros. Ela deter-

mina a forma como os dados são transmitidos entre os dispositivos, influenciando diretamente o desempenho, a confiabilidade e a escalabilidade da rede. As escolhas de topologia podem variar com base nas necessidades de uma rede específica, como o número de dispositivos, a segurança desejada e o custo. Diferentes tipos de topologias oferecem diferentes benefícios e desafios, e entender suas características é fundamental para a construção de redes eficientes.Agora serão abordadas algumas das principais topologias de rede.

2.1.2.1 Topologia em barramento

A topologia em barramento é uma das mais simples e antigas. Nessa configuração, todos os dispositivos da rede compartilham um único cabo de comunicação central, chamado de barramento. Quando um dispositivo envia um sinal, ele é transmitido para todos os outros dispositivos na rede, mas apenas o destinatário apropriado o processa. Essa abordagem é vantajosa em termos de simplicidade e economia de cabos, mas apresenta limitações significativas. Se o cabo principal falhar, toda a rede será interrompida. Além disso, à medida que mais dispositivos são adicionados à rede, o desempenho tende a diminuir, já que o barramento compartilhado se torna um gargalo para o tráfego de dados. A imagem abaixo descreve essa topologia mostrando o trafego de informação, representado pelas seta, por somente por um central de informação.

Figura 3 – Topologia de Barramento.



Fonte: Elaborado pelo autor (2024). .

2.1.2.2 Topologia em anel

A topologia de rede em anel é uma configuração em que os dispositivos são conectados em um circuito fechado, formando um anel. Cada dispositivo se conecta ao próximo, criando

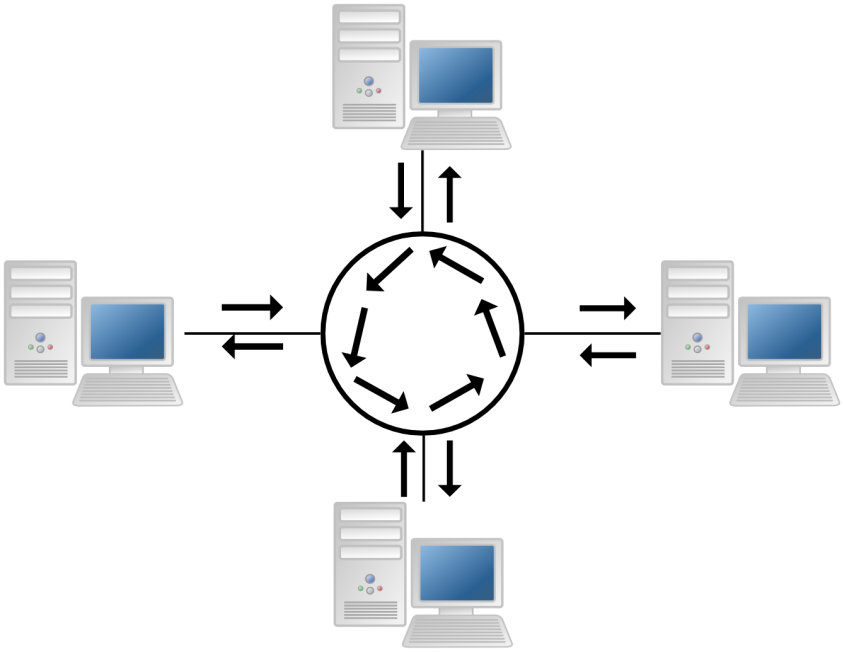
uma estrutura circular. Nesta topologia, os dados trafegam em uma única direção ao longo do anel, passando de um dispositivo para o outro até alcançar o destino final, como demostrado na imagem abaixo. Cada dispositivo participa do roteamento dos dados, o que torna o projeto dos repetidores mais simples, reduzindo a complexidade dos protocolos de comunicação necessários para garantir a entrega correta e em sequência.

Uma característica importante dessa topologia é o uso de tokens, que são pacotes especiais de controle que circulam continuamente pela rede. Um dispositivo só pode enviar dados quando possui o token, o que evita colisões, já que apenas um dispositivo pode transmitir por vez. Esse mecanismo de controle torna o tráfego de dados mais organizado e eficiente, especialmente em redes menores.

No entanto como descrito por (Fernandes, 2015), a topologia em anel tem algumas limitações. A falha de um único dispositivo ou de uma conexão pode comprometer toda a rede, já que o tráfego de dados depende de cada dispositivo estar funcional para manter o fluxo contínuo. Em redes com topologia de anel unidirecional, essa falha interrompe completamente a comunicação. Para mitigar esse problema, algumas redes implementam anéis duplos, permitindo que os dados circulem em ambas as direções, proporcionando uma forma de redundância. Dessa forma, se ocorrer uma falha em um ponto, os dados podem continuar a circular no sentido oposto, garantindo a continuidade da comunicação.

Embora a topologia em anel tenha sido amplamente utilizada no passado, hoje é uma configuração em desuso, especialmente com o surgimento de topologias mais robustas e flexíveis, como a topologia estrela. No entanto, o uso de anéis duplos ainda pode ser encontrado em redes que exigem maior resiliência e controle de tráfego

Figura 4 – Topologia de Anel.

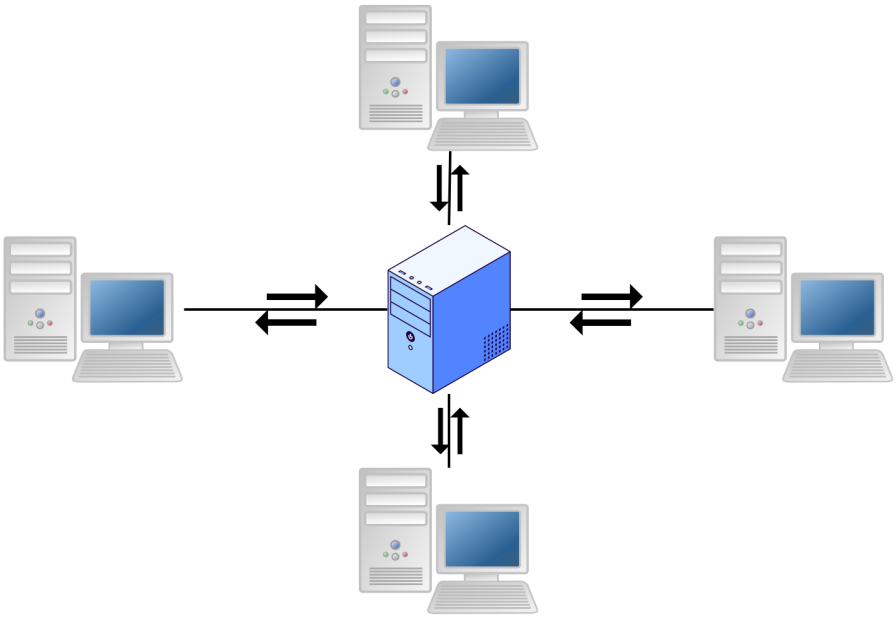


Fonte: Elaborado pelo autor (2024). .

2.1.2.3 Topologia em estrela

A topologia em estrela é amplamente utilizada devido à sua simplicidade de gerenciamento e robustez. Nessa configuração todos os dispositivos da rede são conectados a um dispositivo central, como um switch, que gerencia o tráfego entre os dispositivos. Isso significa que os dados enviados de um dispositivo passam primeiro pelo dispositivo central, como demonstrado na figura a baixo, antes de serem entregues ao destino. A grande vantagem dessa abordagem como descrito em (Kurose; Ross, 2017) é que, se um dispositivo falhar, ele não afeta o restante da rede. Contudo, a dependência do dispositivo central pode ser uma fraqueza; se ele falhar, toda a rede será comprometida.

Figura 5 – Topologia de estrela.

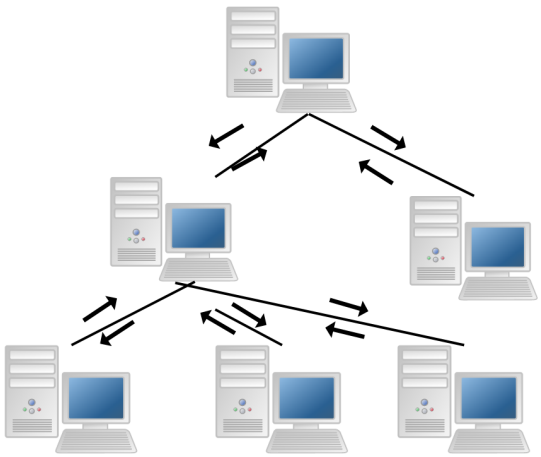


Fonte: Elaborado pelo autor (2024). .

2.1.2.4 Topologia em árvore

A topologia em árvore é uma combinação da topologia em estrela com uma estrutura hierárquica. Nessa configuração grupos de dispositivos em estrela são interconectados de maneira hierárquica, formando uma árvore. A estrutura em árvore é adequada para redes maiores, como as corporativas, onde a rede precisa ser segmentada em grupos menores para facilitar a administração e a manutenção. A falha em um segmento principal pode afetar os dispositivos conectados a ele, mas os outros segmentos permanecem operacionais, permitindo um certo grau de resiliência. A imagem a seguir demonstra exatamente essa hierarquia através da topologia em árvore, onde dispositivos trocam informações somente com dispositivos de hierarquia maior ou que estejam a baixo na hierarquia e conectados entre si.

Figura 6 – Topologia em árvore.



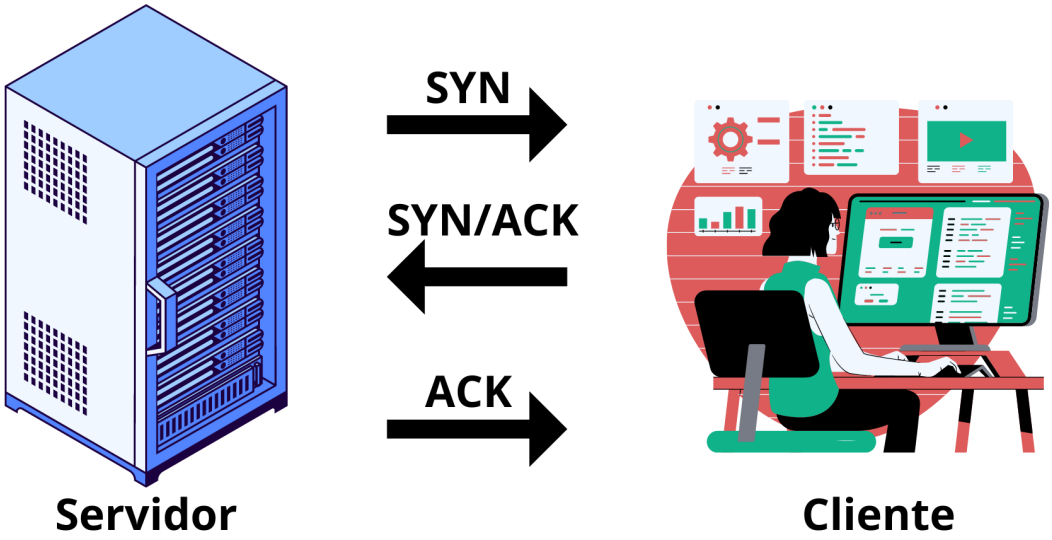
Fonte: Elaborado pelo autor (2024). .

2.2 Transmission Control Protocol

Após a abordagem da camada OSI e algumas das principais tipologias de rede, podemos entrar em um dos conceitos fundamentais dessa estrutura os protocolos de rede, especialmente na camada de transporte: o protocolo TCP (Transmission Control Protocol). O TCP é um dos principais protocolos que operam nessa camada no modelo TCP/IP(Transmission Control Protocol/Internet Protocol), sendo amplamente utilizado para garantir a comunicação confiável e ordenada entre dispositivos em uma rede, como a internet. Diferente de protocolos que não asseguram a integridade dos dados, como o UDP (User Datagram Protocol), o TCP se destaca por sua capacidade de estabelecer uma conexão sólida entre o cliente e o servidor antes de permitir a transmissão de qualquer dado, como destacado por (Tanenbaum; Wetherall, 2011). Esse processo garante que todas as informações enviadas sejam recebidas corretamente e na sequência certa, proporcionando uma comunicação robusta e livre de erros.

O funcionamento do TCP é exemplificado de maneira clara pelo processo inicial de estabelecimento de conexão, conhecido como o "handshake de três vias", como representado na figura 7 a seguir.

Figura 7 – Comunicação TCP.



Fonte: Elaborado pelo autor (2024). .

Esse processo é essencial para a criação de uma conexão confiável, onde a comunicação entre o cliente e o servidor é rigorosamente sincronizada. A comunicação se inicia quando o cliente envia uma solicitação ao servidor por meio de um pacote chamado SYN (Synchronize) como demostrado na figura através de uma seta do cliente para o servidor. Este pacote tem a função de iniciar a conversa entre os dois dispositivos e sincronizar os números de sequência, que são utilizados posteriormente para rastrear a ordem dos pacotes de dados enviados.

Ao receber o pacote SYN, o servidor responde com um pacote SYN-ACK, como demonstrado na imagem com uma seta do servidor para o cliente. Esse pacote cumpre dois papéis essenciais: o SYN indica que o servidor está pronto para estabelecer a conexão, enquanto o ACK (Acknowledgment, ou reconhecimento) confirma que o servidor recebeu o pacote SYN do cliente. Em outras palavras, o pacote SYN-ACK é uma aceitação formal da tentativa de conexão por parte do cliente, além de um reconhecimento de que o primeiro pacote foi recebido com sucesso.

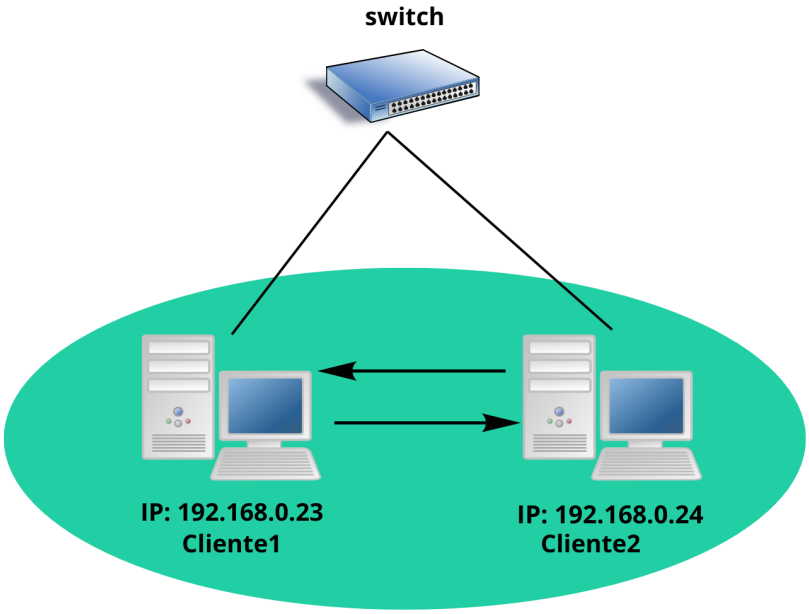
Para completar o processo de handshake, o cliente responde ao servidor com um pacote ACK,como demonstrado na terceira seta (do cliente para o servidor) da figura a cima. Este último pacote serve para confirmar que a resposta SYN-ACK foi recebida com sucesso e que ambos os dispositivos estão prontos para começar a troca de dados reais. Nesse ponto, a conexão está oficialmente estabelecida e qualquer dado enviado entre o cliente e o servidor estará protegido pelo protocolo TCP, que continuará monitorando a sequência dos pacotes e retransmitindo qualquer dado que não seja entregue corretamente.

Como descrito por (Tanenbaum; Wetherall, 2011), esse processo de handshake não é apenas uma formalidade, mas um mecanismo crucial que garante que a comunicação seja confiável e ordenada. Ele assegura que qualquer dado enviado será recebido sem erros e na sequência correta; o que é especialmente importante em aplicações onde a confiabilidade é crítica, como no carregamento de páginas web, envio de e-mails ou transferência de arquivos. Além disso, o TCP oferece mecanismos para corrigir falhas durante a transmissão, como pacotes perdidos ou duplicados, assegurando que o destinatário receba exatamente o que foi enviado, sem quaisquer omissões ou duplicidades.

2.3 Endereçamento IP

O Protocolo de Internet (IP) é um dos principais protocolos da camada de rede do modelo TCP/IP (Transmission Control Protocol/Internet Protocol). Ele é responsável pelo endereçamento e encaminhamento dos pacotes de dados entre dispositivos em uma rede. A figura a seguir ilustra esse processo.

Figura 8 – Comunicação através do IP.

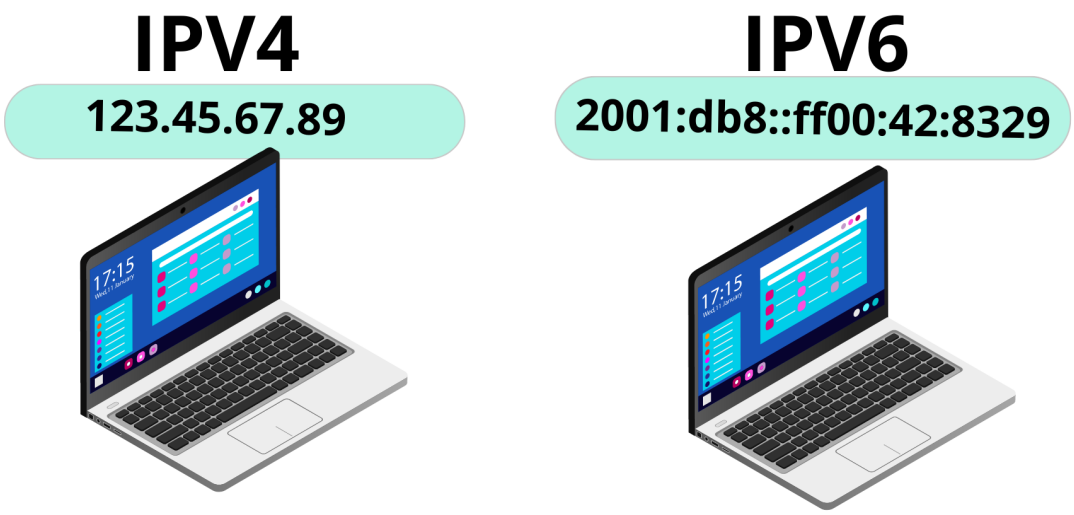


Fonte: Elaborado pelo autor (2024). .

Cada dispositivo em uma rede IP possui um endereço IP único, que consiste em duas partes: a parte de rede e a parte de host. O endereço IP permite que os pacotes sejam encaminhados corretamente do remetente ao destinatário. Além disso, o IP realiza fragmentação e remontagem de pacotes quando necessário, para garantir que os dados sejam entregues independentemente do tamanho dos pacotes originais.

Assim como demonstrado na figura 9, existem duas versões principais do protocolo IP: IPv4 e IPv6. O IPv4 utiliza endereços de 32 bits, resultando em aproximadamente 4,3 bilhões de endereços únicos. Mas como descrito em (Bessani *et al.*, 2017), com o crescimento exponencial de dispositivos conectados à internet, a quantidade de endereços disponíveis no IPv4 começou a se esgotar, o que gerou a necessidade de uma solução para acomodar o número crescente de endereços necessários. Foi para resolver essa limitação que o IPv6 foi desenvolvido.

Figura 9 – Demonstração IPV4 e IPV6.



Fonte: Elaborado pelo autor (2024). .

O IPV6 utiliza endereços de 128 bits, permitindo uma quantidade praticamente ilimitada de endereços. Essa expansão é fundamental para suportar a continuidade do crescimento da internet e a conectividade de uma ampla gama de dispositivos e tecnologias emergentes. Além das melhorias no espaço de endereçamento, o IPV6 também oferece melhorias em termos de eficiência no roteamento, segurança e suporte a novas funcionalidades que não estavam disponíveis no IPV4 como referenciado por (Huitema, 1998).

(Kurose; Ross, 2017) descreve que, enquanto o IP se encarrega do endereçamento e roteamento dos pacotes, o Transmission Control Protocol (TCP) complementa essa função ao garantir a entrega confiável dos dados. O TCP trabalha em conjunto com o IP para estabelecer uma conexão confiável, verificando a entrega de cada pacote e retransmitindo-o, se necessário, para garantir a integridade e a ordem dos dados transmitidos.

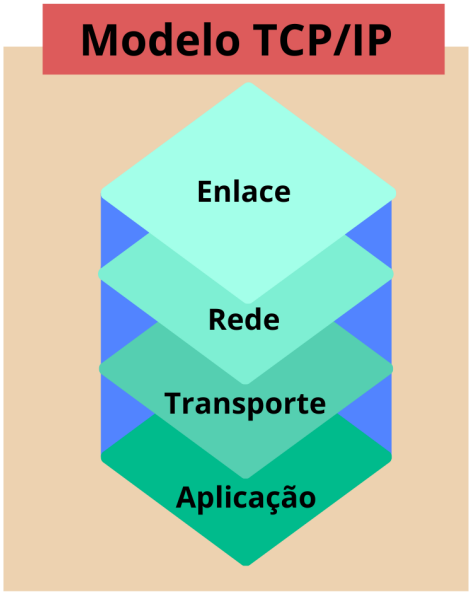
Dessa forma, o modelo TCP/IP fornece uma estrutura robusta para a comunicação na internet, onde o IP cuida do caminho que os dados percorrem e o TCP assegura que esses dados cheguem corretamente ao destino.

2.4 Transmission Control Protocol/Internet Protocol

O Protocolo TCP/IP, ou Transmission Control Protocol/Internet Protocol, é a base da comunicação na Internet e em muitas redes privadas. Desenvolvido nos anos 70 pelo Departamento de Defesa dos Estados Unidos, o TCP/IP tornou-se o padrão universal para a transmissão de dados. Ele é composto por um conjunto de protocolos TCP e IP que trabalham juntos para permitir a comunicação entre dispositivos de rede. De acordo com (STALLINGS, 2005) esse protocolo também é implementado em roteadores e este passa dados de uma rede para outra a partir de uma rota de origem até o destino.

O TCP/IP é dividido em quatro camadas principais como aborado por (Tanenbaum; Wetherall, 2011), conforme demonstrado na imagem abaixo: a camada de enlace, a camada de rede, a camada de transporte e a camada de aplicação. Cada camada tem um papel específico e interage com as outras para garantir a transmissão de dados de maneira eficiente e confiável.

Figura 10 – Camadas TCP/IP.



Fonte:Fonte: Elaborado pelo autor (2024). .

1. Camada de Enlace: A camada de enlace é responsável pela transmissão de dados entre dispositivos na mesma rede física. Ela lida com o endereçamento físico (MAC(Media Access Control)), a detecção e correção de erros e a forma como os dados são formatados para transmissão. Nesta camada, os pacotes de dados são encapsulados em quadros, que são enviados para o próximo salto na rede.

2. Camada de Rede: A camada de rede é responsável pelo roteamento dos pacotes de dados entre diferentes redes. O principal protocolo desta camada é o Protocolo de Internet (IP),

que trata do endereçamento e encaminhamento dos pacotes. O IP determina o melhor caminho para os pacotes através da rede e pode fragmentar e remontar pacotes quando necessário

3. Camada de Transporte: A camada de transporte é responsável pela entrega confiável dos dados entre os dispositivos finais. O Transmission Control Protocol (TCP) é o principal protocolo desta camada, oferecendo serviços como controle de fluxo, controle de congestionamento e correção de erros para garantir que os dados sejam entregues corretamente e na ordem correta

4. Camada de Aplicação: A camada de aplicação é a camada mais próxima do usuário e lida com a interface direta entre o software de aplicação e a rede. Protocolos nesta camada incluem HTTP (Hypertext Transfer Protocol) para a web, FTP (File Transfer Protocol) para transferência de arquivos e SMTP (Simple Mail Transfer Protocol) para e-mails. Esta camada garante que os dados sejam formatados e interpretados de acordo com as necessidades da aplicação final, proporcionando uma interface para que os usuários interajam com a rede.

Após definição e explicação sobre conceito base de protocolos e IP é viável antes de voltar a referir sobre meios de hardware em redes de computadores, abordar mais um conceito de protocolo que será tema principal retratado nesse projeto, o DHCP.

2.5 Protocolo DHCP

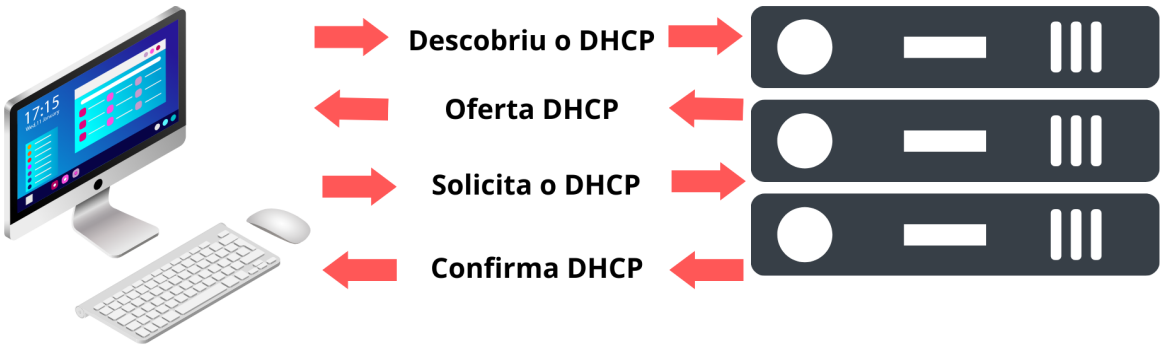
O Protocolo de Configuração Dinâmica de Host (DHCP) é um protocolo de rede essencial que permite a atribuição automática de endereços IP e outras configurações de rede para dispositivos conectados. Como (L.L.Marques; Costa, 2016) descreve, este protocolo simplifica a gestão de redes, permitindo que dispositivos se configurem automaticamente, sem a necessidade de intervenção manual.

Quando um dispositivo, como um computador ou smartphone, se conecta a uma rede, ele inicia o processo de obtenção de um endereço IP enviando uma solicitação DHCP Discover. Esta mensagem é transmitida para toda a rede local, em busca de um servidor DHCP que possa fornecer as informações necessárias para a configuração do dispositivo. Um servidor DHCP que receba essa solicitação responde com uma mensagem DHCP Offer, oferecendo um endereço IP específico ao dispositivo solicitante, junto com outras informações importantes, como a máscara de sub-rede, o gateway padrão e os servidores DNS.

O dispositivo, ao receber a oferta do servidor DHCP, responde com uma mensagem DHCP Request, confirmando que aceita a oferta recebida. Finalmente, o servidor DHCP envia uma mensagem DHCP Acknowledgment, finalizando o processo de configuração e confirmando a alocação do endereço IP e das configurações associadas ao dispositivo. Todo processo descrito

é representado pela imagem abaixo, onde as setas representam as direção dos comandos e informações solicitadas e enviadas.

Figura 11 – Protocolo DHCP.



Fonte: Elaborado pelo autor (2024). .

Esse processo como abordado por (Tanenbaum; Wetherall, 2011), além de automatizar a configuração dos endereços IP, proporciona vários benefícios. Em primeiro lugar, simplifica a administração de redes, eliminando a necessidade de configurar manualmente endereços IP para cada dispositivo, o que é especialmente útil em redes grandes ou dinâmicas, onde dispositivos se conectam e desconectam frequentemente. Além disso, o DHCP facilita a reutilização de endereços IP por meio da alocação dinâmica e de prazos de concessão (lease). Cada endereço IP atribuído possui um prazo de validade, após o qual ele pode ser liberado e reutilizado por outros dispositivos, garantindo uma utilização eficiente do espaço de endereçamento IP disponível.

O DHCP também melhora a flexibilidade e a escalabilidade das redes, facilitando a adição de novos dispositivos. Quando um novo dispositivo é conectado, ele pode ser configurado rapidamente e começar a operar na rede sem demora. Isso é particularmente útil em ambientes corporativos ou em locais onde a conectividade é crítica e os dispositivos precisam ser configurados de forma rápida e eficiente.

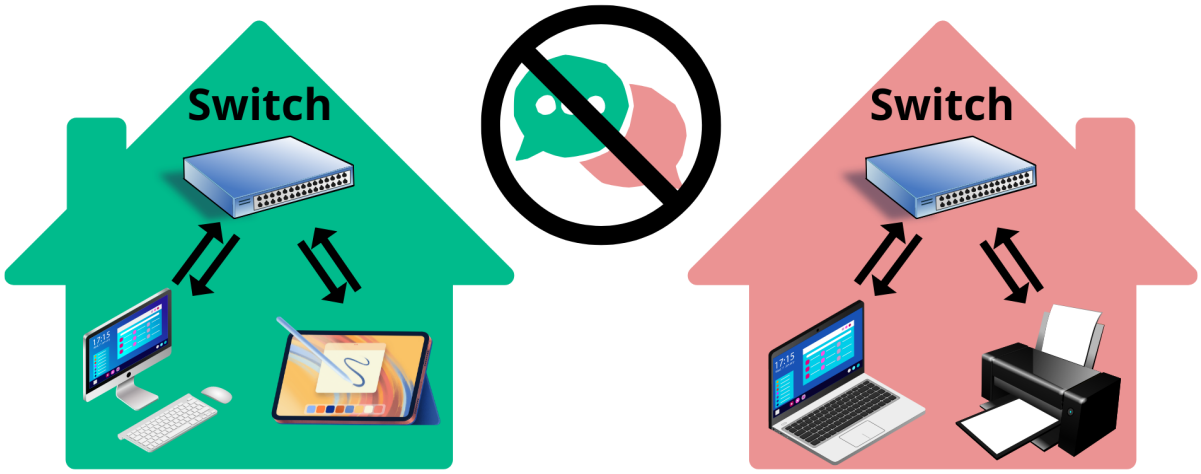
2.6 LAN (Local Area Network)

Após a introdução dos conceitos técnicos relacionados a software, é de suma importância falar sobre os hardwares abordados na figura 1 deste capítulo, começando primeiramente com

um conceito geral de LAN.

Uma LAN (Local Area Network), ou Rede Local, é uma rede que conecta dispositivos em uma área geograficamente limitada. Essa é uma rede que permite a interconexão de dados em uma pequena região (distância entre 100 m e 25 Km), conforme descrito por (Fernando *et al.*, 1995).Para exemplificar podemos citar uma residência, escritório, escola ou campus. O principal objetivo de uma LAN é permitir a comunicação rápida e eficiente entre dispositivos, possibilitando o compartilhamento de recursos como arquivos, impressoras e até mesmo a conexão à internet.

Figura 12 – Representação de LAN.



Fonte: Elaborado pelo autor (2024). .

A figura 8 apresenta duas casas, cada uma com seus dispositivos conectados internamente através de um switch, que é o dispositivo responsável por direcionar o tráfego de dados dentro de uma LAN. Cada casa forma uma rede local individual, onde dispositivos como computadores, tablets, impressoras, entre outros, podem se comunicar entre si de maneira eficiente e rápida.

Em uma LAN, a comunicação entre dispositivos é extremamente rápida, principalmente por causa da proximidade física e da utilização de tecnologias como Ethernet e Wi-Fi, que oferecem alta velocidade de transferência de dados e baixa latência, como descrito por (Stallings, 2013). Isso permite que dispositivos conectados à mesma LAN compartilhem arquivos e recursos, como impressoras, de forma centralizada.

A figura também sugere que, embora haja comunicação interna dentro de cada rede local (cada casa), não há comunicação direta entre as redes das duas casas. O ícone de "não comu-

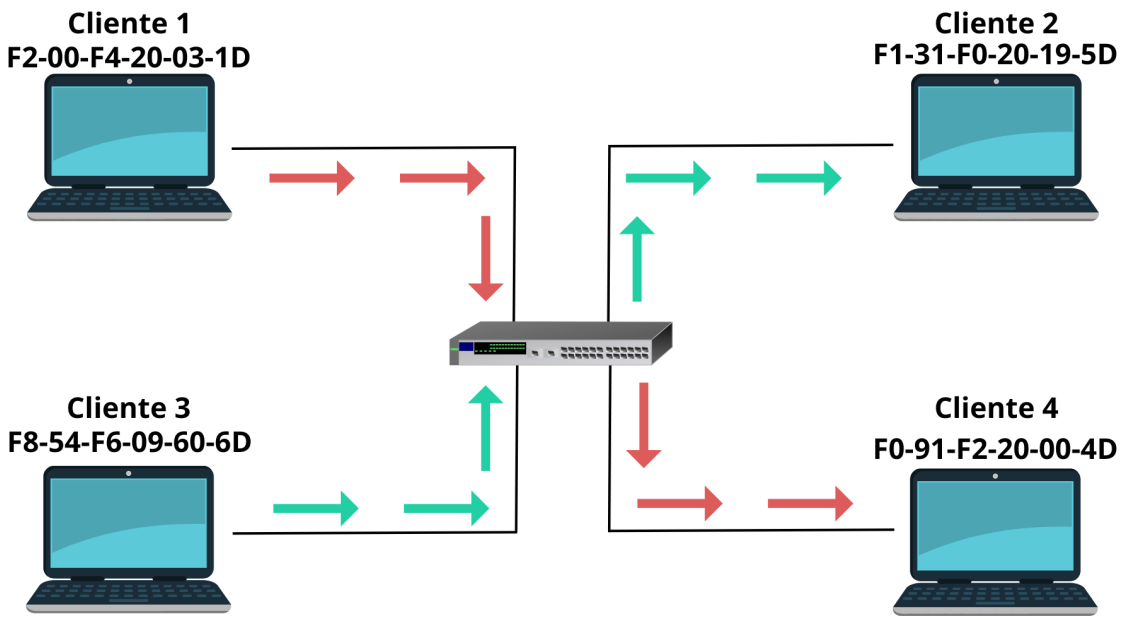
nicação"entre as duas casas representa essa falta de interconexão. Para que essa comunicação fosse possível, seria necessário um roteador ou outro dispositivo que conectasse as duas redes LAN, criando uma rede maior ou permitindo que elas trocassem informações.

Dessa forma, uma LAN pode ser usada para centralizar recursos de maneira eficiente, facilitando a gestão de dispositivos e reduzindo custos, já que todos podem compartilhar o mesmo hardware e software dentro da rede local.

2.7 Switch

Continuando a abordagem sobre a figura 1, neste capitulo, é necessário dar sequência à logica escalável referente aos dispositivos e falar sobre o switch, que é um dispositivo de rede essencial para a comunicação dentro de uma rede local (LAN). Assim como também abordado em (OLIVEIRA, 2022) sua principal função é conectar múltiplos dispositivos, como computadores, impressoras, servidores e gerenciar o fluxo de dados entre eles de maneira eficiente e organizada. Diferente de hubs, que simplesmente retransmitem os dados para todas as portas, os switches são inteligentes e encaminham os dados apenas para o dispositivo de destino específico.

Figura 13 – Representação de envio de dados através de um switch.



Fonte: Elaborado pelo autor (2024). .

Como abordado na figura anterior, o funcionamento de um switch baseia-se no uso de endereços MAC (Media Access Control). Cada dispositivo na rede possui um endereço MAC único, representado pelo endereço de letras e números de cada cliente na imagem. Esse endereço, geralmente fixo e gravado na fábrica, tem 48 bits e é composto por duas partes: a primeira metade

identifica o fabricante do dispositivo e a segunda metade identifica o dispositivo de maneira única dentro daquele fabricante como descrito em (Wright, 2003).

Quando um quadro de dados chega a uma porta do switch, ele lê o endereço MAC de destino contido no quadro. O switch então verifica sua tabela de endereços, que é uma base de dados interna que mapeia endereços MAC para portas específicas. Se o endereço de destino já estiver na tabela, o switch encaminha o quadro de dados representados pelas setas coloridas, diretamente para a porta correspondente. Caso contrário, o switch enviará o quadro para todas as portas (exceto a porta de origem) num processo chamado "flooding", até que o dispositivo de destino responda, permitindo que o switch aprenda em qual porta aquele dispositivo está conectado.

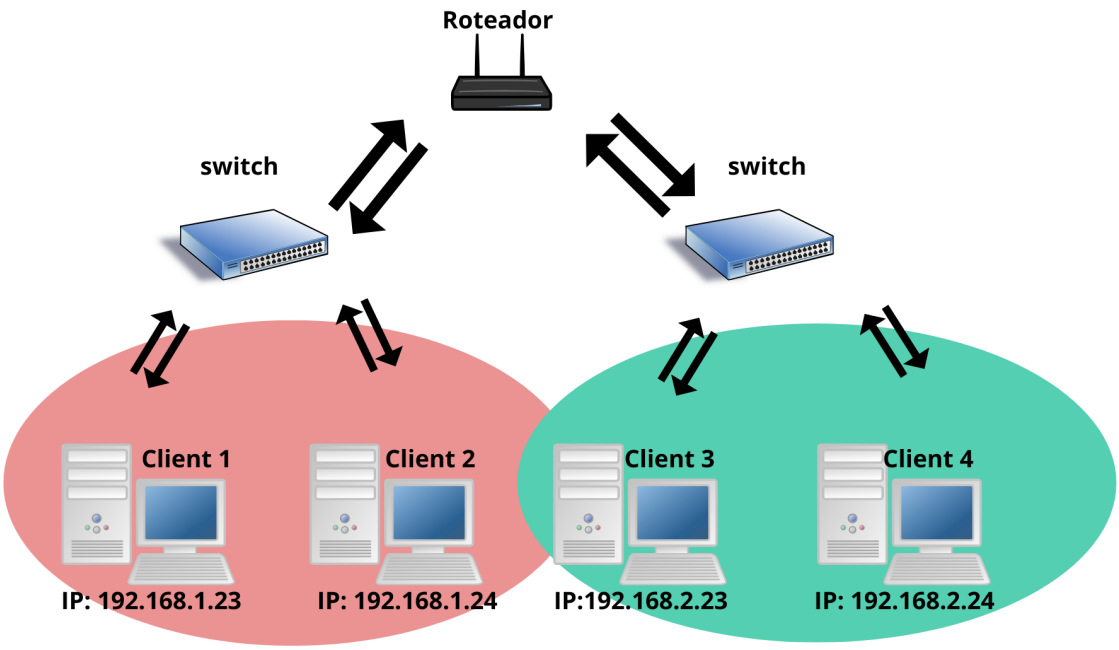
Com o tempo, à medida que o switch aprende mais sobre os dispositivos na rede e seus endereços MAC, ele se torna cada vez mais eficiente no encaminhamento de quadros, reduzindo o tráfego desnecessário e maximizando o desempenho da LAN. A capacidade do switch de armazenar e gerenciar a tabela de endereços MAC é fundamental para seu papel na otimização da comunicação dentro de redes locais.

2.8 Roteador

O roteador é um dispositivo crucial para a comunicação entre redes distintas, atuando na camada de rede do modelo OSI (Camada 3). Sua principal função é receber, analisar e encaminhar pacotes de dados de uma rede para outra. Isso é feito com base nos endereços IP de origem e destino contidos nos pacotes de dados, além de informações armazenadas nas tabelas de roteamento do próprio dispositivo. Esses dados permitem que o roteador identifique o melhor caminho para o pacote, assegurando que ele chegue corretamente ao seu destino final, seja em uma rede local (LAN) ou em uma rede de longa distância (WAN).

Ao contrário de um switch, que trabalha na camada de enlace e é responsável por organizar a comunicação dentro de uma única rede local com base em endereços MAC, o roteador faz a interconexão entre redes diferentes. Ele funciona como uma espécie de "porteiro"da rede, decidindo quais pacotes de dados podem entrar e sair de uma rede, e para onde esses pacotes devem ser direcionados. Quando um roteador recebe um pacote de dados, ele examina o endereço IP de destino e determina a melhor rota para encaminhar esse pacote, muitas vezes passando por vários outros roteadores antes de alcançar o destino final, isso é abordado por (Kurose; Ross, 2017). Esse processo é possível graças a protocolos de roteamento, como OSPF (Open Shortest Path First) ou BGP (Border Gateway Protocol), que definem as regras para a movimentação dos pacotes.

Figura 14 – Comunicação com roteador.



Fonte: Elaborado pelo autor (2024). .

No contexto da imagem vemos um exemplo claro de como o roteador funciona em conjunto com outros dispositivos de rede. O roteador está no centro, conectado a dois switches, que por sua vez estão conectados a grupos de clientes em redes diferentes. À esquerda da imagem, os dispositivos Client 1 e Client 2 estão em uma rede com endereços IP na faixa 192.168.1.x, enquanto à direita, os dispositivos Client 3 e Client 4 estão em uma rede com endereços IP na faixa 192.168.2.x. Essa diferença de redes (identificável pelos diferentes intervalos de IP) ilustra o papel do roteador em interligar essas duas LANs distintas.

Cada switch conectado ao roteador organiza a comunicação dentro de sua própria LAN. Eles utilizam endereços MAC para garantir que os pacotes de dados sejam enviados diretamente ao destinatário correto dentro daquela rede local específica. No entanto, quando um dispositivo da rede 192.168.1.x (como o Client 1) deseja se comunicar com um dispositivo da rede 192.168.2.x (como o Client 3), o roteador entra em ação. Ele encaminha os dados entre essas duas redes separadas, permitindo a comunicação entre os clientes, mesmo que estejam em redes distintas. Sem o roteador, esses dispositivos não poderiam trocar informações, pois cada LAN opera de forma isolada sem uma conexão externa.

2.9 Diferença entre gateway e roteador

Nas aplicações de rede de computadores também pode ser utilizado o gatway, que é um ponto de acesso que atua como uma interface entre duas redes diferentes, muitas vezes de tipos distintos. Ele é responsável por traduzir dados de uma rede para outra, funcionando como

um intermediário entre redes que utilizam protocolos de comunicação diferentes. Por exemplo, em uma rede corporativa, o gateway pode ser a interface entre uma rede interna privada e a Internet. Sua função principal é permitir a comunicação entre redes que de outra forma não seriam compatíveis, facilitando a tradução de endereços IP, protocolos ou formatos de dados.

Já o roteador é um dispositivo responsável por encaminhar pacotes de dados entre redes diferentes, mas que utilizam o mesmo protocolo, normalmente IP (Internet Protocol). Ele toma decisões baseadas nos endereços IP de destino dos pacotes e escolhe o melhor caminho para encaminhar os dados. Dentro de uma rede doméstica, por exemplo, o roteador conecta a rede local à Internet, repassando os dados de e para os dispositivos da rede local, utilizando tabelas de roteamento para determinar o caminho mais eficiente.Um gateway e um roteador são ambos dispositivos utilizados em redes de comunicação, mas servem a propósitos distintos, embora muitas vezes possam ser integrados em um único dispositivo.

A escolha entre utilizar um gateway ou um roteador depende das necessidades da rede. Se a rede envolve a comunicação entre redes distintas, que utilizam protocolos diferentes (como uma rede local conectada a uma rede externa via VPN ou um sistema legado), é necessário o uso de um gateway. Em redes que compartilham o mesmo protocolo, como redes IP, o roteador é suficiente para garantir a comunicação eficiente. Na prática, muitos dispositivos combinam ambas as funcionalidades, como em roteadores domésticos que atuam também como gateways para a Internet, traduzindo protocolos entre a rede local (LAN) e a rede pública (WAN) isso é descrito por (Tanenbaum; Wetherall, 2011).

Portanto, enquanto o roteador é especializado em gerenciar o tráfego de redes utilizando o mesmo protocolo, o gateway é essencial quando há necessidade de comunicação entre redes de naturezas diferentes. O uso de um ou outro depende da complexidade e das características das redes que estão sendo conectadas.

3 METODOLOGIA

3.1 Introdução

A crescente complexidade das redes de computadores exige ferramentas eficientes para simulação e teste. Neste contexto, este trabalho propõe o desenvolvimento de um simulador de rede DHCP com o objetivo de fornecer um ambiente controlado e seguro para realizar testes de estresse e validação de configurações sem danificar a infraestrutura física. Diferente de muitos simuladores comerciais, que podem ser complexos e exigem configurações mais avançadas, este simulador, implementado em Python, destaca-se por sua simplicidade e flexibilidade, permitindo a criação de cenários variados que simulam tanto o comportamento de um servidor DHCP quanto de múltiplos clientes. Dessa forma, é possível analisar o desempenho da rede sob diferentes condições e identificar possíveis problemas antes de uma implementação em ambientes reais, onde qualquer falha poderia acarretar custos e riscos consideráveis.

Além de evitar danos à rede física, o simulador oferece uma plataforma acessível para usuários sem experiência prévia, permitindo uma interação direta com os fundamentos das redes DHCP sem comprometer sistemas de produção. Python foi escolhido devido à sua simplicidade, clareza e grande variedade de bibliotecas que suportam o desenvolvimento de aplicações de rede, conforme destacado por (Nosrati, 2011), (Jambunatha, 2015) e (Rocha, 2017). Isso garante uma fácil personalização, tornando o simulador ideal para outros testes futuros, como a implementação de funcionalidades relacionadas a NAT e DNS. O Visual Studio Code (VS Code) é utilizado como ambiente de desenvolvimento devido à sua extensibilidade e integração com Python, fornecendo um ambiente robusto para testes e depuração. Assim, o simulador não apenas facilita o aprendizado prático de conceitos fundamentais de rede, mas também reduz os custos e riscos associados à experimentação em redes reais, além de oferecer uma abordagem mais direta e prática em comparação com outros simuladores disponíveis no mercado.

3.2 Desenvolvimento

Este capítulo descreve o desenvolvimento do projeto, que se inicia com a definição dos objetivos a serem alcançados:

1. Desenvolver um sistema de rede para comunicação entre dispositivos simulados com interação a um roteador simulado.
2. Implementar um roteador simulado onde seja possível escolher IPs manualmente.
3. Garantir que, caso um IP não seja definido manualmente, o dispositivo simulado receba um IP aleatório via DHCP.
4. Definir uma faixa de IP para segregar a conexão, permitindo um número limitado de dispositivos simulados na rede e restringindo a conexão a um intervalo específico de IPs.

5. Testar a efetividade deste trabalho para uso pedagógico.

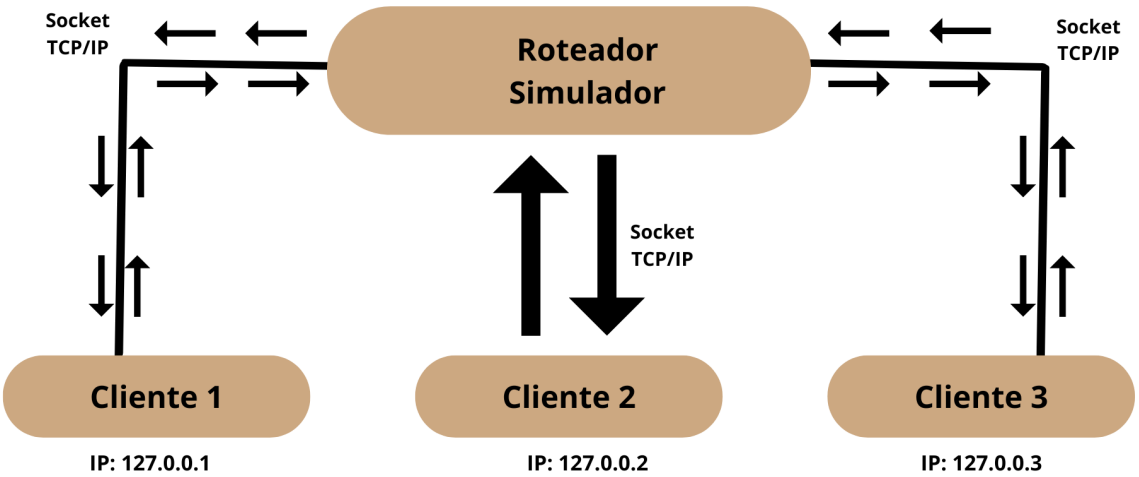
3.3 Abordagem Metodológica

Para implementar a comunicação entre os dispositivos simulados e o servidor simulado, foi utilizada a biblioteca socket do Python, conforme sugerido por Kathiravelu e Sarker (2015). Essa biblioteca oferece as funcionalidades necessárias para criar sockets, estabelecer conexões e trocar dados, sendo ideal para a simulação de redes. Foi escolhido executar o código no terminal do VS Code para simplificar a visualização e o gerenciamento do processo.

A partir dessas definições, foi feita a conexão entre cliente simulado e o servidor simulado. Porém para atender a necessidade de diversas conexões simultâneas para teste de aquisição de IP e DHCP se deu à necessidade da utilização da biblioteca threading que faz com que mais de uma execução possa acontecer simultaneamente dentro do código.Com a implementação das bibliotecas, a interação dos dispositivos simulados com roteador simulado foi possível.

Os IP´s para comunicação foram escolhidos para terem 4 conjuntos de números devido à similaridade com IPV4. E para melhor compressão e facilidade para o código, foi escolhido o endereço IP 127.0.0.x, onde x representa cada dispositivo conectado, como segue na imagem abaixo, onde é demonstrado o reteador simulado em conexão aos clientes simulados. E essa interação só é possível a partir do servido inicializado do socket que faz a ponte de comunicação entre esses dois dispositivos a partir da LAN (Local Area Network).

Figura 15 – Representação das conexões na programação.

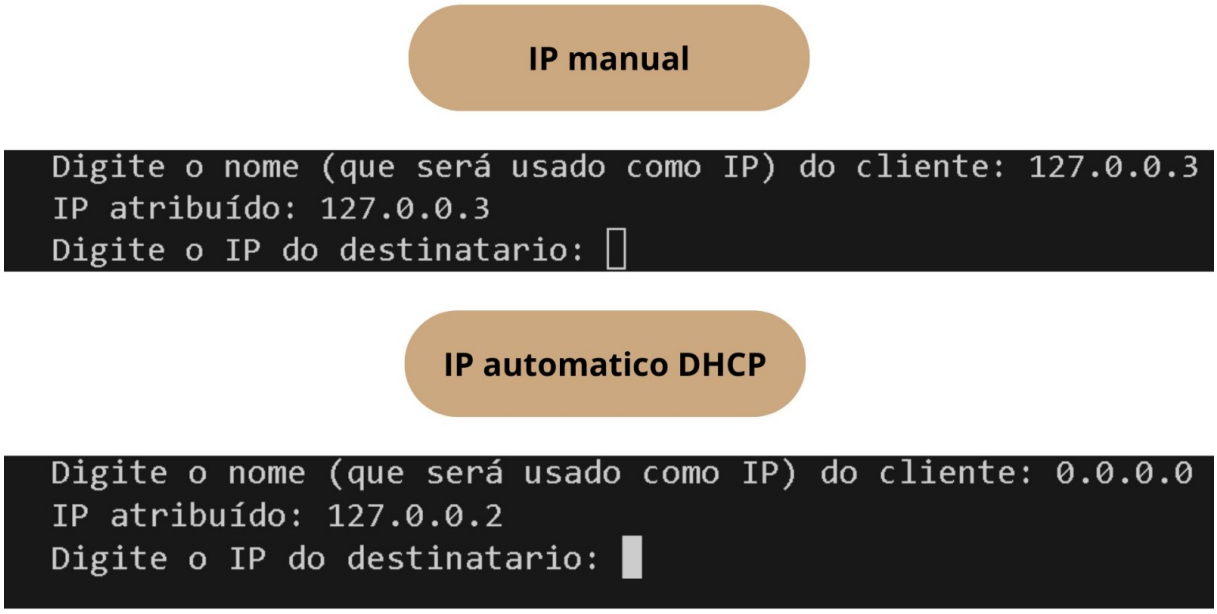


Fonte: Elaborado pelo autor (2024). .

A partir desse ponto, foi dado início a implementação de requisição IP onde usuário poderia assumir manualmente seu IP para representar um cliente que pode dar entrada no servidor com endereço IP especifico o que para ambiente de teste seja uma opção. Essa escolha é definida usando parâmetro de comparação para verificar se o endereço IP escolhido está devidamente preenchido com os 4 conjuntos de números necessários.

Com a implementação do IP manual assim como representado na figura abaixo pelo campo manual definido pelo endereço IP 127.0.0.3, o passo subsequente foi a implementação do DHCP, que devido à natureza do simulador deve ter uma entrada pseudo-vazia, pois o endereço IP assumido pelo cliente deve ser 0.0.0.0 como demonstrado na figura 12, através do console do VScod, devido escolha de se associar de forma mais intuitiva a um IP vazio do que outra entrada. A atuação do DHCP se deve a uma faixa de de IP´s disponíveis predefinidas nas programação, devido ambiente de teste e para melhor entendimento foi definido uma faixa de 1 a 9 para conexões simultâneas, onde o DHCP oferta esses endereços IP´s de forma aleatória, porém sequencial, a cada conexão respeitando a fila de conexões aplicadas.

Figura 16 – Escolha de IP manual e automático no console do python.



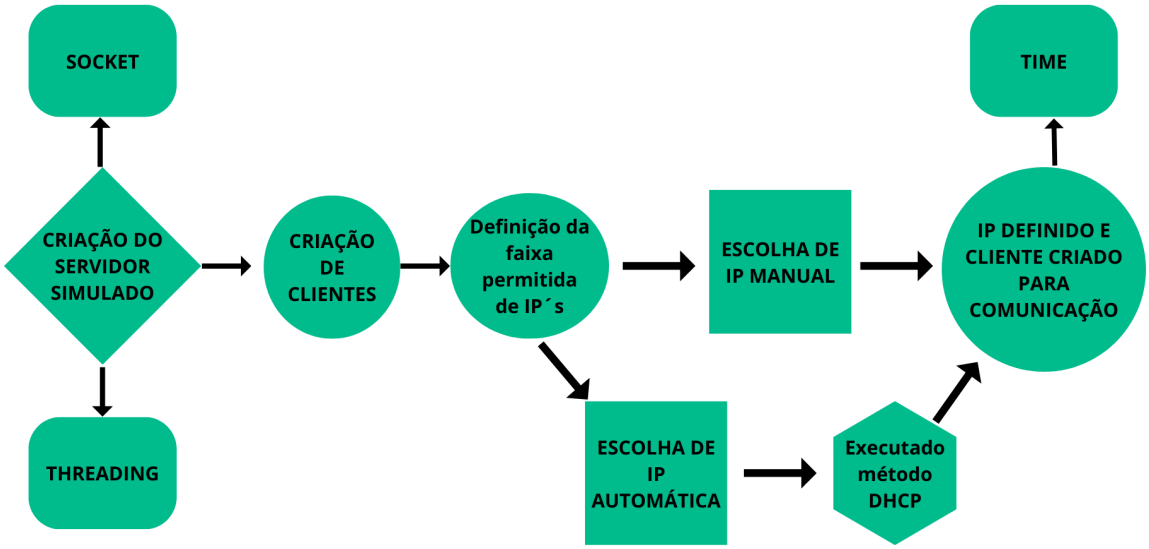
Fonte: Elaborado pelo autor (2024). .

Com o crescimento do código e as funcionalidades feitas para serem aplicadas no simulador de forma rápida e viável, também houve a necessidade de controle de tempo de cada funcionalidade e resposta do servidor para ser algo visual. devido essa necessidade foi aplicada a biblioteca time, disponível para python no VScod.

A imagem 17 representa todas as funcionalidades descritas, desde a criação do servidor simulado que necessitou em paralelo o uso das bibliotecas socket e threading. Com a criação dos

clientes foi necessário definir uma faixa de IP para limitar as conexões aceitas, o que enfatiza o uso desse trabalho para meios de teste onde conexões são normalmente limitadas para evitar sobrecarga de rotas de informação, porém essa funcionalidade pode ser definida como número infinito podendo simular sobrecarga de rota.

Figura 17 – Fluxograma das funções do código.

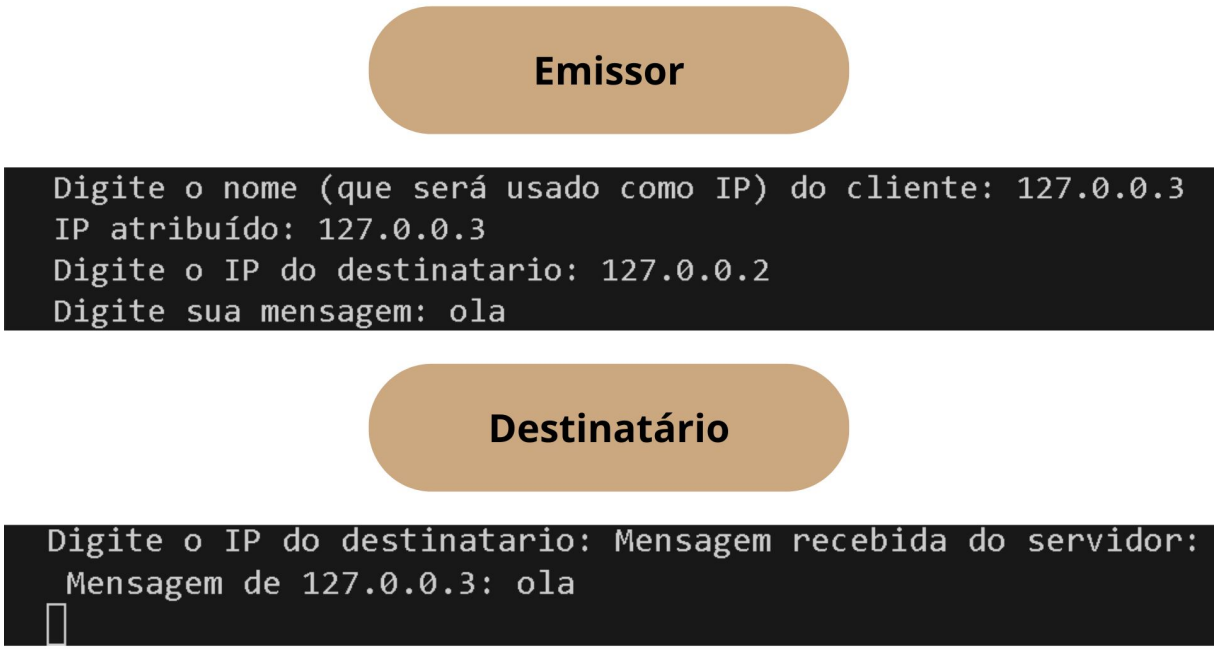


Fonte: Elaborado pelo autor (2024). .

A partir da escolha da faixa de IP, como demonstrado na imagem, é necessário escolher o método que o IP será utilizado, sendo de maneira manual ou automática como representado na figura 16 desse trabalho, caso seja selecionado manualmente o cliente será criado diretamente e, caso selecionado de maneira automática, será utilizado o método DHCP para escolher um IP dentro da faixa permitida segundo a fila de conexão. Após tal escolha, o cliente será criado e para que o servidor dê conta de várias conexões e informe cada uma delas de maneira contínua e em intervalos de tempo, foi escolhida a biblioteca time, que tem como função coordenar a leitura de tempo em tempo das conexões feitas no servidor. Por ser um ambiente teste, esse tempo foi definido como 10 segundos. No entanto, pode ser facilmente alterado, acessando a programação do sistema como descrito no apêndice.

De acordo com as informações já mencionadas, vale ressaltar que este trabalho usa uma tipologia em estrela. Como pode ser observado, todos os clientes simulados se conectam a somente um servidor simulado central onde a informação é tratada e direcionada a outro cliente. Conforme demonstrado na figura do console abaixo, o envio da mensagem é direcionada para o destinatário especifico que o cliente simulado, sendo o emissor, poderá escolher.

Figura 18 – Emissor e destinatário.



Fonte: Elaborado pelo autor (2024). .

Devido à afirmativa descrita por (Macedo *et al.*, 2018) e (Comer, 2015) destacando a compreensão limitadas de algumas pessoas sobre sistema de redes, abordado nesse trabalho, se deu como um pequeno experimento para à averiguação dessa afirmativa em uma aula direcionada aos alunos do Ensino Técnico do IFMG-Campus Ibirité, na qual usando o simulador de redes deste trabalho e os conceitos base para redes abordados anteriormente, com intuito de verificar, mesmo que sutilmente, a efetividade desse simulador e trabalho para uso didático, dentro de sala de aula.

Para a aula teste foi preparado um material didático que incluiu questionários desenvolvidos na plataforma Microsoft Forms, os quais foram disponibilizados para os alunos através de um QR code inserido nos slides de apresentação feitos no PowerPoint. A aula seguiu um formato onde o conhecimento foi construído de maneira progressiva, abordando os seguintes temas, nessa ordem:

1. O que é IP;
2. Comunicação em rede LAN;
3. Comunicação em rede internet;
4. Protocolo de rede;
5. Protocolo TCP/IP;

6. Comportamento do protocolo TCP/IP;
7. DHCP;
8. Funcionamento do DHCP.

Após a exposição teórica desses conceitos, foi demonstrado na prática o código desenvolvido e o seu funcionamento, permitindo aos alunos uma compreensão aplicada do conteúdo apresentado.

4 RESULTADOS

Neste capítulo serão apresentados os resultados alcançados durante o desenvolvimento do sistema de rede para comunicação entre dispositivos simulados, bem como uma discussão dos resultados encontrados em foco a responder aos objetivos propostos e avaliar a coleta de informações da aula experimental ministrada.

4.1 Programação

Vale ressaltar que o código referente ao sumulador é dividido em 5 partes que serão abordadas no geral nesse capitulo e mais especificamente no apêndice.

Em primeiro lugar, atendendo ao objetivo principal deste trabalho, que é a criação de um dispositivo de rede simulado, foi efetivado com êxito, devido à implementação da programação com base em python e as duas bibliotecas principais, socket e threading, como abordado no capítulo anterior. Um dos objetivos específicos de conexão de cliente simulado e servidor simulado atendido a partir dessa proposta, juntamente com objetivos de tratar múltiplas conexões em escala no servidor simulado. Para tal, é executada a função no bloco de código a baixo para o servidor simulado:

Figura 19 – Bloco de código da função para comunicação de servidor simulado e cliente simulado.

```
import socket
import threading
import time  # Adicionando a importação do módulo time

def handle_client_connection(conn, ender, client_name, clients):
    """
   Comunicação entre os clientes.
    """
    try:
        while True:
            data = conn.recv(1024)
            if not data:
                print(f'Connection closed with {client_name}')
                break

            decoded_data = data.decode()
            print(f'Message received from {client_name}: {decoded_data}')

            if decoded_data.startswith('/destinatario'):
                partes = decoded_data.split(' ')
                destinatario = partes[1]
                mensagem = ' '.join(partes[2:])
                if destinatario in clients:
                    clients[destinatario][1].sendall(str.encode(f'Mensagem de {client_name}: {mensagem}'))
                else:
                    print(f'Destinatário {destinatario} não encontrado.')
            else:
                # Process other types of messages if needed
                pass

    except (ConnectionResetError, OSError):
        print(f'Connection lost with {client_name}')
    finally:
        del clients[client_name]  # Remove the disconnected client
        conn.close()
```

Fonte: Elaborado pelo autor (2024). .

A validação de endereço IP manual e automático atendeu bem aos requisitos necessários para simulação de rede. Logo que simulam IP´s de IPV4 e requisição DHCP para um servidor.Isso acontece devido o bloco de comunicação DHCP ou endereçamento IP manual, como esse bloco apresenta repetição quando uma vez executado o DHCP o endereço IP fica anexado ao cliente evitado repetição de ações pelo usuário. Isso pode ser descrito na imagem abaixo:

Figura 20 – Bloco de código para execução do protocolo DHCP e endereçamento IP manual.

```
def assign_client_ip(conn, client_name, clients, used_ips):
    """
    DHCP.
    """
    while True:
        if client_name == '0.0.0.0':
            # Atribuir um IP disponível da faixa permitida ao cliente
            available_ips = [f'127.0.0.{i}' for i in range(2, 10) if f'127.0.0.{i}' not in used_ips]
            if available_ips:
                client_name = available_ips[0]
                clients[client_name] = (client_name, conn)  # Atribuir o IP como nome do cliente
                used_ips.add(client_name)
                # Envia uma mensagem ao cliente sobre o IP definido
                conn.sendall(str.encode(f"IP atribuído: {client_name}"))
                return client_name
            else:
                print('Não há IPs disponíveis. Encerrando conexão.')
                conn.close()
                return None
        elif client_name.startswith('127.0.0.') and int(client_name.split('.')[3]) in range(2, 10):
            if client_name not in used_ips:
                clients[client_name] = (client_name, conn)  # Atribuir o IP como nome do cliente
                used_ips.add(client_name)
                # Envia uma mensagem ao cliente sobre o IP definido
                conn.sendall(str.encode(f"IP atribuído: {client_name}"))
                return client_name  # Retorna o nome do cliente atualizado
            else:
                print('O IP fornecido já está em uso. Solicitando outro IP válido ao cliente...')
                conn.sendall(str.encode("IP está em uso. Por favor, insira outro IP na faixa permitida."))
                client_name = conn.recv(1024).decode().strip()  # Solicita outro IP válido ao cliente
        else:
            print('O IP fornecido não está dentro da faixa permitida (127.0.0.2 a 127.0.0.9).')
            conn.sendall(str.encode("IP inválido. Por favor, forneça um IP dentro da faixa permitida."))
            client_name = conn.recv(1024).decode().strip()
```

Fonte: Elaborado pelo autor (2024). .

Outro ponto abordado com êxito nesta pesquisa foi a definição de uma faixa de endereço IP específica, totalmente configurável pelo usuário da programação de forma fácil e concisa, como demostrado na figura acima, onde basta definir a faixa inicial e final para que os IP´s sejam assumidos, podendo não só trocar a faixa como acrescentar muito mais opções de conexão de cliente sendo totalmente viável para teste de capacidade de conexões em um sistema.

Outra necessidade a ser atendida foi a de bloqueio que um IP substitua outro já existente para isso foi feito um bloco de validação de dados dos IP´s, armazenamento desses dados e condicional para atribuição de endereço IP na faixa permitida. Isso pode ser observado no bloco de código descrito abaixo, que evita também que à requisição de IP manual e DHCP sejam executados juntos, com uma condicional de somente um ou outro.

Figura 21 – Bloco código para condicionais.

```
# Endereço IP do servidor
server_ip_int_parts = [127, 0, 0, 1]
Host = '.'.join(map(str, server_ip_int_parts))
PORT = 50000

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((Host, PORT))
s.listen()

print("Servidor esperando conexões")

clients = {}  # Dicionário para armazenar as conexões dos clientes
used_ips = set()  # Conjunto para armazenar os IPs já atribuídos

# Iniciar a thread para monitorar os clientes e IPs
monitor_thread = threading.Thread(target=monitor_clients, args=(clients, used_ips))
monitor_thread.start()

while True:
    conn, ender = s.accept()

    # Verifica se o IP do cliente está na faixa permitida (127.0.0.1 a 127.0.0.10)
    client_ip = ender[0]
    allowed_ip_range = [f'127.0.0.{i}' for i in range(1, 11)]
    if client_ip not in allowed_ip_range:
        print(f'Conexão recusada do cliente com IP {client_ip}')
        conn.close()
        continue

    client_name = conn.recv(1024).decode()

    # Verifica se o cliente forneceu '0.0.0.0' como IP
    if client_name == '0.0.0.0':
        client_name = assign_client_ip(conn, client_name, clients, used_ips)
    else:
        # Atribui um IP para o cliente e atualiza o nome do cliente no servidor
        client_name = assign_client_ip(conn, client_name, clients, used_ips)

    if client_name is not None:
        client_thread = threading.Thread(target=handle_client_connection, args=(conn, ender, client_name, clients))
        client_thread.start()
```

Fonte: Elaborado pelo autor (2024). .

Devido à alta gama de mensagens suportadas e para a melhoria do código detectou se à necessidade de adicionar a biblioteca time para leitura das informações no console do VScod. O bloco na figura abaixo aborda tal necessidade, colocando um leitor dentro do servidor simulado para monitorar conexões, utilizando time.sleep(10) para referir ao tempo de cada atualização da informação no servidor.

Figura 22 – Bloco código para monitoramento dos clientes no servidor.

```
def monitor_clients(clients, used_ips):
    """
    Monitora os clientes e IPs conectados no servidor.
    """
    while True:
        i = 0
        print("Clientes conectados:")
        for client_name, _ in clients.items():
            i += 1
            print(f"Cliente {i}")

        print("\nIPs atribuídos:")
        for ip in used_ips:
            print(f"IP: {ip}")

        print("\n")

        # Atualiza a cada 10 segundos
        time.sleep(10)
```

Fonte: Elaborado pelo autor (2024). .

Tais blocos de código no servidor permitem que ele execute continuamente até que seja interrompido. No caso deste simulador, uma falha que pode ser descrita é a necessidade exclusiva do servidor simulado central observado acima. Caso esse servidor simulador encontre um erro e falhe a execução, há falha de comunicação com todos os clientes. A vantagem é que, caso um cliente falhe o servidor não depende deles para o funcionamento; o que faz esse simulador se encaixar na topologia estrela.

Abordados os blocos principais descritos no servidor simulado, agora será descrito o código padrão dos clientes, que é divido em duas partes; uma que envia a mensagem ao servidor e a outra que recebe essa mensagem, ambas são descritas na imagem a seguir:

Figura 23 – Bloco código do cliente para envio e recebimento de mensagem.

```python
import socket
import threading

def send_message(s, client_ip):
    try:
        while True:
            destinatario = input("Digite o IP do destinatario: ")
            mensagem = input("Digite sua mensagem: ")
            s.sendall(str.encode(f'/destinatario {destinatario} {mensagem}'))
    except KeyboardInterrupt:
        s.close()

def connect_to_server(ip):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        s.connect((ip, PORT))
        return s
    except ConnectionRefusedError:
        return None

# Endereço IP do servidor
server_ip_int_parts = [127, 0, 0, 1]
Host = '.'.join(map(str, server_ip_int_parts))
PORT = 50000

# Conectando-se ao servidor
s = connect_to_server(Host)

if s is None:
    print("Falha ao conectar ao servidor. Encerrando...")
else:
    while True:
        # Solicitar ao usuário que digite o IP (nome do cliente)
        client_ip = input("Digite o nome (que será usado como IP) do cliente: ")

        # Enviar o IP do cliente para o servidor
        s.sendall(str.encode(client_ip))

        # Receber a resposta do servidor
        response = s.recv(1024).decode().strip()

        # Verificar se o IP fornecido pelo cliente é válido
        if response.startswith("IP atribuído"):
            print(response)
            # Atualizar o nome do cliente com o IP validado fornecido pelo servidor
            client_name = response.split(":")[1].strip()
            break
        else:
            print(response)

    # Iniciar uma thread para enviar mensagens constantemente
    send_thread = threading.Thread(target=send_message, args=(s, client_ip))
    send_thread.start()

    try:
        while True:
            data = s.recv(1024)
            if not data:
                print("Conexão encerrada pelo servidor.")
                break
            print(f'Mensagem recebida do servidor: {data.decode()}')
    except KeyboardInterrupt:
        s.close()
```

Fonte: Elaborado pelo autor (2024). .

Com essa análise, os objetivos de 1 ao 5, propostos na sessão 3.2 de desenvolvimento no capitulo anterior, encontram-se atendidos. Mais detalhes sobre o código serão deixados no apêndice.

4.2 Uso do simulador para didática

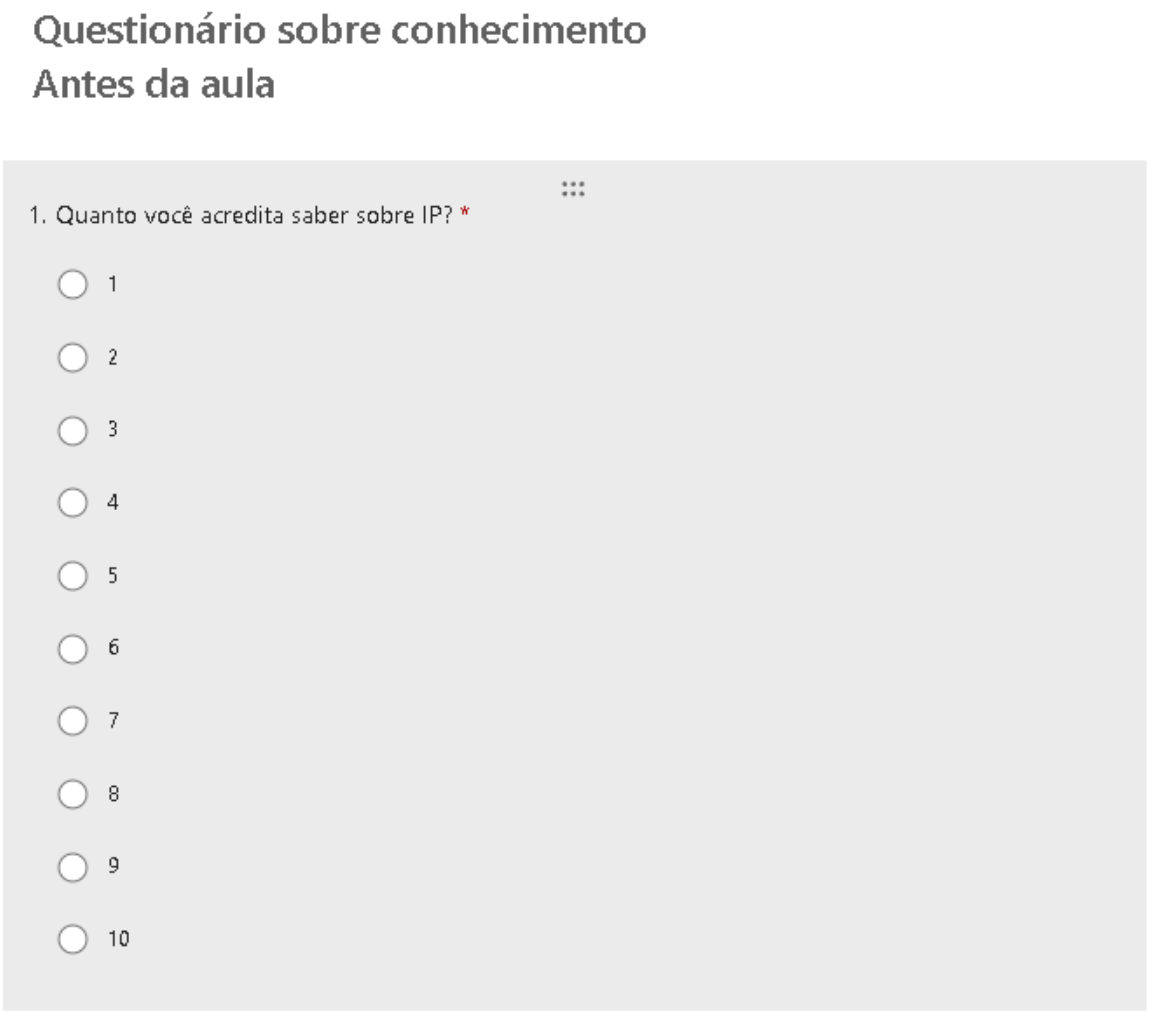
Para avaliar a eficácia do programa aplicado na metodologia didática expositiva, foram coletados dados durante uma aula experimental. Inicialmente, foi elaborado um questionário

sobre conhecimentos presentes nesse trabalho. Os conceitos foram divididos de 1 a 10 sendo 10 conhecimento pleno sobre assuntos abordados e 1 nenhum conhecimento. Tal questionário constava com 5 perguntas sendo elas:

1. Quanto você acredita saber sobre IP?
2. Quanto você acredita saber sobre comunicação em rede?
3. Quanto você acredita saber sobre comunicação em LAN ?
4. Quanto você acredita saber sobre "protocolo de rede"?
5. Quanto você acredita saber sobre DHCP?

E, usando Microsoft Forms foi montada a seguinte estrutura para formulário; pergunta como titulo e campo de seleção de 1 a 10 , como consta na imagem abaixo:

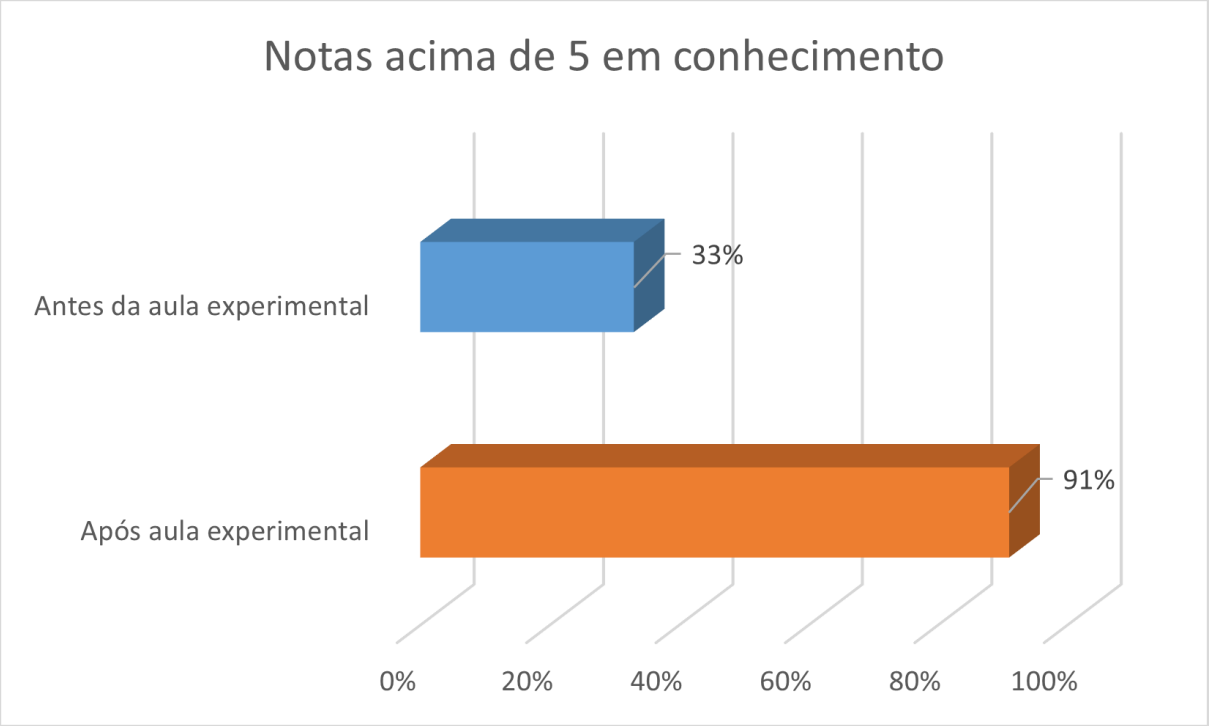
Figura 24 – Modelo de perguntas no Forms.



Fonte: Elaborado pelo autor (2024). .

Observou-se que a maioria dos alunos possuía pouco ou nenhum conhecimento prévio sobre os temas abordados, assim como descrito por (Macedo *et al.*, 2018).Sendo que a maior parte das notas no questionário inicial, situaram-se abaixo de 5,0 pontos. Os tópicos mais desconhecidos incluíam DHCP, seguido de IP e protocolos de rede. Enquanto a comunicação em rede apresentou a maior taxa de conhecimento prévio entre os alunos. O mesmo questionário foi utilizado para respostas antes e depois da aula expositiva. O gráfico abaixo demonstra o percentual dos dados coletados antes e após a aula experimental, onde após a aplicação do conteúdo programático e da execução da aula expositiva, revelou-se um aumento significativo no nível de conhecimento, visto que a maioria das respostas, no segundo questionário, alcançou notas acima de 6 pontos.

Figura 25 – Coleta de dados antes da aula expositiva.



Fonte: Elaborado pelo autor (2024). .

No entanto, cabe destacar que o tema "protocolos de rede"continuou apresentando certa dificuldade para os alunos. Esse assunto, por abranger diversos tipos de protocolos, não pôde ser completamente explorado devido à limitação de tempo disponível na aula experimental. Mesmo assim, os resultados gerais indicam uma melhoria substancial na compreensão dos conceitos, validando a eficácia da metodologia aplicada.Vale ressaltar que o método usado pode não ser o melhor modelo de teste para essa estrutura, pois não apresenta testes de conhecimento como provas. No entanto há escalabilidade desse trabalho onde se pode proporcionar novos modelos para atividades podendo assim suprir conhecimento de protocolos de rede e uma análise mais precisa através de testes.

5 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Neste trabalho, abordou-se a implementação de um sistema de rede utilizando programação em Python, com ênfase na comunicação servidor-cliente.Com este estudo, constatamos que, embora a apresentação teórica de sistemas de rede possa ser complexa, a implementação prática e a demonstração por meio do código oferece uma base sólida e compreensível para a visualização e entendimento do funcionamento de tais sistemas.

O simulador configurou-se como um método valioso para testes de conexões em rede e como projeto base para simuladores mais complexos, onde caso algum usuário possa usá-lo tenha a chance de retrabalhar o código de forma fácil para criar um ambiente sobreposto para sistemas mais robusto. Dito isso, o código pode ser adaptado e aprimorado para incluir funções adicionais, presentes em outras camadas do modelo OSI, permitindo um estudo mais aprofundado e detalhado de sistemas de rede.

Com base nos dados coletados e na análise dos resultados referentes à aula expositiva, pode-se concluir que o simulador foi eficaz em aumentar o nível de conhecimento dos alunos sobre os temas abordados nesse trabalho, como IP, comunicação em rede, LAN, internet, protocolos de rede e DHCP. A aula experimental revelou que, embora a maioria dos alunos, inicialmente apresentassem pouco ou nenhum conhecimento prévio sobre esses tópicos, a aplicação do conteúdo programático e a realização de atividades práticas contribuíram para uma melhoria significativa em sua compreensão do conteúdo abordado.

Assim, este trabalho não só atingiu seu objetivo de demonstrar o funcionamento de um sistema de rede, mas também estabeleceu uma base sólida para futuras explorações e expansões dentro deste campo essencial da Tecnologia da Informação e Engenharia adaptando métodos como programação para criar uma nova ferramenta a ser utilizada e trabalhada em diferentes contextos de aplicação.

REFERÊNCIAS

BESSANI, A.; ROCHA, A. A.; GUEDES, D.; ALMEIDA, J. **Redes de Computadores: Dos Princípios à Prática**. 1. ed. Rio de Janeiro: Elsevier, 2017.

CAVIN YOAV SASSON, A. S. D. On the accuracy of manet simulators. **Proceedings of the 2nd ACM International Workshop on Principles of Mobile Computing (POMC'02)**, p. 38–43, 2002.

COMER, D. E. **INTERLIGAÇÃO DE REDES COM TCP/IP**. 6. ed. Rio de Janeiro: Elsevier Editora Ltda, 2015.

FERES, M. M. **AVALIAÇÃO DO DESEMPENHO DO PROTOCOLO TCP/IP EM REDES COM TOPOLOGIA ESTRELA UTILIZANDO A FERRAMENTA NETWORK SIMULATOR**. Dissertação (Mestrado) — FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"CENTRO UNIVERSITÁRIO "EURÍPIDES DE MARÍLIA" – UNIVEM, 2006.

FERNANDES, F. A. **Um Estudo Sobre o Uso da Topologia Anel em Redes Profinet como Técnica de Implementação de Redundância**. Dissertação (Mestrado) — Universidade de São Paulo, Escola de Engenharia de São Carlos, 2015.

FERNANDO, L.; SOARES, G.; GUIDO, L.; COLCHER, S. **Redes de Computadores: das LANs, MANs e WANs às redes ATM**. 2. ed. Rio de Janeiro: Campus, 1995.

GALLO, M.; HANCOCK, W. **Networking Explained**. 2nd. ed. Burlington, MA: Digital Press, 2002.

HUITEMA, C. **IPv6: The New Internet Protocol**. 1st. ed. Upper Saddle River, NJ: Prentice Hall PTR, 1998.

JAMBUNATHA, K. Design and implement automated procedure to upgrade remote network devices using python. **IEEE International Advance Computing Conference (IACC**, p. 217–220, 2015.

KUROSE, J. F.; ROSS, K. W. **Computer Networking: A Top-Down Approach**. 7. ed. Hoboken,USA: Pearson Education, 2017.

L.L.MARQUES; COSTA, M. Configuração automática de endereços ip com o uso do protocolo dhcp: Um estudo sobre a implementação e a segurança do protocolo em redes locais. **Anais do Congresso Brasileiro de Redes de Computadores e Sistemas Distribuídos (CBRSD)**, p. 45–46, 2016.

MACEDO, R. T.; FRANCISCATTOAND, R.; CUNHA, G. B. da; BERTOLINI, C. **REDES DE COMPUTADORES**. 1. ed. Santa Maria | RS: UAB/NTE/UFSM, 2018.

NOSRATI, M. Python: An appropriate language for real world programming. **WAP journal**, p. 110–117, 2011.

OLIVEIRA, B. M. D. **DESENVOLVIMENTO DE UM AMBIENTE COMPUTACIONAL PARA RABALHAR COM REDES DEFINIDAS POR SOFTWARES NO LABORATÓRIO CIDIG**. Dissertação (Mestrado) — INSTITUTO FEDERAL DO ESPÍRITO SANTO, 2022.

ROCHA, U. V. AutomaÇÃo de redes utilizando python. **Anais III JOIN / Edição Brasil**, p. 1–6, 2017.

STALLINGS, W. **Redes de Sistemas de Comunicação de Dados**. 1. ed. Rio de Janeiro: Elsevier Editora Ltda, 2005.

____. **Data and Computer Communications**. 10th. ed. Upper Saddle River, NJ, USA: Pearson, 2013.

STEMMER, P. D.-I. M. R. **Das 5331- sistemas distribuídos e REDES de computadores paracontrole e AUTOMAÇÃO industrial**. Dissertação (Mestrado) — Universidade Federal de Ouro Preto, 2001.

TANENBAUM, A. S.; WETHERALL, D. **REDES DE COMPUTADORES**. 5. ed. São Paulo: Pearson Prentice Hall, 2011.

WRIGHT, J. Detecting wireless lan mac address spoofing. 2003. Acesso em: 15 out. 2014. Disponível em: <http://target0.be/madchat/reseau/wireless/wlan-mac-spoof.pdf>.

ZIMMERMANN, H. Osi reference model—the iso model of architecture for open systems interconnection. **IEEE Transactions on Communications**, v. 28, n. 4, p. 425–432, 1980.

APÊNDICE A – INFORMAÇÕES SOBRE MONTAGEM DA PROGRAMAÇÃO

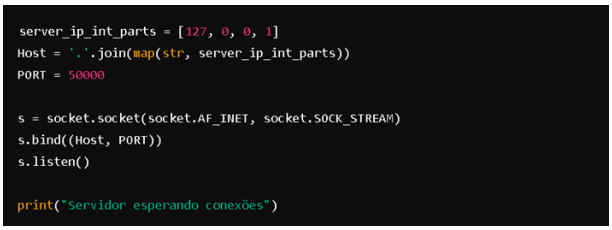
No presente anexo, será abordada toda a montagem passo a passo da programação realizada para o pleno entendimento dos usuários, destacando as capacidades presentes na versão atual e consolidando o entendimento para que possa ser modificada, se necessário.Vale ressaltar que todas as imagens presentes nesse apêndice serão explicadas abordando a programação presentes, de forma a destrinchar cada pedaço trabalhado com uma explicação minuciosa. Sempre quando uma imagem aparecer no apêndice referente ao console do VScod ela será explicada detalhadamente a baixo.

A.1 Criação do servidor

Em primeiro lugar, é criado um servidor local para interação entre cliente e servidor, através do uso da biblioteca socket.Como demonstrado na imagem abaixo.

Configuração do Servido:

Figura 26 – Criação do servidor.



Fonte: Elaborado pelo autor (2024). .

server ip int parts = [127, 0, 0, 1]

- Definição do endereço IP do servidor como uma lista de inteiros

  Host = '.'.join(map(str, server ip int parts))

- Converte a lista de inteiros para uma string no formato "127.0.0.1"usando a função

  PORT = 50000

- Cria um objeto de socket utilizando IPv4 (AF INET) e TCP (SOCK STREAM)

  s = socket.socket(socket.AF INET, socket.SOCK STREAM)

- Associa o socket ao endereço IP e porta definidos anteriormente

s.bind((Host, PORT))

- Coloca o servidor em modo de escuta, pronto para aceitar conexões de clientes

s.listen()

- Coloca o servidor em modo de escuta, pronto para aceitar conexões de clientes

print("Servidor esperando conexões")

- Exibe uma mensagem indicando que o servidor está esperando por conexões

A.2 Inicialização das Estruturas de Dados

Essa parte está responsável por inicializar os dicionários e conjuntos para informações do clientes.

Figura 27 – Inicialização das Estruturas de Dados.

```
clients = {}  # Dicionário para armazenar as conexões dos clientes
used_ips = set()  # Conjunto para armazenar os IPs já atribuídos
```

Fonte: Elaborado pelo autor (2024). .

clients =

- Dicionário para armazenar as conexões dos clientes

used ips = set()

- Conjunto para armazenar os IPs já atribuídos

A.3 Thread de Monitoramento

Essa parte está responsável poriIniciar uma thread para monitorar os clientes e IPs conectados

Figura 28 – Thread de Monitoramento.

```
# Iniciar a thread para monitorar os clientes e IPs
monitor_thread = threading.Thread(target=monitor_clients, args=(clients, used_ips))
monitor_thread.start()
```

Fonte: Elaborado pelo autor (2024). .

monitor thread = threading.Thread(target=monitor clients, args=(clients, used ips))

Nesta linha, utiliza-se a classe **Thread** do módulo **“threading”** para criar uma nova thread. A função que será executada pela thread é **“monitor clients”**, e os argumentos passados para essa função são **“clientes”** e **“used ips”**. A função **“monitor clients”** será responsável por monitorar os clientes conectados e os IPs atribuídos, atualizando essas informações periodicamente.

monitor thread.start()

Esta linha inicia a execução da thread que foi criada anteriormente. A partir deste ponto, a função **“monitor clients”** começa a ser executada em paralelo com o restante do programa, permitindo que o servidor continue aceitando novas conexões de clientes enquanto monitora os clientes conectados e os IPs utilizados. Essas etapas são cruciais para garantir que o servidor possa manter um registro atualizado dos clientes conectados e dos IPs atribuídos, sem bloquear a execução principal do servidor. Isso é feito através do uso de threading, que permite a execução concorrente de múltiplas partes do programa.

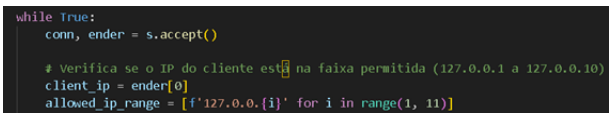
Juntas essas partes estão em comum acordo para funcionamento do servidor base.

A.4 Loop de Aceitação de Conexões

Esta seção do código é responsável por aceitar novas conexões de clientes, validar seus endereços IP, atribuir IPs quando necessário, e iniciar threads para lidar com cada cliente de forma independente.

Para melhor explicação essa parte será repartida:

Figura 29 – Loop de Aceitação de Conexões.



Fonte: Elaborado pelo autor (2024). .

while True:

conn, ender = s.accept()

- O loop **“while True”** mantém o servidor em execução contínua, aceitando novas conexões de clientes. A função **“s.accept()”** bloqueia até que uma nova conexão seja aceita, retornando um novo objeto de conexão **“conn”** e o endereço do cliente **“ender”**.

client ip = ender[0]

allowed ip range = [f’127.0.0.i’ for i in range(1, 11)]

- O endereço IP do cliente é extraído do objeto **“ender”**.
- Define-se a faixa de IPs permitidos para os clientes, que vai de 127.0.0.1 a 127.0.0.10.

Continua do o loop de aceitação temos:

Figura 30 – Loop de Aceitação de Conexões parte 2.

```
    if client_ip not in allowed_ip_range:
        print(f'Conexão recusada do cliente com IP {client_ip}')
        conn.close()
        continue

    client_name = conn.recv(1024).decode()
```

Fonte: Elaborado pelo autor (2024). .

if client ip not in allowed ip range:

print(f’Conexão recusada do cliente com IP client ip’)

conn.close()

continue

- Esta parte ,verifica-se se o IP do cliente está dentro da faixa permitida. Se não estiver, a conexão é recusada, o socket é fechado, e o loop recomeça para aceitar outra conexão.

client name = conn.recv(1024).decode()

- recebe o nome do cliente enviado pelo mesmo através do socket. O nome é decodificado de bytes para string.

Figura 31 – Condicional 0.0.0.0.

```
# Verifica se o cliente forneceu '0.0.0.0' como IP
if client_name == '0.0.0.0':
    client_name = assign_client_ip(conn, client_name, clients, used_ips)
```

Fonte: Elaborado pelo autor (2024). .

if client name == ’0.0.0.0’:

client name = assign client ip(conn, client name, clients, used ips)

- Através da condicional **if client name ==’0.0.0.0’:** verifica-se se o nome do cliente é 0.0.0.0, o que indica que o cliente precisa de um IP atribuído pelo servidor.

Figura 32 – Inicialização do client.

Fonte: Elaborado pelo autor (2024). .

if client name is not None:

client thread=threading.Thread(target=handle client connection,args=(conn,ender, client name, clients)

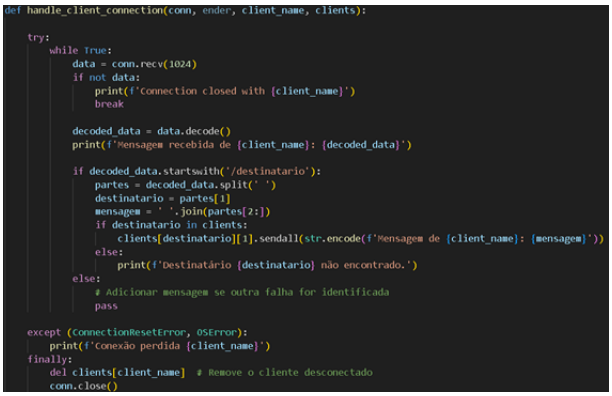
client thread.start()

- Se o cliente recebeu um nome válido (ou seja, um IP válido foi atribuído), cria-se uma nova thread para lidar com a comunicação com esse cliente. A função **handle client connection** será executada nessa thread, permitindo que o servidor continue aceitando novas conexões enquanto trata das existentes. A thread é então iniciada com **client thread.start()**.

A.5 Switch

Sendo umas das partes principais desse projeto , esse código define o switch onde a comunicação é endereçada a cada ip , conexões são armazenadas e conexões perdidas são adicionadas as bibliotecas informando que o cliente especifico não está mais conectado.

Figura 33 – Switch.



Fonte: Elaborado pelo autor (2024). .

Ou seja, é a parte de atualização de sistema, comunicação entre os ip´s , gerenciamento dos clientes.

Para explicação mais detalhada será repartido o código para melhor entendimento.

A.5.1 Entrada de dados

Figura 34 – Entrada de dados.

Fonte: Elaborado pelo autor (2024). .

def handle client connection(conn, ender, client name, clients):

A função **handle client connection** é responsável por gerenciar a comunicação entre o servidor e um cliente específico. **A função handle client connection** aceita quatro parâmetros:

- **conn**: o objeto de conexão do cliente.
- **ender**: o endereço do cliente (não utilizado diretamente na função).
- **client name**: o nome/IP atribuído ao cliente.
- **clients**: um dicionário contendo todos os clientes conectados.

try:

while True:

data = conn.recv(1024)

Dentro de um bloco **try**, a função entra em um loop **while True**, indicando que vai continuar recebendo dados do cliente até que ocorra uma condição de parada. A função **conn.recv(1024)** bloqueia a execução até que cheguem dados do cliente, recebendo até 1024 bytes de cada vez.

if not data:

print(f'Connection closed with client name')

break

Se data estiver vazio, isso indica que o cliente fechou a conexão. Nesse caso, imprime-se uma mensagem indicando que a conexão foi fechada e sai do loop usando break.

A.5.2 Decodificação da mensagem

Figura 35 – Decodificação da mensagem.

Fonte: Elaborado pelo autor (2024). .

decoded data = data.decode()

print(f'Mensagem recebida de client name: decoded data')

Os dados recebidos são decodificados de bytes para string e impressos no console

A.5.3 Verificações e condições para cliente

Figura 36 – Verificações e condições para cliente.

```
if decoded_data.startswith('/destinatario'):
    partes = decoded_data.split(' ')
    destinatario = partes[1]
    mensagem = ' '.join(partes[2:])
    if destinatario in clients:
        clients[destinatario][1].sendall(str.encode(f'Mensagem de {client_name}: {mensagem}'))
    else:
        print(f'Destinatário {destinatario} não encontrado.')
else:
    # Adicionar mensagem se outra falha for identificada
    pass
```

Fonte: Elaborado pelo autor (2024). .

if decoded data.startswith('/destinatario'):

- O código verifica se a mensagem recebida **(decoded data)** começa com a string **'/destinatario'**. Isso indica que a mensagem é destinada a um cliente específico.

partes = decoded data.split(' ')

A.5.4 A mensagem é dividida em partes usando o espaço como delimitador. O resultado é uma lista de strings, armazenada em partes.

destinatario = partes[1]

- O segundo elemento da lista **(partes[1])** é o destinatário da mensagem. Este é o nome/IP do cliente para quem a mensagem deve ser enviada.

mensagem = ' '.join(partes[2:])

- Os elementos restantes da lista (partes[2:]) são juntados novamente em uma única string, formando a mensagem real que deve ser enviada ao destinatário.

if destinatario in clients:

clients[destinatario][1].sendall(str.encode(f'Mensagem de client name: mensagem'))

- O código verifica se o destinatário está no dicionário **clients**, que contém todas as conexões de clientes atualmente ativas.

- Se o destinatário for encontrado, a mensagem é enviada para ele. O método **sendall** é usado para enviar a mensagem, que é codificada em bytes.
- A mensagem enviada tem o formato Mensagem de **client name: mensagem**, onde **client name** é o nome/IP do remetente e mensagem é a mensagem real.

else:

print(f'Destinatário destinatario não encontrado.')

- Se o destinatário não estiver no dicionário clients, uma mensagem de erro é impressa no console, indicando que o destinatário não foi encontrado.

else:

pass

- Else apenas para fechar o ciclo do IF e pode ser útil caso queira adicionar algo sobre alguma falha , que não foi abordado no projeto atual.

A.5.5 Exceções e a finalização da conexão com o cliente

Este bloco captura exceções que podem ocorrer durante a comunicação com o cliente

Figura 37 – Exceções e a finalização da conexão com o cliente.

```
except (ConnectionResetError, OSError):
    print(f'Conexão perdida {client_name}')
finally:
    del clients[client_name]  # Remove o cliente desconectado
    conn.close()
```

Fonte: Elaborado pelo autor (2024). .

except (ConnectionResetError, OSError):

print(f'Conexão perdida client name')

As exceções específicas tratadas aqui são:

- **ConnectionResetError**: Ocorre quando a conexão é abruptamente interrompida pelo cliente. Por exemplo, se o cliente fechar a conexão de forma inesperada.
- **OSError**: Abrange uma variedade de erros do sistema operacional que podem ocorrer durante operações de rede, incluindo falhas de conexão.

Quando uma dessas exceções é capturada, uma mensagem é impressa no console indicando que a conexão com o cliente foi perdida. client name é o identificador do cliente cuja conexão foi interrompida.

finally:

del clients[client name]

conn.close()

- O bloco **finally** é executado independentemente de uma exceção ter sido lançada ou não. Ele garante que certas ações sejam realizadas sempre, como a limpeza de recursos.
- A parte **del clients[client name]** remove o cliente do dicionário **clients**. Isso é importante para garantir que o servidor não tente mais se comunicar com um cliente que não está mais conectado e para liberar a memória associada a essa conexão.
- **conn.close()** fecha a conexão de **socke**t com o cliente.

A.6 Servidor DHCP

Essa parte é responsável pela execução do DHCP dentro do servidor , nela é feita divisão de ip´s segundo a faixa permitida pelo servidor. A função **assign client ip** é projetada para garantir que um cliente que solicita um IP dinâmico (usando '0.0.0.0') receba um IP válido dentro de um intervalo específico.

Figura 38 – Servidor DHCP.

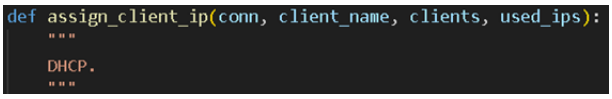
```
def assign_client_ip(conn, client_name, clients, used_ips):
    """
    DHCP.
    """
    while True:
        if client_name == '0.0.0.0':
            # Atribuir um IP disponível da faixa permitida ao cliente
            available_ips = [f'127.0.0.{i}' for i in range(2, 10) if f'127.0.0.{i}' not in used_ips]
            if available_ips:
                client_name = available_ips[0]
                clients[client_name] = (client_name, conn)  # Atribuir o IP como nome do cliente
                used_ips.add(client_name)
                # Envia uma mensagem ao cliente sobre o IP definido
                conn.sendall(str.encode(f"IP atribuído: {client_name}"))
                return client_name
            else:
                print('Não há IPs disponíveis. Encerrando conexão.')
                conn.close()
                return None
        elif client_name.startswith('127.0.0.') and int(client_name.split('.')[3]) in range(2, 10):
            if client_name not in used_ips:
                clients[client_name] = (client_name, conn)  # Atribuir o IP como nome do cliente
                used_ips.add(client_name)
                # Envia uma mensagem ao cliente sobre o IP definido
                conn.sendall(str.encode(f"IP atribuído: {client_name}"))
                return client_name  # Retorna o nome do cliente atualizado
            else:
                print('O IP fornecido já está em uso. Solicitando outro IP válido ao cliente...')
                conn.sendall(str.encode("IP está em uso. Por favor, insira outro IP na faixa permitida."))
                client_name = conn.recv(1024).decode().strip()  # Solicita outro IP válido ao cliente
        else:
            print("O IP fornecido não está dentro da faixa permitida (127.0.0.2 a 127.0.0.9).")
            conn.sendall(str.encode("IP inválido. Por favor, forneça um IP dentro da faixa permitida."))
            client_name = conn.recv(1024).decode().strip()
```

Fonte: Elaborado pelo autor (2024). .

Para melhor explicação será dividida em partes:

A.6.1 Entradas

Figura 39 – Entrada de parâmetros para DHCP.



Fonte: Elaborado pelo autor (2024). .

Primeiramente a função recebe as seguintes entradas.

- **Conn**: O objeto de conexão do cliente.
- **client name**: O nome do cliente, inicialmente recebido como ’0.0.0.0’ se o cliente quer um IP dinâmico.
- **clients**: Um dicionário que mapeia nomes de clientes (IPs) para suas conexões.
- **used ips**: Um conjunto que contém os IPs que já foram atribuídos.

A.6.2 Loop Faixa de IP

Este loop garante que o processo de atribuição continue até que um IP válido seja atribuído ou a conexão seja encerrada.

Figura 40 – Loop Faixa de IP.

Fonte: Elaborado pelo autor (2024). .

if client name == ’0.0.0.0’:

- verifica se o cliente solicitou um ip digitando 0.0.0.0

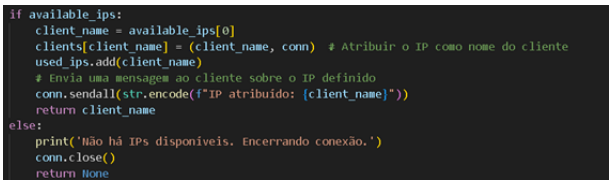
available ips = [f’127.0.0.i’ for i in range(2, 10) if f’127.0.0.i’ not in used ips]

Se a condição for verdadeira logo é atribuído um IP para cliente segundo a faixa permitida

A.6.3 Resposta do servido sobre o IP selecionado.

Essa parte é responsável por definir o ip a da uma resposta ao cliente com ip definido.

Figura 41 – Resposta do servido sobre o IP selecionado.



Fonte: Elaborado pelo autor (2024). .

if available ips:

- É checar se a lista **available ips** não está vazia, ou seja, se existem IPs disponíveis.

  client name = available ips[0]

- O primeiro IP disponível na lista é atribuído ao cliente.

  clients[client name] = (client name, conn)

- Adiciona uma entrada no dicionário **client**s com o IP como chave e uma tupla **(client name, conn)** como valor.

  used ips.add(client name)

- Adiciona o IP ao conjunto **used ips** para marcar que ele está em uso.

  conn.sendall(str.encode(f"IP atribuído: client name"))

- Envia uma mensagem ao cliente informando o IP que foi atribuído a ele.

  return client name

- Retornar o IP atribuído (agora o novo nome do cliente) para o chamador da função.

  else:

  print('Não há IPs disponíveis. Encerrando conexão.')

  conn.close()

  return None

Essa parte lida com a situação em que não há IPs disponíveis.Logo,imprimi uma mensagem no servidor, para encerrar a conexão com o cliente e retornar None para indicar que a atribuição falhou.

A.6.4 Caso a condição de 0.0.0.0 não seja atendida

A função verifica se o cliente forneceu um IP específico dentro da faixa permitida (127.0.0.2 a 127.0.0.9),se o IP fornecido estiver dentro da faixa e não estiver em uso, ele é atribuído ao cliente. Se o IP já estiver em uso, o cliente é solicitado a fornecer outro IP,se o IP fornecido estiver fora da faixa permitida, o cliente é informado e solicitado a fornecer um IP válido.

Figura 42 – Caso a condição de 0.0.0.0 não seja atendida.

```
elif client_name.startswith('127.0.0.') and int(client_name.split('.')[3]) in range(2, 10):
    if client_name not in used_ips:
        clients[client_name] = (client_name, conn)  # Atribuir o IP como nome do cliente
        used_ips.add(client_name)
        # Envia uma mensagem ao cliente sobre o IP definido
        conn.sendall(str.encode(f"IP atribuído: {client_name}"))
        return client_name  # Retorna o nome do cliente atualizado
    else:
        print('O IP fornecido já está em uso. Solicitando outro IP válido ao cliente...')
        conn.sendall(str.encode("IP está em uso. Por favor, insira outro IP na faixa permitida."))
        client_name = conn.recv(1024).decode().strip()  # Solicita outro IP válido ao cliente
else:
    print('O IP fornecido não está dentro da faixa permitida (127.0.0.2 a 127.0.0.9).')
    conn.sendall(str.encode("IP inválido. Por favor, forneça um IP dentro da faixa permitida."))
    client_name = conn.recv(1024).decode().strip()
```

Fonte: Elaborado pelo autor (2024). .

elif client name.startswith('127.0.0.') and int(client name.split('.')[3]) in range(2, 10):

- Verificar se o cliente forneceu um IP específico dentro da faixa permitida.

  client name.startswith('127.0.0.'):

- Verifica se o IP fornecido começa com 127.0.0.

  int(client name.split('.')[3]) in range(2, 10):

- Verifica se o último octeto do IP (a quarta parte do IP) está dentro do intervalo permitido (2 a 9).

  if client name not in used ips:

- Certificar-se de que o IP fornecido pelo cliente não está em uso.

  clients[client name] = (client name, conn)

  used ips.add(client name)

- Adiciona uma entrada no dicionário clients com o IP como chave e uma tupla (**client name, conn**) como valor e adiciona o IP ao conjunto **used ips** para marcar que ele está em uso.

conn.sendall(str.encode(f"IP atribuído: client name"))

- Informar o cliente que o IP fornecido foi aceito e atribuído a ele.
-

return client name

- Retornar o IP fornecido (nome do cliente) para o chamador da função.

Caso o IP Não Esteja na Faixa Permitida

else:

print('O IP fornecido não está dentro da faixa permitida (127.0.0.2 a 127.0.0.9).')

conn.sendall(str.encode("IP inválido. Por favor, forneça um IP dentro da faixa permitida."))

client name = conn.recv(1024).decode().strip()

- Essa parte ,informar o cliente que o IP fornecido é inválido e solicitar um IP dentro da faixa permitida,depois aguarda o cliente enviar um novo IP válido.

Com isso termina a revisão do codigo base para esse projeto.