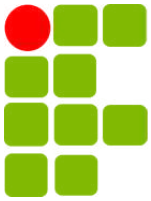


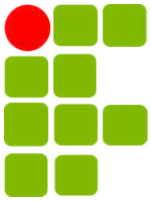
Instituto Federal de Educação, Ciência e Tecnologia da Paraíba
Campus Campina Grande
Coordenação do Curso Superior de Tecnologia em Telemática

Programação e Redes de Computadores: Uma Abordagem Integrada para Automação e Otimização de Processos de Rede

JOSEFFER MAXWEL OLIVEIRA DAS MERCÊS

Orientador: DR. MARCELO PORTELA SOUSA

Campina Grande, dezembro de 2025



Instituto Federal de Educação, Ciência e Tecnologia da Paraíba
Campus Campina Grande
Coordenação do Cursos Superior de Tecnologia em Telemática

Programação e Redes de Computadores: Uma Abordagem Integrada para Automação e Otimização de Processos de Rede

JOSEFFER MAXWEL OLIVEIRA DAS MERCÊS

Monografia apresentada à Coordenação do Curso Superior de Tecnologia em Telemática do IFPB - Campus Campina Grande, como requisito parcial para conclusão do curso de Tecnologia em Telemática.

Orientador: DR. MARCELO PORTELA SOUSA

Campina Grande, dezembro de 2025

Catalogação na fonte:
Ficha catalográfica

M554p Mercês, Joseffer Maxwel Oliveira das.
Programação e redes de computadores: uma abordagem integrada para automação e otimização de processos de rede / Joseffer Maxwel Oliveira das Mercês. - Campina Grande, 2025.
76f.: il.

Trabalho de Conclusão de Curso (Graduação em Tecnologia em Telemática) - Instituto Federal da Paraíba, 2025.
Orientador: Prof. Dr . Marcelo Portela Sousa.

1. Redes de computadores - Automação. 2. Programação de computadores - Phyton. 3. Protocolos de rede. 4. Otimização de processos. 5. Telemática. I. Sousa, Marcelo Portela. II Título.

CDU 004.7:004.42

Programação e Redes de Computadores: Uma Abordagem Integrada para Automação e Otimização de Processos de Rede

JOSEFFER MAXWEL OLIVEIRA DAS MERCÊS

DR. MARCELO PORTELA SOUSA
Orientador

DR. ANDERSON FABIANO BATISTA FERREIRA DA COSTA
Membro da Banca

MRS IANA DAYA CAVALCANTE FACUNDO PASSOS
Membro da Banca

Campina Grande, Paraíba, Brasil
dezembro/2025

Este trabalho é dedicado a todos que contribuíram para meu desenvolvimento acadêmico e profissional.

"Uma máquina consegue fazer o trabalho de 50 homens ordinários. Nenhuma máquina consegue fazer o trabalho de um homem extraordinário"

Elbert Hubbard

Agradecimentos

A realização deste trabalho de conclusão de curso representa não apenas o fechamento de um ciclo acadêmico, mas também a culminância de uma jornada marcada por aprendizado, desafios e apoio incondicional de pessoas que foram fundamentais para que eu chegasse até aqui. Este é um momento especial para expressar minha gratidão a todos que, de alguma forma, contribuíram para o desenvolvimento deste projeto e para o meu crescimento pessoal e profissional.

Agradeço, com profunda admiração, ao meu orientador, Professor Dr. Marcelo Portela Sousa, cuja dedicação, paciência e conhecimento foram essenciais para a realização deste trabalho. Sua orientação precisa, com sugestões de melhorias e incentivo constante, foi decisiva para que eu pudesse estruturar e aprimorar este TCC. Além disso, sua disponibilidade para esclarecer dúvidas, mesmo nos momentos mais corridos, e seu apoio incondicional durante minha participação na ICT Competition (2024-2025, 2025-2026), onde consegui me classificar consecutivamente, foram fundamentais. O Professor Portela não apenas compartilhou seu vasto conhecimento, mas também me inspirou a buscar sempre o melhor, com uma postura ética e profissional que levarei como exemplo ao longo de minha carreira.

Aos membros da banca, Dr. Anderson Fabiano Batista Ferreira da Costa e Mrs. Iana Daya Cavalcante Facundo Passos, meu sincero agradecimento por aceitarem o convite para avaliar este trabalho. Suas críticas construtivas e sugestões valiosas enriqueceram significativamente o projeto, permitindo que ele atingisse um nível de qualidade que não seria possível sem suas contribuições. Agradeço pela atenção, pelo tempo dedicado e pelo compromisso em me ajudar a tornar este trabalho ainda melhor.

Não poderia deixar de expressar minha gratidão aos meus amigos João Vitor Nepomuceno Máximo e Adonias Alves Cardoso Junior, companheiros inseparáveis ao longo dos três anos de curso. Juntos, enfrentamos momentos de alegria, desafios e até mesmo desespero, mas a amizade e o apoio mútuo fizeram com que cada obstáculo fosse mais leve. Vocês foram mais do que colegas de curso; foram irmãos que estiveram ao meu lado em cada etapa, tornando esta jornada inesquecível. Agradeço por cada conversa, cada ajuda nos estudos e por todos os momentos compartilhados que ficarão guardados em minha memória.

Também deixo meu agradecimento especial aos meus colegas de classe, que fizeram parte dessa caminhada desde o início. Compartilhamos momentos de felicidade, conquistas, agonia e desespero, mas, acima de tudo, aprendemos e crescemos juntos. Foram quatro anos de convivência, aprendizado e amizade, que tornaram essa trajetória ainda mais significativa e

repleta de boas lembranças. Foi muito bom ter conhecido cada um de vocês e poder dividir essa etapa tão importante da vida lado a lado.

À minha família, dedico um agradecimento especial, pois sem vocês nada disso seria possível. À minha mãe, Silvana Xavier de Oliveira, e ao meu pai, Josinaldo Figueiredo das Mercês, agradeço pelo amor incondicional, pelo suporte emocional e financeiro, e pelos valores que me ensinaram e que moldaram quem sou hoje. Vocês sempre acreditaram em mim, mesmo quando eu duvidava de mim mesmo, e isso foi a base para que eu chegasse até aqui.

Aos meus irmãos, Jefferson Maximiliano Oliveira das Mercês e Jennifer Beatriz Oliveira das Mercês, meu carinho e gratidão. Jefferson, sua orientação na área de Telemática, desde a sugestão para ingressar no curso até os conselhos valiosos sobre protocolos e preparação para a ICT Competition, despertou em mim uma paixão ainda maior pelo aprendizado. Jennifer, sua ajuda direta na conclusão deste trabalho, com orientações sobre normas, regras e pontos de melhoria, foi indispensável para que eu pudesse finalizar esta etapa com confiança.

Um agradecimento especial à Emanuelle Araújo Sousa, que desempenhou um papel fundamental na primeira etapa deste trabalho. Sua colaboração inicial, com ideias e apoio, foi essencial para dar os primeiros passos nesta pesquisa. Mesmo com a mudança de rumo do projeto, conforme sugestões da banca, sua contribuição foi valiosa e será sempre lembrada com carinho.

Por fim, agradeço ao Instituto Federal da Paraíba (IFPB), que proporcionou não apenas o ambiente acadêmico necessário para a realização deste trabalho, mas também oportunidades de crescimento, aprendizado e conexões que transformaram minha trajetória. A todos que, de alguma forma, cruzaram meu caminho e contribuíram para esta conquista, meu mais sincero obrigado. Vocês fizeram parte de um capítulo muito importante da minha vida.

Resumo

Este trabalho apresenta o desenvolvimento da ferramenta *Huawei Network Automation Tool*, uma aplicação voltada para a automação de redes de computadores, projetada para otimizar o tempo, ampliar a segurança e aumentar a escalabilidade nas tarefas de configuração e gerenciamento de dispositivos de rede. A ferramenta foi desenvolvida em *Python*, integrando protocolos amplamente utilizados, como o *Network Configuration Protocol* (NETCONF) e o *Secure Shell version 2 (SSHv2)*, aliados às bibliotecas *Paramiko* e *ncclient*, que viabilizam a comunicação remota e segura com os equipamentos. O sistema também incorpora o uso de chaves criptográficas do tipo *RSA*, essenciais para a autenticação e o estabelecimento de conexões protegidas. Sua arquitetura modular reúne, em uma única interface, funcionalidades como aplicação de configurações, execução de comandos, realização de cópias de segurança (*backup*), verificação de conectividade e registro de logs do sistema. Os testes realizados no simulador *Enterprise Network Simulation Platform* (*eNSP*) demonstraram que a ferramenta reduz significativamente o tempo de configuração, padroniza processos e minimiza erros manuais, tornando a administração de redes mais eficiente e confiável. O estudo reforça que a integração entre programação e redes de computadores é essencial para o desenvolvimento de soluções modernas, seguras e escaláveis no contexto da engenharia de redes.

Palavras-chave: automação de redes; programação em Python; *scripts* de automação; protocolos de comunicação; lógica de programação; eficiência operacional.

Abstract

This work presents the development of the *Huawei Network Automation Tool*, an application designed to automate computer networks, aiming to optimize time, enhance security, and increase scalability in configuration and device management tasks. The tool was developed in *Python*, integrating widely used protocols such as the *Network Configuration Protocol* (NETCONF) and *Secure Shell version 2 (SSHv2)*, together with the *Paramiko* and *ncclient* libraries, which enable remote and secure communication with network equipment. The system also incorporates the use of *RSA* cryptographic keys, essential for authentication and the establishment of protected connections. Its modular architecture combines, in a single interface, essential functionalities such as configuration deployment, command execution, backup operations, connectivity verification, and system log recording. Tests performed using the *Enterprise Network Simulation Platform* (*eNSP*) demonstrated that the tool significantly reduces configuration time, standardizes processes, and minimizes human errors, making network management more efficient and reliable. The study highlights that the integration between programming and computer networks is crucial for the development of modern, secure, and scalable solutions within the context of network engineering.

Keywords: network automation; Python programming; automation *scripts*; communication protocols; programming logic; operational efficiency.

Sumário

Lista de Siglas

ACE	Máquina de Computação Automática (*Automatic Computing Engine*)
ACK	Número de Confirmação (*Acknowledgment Number*)
ARP	Protocolo de Resolução de Endereços (*Address Resolution Protocol*)
ARP SPOOFING	Falsificação de ARP (*Address Resolution Protocol Spoofing*)
ARPANET	Rede da Agência de Projetos de Pesquisa Avançada (*Advanced Research Projects Agency Network*)
ASSEMBLY	Linguagem de Montagem (*Assembly Language*)
ATM	Caixa Eletrônico (*Automated Teller Machine*)
AWS CLI	Interface de Linha de Comando da AWS (*Amazon Web Services Command Line Interface*)
BASH	*Bourne Again Shell*
CIDR	Roteamento Sem Classe Entre Domínios (*Classless Inter-Domain Routing*)
CLI	Interface de Linha de Comando (*Command Line Interface*)
CERN	Organização Europeia para Pesquisa Nuclear (*European Organization for Nuclear Research*)
CPU	Unidade Central de Processamento (*Central Processing Unit*)
DOS	Negação de Serviço (*Denial of Service*)
DOCKER CLI	Interface de Linha de Comando do Docker (*Docker Command Line Interface*)
ENSP	Plataforma de Simulação de Redes Empresariais (*Enterprise Network Simulation Platform*)
FTP	Protocolo de Transferência de Arquivos (*File Transfer Protocol*)
GIT	Rastreador de Informações Globais (*Global Information Tracker*)

GIT BASH	Terminal Bash do Git (*Global Information Tracker Bash*)
HTTP	Protocolo de Transferência de Hipertexto (*Hypertext Transfer Protocol*)
IETF	Força-Tarefa de Engenharia da Internet (*Internet Engineering Task Force*)
ICMP	Protocolo de Mensagens de Controle da Internet (*Internet Control Message Protocol*)
IHC	Interação Humano-Computador (*Human–Computer Interaction*)
IP	Protocolo da Internet (*Internet Protocol*)
IPV4	Protocolo da Internet Versão 4 (*Internet Protocol Version 4*)
LAN	Rede Local (*Local Area Network*)
MAC	Controle de Acesso ao Meio (*Media Access Control*)
MITM	Ataque Homem-no-Meio (*Man-in-the-Middle*)
NAPALM	Abstração de Automação e Programabilidade de Rede Multivendor (*Network Automation and Programmability Abstraction Layer with Multivendor Support*)
NCP	Protocolo de Controle de Rede (*Network Control Protocol*)
NETBIOS	Sistema Básico de E/S da Microsoft (*Network Basic Input/Output System*)
NETCONF	Protocolo de Configuração de Rede (*Network Configuration Protocol*)
NUMPY	Python Numérico (*Numerical Python*)
OSI	Interconexão de Sistemas Abertos (*Open Systems Interconnection*)
RAM	Memória de Acesso Aleatório (*Random Access Memory*)
RPCS	Chamadas de Procedimento Remoto (*Remote Procedure Calls*)
RSA	Algoritmo *Rivest–Shamir–Adleman*
SDN	Rede Definida por Software (*Software-Defined Networking*)
SEQ	Número de Sequência (*Sequence Number*)
SIG	Sistema de Informação Geográfica (*Geographic Information System*)
SMTP	Protocolo de Transferência de Correspondência Simples (*Simple Mail Transfer Protocol*)
SNMP	Protocolo Simples de Gerenciamento de Rede (*Simple Network Management Protocol*)

SSH	Shell Seguro (*Secure Shell*)
SSH-TRANS	Protocolo de Transporte do SSH (*SSH Transport Layer Protocol*)
SSH-USERAUTH	Protocolo de Autenticação de Usuário do SSH (*SSH User Authentication Protocol*)
SSHV1	Protocolo *Secure Shell* Versão 1
SSHV2	Protocolo *Secure Shell* Versão 2
TCP	Protocolo de Controle de Transmissão (*Transmission Control Protocol*)
TCP/IP	Suíte de Protocolos TCP/IP (*Transmission Control Protocol/Internet Protocol*)
TELNET	Rede de Telecomunicações (*Telecommunication Network*)
UNIX	Sistema Operacional Unix (*Uniplexed Information and Computing System*)
UNIX-LIKE	Similar ao Unix (*Uniplexed Information and Computing System-like*)
VIRTUALBOX	Máquina Virtual Oracle (*Oracle VM VirtualBox*)
WWW	Rede Mundial de Computadores (*World Wide Web*)
XML	Linguagem de Marcação Extensível (*Extensible Markup Language*)
XML-RPC	Chamada de Procedimento Remoto em XML (*Extensible Markup Language – Remote Procedure Call*)
YANG	Outra Geração de Modelagem (*Yet Another Next Generation*)

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A trajetória da computação moderna é marcada pela busca constante por formas mais eficientes de comunicação entre homem e máquina. Desde os fundamentos teóricos propostos por Alan Turing, na década de 1930, a ideia de instruir uma máquina a partir de sequências lógicas e estruturadas impulsionou o surgimento das linguagens de programação de alto nível, que tornaram o desenvolvimento de sistemas mais acessível, compreensível e automatizado. Essa evolução conceitual não apenas transformou o modo como os programas são construídos, mas também influenciou diretamente áreas como redes de computadores, segurança da informação e automação de processos.

Ao longo das últimas décadas, as redes de computadores evoluíram de simples conexões experimentais, como a *Advanced Research Projects Agency Network (ARPANET)* , para sistemas globais e complexos que sustentam praticamente todas as atividades digitais contemporâneas. Essa expansão trouxe o desafio da administração eficiente de dispositivos e serviços interconectados. O aumento do tráfego de dados, a diversidade de protocolos e a necessidade de confiabilidade exigiram abordagens cada vez mais padronizadas e seguras para o gerenciamento das infraestruturas de rede. Nesse cenário, a integração entre programação e redes consolidou-se como um eixo essencial para o avanço tecnológico e a redução de custos operacionais.

A automação de redes surge, assim, como um reflexo direto dessa integração. Por meio de linguagens como o *Python*, é possível transformar tarefas manuais e repetitivas como configuração, monitoramento e manutenção de equipamentos em rotinas automáticas executadas com rapidez, consistência e segurança. O uso de bibliotecas especializadas, como *Paramiko* e *ncclient*, amplia as possibilidades dessa automação ao permitir o controle remoto de dispositivos por meio de protocolos como o *Secure Shell version 2 (SSHv2)* e o *Network Configuration Protocol (NETCONF)*, garantindo autenticidade, integridade e confiabilidade às operações.

Esses avanços também se relacionam ao surgimento de ferramentas e ambientes de simulação que favorecem a prática e o aprendizado. O *Enterprise Network Simulation Platform (eNSP)*, desenvolvido pela Huawei, permite a criação de topologias virtuais que simulam o comportamento de dispositivos reais, enquanto o *Git Bash* fornece um ambiente de linha de

comando compatível com sistemas baseados em *Unix*, facilitando o uso de protocolos seguros e o controle de versões em projetos de automação. A combinação dessas ferramentas cria um ecossistema robusto para o desenvolvimento, teste e validação de soluções voltadas à administração de redes.

Dessa forma, este trabalho aborda o papel da programação como instrumento de automação e otimização de redes de computadores, analisando conceitos teóricos, protocolos e ferramentas que sustentam essa integração tecnológica. Busca-se compreender como a evolução das linguagens de programação e dos modelos de comunicação, aliados a protocolos de gerenciamento como o *NETCONF*, pode contribuir para a construção de infraestruturas mais eficientes, seguras e escaláveis.

1.1 Justificativa e Relevância do Trabalho

A administração de redes de computadores ainda é, em muitos ambientes corporativos e educacionais, realizada por meio de métodos tradicionais, que envolvem a configuração manual de dispositivos e a execução repetitiva de comandos diretamente nos equipamentos. Esse modelo de gerenciamento, além de demandar tempo e esforço humano consideráveis, aumenta o risco de falhas operacionais e inconsistências entre os dispositivos. Em uma infraestrutura moderna, composta por dezenas ou centenas de elementos interconectados, a abordagem manual torna-se inviável e ineficiente.

Nesse contexto, a automação de redes surge como uma resposta necessária às limitações desses métodos convencionais. Por meio do uso de linguagens de programação como o *Python* e de protocolos padronizados, como o *NETCONF* e o *SSHv2*, é possível criar rotinas que executam tarefas de configuração, monitoramento e manutenção de forma automática, precisa e reproduzível. A execução dessas operações por meio de *scripts* programados reduz drasticamente o tempo necessário para aplicar configurações, que antes poderiam levar horas ou dias , passando a ser concluídas em poucos segundos ou minutos, com maior eficiência e segurança.

Além de otimizar o tempo e reduzir falhas humanas, a automação proporciona ganhos significativos em escalabilidade e padronização. Em redes corporativas ou institucionais, onde múltiplos dispositivos precisam ser configurados de maneira uniforme, o uso de ferramentas automatizadas assegura consistência entre os equipamentos e facilita futuras atualizações. Essa padronização também contribui para a segurança operacional, uma vez que diminui a exposição a erros de digitação e a configurações incorretas que poderiam comprometer a disponibilidade da rede.

A relevância deste trabalho está em demonstrar e fundamentar o potencial da automação de redes como um caminho para a modernização das práticas de gerenciamento. Ao integrar programação, protocolos de comunicação e ferramentas de simulação, o estudo evidencia como a automação pode tornar o ambiente de rede mais eficiente, escalável e seguro. Além

disso, a utilização de tecnologias acessíveis, como o *Python*, o *Paramiko*, o *ncclient* e o simulador *eNSP*, reforça a aplicabilidade prática da proposta, permitindo sua adoção tanto em contextos acadêmicos quanto em ambientes profissionais.

Assim, o presente trabalho justifica-se pela necessidade de acompanhar a evolução tecnológica e incorporar soluções automatizadas ao gerenciamento de redes de computadores. A abordagem proposta contribui não apenas para o aprimoramento técnico dos processos, mas também para a formação de profissionais mais capacitados e alinhados às demandas contemporâneas de eficiência, segurança e inovação no campo da Engenharia de Redes.

1.2 Objetivos

1.2.1 Objetivo Geral

Desenvolver uma ferramenta de automação para redes de computadores capaz de realizar, de forma segura e eficiente, a configuração, o monitoramento e o gerenciamento remoto de dispositivos por meio de linguagens de programação e protocolos padronizados.

1.2.2 Objetivos Específicos

- Compreender os fundamentos teóricos de programação, algoritmos e redes de computadores, identificando sua integração no contexto da automação;
- Estudar os protocolos e arquiteturas essenciais para o gerenciamento de redes, com destaque para o *TCP*, *IPv4*, *ARP*, *ICMP*, *SSHv2* e *NETCONF*;
- Analisar o papel das linguagens de alto nível, em especial o *Python*, na automação de tarefas e na criação de soluções escaláveis e seguras;
- Utilizar bibliotecas como *Paramiko* e *ncclient* para estabelecer conexões seguras e realizar operações automatizadas sobre dispositivos de rede;
- Empregar o simulador *eNSP* para testar e validar o funcionamento da ferramenta desenvolvida em um ambiente controlado e realista;
- Utilizar o *Git Bash* como ambiente de apoio para a criação de chaves criptográficas e execução de comandos remotos via *SSHv2*, assegurando a comunicação segura entre os sistemas;
- Demonstrar, por meio da ferramenta desenvolvida, a eficiência e a praticidade da automação de redes, evidenciando sua capacidade de otimizar processos simples e complexos em comparação aos métodos manuais tradicionais.

1.3 Metodologia

Este estudo caracteriza-se como uma pesquisa qualitativa, de natureza aplicada, voltada ao desenvolvimento e validação funcional de uma ferramenta de automação para redes de computadores. A metodologia adotada busca demonstrar, de forma prática, como a integração entre linguagens de programação, protocolos de comunicação e bibliotecas especializadas pode otimizar processos de gerenciamento e configuração de dispositivos.

A condução do trabalho teve início com uma pesquisa teórica contemplando materiais acadêmicos, documentações técnicas e artigos relacionados à automação de redes, ao protocolo *SSHv2*, ao *NETCONF*, às bibliotecas *Paramiko* e *ncclient*, e aos fundamentos de programação necessários para a implementação da solução. Essa etapa fundamentou as decisões técnicas adotadas no desenvolvimento da aplicação.

A etapa seguinte consistiu no desenvolvimento da ferramenta, implementada em *Python* e estruturada em módulos independentes, cada um responsável por uma funcionalidade específica. A aplicação reúne em uma única interface operações como aplicação de configurações, extração de *backups*, testes de conectividade, acesso ao console *SSH*, gerenciamento de conexões e registro de logs. A modularização foi adotada para facilitar a manutenção, a organização interna do código e a escalabilidade do sistema.

Após o desenvolvimento, a ferramenta passou pela etapa de validação funcional, realizada em um ambiente controlado de testes utilizando um equipamento configurado como servidor de rede. Foram observados aspectos como estabilidade das conexões, respostas dos dispositivos, execução das rotinas automatizadas e funcionamento integrado dos módulos. A análise teve caráter qualitativo, priorizando a verificação prática das funcionalidades implementadas e sua coerência com os objetivos da aplicação.

A coleta das informações ocorreu por meio da observação direta dos resultados produzidos em cada módulo da ferramenta, sem necessidade de medições quantitativas ou comparações com ambientes complexos. A validação concentrou-se na aplicabilidade prática da solução, evidenciando ganhos em agilidade, padronização e confiabilidade proporcionados pela automação.

Dessa forma, a metodologia adotada integra pesquisa teórica, desenvolvimento da aplicação, modelagem modular e validação prática em ambiente controlado. Essa abordagem permitiu estruturar o sistema, testar suas funcionalidades e verificar sua aderência aos requisitos definidos.

1.4 Organização do Documento

No **Capítulo 1**, é apresentada a introdução do trabalho, abordando o contexto, a justificativa, os objetivos e a relevância da proposta, bem como a delimitação do problema e o escopo da ferramenta desenvolvida.

O **Capítulo 2** apresenta a fundamentação teórica que embasa o estudo, abordando os

principais conceitos relacionados à programação, automação de redes, protocolos de comunicação e bibliotecas utilizadas na implementação da ferramenta *eNS*. Este capítulo também contextualiza a importância da integração entre programação e redes de computadores no cenário atual da tecnologia.

No **Capítulo 3**, são descritos os resultados obtidos a partir da aplicação prática da ferramenta, incluindo a explicação de suas funcionalidades, os testes realizados no simulador *eNSP* e a análise da eficiência, segurança e usabilidade da solução proposta.

O **Capítulo 4** apresenta as considerações finais do trabalho, destacando as conclusões alcançadas com o desenvolvimento da ferramenta e sugerindo possíveis aprimoramentos e extensões para trabalhos futuros, como a inclusão de novos módulos e mecanismos de segurança.

Por fim, o **Apêndice A** reúne os registros técnicos e as configurações utilizadas durante o processo de implementação e testes, incluindo os comandos aplicados no simulador *eNSP*, as configurações realizadas na nuvem do *eNSP*, os procedimentos executados no *Git Bash* e o link para o repositório oficial da ferramenta no *GitHub*.

Capítulo 2

Fundamentação Teórica

2.1 O Que é Programação? Uma Perspectiva Histórica e Conceitual

A programação pode ser compreendida como o processo de construção de instruções lógicas que orientam o computador na execução de tarefas específicas ou na resolução de determinados problemas. Essas instruções são organizadas em algoritmos, sequências estruturadas de passos que visam alcançar um objetivo definido. Na atualidade, a programação está presente em praticamente todas as áreas da tecnologia, desde os *sistemas operacionais* e dispositivos embarcados até aplicações complexas de *inteligência artificial* e automação de redes.

A origem da programação está intimamente ligada ao desenvolvimento da computação moderna e à busca por automatizar tarefas que exigiam raciocínio humano. Essa trajetória teve início de forma mais clara durante a Segunda Guerra Mundial, quando avanços científicos e matemáticos foram impulsionados pela necessidade de processar grandes volumes de informações em pouco tempo. Nesse cenário, o matemático britânico *Alan Turing* teve papel determinante ao formular os princípios teóricos que fundamentam toda a lógica computacional contemporânea [Bowen 2018, p. 4–5].

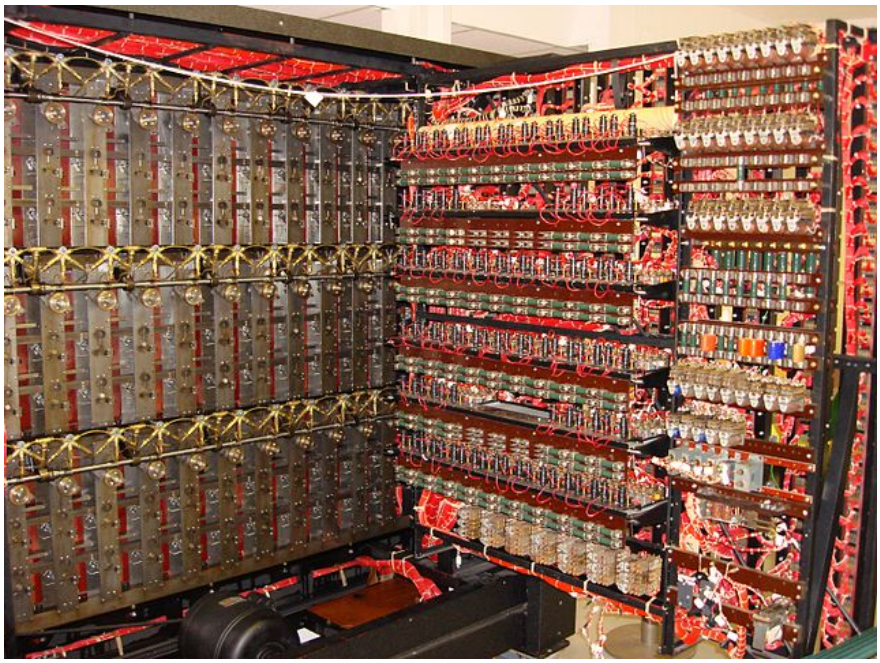
Em seu artigo *On Computable Numbers*, publicado em 1936, Turing demonstrou que qualquer problema passível de ser resolvido de forma lógica poderia também ser executado por uma máquina, desde que essa máquina seguisse um conjunto de instruções bem definidas [Silva *et al.* 2024, p. 1048]. Essa proposta deu origem ao conceito de *Máquina de Turing*, um modelo teórico de computador universal, capaz de executar qualquer cálculo computável. A partir dessa formulação, surgiu o princípio central da programação: toda tarefa, por mais complexa que seja, pode ser decomposta em instruções simples, ordenadas e lógicas.

Durante a Segunda Guerra Mundial, Turing aplicou esses conceitos na prática, ao trabalhar em *Bletchley Park* no desenvolvimento da máquina *Bombe*, projetada para decifrar os códigos da máquina alemã *Enigma* [Silva *et al.* 2024, p. 1049–1050]. A *Bombe* representou um dos primeiros exemplos de automação baseada em lógica algorítmica: ela executava re-

petitivamente instruções específicas até encontrar a combinação correta. Dessa forma, pode-se considerar que Turing não apenas criou os fundamentos teóricos da computação, mas também introduziu a noção prática de "programar" uma máquina para executar tarefas complexas sem intervenção humana direta.

A Figura 2.1 apresenta a máquina *Bombe*, projetada por *Alan Turing* e sua equipe em *Bletchley Park* durante a Segunda Guerra Mundial. O equipamento possuía uma estrutura predominantemente metálica, composta por centenas de cilindros eletromecânicos, fios de cobre e painéis de controle, funcionando como um sistema automatizado de decodificação. Sua operação combinava elementos mecânicos e elétricos, permitindo testar milhares de combinações de cifras por segundo para quebrar os códigos da máquina alemã *Enigma*. Esse processo de repetição lógica e automatizada representou um marco na história da computação, pois demonstrou, pela primeira vez, a aplicação prática de conceitos algorítmicos em uma máquina física [Silva *et al.* 2024, p. 1049–1050].

Figura 2.1: *Máquina Bombe.*



Fonte: Wikimedia Commons, 2007.

Após o término da guerra, Turing deu continuidade às suas pesquisas e propôs o projeto *Automatic Computing Engine (ACE)*, um dos primeiros computadores projetados para ser totalmente programável [Brandão 2019, p. 14–15]. Nessa proposta, ele já antecipava ideias que, décadas mais tarde, estariam associadas à automação e à inteligência artificial. Turing acreditava que as máquinas poderiam aprender e adaptar-se a novas situações, desde que programadas com base em instruções adequadas, um conceito que, na prática, deu origem ao pensamento algorítmico e à lógica de controle que sustentam as tecnologias atuais de automação e redes.

De maneira simplificada, pode-se comparar um algoritmo a uma receita de bolo: uma

sequência de etapas a serem seguidas para alcançar um resultado específico. A programação, portanto, é o processo de traduzir essa sequência em linguagem que o computador possa compreender e executar. Como afirmou Turing, "uma máquina pode ser programada para realizar qualquer tarefa que possa ser descrita de forma algorítmica" [Brandão 2019, p. 79]. Essa visão continua atual e fundamenta desde a criação de pequenos programas até a automação de sistemas complexos de comunicação e redes.

Assim, compreender o papel histórico de Turing e o surgimento da programação não é apenas um exercício de memória científica, mas uma forma de entender como a lógica algorítmica evoluiu para sustentar tecnologias contemporâneas, como os sistemas distribuídos, o monitoramento automatizado e as soluções baseadas em *Python* e *SSHv2* que integram o contexto moderno de redes de computadores.

2.2 De Turing ao Python: A Evolução das Linguagens de Alto Nível

A evolução das linguagens de programação está diretamente ligada à busca humana por simplificar a comunicação com as máquinas. Desde os primeiros conceitos teóricos desenvolvidos por *Alan Turing*, a ideia central sempre foi permitir que o ser humano pudesse expressar instruções de forma lógica, compreensível e executável por um computador. Turing, ao formular a *Máquina de Turing*, estabeleceu não apenas um modelo matemático de computação, mas também o princípio da lógica sequencial, a noção de que todo problema computável pode ser resolvido por meio de um conjunto de passos organizados, isto é, por um algoritmo [Bowen 2018, p. 4–5].

Esse conceito serviu como base para o desenvolvimento das linguagens de programação. Nas décadas seguintes à Segunda Guerra Mundial, programar ainda significava lidar diretamente com o *hardware*, escrevendo instruções em código de máquina ou linguagem de montagem (*Assembly*). Esse processo era extremamente trabalhoso e sujeito a erros. Assim, surgiu a necessidade de criar linguagens mais próximas da linguagem humana, capazes de traduzir automaticamente os comandos do programador para o código binário compreendido pelo computador. Foi o início das chamadas linguagens de alto nível, que tornaram o ato de programar mais acessível e produtivo [Chun 2001, p. 12].

A lógica de programação, que deriva diretamente do pensamento matemático de Turing, constitui a base dessas linguagens. Ela envolve organizar o raciocínio de forma estruturada, definindo etapas lógicas para resolver um problema, o que conhecemos como algoritmo. Antes de escrever um código, o programador precisa compreender essa lógica, pois ela representa a transição entre o pensamento humano e a execução automatizada. Essa estrutura de raciocínio é o que permite transformar uma ideia ou necessidade em uma solução computacional, independentemente da linguagem utilizada [Menezes 2010, p. 21].

Dentro dessa evolução histórica, o *Python* representa uma das etapas mais significativas.

Criado por Guido van Rossum no final da década de 1980, o *Python* foi projetado para ser uma linguagem de alto nível, poderosa, mas também simples e legível. Segundo o próprio Rossum, a proposta era criar uma linguagem "que combinasse clareza e eficiência, sem sacrificar a expressividade" [Rossum e Jr 2017, p. 8]. Essa simplicidade é justamente o que a torna uma das linguagens mais populares entre iniciantes e profissionais. O *Python* foi pensado para abstrair as complexidades do *hardware* e permitir que o programador se concentrasse mais na lógica, um reflexo direto dos princípios que Turing propôs quase meio século antes.

A Figura 2.2 apresenta um exemplo simples de código escrito em *Python*, cujo objetivo é comparar um número informado pelo usuário com o valor 70. O programa solicita a entrada de um número, armazena-o em uma variável e, por meio de uma estrutura condicional, verifica se o valor é maior, igual ou menor que 70. Caso a condição seja verdadeira, o programa exibe a mensagem correspondente na tela.

Figura 2.2: *Exemplo de programa simples em Python.*

TCC.py > ...

```
1  numero = float(input("Digite um número: "))
2
3  if numero >= 70:
4      print(f"O número {numero} é maior ou igual a 70.")
5  else:
6      print(f"O número {numero} é menor que 70.")
```

Fonte: elaborado pelo autor, 2025.

Esse pequeno exemplo ilustra como o *Python* é uma linguagem intuitiva e de fácil leitura, pois utiliza uma sintaxe próxima da linguagem natural e dispensa o uso de comandos complexos. O código é curto, direto e compreensível até mesmo para quem está iniciando na programação, refletindo o principal objetivo do *Python*: tornar o aprendizado e a automação de tarefas mais acessíveis [Chun 2001, p. 12].

Um dos principais diferenciais do *Python* está em sua ampla biblioteca padrão, frequentemente descrita como "*pilhas incluídas*" [Menezes 2010, p. 22]. Isso significa que a linguagem já vem equipada com um conjunto robusto de módulos e funções, capazes de atender às mais diversas aplicações, desde o acesso a arquivos e manipulação de dados até a criação de interfaces gráficas e o gerenciamento de redes. Além disso, o *Python* possui suporte a inúmeras bibliotecas externas, como *Numerical Python (NumPy)* para computação numérica, *Matplotlib* (para visualização de dados) e *Paramiko* (para automação via *SSHv2*), que ampliam ainda mais suas possibilidades [Chun 2001, p. 9–14].

Graças a essa combinação de simplicidade, poder e flexibilidade, o *Python* consolidou-se como uma das linguagens mais influentes e utilizadas da atualidade. Sua popularidade não se deve apenas à facilidade de aprendizado, mas também à sua aplicabilidade em diversas áreas, desde o desenvolvimento de aplicações *web* e análise de dados até a automação de sistemas e o gerenciamento de redes corporativas. Assim, é possível perceber que a visão de Turing

sobre máquinas capazes de executar qualquer tarefa algorítmica encontrou, no *Python*, uma de suas materializações mais acessíveis e práticas no cenário tecnológico contemporâneo.

2.3 Programação como Ferramenta de Automação e Otimização de Redes

A programação exerce um papel central na automação de redes de computadores, pois possibilita a transformação de tarefas manuais e repetitivas, que são "frequentemente demoradas e propensas a erros" [Prabowo e Ariyanto 2025, p. 1], em processos automáticos. Essa transformação reduz a intervenção humana, aumenta a produtividade e otimiza o tempo de execução das operações. A automação, nesse contexto, consiste no uso de *scripts*, linguagens e bibliotecas capazes de executar configurações, monitoramentos e diagnósticos de forma autônoma, sem a necessidade de comandos diretos e contínuos por parte do administrador de rede.

A aplicação da programação nesse domínio permite a criação de rotinas inteligentes que automatizam ações críticas. O uso de linguagens de alto nível, como o *Python*, possibilita que administradores e engenheiros desenvolvam programas voltados à execução automática de tarefas de gerenciamento, como "configurar *roteadores* e *switches*, monitorar o desempenho da rede e coletar dados de conectividade" [Khoa 2023, p. 3]. Em vez de aplicar comandos manualmente em cada equipamento, é possível desenvolver *scripts* que centralizam e executam múltiplas operações simultaneamente por meio de protocolos seguros, como o *SSHv2*. Essa abordagem é essencial em ambientes complexos, onde o aumento do número de dispositivos torna a configuração individual inviável e suscetível a falhas humanas.

A principal contribuição da programação para a automação está na eficiência operacional e na mitigação de erros. Cañas (2025) destaca que o *Python*, ao facilitar a execução de tarefas concorrentes (ou simultâneas), "maximiza a eficiência do tempo de processamento e minimiza o tempo de inatividade", proporcionando ganhos técnicos e econômicos diretos [Cañas 2025, p. 125]. De forma complementar, Mihailă *et al.* (2017) afirmam que a automação não apenas reduz o tempo de configuração e manutenção de dispositivos, mas também "melhora a segurança da rede" ao permitir a identificação e correção programática de vulnerabilidades, aumentando a estabilidade geral da infraestrutura [Mihăilă *et al.* 2017, p. 1].

Outro aspecto relevante é a confiabilidade obtida por meio da execução programada, especialmente em tarefas críticas como o *backup* de configurações. O *Python* permite a integração de bibliotecas específicas, como *Paramiko*, *Netmiko* e *Network Automation and Programmability Abstraction Layer with Multivendor support (NAPALM)* [Khoa 2023, p. 3], que simplificam o controle remoto de dispositivos e a comunicação via *SSH*. Essas bibliotecas funcionam como interfaces de abstração entre o *software* e o equipamento físico, permitindo que os *scripts* interajam diretamente com os dispositivos de rede [Khoa 2023, p. 3]. Osei-Wusu *et al.* (2020) demonstraram a eficácia dessa abordagem ao implementar um esquema

de automação de *backup* em uma rede local hospitalar, comprovando a robustez e a confiabilidade do método em um ambiente de missão crítica [Osei-Wusu *et al.* 2025, p. 1].

Além de aumentar a eficiência e a confiabilidade, a programação aplicada à automação de redes favorece a escalabilidade e a padronização de processos. Estudos recentes indicam que a automação permite a criação de ferramentas *web* customizadas para gerenciamento de inventário, monitoramento de desempenho e controle de conectividade [Andrade 2024, p. 1]. Essas soluções centralizadas simplificam o gerenciamento de redes complexas e reduzem custos operacionais, tornando a administração mais ágil e segura.

Em síntese, a integração entre lógica de programação e infraestrutura de redes constitui um dos pilares da engenharia de redes moderna. A automação, quando apoiada em linguagens como o *Python*, promove uma combinação de eficiência, padronização e segurança operacional, demonstrando que o domínio da programação é um fator determinante para o avanço e a sustentabilidade das infraestruturas de comunicação atuais.

2.4 Automação de Redes com a Biblioteca Paramiko

A biblioteca *Paramiko*, desenvolvida em *Python*, implementa o protocolo *SSHv2*, permitindo que *scripts* escritos nessa linguagem atuem como clientes *SSH*. O *SSHv2* é um protocolo de rede amplamente utilizado na administração de sistemas e equipamentos, proporcionando conexões remotas seguras entre dispositivos. Conforme destacado por [Campos e Paixo 2019, p. 1, 5, 9], o *Paramiko* facilita o estabelecimento de conexões seguras com dispositivos de rede via *SSHv2*, sendo essencial para a automação de tarefas de gerenciamento e configuração, além de garantir operações mais eficientes e confiáveis.

O uso do *Paramiko* possibilita o gerenciamento e a criação de conexões seguras diretamente em *scripts Python*, permitindo a automação de tarefas que, de outra forma, seriam executadas manualmente e, portanto, sujeitas a erros humanos. Essa automação é particularmente relevante em ambientes de grande escala, como provedores de serviços e redes corporativas, nos quais a estabilidade e a escalabilidade são fundamentais. De acordo com [Campos e Paixo 2019, p. 5] e [Khoa 2023, p. 7, 12, 33], a adoção dessa biblioteca contribui para uma administração de rede mais ágil, padronizada e segura, reduzindo o tempo de configuração e aumentando a eficiência operacional.

Um dos aspectos mais importantes do *Paramiko* está relacionado à segurança proporcionada pelo protocolo *SSHv2*. Diferentemente do *Telecommunication Network (Telnet)*, protocolo mais antigo que transmite dados em texto simples e é suscetível à interceptação, o *SSHv2* realiza a criptografia das informações trocadas entre cliente e servidor, garantindo a confidencialidade, a integridade e a autenticação das comunicações. Conforme [Zadka 2019, p. 111], o uso do *SSHv2* torna as conexões significativamente mais seguras, sendo o padrão recomendado em ambientes que envolvem dados sensíveis. O *Paramiko* oferece múltiplos métodos de autenticação, incluindo senha e chave pública, fortalecendo as práticas de segurança e

dificultando o acesso não autorizado. O uso combinado de chaves públicas e privadas, baseadas em criptografia assimétrica, assegura um controle de acesso eficiente e confiável, mesmo quando há múltiplas conexões simultâneas [Khoa 2023].

Como uma implementação pura em *Python* do protocolo *SSHv2*, o *Paramiko* fornece funcionalidades tanto de cliente quanto de servidor, encapsulando os principais conceitos de comunicação segura e utilizando extensões em linguagem C para operações de criptografia de baixo nível [Mihăilă *et al.* 2017, p. 97]. Quando comparado à biblioteca *Telnetlib*, estudos apontam que, embora esta possa apresentar desempenho ligeiramente superior em alguns contextos devido à ausência de criptografia, o *Paramiko* é amplamente preferido por sua robustez e pela segurança inerente do protocolo *SSHv2* [Alfaresa, Arifwidodo e Khair 2023, p. 227–230].

Para a utilização do *Paramiko* em *scripts Python*, é necessário importar a biblioteca, criar uma instância do cliente *SSH* (*SSHClient*), definir a política de aceitação de chaves de *host* ausentes e, em seguida, conectar-se ao dispositivo desejado, informando o endereço *Internet Protocol* (*IP*) ou o *hostname*, o nome de usuário e o método de autenticação, seja por senha, seja por chave pública. É imprescindível que o dispositivo de rede possua o serviço *SSHv2* devidamente configurado e ativo, garantindo, assim, uma comunicação segura e criptografada entre as partes [Choi 2024].

2.4.1 Arquitetura e Funcionamento do Protocolo SSH

O protocolo *SSH* é um mecanismo amplamente utilizado para garantir conexões remotas seguras entre clientes e servidores. Seu funcionamento é baseado em três princípios fundamentais: autenticação, confidencialidade e integridade. Como afirma Ylonen (2006), o SSH foi projetado para "prover comunicação segura sobre redes inseguras, garantindo confidencialidade, integridade e autenticação das partes envolvidas" [Ylonen 2006, p. 3].

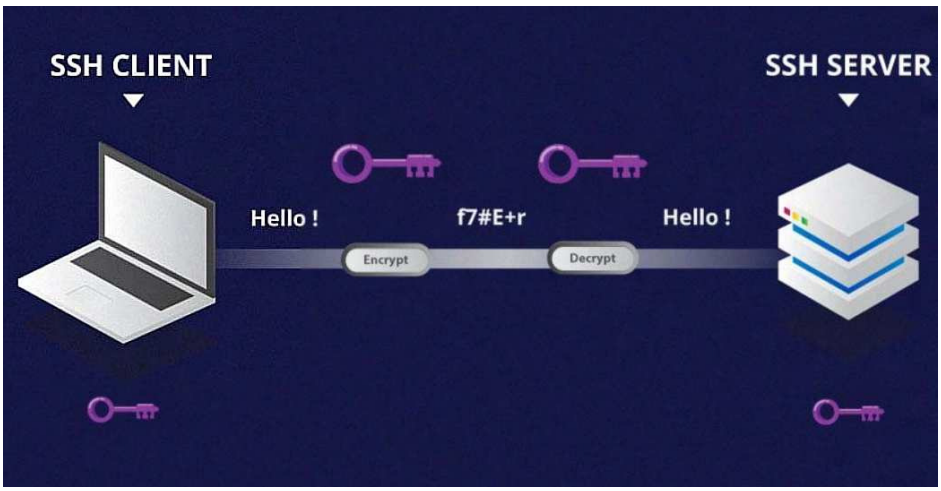
A primeira versão do protocolo, chamada *Secure Shell Protocol Version 1 (SSHv1)*, apresentava limitações importantes, principalmente relacionadas à segurança e à falta de modularidade. De acordo com Smaldone (2004), "a versão inicial do SSH possuía problemas estruturais e algoritmos que rapidamente se tornaram insuficientes" [Smaldone 2004, p. 2]. Para resolver essas falhas, foi desenvolvida a segunda versão, o *SSHv2*, que trouxe melhorias significativas e se tornou a versão padrão utilizada atualmente. Essa evolução é destacada pelo RFC 4251 ao afirmar que a nova versão fornece mecanismos mais sólidos e atualizados de proteção, incluindo sigilo perfeito para frente, autenticação reforçada e melhor negociação de algoritmos [Ylonen 2006, p. 1].

O SSH é composto por três camadas principais. A primeira é o *Transport Layer Protocol (SSH-TRANS)*, responsável por criar o túnel seguro e criptografado. A segunda é o *User Authentication Protocol (SSH-USERAUTH)*, encarregado de verificar a identidade do usuário. A terceira é o *Connection Protocol (SSH-CONNECT)*, que organiza múltiplas sessões dentro do mesmo canal. Essa divisão modular é fundamental para oferecer flexibilidade e

segurança [Ylonen 2006, p. 3].

A Figura 2.3 ilustra o funcionamento básico de uma conexão SSH. O cliente (*SSH Client*) é quem inicia o processo, enviando uma requisição ao servidor (*SSH Server*) para estabelecer o túnel seguro. Antes que qualquer mensagem útil seja trocada, ambos realizam uma negociação de algoritmos e criam chaves que serão utilizadas para criptografar e descriptografar a comunicação.

Figura 2.3: *Comunicação entre cliente e servidor SSH.*



Fonte: Hostinger, 2025.

A imagem mostra a mensagem "Hello!" enviada pelo cliente, que é convertida em um texto ilegível (como "f7#E+r") antes de trafegar pela rede. Esse processo de criptografia impede que invasores possam interpretar o conteúdo. No servidor, a mensagem é descriptografada, retornando novamente ao formato original. Tanto o cliente quanto o servidor utilizam chaves próprias para proteger o conteúdo, reforçando a confidencialidade. Smaldone (2004) afirma que "todo o tráfego dentro de um túnel SSH é automaticamente cifrado e protegido contra inspeção e alteração" [Smaldone 2004, p. 5].

Esse funcionamento é possível graças à troca de chaves que ocorre no início da sessão. Em vez de enviar a chave secreta diretamente, cliente e servidor geram valores que permitem a construção de um mesmo segredo, sem que ele seja transmitido pela rede. Ylonen e Lonvick (2006) explica que esse método evita ataques de interceptação e garante autenticidade do servidor antes de qualquer dado sensível ser enviado [Ylonen e Lonvick 2006, p. 21]. A partir desse segredo comum, são criadas as chaves simétricas utilizadas ao longo da comunicação.

O SSH utiliza algoritmos modernos de criptografia, como AES, e mecanismos de verificação de integridade conhecidos como HMAC, garantindo que nenhuma mensagem seja modificada durante o trajeto. A evolução desses mecanismos também pode ser observada com a adoção de algoritmos mais seguros, como SHA-256, substituindo versões antigas e vulneráveis. Bider destaca que "o uso de HMACs baseados em SHA-2 fortalece a proteção dos dados e reduz riscos de colisão" [Bider e Baushke 2012, p. 2].

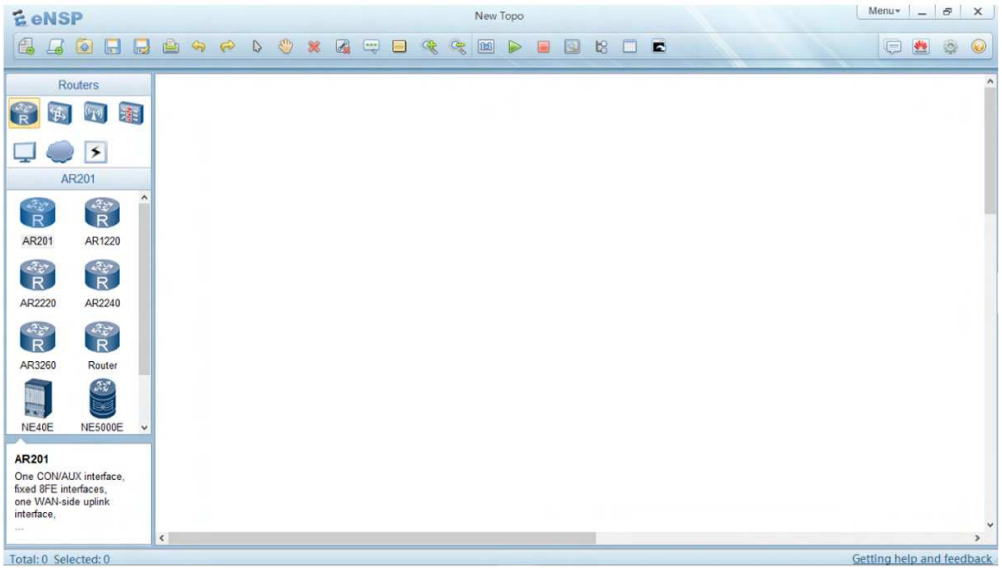
A autenticação no SSH pode ocorrer por senha ou por chaves públicas, sendo este último o método mais seguro. Na autenticação por chave pública, o usuário possui uma chave privada, enquanto o servidor armazena a chave pública correspondente. Somente quem possui a chave privada pode autenticar-se com sucesso. Smaldone (2004) aponta que esse método "evita o envio de senhas pela rede e reduz consideravelmente a possibilidade de ataques" [Smaldone 2004, p. 7].

2.5 Simulação de Redes com o eNSP

O *eNSP*, desenvolvido pela Huawei, é um simulador de redes que possibilita a criação de ambientes virtuais capazes de reproduzir o comportamento de dispositivos reais, como *roteadores*, *switches*, *firewalls* e terminais. Essa ferramenta permite o desenvolvimento de habilidades em gerenciamento de redes sem a necessidade de *hardware* físico, o que a torna especialmente útil em contextos educacionais e de treinamento profissional [Álvarez 2021].

A Figura 2.4 apresenta a interface gráfica do simulador *eNSP*, que se destaca por sua organização intuitiva e facilidade de uso. Por meio dela, é possível criar e manipular topologias de rede completas, interligando dispositivos virtuais e configurando conexões de maneira prática. O *eNSP* virtualiza equipamentos Huawei, fornecendo suporte aos mesmos comandos disponíveis em dispositivos reais por meio da *Command Line Interface* (*CLI*) [CINGOLANI 2019, p. 14]. Essa característica permite que o ambiente simulado se aproxime bastante do comportamento de redes reais, contribuindo para o aprendizado e a prática de configurações complexas.

Figura 2.4: *Interface gráfica do simulador eNSP.*

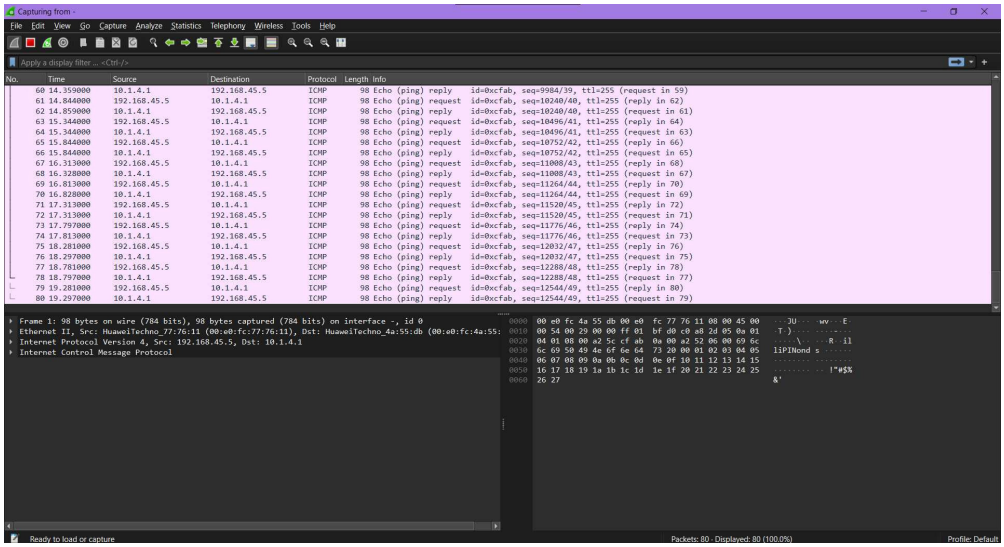


Fonte: elaborado pelo autor, 2025.

Além disso, o simulador possui integração nativa com o *Wireshark*, uma ferramenta am-

plamente utilizada para a captura e análise de pacotes de rede. Essa integração possibilita a observação em tempo real do tráfego entre os dispositivos simulados, permitindo compreender o funcionamento interno dos protocolos de comunicação [Álvarez 2021, p. 18–22]. A Figura 2.5 demonstra a análise de pacotes realizada dentro do ambiente do *eNSP*.

Figura 2.5: *Análise de pacotes no Wireshark integrado ao eNSP.*



Fonte: elaborado pelo autor, 2025.

Essa funcionalidade é essencial para a depuração e validação de protocolos, uma vez que permite acompanhar a troca de mensagens entre dispositivos, identificar falhas de comunicação e analisar cabeçalhos de pacotes em diferentes camadas do modelo *Open Systems Interconnection (OSI)* [CINGOLANI 2019, p. 40]. O recurso de captura integrada também facilita o ensino prático, pois torna visível o fluxo real de dados e a interação entre protocolos.

O *eNSP* é amplamente adotado em treinamentos técnicos e acadêmicos devido à sua capacidade de simular cenários corporativos com fidelidade e baixo custo. Entre suas principais vantagens, destacam-se:

- **Custo reduzido**: elimina a necessidade de equipamentos físicos, tornando-se acessível a instituições e profissionais em formação;
- **Fidelidade**: simula com precisão dispositivos Huawei, proporcionando experiências alinhadas às demandas do mercado de trabalho;
- **Flexibilidade**: suporta topologias que variam de redes simples a ambientes corporativos de grande escala;
- **Acessibilidade**: por ser gratuito, é amplamente utilizado em universidades, cursos técnicos e por profissionais autônomos.

Apesar de suas vantagens, o *eNSP* apresenta algumas limitações técnicas decorrentes de sua dependência de virtualização, o que pode afetar o desempenho em simulações de maior

complexidade [Maldonado 2022, p. 6]. A arquitetura baseada em *Oracle VM VirtualBox (VirtualBox)* introduz um *overhead* computacional, uma vez que a emulação de múltiplos dispositivos consome intensivamente recursos de *Central Processing Unit (CPU)* e memória *Random Access Memory (RAM)*. Em sistemas operacionais mais recentes, como o *Windows 11*, essa dependência pode gerar instabilidades e reduzir a escalabilidade das topologias criadas. As principais limitações identificadas são:

- **Perda de arquivos**: arquivos de topologia (`.topo`) podem ser corrompidos em determinadas versões do simulador;
- **Recursos computacionais**: simulações complexas requerem *hardware* robusto para manter a estabilidade e o desempenho;
- **Compatibilidade**: o *eNSP* é mais estável em sistemas *Windows 10*, apresentando falhas ocasionais no *Windows 11* devido à integração com o *VirtualBox*;
- **Curva de aprendizado**: o uso intensivo da *CLI* pode representar um desafio inicial para usuários iniciantes.

Dessa forma, o *eNSP* consolida-se como uma ferramenta de grande relevância no ensino e na prática de redes de computadores, permitindo que estudantes e profissionais desenvolvam habilidades de configuração e automação em um ambiente seguro, acessível e tecnicamente fiel à realidade.

2.6 Git Bash: Conceitos Fundamentais e Uso Prático

O controle de versão é um elemento essencial no desenvolvimento de *software* e na gestão de projetos, pois possibilita o rastreamento e a organização de alterações realizadas em arquivos ao longo do tempo. Essa prática permite manter um histórico detalhado de modificações, autores e datas, promovendo maior controle e segurança no desenvolvimento colaborativo [Chacon e Straub 2014, p. 1]. Nesse contexto, o *Global Information Tracker (Git)*, sistema de controle de versão distribuído, consolidou-se como uma das ferramentas mais populares e eficientes do mercado.

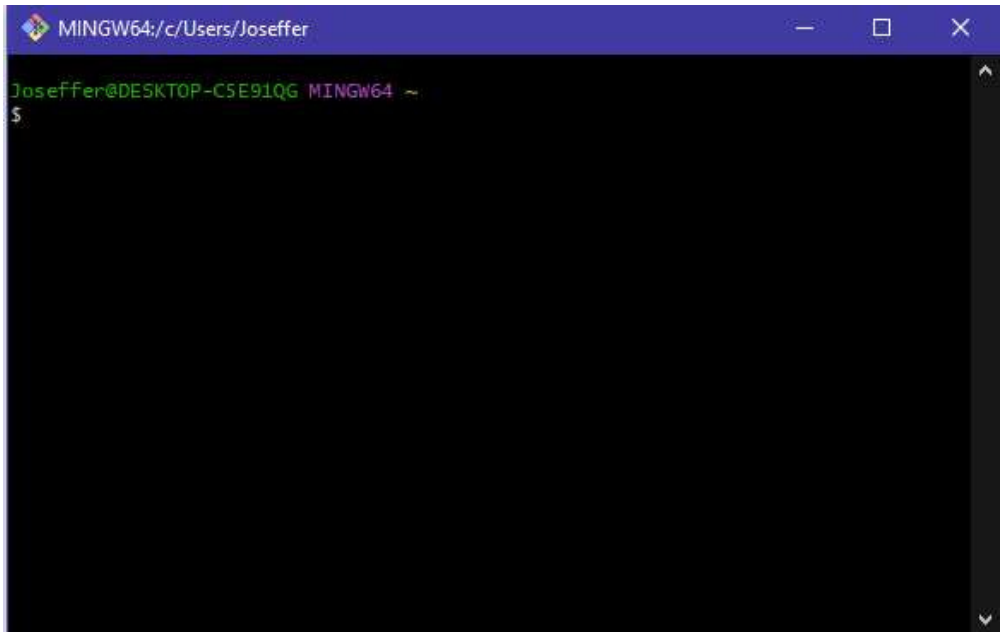
O *Git* foi criado por Linus Torvalds em 2005, com o objetivo de gerenciar o desenvolvimento do núcleo do sistema operacional *Linux*, priorizando velocidade, integridade de dados e suporte a fluxos de trabalho distribuídos e não lineares [Schmitzhaus 2024, p. 15]. Diferentemente de sistemas centralizados ou baseados em diferenças entre arquivos, o *Git* adota um modelo de armazenamento baseado em *snapshots*, registrando o estado completo do projeto a cada *commit*. Conforme descrito na documentação oficial, "o *Git* não armazena dados como uma série de alterações, mas como uma série de instantâneos" [Chacon e Straub 2010, p. 43]. Esse modelo garante maior integridade e facilidade de restauração de versões anteriores, mesmo em ambientes distribuídos.

No funcionamento interno do *Git*, os arquivos passam por três estados principais: *modified*, *staged* e *committed*. Um arquivo *modified* foi alterado, mas ainda não está preparado para o próximo registro. Ao ser adicionado à área de *staging* (*index*), ele passa a compor o próximo *commit*, que, por sua vez, grava as alterações de forma permanente no repositório local. Essa estrutura favorece o controle refinado sobre cada etapa do processo de versionamento e permite que o usuário defina com precisão quais mudanças serão registradas. Os comandos do *Git* dividem-se entre operações de baixo nível, conhecidas como *plumbing*, voltadas a tarefas internas e automação, e operações de alto nível, denominadas *porcelain*, utilizadas nas rotinas diárias, como `git add`, `git commit`, `git push` e `git pull` [Chacon e Straub 2010, cap. 9].

No ambiente *Windows*, o *Global Information Tracker Bash (Git Bash)* constitui uma ferramenta amplamente utilizada para a interação com o *Git*. Trata-se de uma *CLI* que emula um terminal *Bourne Again Shell (Bash)*, característico dos sistemas baseados em *Uniplexed Information and Computing System (Unix)*. Essa emulação permite executar comandos nativos do *Git* e integrar outras ferramentas complementares, como o *SSH*, essencial para conexões seguras entre repositórios remotos [Schmitzhaus 2024, p. 15].

A Figura 2.6 apresenta a interface do *Git Bash* em execução no sistema *Windows*, destacando sua semelhança visual e funcional com os terminais dos sistemas *Unix*-like.

Figura 2.6: *Interface do Git Bash no ambiente Windows.*



Fonte: elaborado pelo autor, 2025.

A Figura 2.6 evidencia como o *Git Bash* fornece um ambiente unificado para a execução de comandos de controle de versão e automação de tarefas. A ferramenta segue princípios de design aplicados a interfaces de linha de comando, como consistência, simplicidade e previsibilidade. Esses princípios foram analisados em estudos de Interação Humano-Computador

(*IHC*), que avaliaram a experiência de uso de ferramentas de terminal, como *Git Bash*, *Docker Command Line Interface (Docker CLI)* e *Amazon Web Services Command Line Interface (AWS CLI)*, especialmente entre usuários técnicos e desenvolvedores experientes [Schmitzhaus 2024, p. 15–25].

Dessa forma, o *Git Bash* não apenas viabiliza o uso do *Git* em sistemas *Windows*, mas também proporciona uma interface familiar para quem já atua em ambientes *Unix*-like, reforçando sua importância como ferramenta de integração entre plataformas e automação de processos de desenvolvimento.

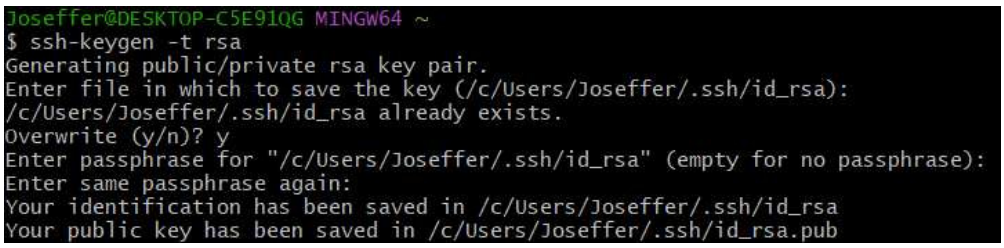
2.6.1 Geração de Chaves Públicas e Privadas SSHv2

A autenticação segura em conexões com repositórios remotos, como os hospedados em plataformas *GitHub* ou *GitLab*, é comumente realizada por meio do protocolo *SSHv2*. Nesse processo, o usuário utiliza um par de chaves criptográficas: uma pública e outra privada que, em conjunto, garantem a confidencialidade e a integridade da comunicação entre a máquina local e o servidor remoto [Milanesio 2013, p. 51–54].

A geração desse par de chaves *SSHv2* é uma prática amplamente adotada para estabelecer conexões seguras com repositórios remotos, eliminando a necessidade de inserir credenciais de autenticação manualmente em cada operação, como no comando *git push*. Em sistemas *Uniplexed Information and Computing System-like (Unix-like)*, incluindo o terminal *Bash* disponível no *Git Bash* para o sistema *Windows*, o comando utilizado para criar essas chaves é o `ssh-keygen` [Baarsen 2014, p. 22–28].

A Figura 2.7 apresenta o processo de criação de chaves criptográficas no ambiente *Git Bash*. Para gerar um par de chaves utilizando o algoritmo *Rivest–Shamir–Adleman (RSA)*, aplica-se o comando `ssh-keygen -t rsa`, que inicia a geração das chaves com tamanho padrão de 2048 bits [Baarsen 2014, p. 22]. Caso se deseje uma criptografia mais robusta, é possível especificar um tamanho superior, como 4096 bits, utilizando a opção `-b 4096`.

Figura 2.7: *Criação de chaves pública e privada no Git Bash.*



Fonte: elaborado pelo autor, 2025.

Durante o processo, o terminal solicita o diretório de armazenamento das chaves, sendo os caminhos padrão:

- `.ssh/id_rsa` — chave privada;

- `.ssh/id_rsa.pub` — chave pública.

A *passphrase* é uma senha opcional que acrescenta uma camada adicional de segurança à chave privada, criptografando-a em disco. Sempre que a chave for utilizada, a *passphrase* será solicitada por exemplo, durante o envio de alterações a um repositório remoto por meio de *SSHv2*. Em contextos de automação, como na execução de *scripts* que realizam operações automáticas de *push*, é possível omitir a *passphrase* para evitar interrupções manuais. No entanto, essa escolha reduz o nível de segurança caso a chave privada seja comprometida [Chacon e Straub 2014, p. 85].

A chave privada (`id_rsa`) deve ser mantida em segurança e jamais compartilhada, permanecendo armazenada localmente. Já a chave pública (`id_rsa.pub`) pode ser distribuída sem riscos e deve ser registrada em plataformas remotas, como *GitHub* ou *GitLab*. Ao adicionar a chave pública ao perfil do usuário nessas plataformas, o sistema passa a reconhecer a chave privada correspondente, permitindo o estabelecimento de conexões seguras via *SSHv2* e possibilitando autenticação automática, sem a necessidade de inserir nome de usuário e senha [Milanesio 2013, p. 51–54].

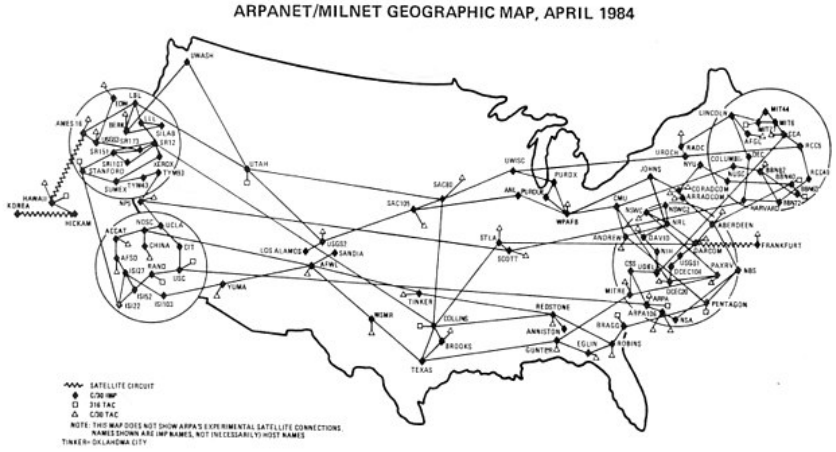
2.7 Origens e Evolução das Redes de Computadores

A evolução das redes de computadores está diretamente associada à necessidade humana de otimizar a comunicação e automatizar processos. Desde os primeiros avanços tecnológicos do século XX, buscou-se estabelecer formas mais rápidas, seguras e eficientes de troca de informações. Assim, o desenvolvimento das redes não representou apenas um marco técnico, mas também um reflexo da busca constante por eficiência, conectividade e redução do tempo de execução das atividades humanas.

Durante a Segunda Guerra Mundial, surgiram os primeiros sistemas computacionais com propósito estratégico, como o computador *Colossus*, desenvolvido para decifrar códigos militares. Essas inovações, associadas à crescente demanda por comunicação segura e confiável durante a Guerra Fria, impulsionaram o desenvolvimento de infraestruturas de rede de longa distância. Nesse contexto, o Departamento de Defesa dos Estados Unidos criou a *ARPANET*, considerada o embrião da atual *Internet* [Winston 2002, p. 170, 321–328]. O objetivo inicial era permitir a troca de informações entre centros de pesquisa e bases militares, garantindo comunicação contínua mesmo em cenários de falha de um dos nós da rede.

A Figura 2.8 apresenta o mapa da *ARPANET* em operação, evidenciando sua topologia e a distribuição dos nós conectados.

Figura 2.8: *Mapa da ARPANET em operação.*



Fonte: Wikimedia Commons, 1984.

A *ARPANET* foi uma das primeiras redes de comutação de pacotes do mundo e estabeleceu os fundamentos da comunicação distribuída. Embora tenha sido a mais conhecida, outras redes acadêmicas, comerciais e militares também estavam em desenvolvimento entre o final da década de 1950 e o início da de 1990 [Curran 2012, p. 36–37]. A introdução do *Network Control Protocol* (*NCP*) foi um marco importante, pois permitiu a comunicação entre computadores de diferentes instituições, estabelecendo as bases para os futuros protocolos de rede.

O termo *Internet* começou a ser utilizado na década de 1970, como abreviação de *Internetworking*, designando a interconexão de várias redes independentes [Cohen-Almagor 2013, p. 50]. Nesse período, Vint Cerf e Robert Kahn desenvolveram o *Transmission Control Protocol* (*TCP*) e o *IP*, formando a arquitetura *Transmission Control Protocol/Internet Protocol (TCP/IP)*, que se tornou o padrão para a comunicação global entre computadores [Cohen-Almagor 2013, p. 50]. Esses protocolos possibilitaram o envio confiável de dados em pacotes e a padronização das comunicações entre sistemas distintos.

Com o tempo, o conjunto *TCP/IP* superou outros protocolos concorrentes, consolidando-se como o modelo dominante nas redes de longa distância [Leiner *et al.* 1997, p. 106]. A desativação da *ARPANET* em 1990 marcou o início de uma nova era para a conectividade digital. No ano seguinte, em 1991, foi criada a *World Wide Web* (*WWW*) no *European Organization for Nuclear Research* (*CERN*), introduzindo uma interface gráfica que simplificava o acesso e a organização das informações distribuídas em rede [Curran 2012, p. 40–41]. Essa inovação expandiu o uso da *Internet* para além dos meios acadêmicos, alcançando empresas e usuários domésticos, e inaugurando o cenário digital contemporâneo [Platform 2020, p. 11–14].

A consolidação das redes de computadores, portanto, não se restringiu à interligação de máquinas, mas transformou-se em um processo que integra comunicação, automação e processamento de dados em escala global. As tecnologias atuais como a automação de redes, os protocolos de segurança e as linguagens de programação de alto nível são resultados

diretos dessa evolução. Cada avanço histórico contribuiu para o desenvolvimento de soluções que hoje permitem monitorar, configurar e gerenciar dispositivos de forma remota, como ocorre com o uso de ferramentas modernas baseadas em *Python* e *SSHv2*. Dessa forma, compreender a origem das redes é essencial para entender os fundamentos das tecnologias aplicadas à automação e ao gerenciamento inteligente de sistemas.

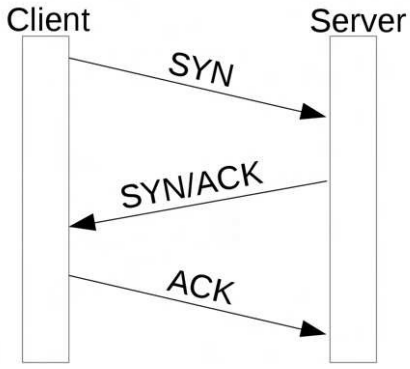
2.8 Protocolo TCP: Estrutura Operacional e Papel na Internet

O *TCP* constitui um dos principais componentes da suíte de protocolos *TCP/IP*, amplamente empregada em sistemas de comunicação digital. Sua função é garantir que os dados enviados entre dispositivos em uma rede sejam entregues de forma íntegra, ordenada e confiável. A maior parte das comunicações realizadas pela *Internet*, como o acesso a páginas, o envio de mensagens e a transferência de arquivos, ocorre segundo as regras estabelecidas pelos protocolos dessa suíte, sendo o *TCP* o responsável por assegurar a integridade e a confiabilidade das trocas de informações.

O *TCP* opera na camada de transporte do modelo de referência, oferecendo um serviço confiável e orientado à conexão. Essa característica permite a comunicação de ponta a ponta entre dispositivos pertencentes a redes heterogêneas, com diferentes tecnologias e arquiteturas [Tang, Liu e Zhu 2004, p. 1352]. Após o colapso da rede em 1986, provocado por um grave congestionamento de tráfego, o protocolo passou por melhorias significativas. Foram incorporados mecanismos de controle de congestionamento, como os introduzidos pelo *TCP Tahoe*, que implementou as estratégias de *slow start* e *congestion avoidance*, contribuindo para um funcionamento mais estável e adaptativo [Tang, Liu e Zhu 2004, p. 1352].

A Figura 2.9 ilustra o processo de estabelecimento de conexão do protocolo *TCP*, conhecido como *three-way handshake* ou aperto de mão em três vias, essencial para garantir uma comunicação confiável entre cliente e servidor.

Figura 2.9: *Representação do handshake de três vias do protocolo TCP.*



Fonte: Mungfali, 2025.

Esse procedimento inicia-se quando o cliente envia um número de sequência inicial, denominado *Initial Sequence Number client* (*ISNc*). O servidor, ao receber a solicitação, responde com um número de sequência próprio, o *Initial Sequence Number server* (*ISNs*), reconhecendo o valor recebido do cliente. Por fim, o cliente confirma o recebimento da resposta do servidor, completando o processo de sincronização. Após a troca dessas três mensagens, a conexão é estabelecida e a transmissão de dados pode ocorrer [Bellovin 1989, p. 32–34].

Os números *Initial Sequence Number (ISNc)* e *Initial Sequence Number (ISNs)* são utilizados nos campos *Sequence Number* (*Seq*) e *Acknowledgment Number* (*Ack*) presentes no cabeçalho *TCP*. O campo *Seq* indica o número de sequência do segmento enviado, enquanto o campo *Ack* confirma o recebimento do número de sequência anterior. O *handshake* tem, portanto, a função de sincronizar esses números, assegurando uma comunicação ordenada, confiável e com controle de fluxo adequado entre as partes envolvidas [Bellovin 1989, p. 32–34].

Apesar de sua relevância e robustez, o *TCP* apresenta vulnerabilidades inerentes ao seu modelo de funcionamento. Parte dessas fragilidades decorre da confiança excessiva dos *hosts* nos endereços *IP* de origem como forma de autenticação, o que pode ser explorado por agentes maliciosos para realizar ataques de falsificação de identidade ou interceptação de dados [Bellovin 1989, p. 1]. Ainda assim, o protocolo permanece fundamental para o funcionamento da *Internet*, sendo continuamente aprimorado a fim de equilibrar desempenho, compatibilidade e segurança.

2.9 Protocolo IPv4: Princípios de Operação e Classificação de Endereços

O *Internet Protocol version 4* (*IPv4*) constitui o principal protocolo de endereçamento da camada de rede, sendo responsável por identificar logicamente as interfaces de dispositivos em uma infraestrutura de comunicação. Em outras palavras, o *IPv4* define o modo como os dispositivos localizam-se mutuamente dentro de uma rede, atribuindo endereços únicos que possibilitam a troca de informações de forma organizada e eficiente [Wegner e Rockell 2000, cap. 1, p. 2, 10].

Nas redes *Ethernet*, a comunicação entre os dispositivos ocorre por meio de endereços físicos únicos associados a cada placa de rede, conhecidos como endereços *Ethernet* ou *Media Access Control (MAC)*. Para que um dispositivo envie dados a outro, é necessário que conheça o endereço físico de destino. Nesse contexto, enquanto protocolos como o *Microsoft Network Basic Input/Output System (NetBIOS)* realizam essa comunicação de forma direta, o *IPv4* utiliza o *Address Resolution Protocol* (*ARP*) para traduzir o endereço lógico (*IP*) em endereço físico (*MAC*), garantindo assim a entrega correta dos pacotes [Wegner e Rockell 2000, cap. 1, p. 2].

A configuração de endereços *IP* pode variar conforme o sistema operacional ou o tipo

de dispositivo empregado. Por essa razão, recomenda-se sempre a consulta à documentação oficial do fabricante, a fim de assegurar que as instruções sejam aplicadas corretamente. Além disso, manter um controle sistemático da alocação de endereços é uma prática essencial para evitar conflitos dentro da rede e preservar sua estabilidade [Wegner e Rockell 2000, cap. 2, p. 56].

A Tabela 2.1 apresenta as classes de endereçamento *IPv4*, originalmente definidas para organizar de forma hierárquica o espaço de endereços disponíveis. Cada classe possui uma faixa específica de endereços, uma máscara padrão e uma quantidade máxima de *hosts* suportada [Class *et al.* 1996, p. 5–7]. Por exemplo, o endereço `192.168.1.10` pertence à Classe C, pois está situado entre `192.0.0.0` e `223.255.255.255`, com máscara padrão `255.255.255.0`. Isso significa que ele pode integrar uma rede com até 254 *hosts*, o que o torna adequado para pequenos escritórios ou redes domésticas. Essa divisão por classes também otimizava o roteamento, agrupando endereços de maneira padronizada e eficiente para a época [Singh 2015, p. 87–88].

Tabela 2.1: *Classes de endereçamento IPv4.*

Classe	**Faixa de Endereços**	**Máscara Padrão**	**Nº Máx. de *Hosts***
A	0.0.0.0 a 127.255.255.255	255.0.0.0 (/8)	16.777.214
B	128.0.0.0 a 191.255.255.255	255.255.0.0 (/16)	65.534
C	192.0.0.0 a 223.255.255.255	255.255.255.0 (/24)	254

Fonte: elaborado pelo autor, 2025.

As máscaras fixas associadas a cada classe /8 para a Classe A, /16 para a Classe B e /24 para a Classe C simplificavam o encaminhamento de pacotes, pois seguiam uma estrutura uniforme nos 32 bits do endereço *IPv4*. Essa padronização permitia distinguir facilmente a parte da rede e a parte do *host*, reduzindo a complexidade dos roteadores [Thomas 2000, p. 1–2].

A divisão em classes também buscava equilibrar a distribuição dos endereços disponíveis, de modo que cada organização recebesse um bloco proporcional ao seu tamanho e às suas necessidades. Redes de grande porte, como provedores ou corporações, eram alocadas na Classe A, que possuía 24 bits destinados à identificação de *hosts*, permitindo milhões de dispositivos [Kumar e Shinde 2016, p. 242]. Já a Classe C, com apenas 8 bits para *hosts*, era voltada a redes menores, nas quais poucos dispositivos precisavam ser endereçados. Embora o modelo por classes tenha sido posteriormente substituído pelo *Classless Inter-Domain Routing* (*CIDR*), ele representou um marco importante na organização e na escalabilidade inicial da *Internet*.

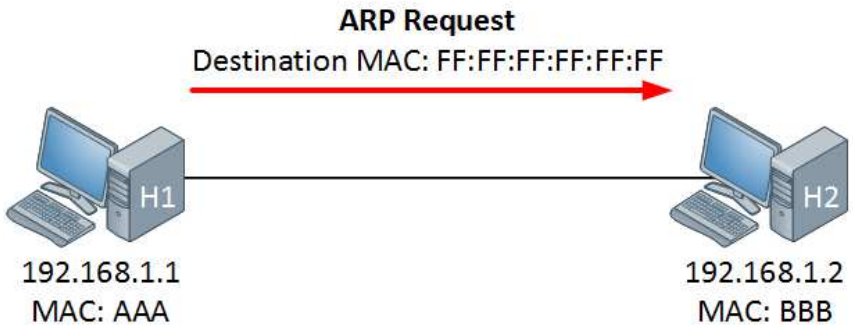
2.10 Funcionamento e Vulnerabilidades do Protocolo ARP

O *ARP* é um protocolo clássico, porém indispensável nas redes modernas, especialmente em *Local Area Networks* (*LANs*) baseadas em *Ethernet* [Parameswari e SURESH 2009, p. 26]. Sua principal função consiste em mapear endereços lógicos *IP* (camada de rede) para endereços físicos *MAC* (camada de enlace), possibilitando que os pacotes sejam corretamente entregues ao destino. Essa função atua como um elo entre a camada 3 e a camada 2 do modelo *OSI*, garantindo a comunicação eficiente entre dispositivos [Xi 2017, p. 2].

Em redes *Ethernet*, nas quais os endereços *MAC* possuem 48 *bits*, o mapeamento é essencial para o envio correto dos quadros. O funcionamento do *ARP* baseia-se em um mecanismo de "requisição e resposta" e no uso de uma tabela de correspondências denominada *cache ARP*. Quando um dispositivo precisa enviar um pacote para um endereço *IP* localizado na mesma rede, ele consulta essa tabela. Se encontrar a associação entre o endereço *IP* e o *MAC* correspondente, o pacote é encaminhado diretamente. Caso contrário, inicia-se o processo de descoberta [Parameswari e SURESH 2009, p. 26].

A Figura 2.10 apresenta o processo de requisição *ARP*. Quando a correspondência *IP*/*MAC* não é encontrada no *cache*, o dispositivo envia uma mensagem de *ARP Request* em modo *broadcast*, ou seja, destinada a todos os dispositivos da sub-rede. Essa mensagem contém o endereço *IP* do destino e solicita o endereço físico correspondente, em um formato semelhante à pergunta: "Quem possui o *IP* X.X.X.X? Informe o seu *MAC*." [Vidya, Gowri e Bhaskaran, p. 2–3].

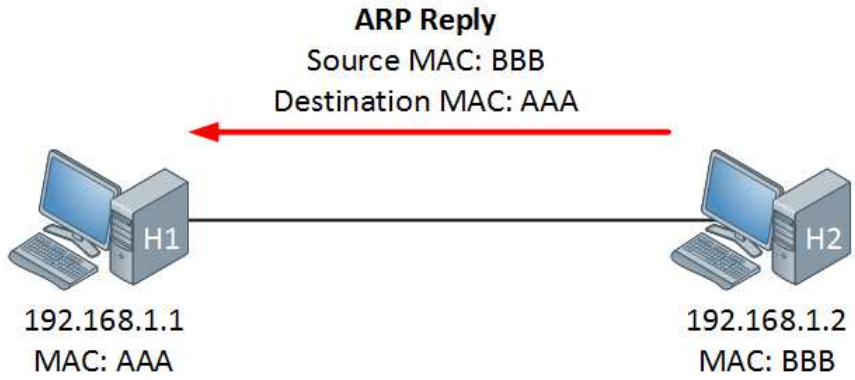
Figura 2.10: *Requisição ARP (ARP Request).*



Fonte: NetworkLessons, s.d.

A Figura 2.11 ilustra a resposta ao processo anterior. O dispositivo que possui o endereço *IP* solicitado envia uma mensagem de *ARP Reply* em modo *unicast*, direcionada exclusivamente ao remetente da requisição [Sukkar *et al.* 2016, p. 119]. Essa resposta contém o endereço físico *MAC*, que é então armazenado na tabela *cache ARP*. Esse registro permanece ativo por um período determinado, chamado *aging time*, e é reutilizado em comunicações futuras, reduzindo a necessidade de novas requisições.

Figura 2.11: *Resposta ARP (ARP Reply).*



Fonte: NetworkLessons, s.d.

Apesar de sua importância, o *ARP* apresenta vulnerabilidades significativas devido à sua natureza *stateless*, ou seja, à ausência de controle sobre as requisições e respostas anteriores [Alharbi *et al.* 2016, p. 523]. O protocolo não possui mecanismos nativos de autenticação, aceitando qualquer resposta recebida como válida. Essa fragilidade permite que agentes mal-intencionados introduzam mapeamentos falsos na tabela *cache ARP*, prática conhecida como *Address Resolution Protocol Spoofing (ARP Spoofing)* ou *Envenenamento de Cache ARP* [Trabelsi e Shuaib 2008, p. 81].

Durante esse tipo de ataque, o invasor envia respostas *ARP* fraudulentas, convencendo os dispositivos da rede a associar um endereço *IP* legítimo ao seu próprio endereço *MAC*. Esse comportamento possibilita ataques do tipo *Man-in-the-Middle* (*MITM*), nos quais o invasor intercepta, altera ou redireciona as comunicações entre os *hosts*. Em cenários mais críticos, pode até resultar em ataques de *Denial of Service* (*DoS*), comprometendo a disponibilidade e a estabilidade da rede [Yang 2014, p. 193].

2.11 Protocolo ICMP: Estrutura, Funcionamento e Implicações de Segurança

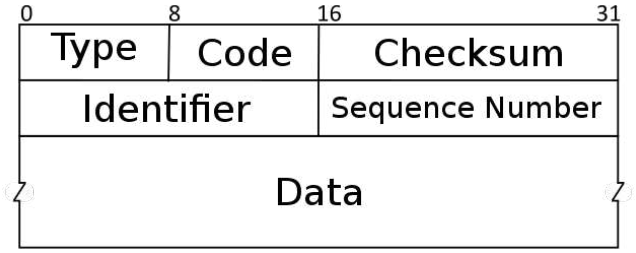
O *Internet Control Message Protocol* (*ICMP*) é um protocolo fundamental para o funcionamento da *Internet*, atuando de forma complementar na camada de rede e sendo encapsulado em datagramas *IP* [Saputro 2015, p. 1]. Integrante da suíte *TCP/IP*, o *ICMP* é utilizado principalmente para o envio de mensagens de erro e controle, bem como para o fornecimento de informações sobre o estado de conectividade entre dispositivos. Ele indica, por exemplo, a inacessibilidade de um destino ou a falha na entrega de pacotes devido a problemas de roteamento [Delaval, Martins e Melo 2024].

O protocolo não foi projetado para o uso direto por aplicações de usuário final, mas sim para atuar como suporte ao protocolo *IP*, que não dispõe de mecanismos nativos de gerenciamento e notificação de erros. Dessa forma, o *ICMP* estabelece um canal de comunicação

entre os dispositivos da rede, permitindo a troca de mensagens relacionadas ao controle e à entrega de pacotes [Saputro 2015]. Essa funcionalidade é essencial para o diagnóstico e a manutenção de redes, viabilizando a detecção de falhas e a análise do comportamento dos enlaces de comunicação entre *hosts*.

A Figura 2.12 apresenta a estrutura básica de um pacote *ICMP*, composta por campos que definem o tipo e o propósito da mensagem transmitida.

Figura 2.12: *Estrutura básica de um pacote ICMP.*



Fonte: Wikimedia Commons, 2020.

Cada mensagem *ICMP* inicia com o campo *Type*, de 1 *octeto* (8 bits), responsável por indicar a natureza da mensagem. Em seguida, o campo *Code*, também de 1 *octeto*, fornece informações adicionais que detalham o tipo de erro ou evento reportado. O campo *Checksum*, com 2 *octetos*, é utilizado para verificar a integridade dos dados, assegurando que o pacote não tenha sido corrompido durante o transporte [Kumar *et al.* 2010, p. 47], [Baltatu *et al.* 2000, p. 882].

Além desses campos, algumas mensagens *ICMP* incluem os parâmetros *Identifier*, *Sequence Number* e uma seção de *Data*, cujo tamanho varia conforme o tipo de mensagem. Em geral, a mensagem *ICMP* incorpora o cabeçalho do pacote *IP* original e os primeiros 8 *octetos* (64 bits) do datagrama que gerou o erro, permitindo que protocolos de camadas superiores, como o *TCP*, possam identificar e reagir adequadamente à falha detectada.

O *ICMP* é um protocolo sem conexão, ou seja, não requer o estabelecimento prévio de um canal de comunicação entre as partes envolvidas. Essa característica reduz a sobrecarga na rede, embora não garanta a entrega das mensagens [Kumar *et al.* 2010, p. 47]. Apesar de sua simplicidade e importância para o diagnóstico de redes, o *ICMP* apresenta vulnerabilidades inerentes. Por ser transportado dentro de datagramas *IP*, ele está sujeito a erros, interferências e falsificações [Chen 2001, p. 94].

Uma das fragilidades mais significativas está na ausência de mecanismos de autenticação, o que possibilita a falsificação de mensagens, prática conhecida como *spoofing*. Nesse tipo de ataque, um invasor envia pacotes *ICMP* forjados, simulando a origem de um dispositivo legítimo, com o intuito de enganar sistemas e provocar respostas incorretas ou até mesmo a indisponibilidade de serviços [Baltatu *et al.* 2000, p. 882–886]. Essa vulnerabilidade reforça a importância da implementação de políticas de segurança que limitem o uso irrestrito de mensagens *ICMP*, especialmente em redes corporativas, onde ataques dessa natureza podem

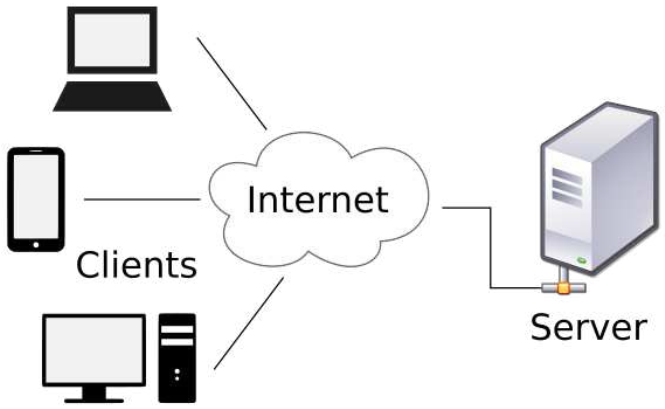
comprometer a disponibilidade e a confiabilidade das comunicações.

2.12 Modelo Cliente-Servidor: Estrutura e Operação

A arquitetura *cliente-servidor* representa um dos pilares fundamentais da computação distribuída moderna, sendo amplamente utilizada em sistemas de informação, aplicações corporativas e serviços de rede. Nessa estrutura, duas entidades autônomas: o *cliente* e o *servidor* interagem de forma coordenada para viabilizar o acesso remoto a dados e recursos computacionais [Kumar 2019, p. 33857]. O *cliente* é responsável por enviar requisições de serviço, enquanto o *servidor* processa essas solicitações e responde com as informações ou resultados esperados [Oluwatosin 2014, p. 67]. Essa relação permite uma comunicação eficiente entre processos distintos, distribuídos em diferentes dispositivos conectados à rede.

A Figura 2.13 ilustra o funcionamento básico dessa arquitetura, evidenciando a troca de informações entre os dois componentes.

Figura 2.13: *Exemplo de conexão cliente-servidor via Internet.*



Fonte: Wikimedia Commons, 2011.

O *servidor* é responsável por hospedar bancos de dados e executar *softwares* que processam as requisições enviadas pelos *clientes* [Nyabuto 2023, p. 1]. Essa comunicação ocorre mediante protocolos padronizados, como o *Hypertext Transfer Protocol* (*HTTP*), o *File Transfer Protocol* (*FTP*) e o *Simple Mail Transfer Protocol* (*SMTP*), que garantem a integridade e a interoperabilidade na troca de dados [Nyabuto 2023, p. 1]; [Oluwatosin 2014, p. 67]. A compreensão desse modelo é essencial para o estudo de redes e programação, pois constitui a base conceitual de diversos sistemas distribuídos contemporâneos.

A arquitetura *cliente-servidor* apresenta vantagens significativas. Entre as principais estão o acesso simultâneo de múltiplos usuários a um mesmo recurso e a facilidade de escalabilidade, permitindo que o sistema cresça conforme a demanda [Nyabuto 2023, p. 1]. Além disso, a separação entre os papéis de *cliente* e *servidor* possibilita maior organização e segurança

no processamento das informações. Historicamente, essa arquitetura desempenhou papel essencial na evolução da tecnologia da informação e continua sendo um modelo amplamente empregado nas infraestruturas atuais [Kumar 2019, p. 33857]. Outra característica vantajosa é sua relativa simplicidade de implementação, uma vez que pode ser construída utilizando tecnologias amplamente acessíveis e compatíveis com diferentes plataformas [Bouju *et al.* 1999, p. 2].

Entretanto, o modelo *cliente-servidor* também apresenta desafios. Um dos principais está relacionado ao desempenho da comunicação, que pode sofrer degradação em cenários de alta demanda ou conexões de rede instáveis [Bouju *et al.* 1999, p. 2–3]. Além disso, aspectos de segurança e confiabilidade são críticos, pois a integridade das informações e a disponibilidade dos serviços dependem diretamente da robustez e do gerenciamento do *servidor*. A análise e o aperfeiçoamento da confiabilidade nesse tipo de arquitetura constituem, inclusive, um campo de pesquisa ativo dentro das áreas de redes e sistemas distribuídos.

A presença da arquitetura *cliente-servidor* é amplamente perceptível no cotidiano digital. Diversos serviços e aplicações, como *websites*, plataformas de *electronic commerce (e-commerce)*, sistemas bancários, serviços de *electronic mail (e-mail)*, aplicações móveis e até caixas eletrônicos (*ATMs*), utilizam esse modelo para assegurar o funcionamento contínuo e integrado de suas operações. Em áreas especializadas, como nos *Sistemas de Informação Geográfica* (*SIG*), o modelo permite o acesso remoto a grandes bancos de dados *multimídia* e geográficos, otimizando a manipulação e a análise de volumes expressivos de informações [DeWitt *et al.* 1994, p. 1, 6].

Dessa forma, o modelo *cliente-servidor* consolida-se como um paradigma essencial da computação distribuída, unindo simplicidade estrutural, eficiência operacional e adaptabilidade. Sua capacidade de escalabilidade, integração e suporte a múltiplos usuários o mantém como uma das arquiteturas mais relevantes para o desenvolvimento de aplicações modernas e infraestruturas de rede.

2.13 Protocolo NETCONF: Estrutura, Funcionamento e Aplicações em Automação de Redes

O *NETCONF* é um protocolo de gerenciamento de redes padronizado pela *Internet Engineering Task Force* (*IETF*) com o objetivo de simplificar, automatizar e padronizar o processo de configuração de dispositivos de rede [Hedstrom, Watwe e Sakthidharan 2011, p. 1]. Desenvolvido para superar as limitações do *Simple Network Management Protocol* (*SNMP*), o *NETCONF* fornece um meio estruturado e seguro de instalar, modificar e remover configurações em dispositivos heterogêneos de maneira automatizada [Chappell 2013, p. 5–6].

O *NETCONF* baseia-se em uma arquitetura do tipo *cliente-servidor*, na qual o cliente geralmente uma aplicação de gerenciamento ou automação, envia comandos para um servidor embutido no dispositivo de rede [Hedstrom, Watwe e Sakthidharan 2011, p. 3]. A comuni-

cação ocorre por meio de *Remote Procedure Calls* (*RPCs*) em formato *Extensible Markup Language (XML)*, encapsulados dentro de sessões seguras, normalmente transportadas sobre o protocolo *SSH*. Essa abordagem garante autenticação, integridade e confidencialidade, características essenciais em ambientes de automação de rede. A estrutura em camadas do protocolo composta por transporte, mensagens, operações e conteúdo permite modularidade, interoperabilidade e fácil extensão [Tran, Tumar e Schönwälder 2009, p. 85].

O *NETCONF* introduz o conceito de *datastores*, que são repositórios lógicos de configuração utilizados para manter diferentes estados do dispositivo. Entre os principais estão o *running*, que representa a configuração ativa; o *startup*, carregado durante a inicialização; e o *candidate*, utilizado como um ambiente temporário para testar configurações antes de aplicá-las definitivamente [Schönwälder, Björklund e Shafer 2010, p. 168]. Essa estrutura assegura que as modificações possam ser validadas, revertidas ou confirmadas de forma controlada, reduzindo o risco de falhas durante operações críticas na rede.

As principais operações do *NETCONF* incluem: `<get-config>`, utilizada para leitura da configuração; `<edit-config>`, responsável por criar, substituir ou remover parâmetros; `<copy-config>` e `<delete-config>`, que permitem copiar ou excluir repositórios inteiros; e `<lock>` e `<unlock>`, que implementam mecanismos de bloqueio durante alterações concorrentes [**?**, p. 86]. Além disso, o protocolo oferece suporte a notificações e transações atômicas, garantindo que todas as mudanças sejam aplicadas integralmente ou revertidas em caso de erro [**?**, p. 169].

Uma das principais inovações do *NETCONF* é o uso da linguagem de modelagem *Yet Another Next Generation* (*YANG*), também padronizada pela IETF, para estruturar e descrever os dados de configuração [Chappell 2013, p. 9]. O *YANG* define a hierarquia dos elementos de configuração em formato de árvore e permite a criação de modelos compatíveis entre dispositivos de diferentes fabricantes, promovendo a interoperabilidade e a automação. Essa integração entre *NETCONF* e *YANG* constitui a base dos sistemas de gerenciamento orientados a modelos (*model-driven management*), fundamentais para a virtualização e a automação em redes definidas por software (*SDN*) [Wallin e Wikström 2011, p. 2].

Em comparação ao *SNMP*, o *NETCONF* apresenta ganhos expressivos em eficiência e escalabilidade. Em um estudo comparativo conduzido por Hedstrom, Watwe e Sakthidharan (2011), verificou-se que o *NETCONF* é capaz de configurar até 100.000 objetos gerenciados em uma única transação, enquanto o *SNMP* exige cerca de 2.779 transações para o mesmo volume [Hedstrom, Watwe e Sakthidharan 2011, p. 11–12]. Embora o *NETCONF* utilize maior largura de banda devido às suas funções de segurança e troca de capacidades, essas características são consideradas essenciais para ambientes complexos, garantindo confiabilidade e controle transacional.

Outro diferencial está na compatibilidade com ambientes *multi-vendor*. O protocolo foi projetado para ser independente de fabricante, permitindo que dispositivos de diferentes origens sejam configurados de maneira uniforme, eliminando a necessidade de scripts específicos para cada interface proprietária [Chappell 2013, p. 8]. Essa padronização favorece a

redução de custos operacionais e acelera a implantação de serviços automatizados.

2.13.1 Automação NETCONF com a Biblioteca *ncclient*

A biblioteca *ncclient*, desenvolvida em linguagem *Python*, é uma implementação cliente do protocolo *NETCONF*, projetada para simplificar a comunicação entre aplicações de automação e dispositivos de rede compatíveis com esse padrão. Ela fornece uma interface de alto nível que abstrai a complexidade das operações *Extensible Markup Language – Remote Procedure Call (XML-RPC)* e das mensagens *NETCONF*, permitindo que o desenvolvedor concentre-se nas lógicas de automação e gerenciamento [Bhushan, Tran e Schönwälder 2009, p. 2].

O *ncclient* opera como um intermediário entre o sistema de automação e o servidor *NETCONF* incorporado nos equipamentos de rede. Por meio dele, é possível estabelecer sessões seguras sobre *SSH*, enviar e receber configurações no formato *XML*, e manipular os diferentes *datastores* definidos pelo protocolo, como o *running*, o *candidate* e o *startup*. Segundo Raza e Fei (2016), "o uso do *ncclient* permite o controle programático completo sobre o ciclo de configuração e validação do dispositivo, promovendo automação confiável e reprodutível" [Bhushan, Tran e Schönwälder 2009, p. 3].

A biblioteca segue uma estrutura baseada no modelo *cliente-servidor*, na qual o *ncclient* atua como cliente e envia chamadas remotas de procedimento (*RPCs*) encapsuladas em mensagens *XML*. O servidor, por sua vez, processa as solicitações e retorna respostas formatadas conforme as especificações do protocolo [Bhushan, Tran e Schönwälder 2009, p. 3–4]. Essa arquitetura modular proporciona interoperabilidade entre diferentes fabricantes e plataformas, garantindo que os processos de automação sejam executados de maneira uniforme e padronizada.

Entre as principais funcionalidades disponibilizadas pela biblioteca estão: o gerenciamento de sessões seguras, a execução de operações *RPC* como `get-config`, `edit-config`, `lock`, `commit` e `discard-changes`, além do suporte a capacidades específicas de cada fabricante. De acordo com Raza e Fei (2016), "essas operações são implementadas em métodos de fácil uso, que tornam o processo de automação acessível mesmo a profissionais sem conhecimento aprofundado de *XML* ou *RPC*" [Bhushan, Tran e Schönwälder 2009, p. 5].

A adoção do *ncclient* é particularmente vantajosa em ambientes que requerem automação de tarefas complexas, pois reduz o tempo de configuração, garante consistência entre execuções e diminui a probabilidade de falhas humanas. Em experimentos conduzidos por Shrestha *et al.* (2020), o uso combinado do *NETCONF* e do *ncclient* em dispositivos integrados a um servidor *Raspberry Pi* demonstrou aumento significativo na eficiência de transmissão de dados e na estabilidade das sessões de gerenciamento remoto [Đumić e Lubura 2021, p. 179].

Essa flexibilidade técnica contribui para que o *ncclient* seja amplamente adotado em projetos de automação e orquestração de redes, tanto no meio acadêmico quanto no setor corporativo. Sharma *et al.* (2018) destacam que o uso de bibliotecas baseadas em *Python*,

como o *ncclient*, "tem impulsionado o desenvolvimento de sistemas de gerenciamento mais dinâmicos, reconfiguráveis e resilientes em grandes infraestruturas de rede" [Stancu, Shevrikuko e Rueda 2019, p. 5].

Em síntese, a biblioteca *ncclient* consolida-se como uma solução essencial para o desenvolvimento de aplicações de automação em redes gerenciadas via *NETCONF*. Sua combinação de simplicidade, interoperabilidade e compatibilidade com diferentes plataformas faz dela uma ferramenta indispensável para projetos que visam aumentar a eficiência operacional e a padronização de processos em ambientes de rede modernos.

Capítulo 3

Resultados Obtidos

A crescente complexidade das infraestruturas de rede em ambientes corporativos, acadêmicos e de pesquisa tem exigido soluções cada vez mais eficazes para o gerenciamento e a configuração de dispositivos. Tradicionalmente, essas tarefas são executadas manualmente por administradores de rede, o que demanda tempo, atenção e conhecimento técnico especializado. Além disso, a repetição constante de comandos por meio de *CLI* aumenta a probabilidade de falhas humanas e compromete a padronização das configurações. Nesse contexto, o desenvolvimento de ferramentas que automatizem a comunicação e o controle de equipamentos de rede torna-se um recurso essencial para aprimorar a eficiência operacional e a segurança administrativa.

Com base nessa demanda, foi desenvolvida a aplicação **Huawei Network Automation Tool**, um sistema voltado para a automação e o gerenciamento remoto de dispositivos de rede, com foco em roteadores e *switches*. O software foi construído com o objetivo de centralizar, em uma única interface gráfica, operações que antes eram realizadas de forma manual e fragmentada, permitindo ao administrador executar tarefas como envio de configurações, execução de comandos remotos, extração de *backups* e testes de conectividade de forma prática e segura.

O desenvolvimento foi realizado em *Python*, integrando bibliotecas amplamente utilizadas na automação de redes, como a *ncclient*, voltada às operações com o protocolo *NETCONF*, e a *Paramiko*, responsável por conexões seguras via *SSHv2*. Também foi incorporado o uso de chaves criptográficas do tipo RSA, geradas no ambiente *Git Bash*, para autenticação sem o uso de senha. Além disso, todo o processo de testes, validações e execução modular dos componentes da ferramenta foi conduzido no ambiente **Jupyter Notebook**, que permitiu organizar, visualizar e depurar o código de forma estruturada e iterativa, contribuindo para maior clareza e confiabilidade no desenvolvimento.

A arquitetura da ferramenta foi organizada em módulos interdependentes, que cobrem as principais etapas de administração de um equipamento de rede. O módulo de Aplicar Configuração concentra as funções de edição e envio de blocos *XML*, permitindo que o usuário modifique parâmetros e aplique configurações de forma direta no dispositivo. Caso

o bloco *XML* seja muito extenso ou exija comandos específicos, o próprio código-fonte pode ser ajustado para adequar as instruções ao cenário desejado. Um diferencial importante é a possibilidade de modificar, durante a execução, as credenciais de acesso utilizadas pelos protocolos *NETCONF* e *SSH*, sem a necessidade de reiniciar o aplicativo ou alterar arquivos internos, conferindo maior flexibilidade ao processo de administração.

Além desse módulo, a aplicação conta com outros recursos complementares que ampliam sua funcionalidade: extração de configurações e geração de *backups* automáticos via *SSH*, *console* interativo para comandos diretos, painel de Testes de Conectividade (com suporte a *ping* e *traceroute*) e registro centralizado de *logs* de execução. Essa estrutura modular permite que o administrador gerencie todas as etapas de configuração, monitoramento e diagnóstico em um único ambiente, mantendo a rastreabilidade das ações e a consistência das informações.

Durante a fase de testes, realizada em ambiente de simulação (*eNSP*), a ferramenta apresentou desempenho estável e comportamento coerente com os objetivos propostos. Foi possível enviar blocos *XML* padronizados, extrair configurações completas, armazenar *backups* organizados em diretórios específicos e realizar verificações de conectividade de maneira integrada à interface. A comunicação simultânea entre os módulos de automação e *console* interativo demonstrou que o sistema é capaz de unir tarefas automatizadas e intervenções manuais num fluxo contínuo e eficiente.

Os resultados práticos alcançados até o momento confirmam a viabilidade da proposta e reforçam a relevância da automação como ferramenta de apoio à administração de redes. A aplicação apresenta potencial para futuras expansões, como integração com sistemas de versionamento e implementação de auditorias automatizadas. Tais avanços poderão ampliar ainda mais o alcance e a aplicabilidade do projeto em ambientes corporativos e institucionais.

Nas seções seguintes, são apresentadas as principais telas e funcionalidades da ferramenta desenvolvida, detalhando o funcionamento de cada módulo e os resultados observados durante sua utilização prática.

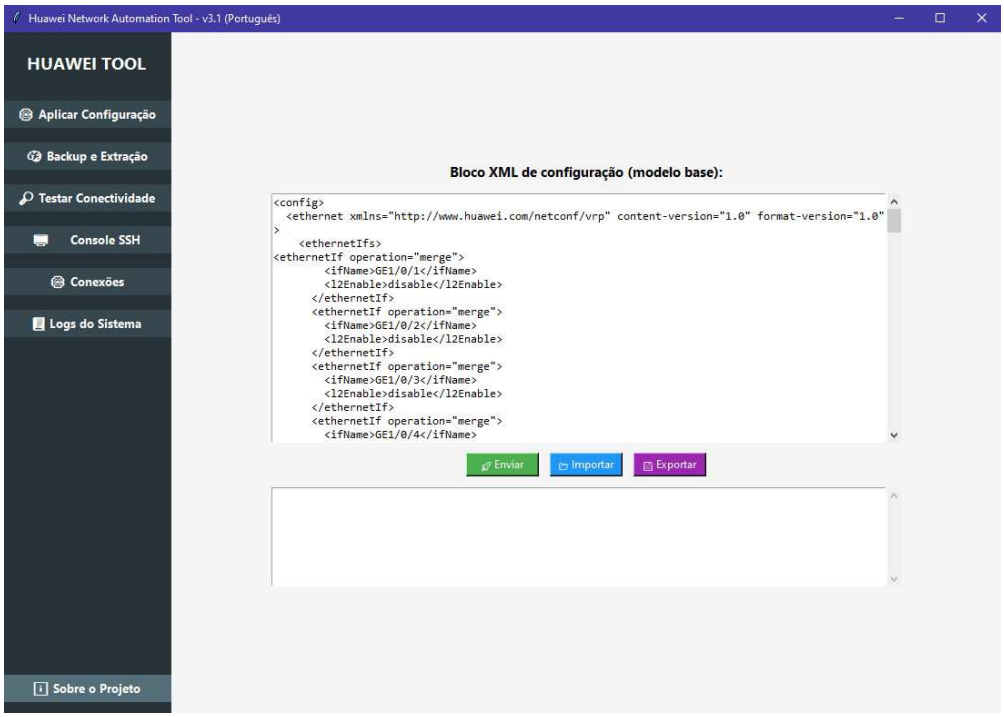
3.1 Módulo de Aplicar Configuração

Ao iniciar a execução da aplicação, o usuário é direcionado automaticamente para a tela inicial do sistema, correspondente ao módulo **Aplicar Configuração**. Essa interface faz parte do conjunto de funcionalidades da Huawei Network Automation Tool e tem como objetivo permitir a comunicação direta com o equipamento de rede e a execução de *scripts* de configuração de maneira automatizada. Nessa tela, o usuário tem acesso a um bloco *XML* base, que serve como modelo padrão para a criação e modificação das instruções que serão enviadas ao dispositivo.

A Figura 3.1 apresenta a interface inicial do módulo, na qual o usuário pode visualizar o bloco *XML* modelo e acessar as principais opções de operação. A interface foi desenvolvida

com o objetivo de manter clareza e praticidade, concentrando todas as ações principais em um único espaço de trabalho.

Figura 3.1: *Tela inicial do módulo Aplicar Configuração.*

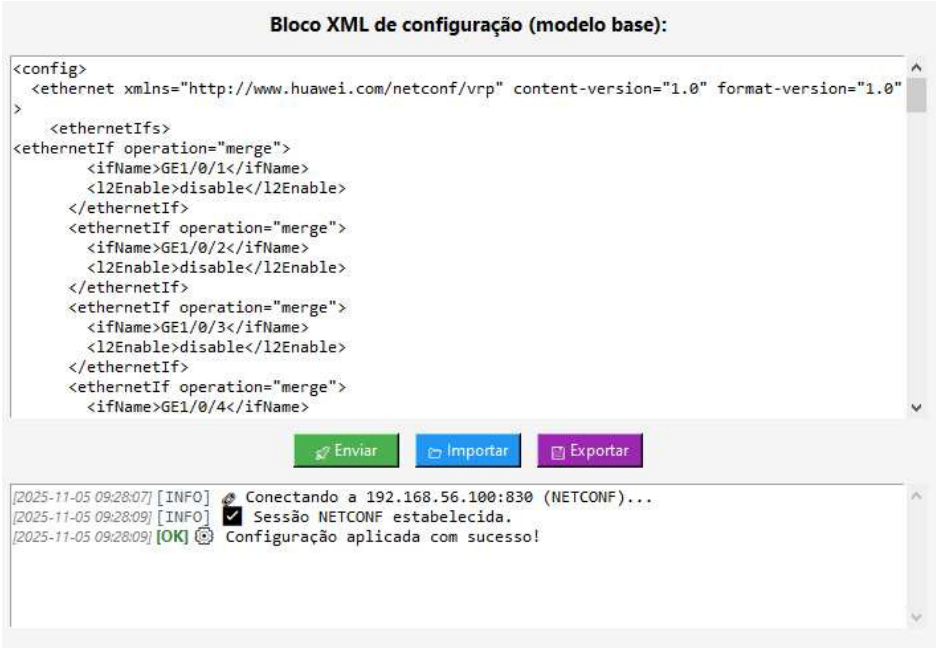


Fonte: elaborado pelo autor, 2025.

O editor *XML* integrado permite editar, importar ou exportar arquivos de configuração conforme a necessidade do administrador. A opção **Importar** possibilita carregar arquivos *XML* já existentes no sistema, enquanto a função **Exportar** permite salvar o conteúdo produzido ou ajustado pelo usuário, garantindo a reutilização e o versionamento das configurações. Por sua vez, o botão **Enviar** é responsável por iniciar o processo de envio do bloco *XML* ao equipamento, utilizando o protocolo *NETCONF* como canal de comunicação.

Durante o envio, o sistema estabelece uma conexão segura com o endereço configurado, exibindo em tempo real as mensagens de status na área de *log*. Caso o *host* esteja inacessível ou haja falhas de autenticação, são geradas mensagens de erro que informam a causa da interrupção. Por outro lado, quando o processo ocorre com sucesso, a ferramenta apresenta uma notificação positiva, confirmando o envio e a aplicação da configuração no dispositivo remoto, conforme ilustrado na Figura 3.2. Esse retorno imediato fornece ao usuário segurança e controle sobre as ações executadas, reduzindo o risco de inconsistências na rede.

Figura 3.2: *Envio bem-sucedido de configuração via protocolo NETCONF.*



Fonte: elaborado pelo autor, 2025.

Na execução prática demonstrada, o sistema foi configurado para se conectar ao endereço `192.168.56.100` na porta `830`, correspondente à comunicação *NETCONF* do equipamento Huawei simulado no ambiente *eNSP*. O teste apresentou um tempo médio inferior a dois segundos entre o início da conexão e a aplicação do *script*, evidenciando o desempenho e a eficiência da ferramenta. A mensagem "Sessão *NETCONF* estabelecida" seguida de "Configuração aplicada com sucesso" confirma o correto funcionamento do módulo e a estabilidade do processo de automação.

O bloco *XML* utilizado na simulação foi responsável por converter as interfaces Ethernet físicas em modo de camada 3, definir endereços *IP* específicos e inserir descrições automáticas em cada uma delas. O *script* foi aplicado nas nove interfaces disponíveis do equipamento, configurando-as de forma padronizada e simultânea, conforme exemplificado no ambiente *eNSP*:

```
1 interface GE1/0/2
2  undo portswitch
3  description Interface configurada via SCRIPT (GE1/0/2)
4  shutdown
5  ip address 192.168.2.1 255.255.255.0
```

Para a geração e envio desse bloco de configuração, a ferramenta utiliza uma função interna denominada `gerar_xml_modelo()`, que estrutura automaticamente os parâmetros do arquivo *XML* com base em um modelo pré-definido no código-fonte. Caso o administrador deseje aplicar configurações mais avançadas ou adicionar novos comandos não contemplados no modelo padrão, é possível editar essa função diretamente no código, ajustando o conte-

údo dinâmico que compõe o arquivo *XML*. Essa característica confere maior flexibilidade ao sistema, permitindo sua adaptação a diferentes cenários de rede e tipos de dispositivos.

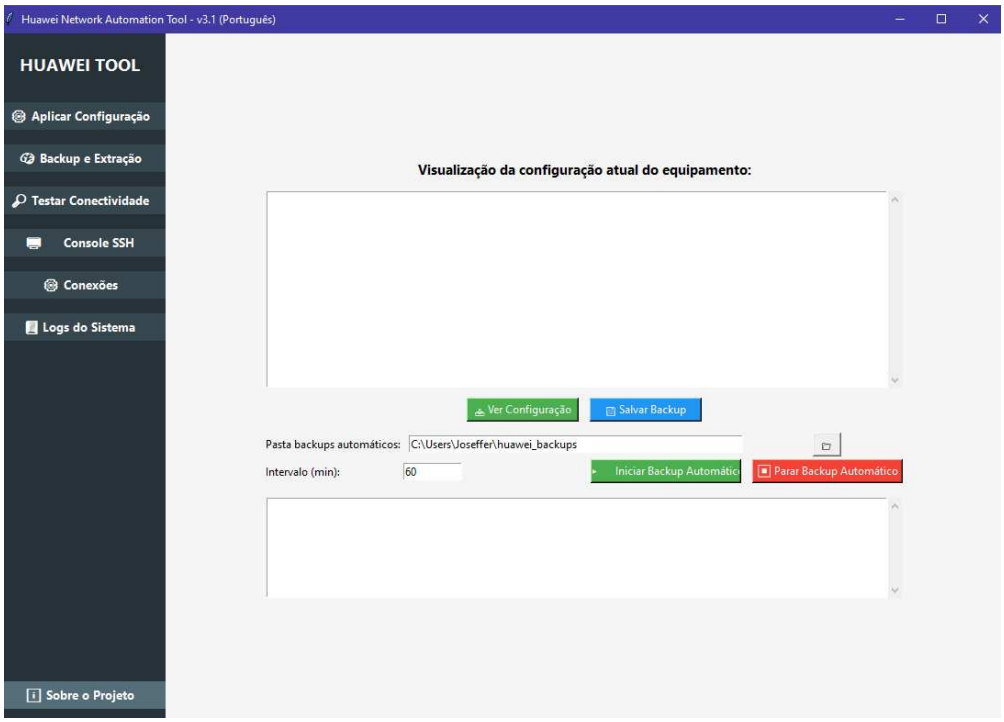
Essas configurações, embora simples, demonstram de forma clara a eficiência do processo automatizado. O envio foi concluído com sucesso em menos de dois segundos, mostrando a capacidade da ferramenta de aplicar *scripts* de configuração de forma rápida, segura e confiável. Essa performance é especialmente relevante em cenários que exigem repetição de comandos ou padronização de configurações em múltiplos dispositivos, onde a automação reduz significativamente o tempo de operação e a possibilidade de falhas humanas.

3.2 Módulo de Backup e Extração

O módulo **Backup e Extração** tem como objetivo permitir ao administrador de rede visualizar e armazenar a configuração atual do equipamento de forma rápida e segura. Essa funcionalidade é essencial em ambientes corporativos, acadêmicos e laboratoriais, nos quais a manutenção de cópias de segurança é um requisito fundamental para a continuidade dos serviços e a recuperação em situações de falha.

A Figura 3.3 apresenta a interface do módulo, que possibilita tanto a visualização direta da configuração quanto a execução de cópias de segurança automáticas e manuais. O design da tela foi desenvolvido de modo a concentrar as opções principais: **Ver Configuração**, **Salvar Backup** e o controle de **Backup Automático**, em um ambiente unificado e de fácil compreensão.

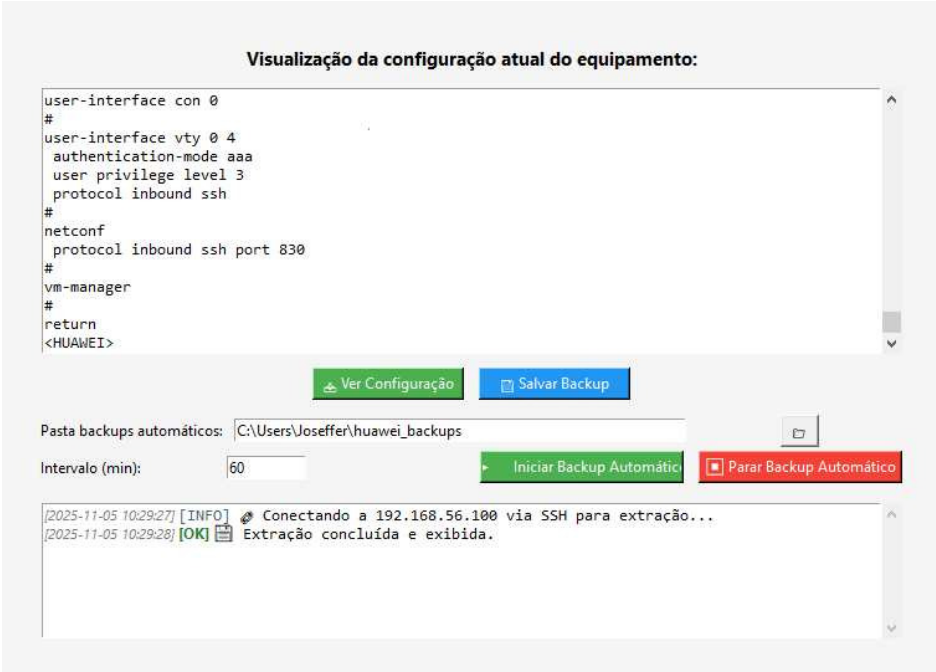
Figura 3.3: *Interface do módulo Backup e Extração.*



Fonte: elaborado pelo autor, 2025.

Ao selecionar a opção **Ver Configuração**, o sistema estabelece uma conexão segura com o equipamento por meio do protocolo *SSHv2*, utilizando a biblioteca *Paramiko* e o par de chaves *RSA* previamente gerado no ambiente *Git Bash*. Essa operação realiza a extração da configuração atual do dispositivo, exibindo-a diretamente na interface da aplicação. Em caso de sucesso, são apresentadas mensagens informativas que demonstram o tempo médio de resposta do processo, conforme ilustrado na Figura 3.4.

Figura 3.4: *Visualização da configuração e mensagens de log no módulo Backup e Extração.*



Fonte: elaborado pelo autor, 2025.

Como pode ser observado, a configuração extraída é exibida na área de visualização superior, apresentando informações referentes às interfaces, permissões de acesso e protocolos habilitados no equipamento. Logo abaixo, na área de mensagens, o sistema registra em tempo real as etapas do processo, incluindo o início da conexão e a conclusão bem-sucedida da extração. O exemplo apresentado demonstra que o procedimento levou aproximadamente um segundo para se conectar ao dispositivo e exibir a configuração, evidenciando a eficiência do módulo.

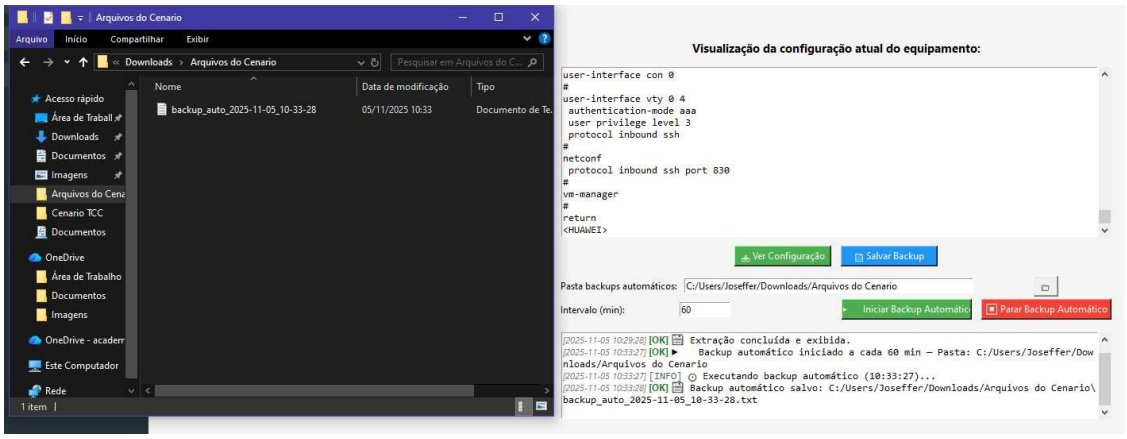
Caso o *host* esteja inacessível, o módulo apresenta mensagens de erro detalhadas, permitindo ao administrador identificar facilmente falhas de autenticação, interrupções de rede ou indisponibilidade do equipamento. Essa resposta imediata fornece ao usuário um controle preciso sobre o estado da operação e contribui para a confiabilidade do processo de extração e *backup*.

Além da visualização direta, o usuário pode salvar manualmente o conteúdo extraído por meio do botão **Salvar Backup**. Essa ação abre uma janela para seleção do diretório de destino, recomendando-se que o arquivo seja armazenado em um local seguro, uma vez que

contém a configuração integral do equipamento. Essa funcionalidade é particularmente útil em rotinas de manutenção, auditoria e restauração de sistemas.

Outra funcionalidade implementada é o **Backup Automático**, que possibilita a criação periódica de cópias de segurança de forma programada. O usuário pode definir um intervalo em minutos e selecionar a pasta de destino onde os arquivos serão salvos, conforme ilustrado na Figura 3.5. Após o acionamento do botão **Iniciar Backup Automático**, o sistema realiza o processo de extração e armazenamento no intervalo definido, repetindo a operação de modo contínuo até que o comando **Parar Backup Automático** seja acionado.

Figura 3.5: *Configuração do intervalo e destino do Backup Automático.*



Fonte: elaborado pelo autor, 2025.

O módulo foi projetado para operar com estabilidade e baixo consumo de recursos, tornando-se especialmente útil em ambientes corporativos e institucionais, onde a perda de configurações pode resultar em interrupções prolongadas e custos operacionais elevados. Com o recurso de backup automatizado, é possível minimizar riscos e assegurar a rápida recuperação de dispositivos em caso de falhas. Além disso, os arquivos de backup gerados podem ser reutilizados para restaurar a configuração original ou aplicá-la em outro equipamento, simplificando a manutenção e a padronização da infraestrutura de rede.

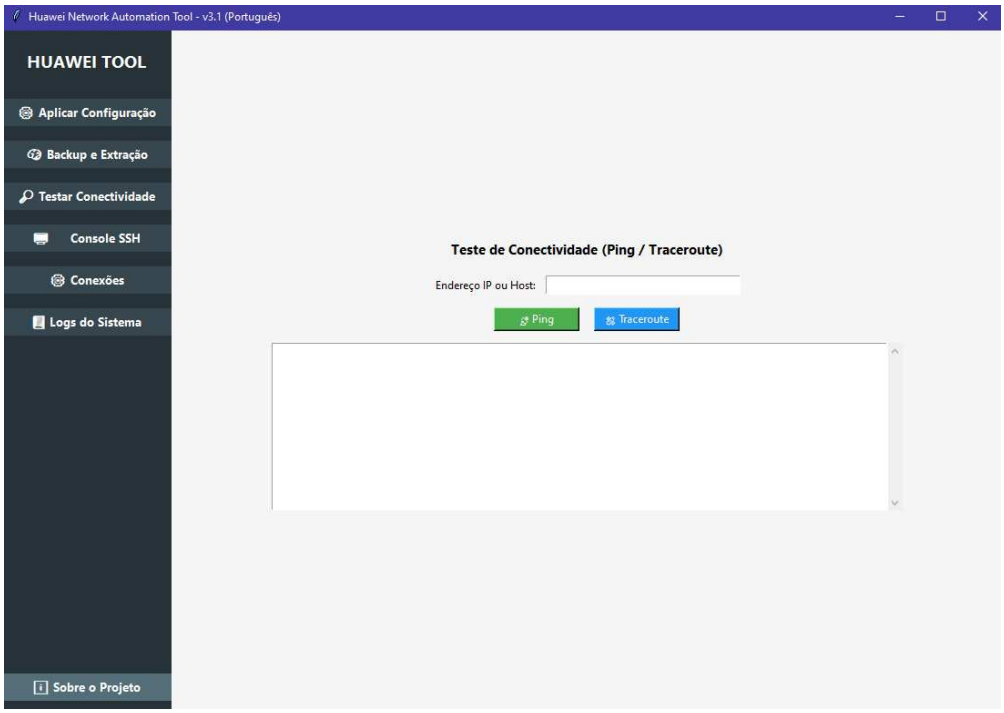
3.3 Módulo de Testar Conectividade

O módulo **Testar Conectividade** tem como finalidade auxiliar o administrador de rede na verificação do alcance e do desempenho da comunicação entre dispositivos, permitindo identificar eventuais falhas ou instabilidades em enlaces de rede. Essa funcionalidade incorpora dois dos principais utilitários utilizados em diagnósticos de rede: o *ping* e o *traceroute*, integrados de forma prática à interface da aplicação.

A Figura 3.6 apresenta a interface inicial do módulo, onde o usuário pode inserir o endereço *IP* ou o nome do *host* que deseja testar. Logo abaixo, encontram-se os botões correspondentes às duas operações disponíveis: **Ping** e **Traceroute** e uma área de saída que

exibe em tempo real os resultados obtidos durante o teste.

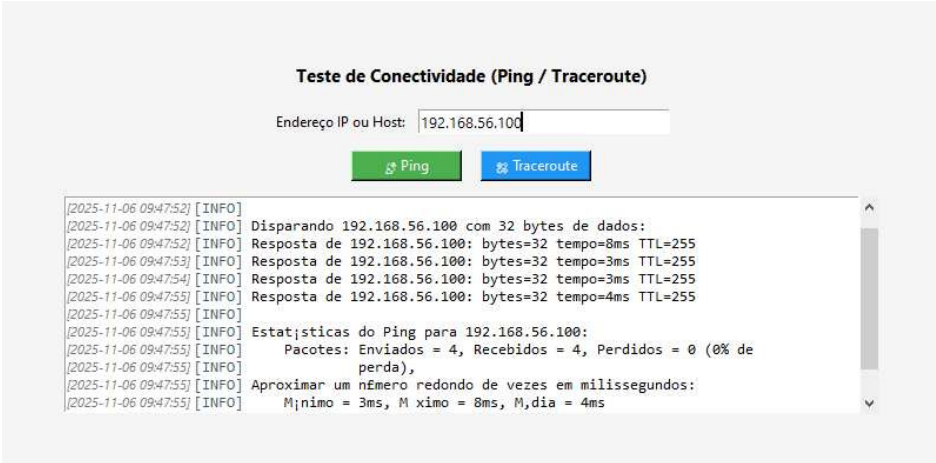
Figura 3.6: *Interface do módulo Testar Conectividade.*



Fonte: elaborado pelo autor, 2025.

Ao acionar o botão **Ping**, o sistema executa um teste de comunicação com o endereço informado, enviando pacotes de dados e registrando as respostas recebidas. Essa operação permite avaliar se o equipamento está alcançável e medir o tempo médio de resposta entre as requisições. A Figura 3.7 demonstra um teste bem-sucedido com o endereço `192.168.56.100`, em que todos os pacotes foram enviados e recebidos com tempo médio de resposta de 4,5 milissegundos.

Figura 3.7: *Execução de teste de ping no módulo Testar Conectividade.*

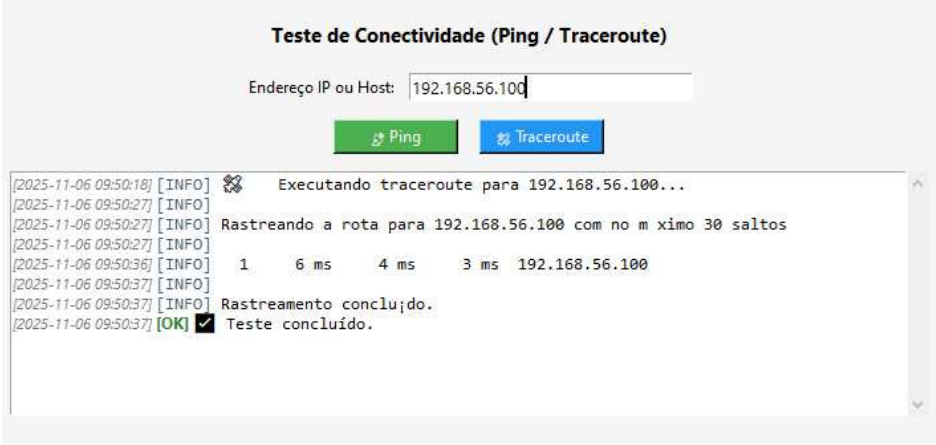


Fonte: elaborado pelo autor, 2025.

Durante a execução, a ferramenta apresenta os resultados de forma detalhada, exibindo mensagens informativas sobre cada pacote transmitido, o número de respostas recebidas e a estatística final de desempenho. O retorno em tempo real auxilia o administrador a detectar de imediato perdas de pacotes, atrasos ou falhas de conectividade, facilitando a identificação de problemas relacionados à latência ou à indisponibilidade de dispositivos.

O segundo recurso disponível é o comando **Traceroute**, utilizado para mapear a rota percorrida pelos pacotes até o destino especificado. Essa funcionalidade é especialmente útil para identificar gargalos de desempenho ou falhas em determinados pontos do caminho de rede. A Figura 3.8 apresenta o resultado de um rastreamento realizado para o mesmo endereço `192.168.56.100`, no qual o teste foi concluído com sucesso, mostrando que a rota é alcançável sem interrupções. No cenário de simulação utilizado, o rastreamento apresenta apenas um salto, pois há apenas um equipamento ativo na topologia, representando o destino final da comunicação.

Figura 3.8: *Execução de teste de traceroute no módulo Testar Conectividade.*



Fonte: elaborado pelo autor, 2025.

Durante a execução do *traceroute*, cada salto da rota é exibido na tela, apresentando o tempo de resposta de cada nó intermediário até o destino final. Ao término do processo, o sistema exibe uma mensagem indicando o sucesso do rastreamento.

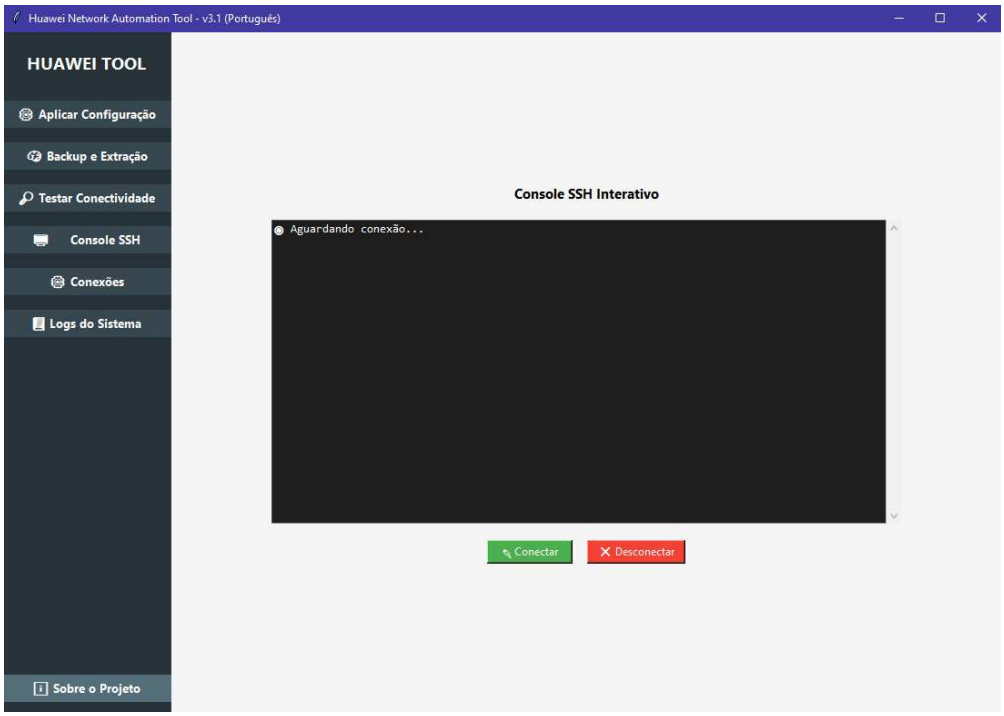
Ambos os testes demonstraram desempenho consistente, com tempo de resposta médio inferior a dez milissegundos e nenhuma perda de pacotes, evidenciando a estabilidade da comunicação entre o equipamento local e o dispositivo remoto. Essa capacidade de diagnóstico rápido e integrado reforça a importância do módulo para a verificação prévia da conectividade, uma etapa essencial antes da aplicação de *scripts* de configuração ou da execução de rotinas de *backup*. Garantir que o dispositivo esteja alcançável é um requisito indispensável para o funcionamento adequado das demais funções da aplicação.

3.4 Módulo de Console SSH Interativo

O módulo **Console SSH Interativo** representa uma das funcionalidades mais poderosas e versáteis da Huawei Network Automation Tool, permitindo o acesso direto ao terminal do equipamento de rede. A interface foi projetada para oferecer ao administrador uma experiência semelhante à de um terminal nativo do sistema operacional, possibilitando a execução manual de comandos e a observação imediata de suas respostas no dispositivo remoto.

A Figura 3.9 apresenta a tela inicial do módulo, na qual é exibida a área de terminal e os botões **Conectar** e **Desconectar**. Ao iniciar o módulo, o sistema aguarda a conexão com o equipamento, exibindo a mensagem "Aguardando conexão...". O botão **Conectar** estabelece uma sessão segura por meio do protocolo *SSH*, utilizando a biblioteca *Paramiko* e o par de chaves *RSA* previamente configurado no ambiente *Git Bash*.

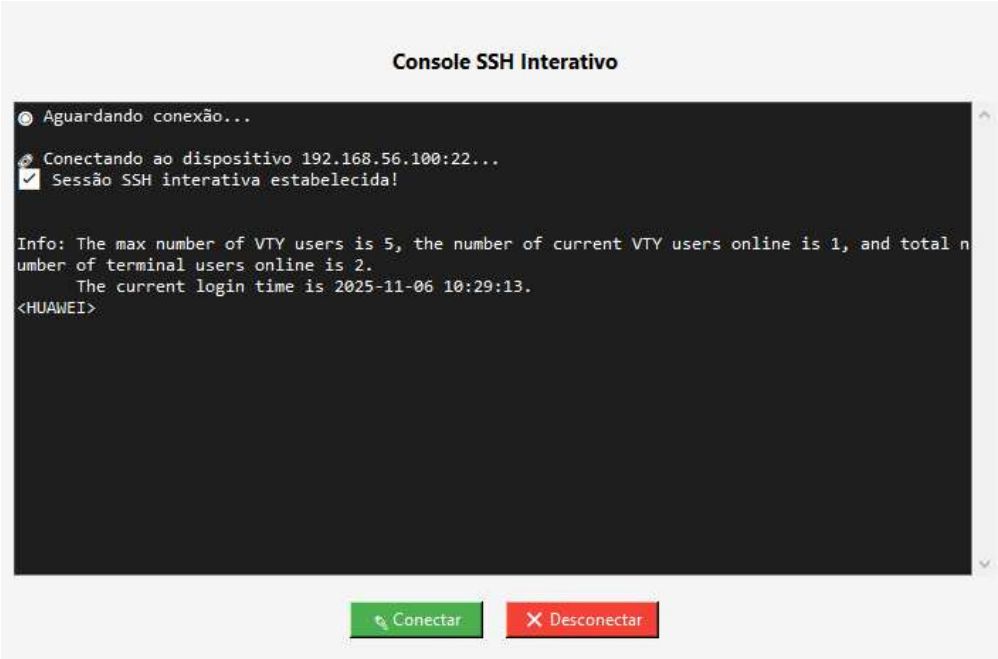
Figura 3.9: *Tela inicial do módulo Console SSH Interativo.*



Fonte: elaborado pelo autor, 2025.

Após a autenticação, o sistema confirma a conexão bem-sucedida com a mensagem "Sessão SSH interativa estabelecida", conforme apresentado na Figura 3.10. A partir desse momento, o usuário pode interagir diretamente com o equipamento, emitindo comandos de configuração, diagnóstico e gerenciamento de rede em tempo real.

Figura 3.10: *Sessão SSH estabelecida com o equipamento.*



Fonte: elaborado pelo autor, 2025.

A Figura 3.11 ilustra uma sessão interativa ativa, na qual o administrador realizou comandos diretamente no equipamento de rede por meio do terminal integrado. Nessa execução, foi demonstrado o processo de criação de uma *VLAN 10* com a inserção da descrição "Feito via console". Assim que o comando foi emitido, a ferramenta refletiu a modificação em tempo real no dispositivo, comprovando a comunicação direta e o correto funcionamento da conexão *SSH*.

Figura 3.11: *Execução de comandos no Console SSH Interativo.*

Fonte: elaborado pelo autor, 2025.

Logo após a execução no terminal, foi possível verificar que o seguinte comando foi devidamente aplicado e registrado na configuração do equipamento, confirmando o reflexo imediato da ação realizada:

```
1 vlan 10
2  description Feito via console
3 #
```

Além de demonstrar a interação em tempo real, essa funcionalidade comprova a integração direta entre o ambiente de automação e o equipamento de rede, garantindo que as ações executadas sejam aplicadas de forma confiável e auditável. O uso do *SSH* garante comunicação criptografada, preservando a segurança das credenciais e dos dados trocados durante a sessão.

Por fim, ao encerrar a sessão, o sistema exibe a mensagem "Conexão encerrada. Nenhum usuário conectado ao equipamento.", confirmando a finalização da comunicação, como mostrado na Figura 3.12. Esse retorno visual assegura que o canal de comunicação foi encerrado de forma segura e controlada.

Figura 3.12: *Encerramento da sessão no Console SSH Interativo.*

Fonte: elaborado pelo autor, 2025.

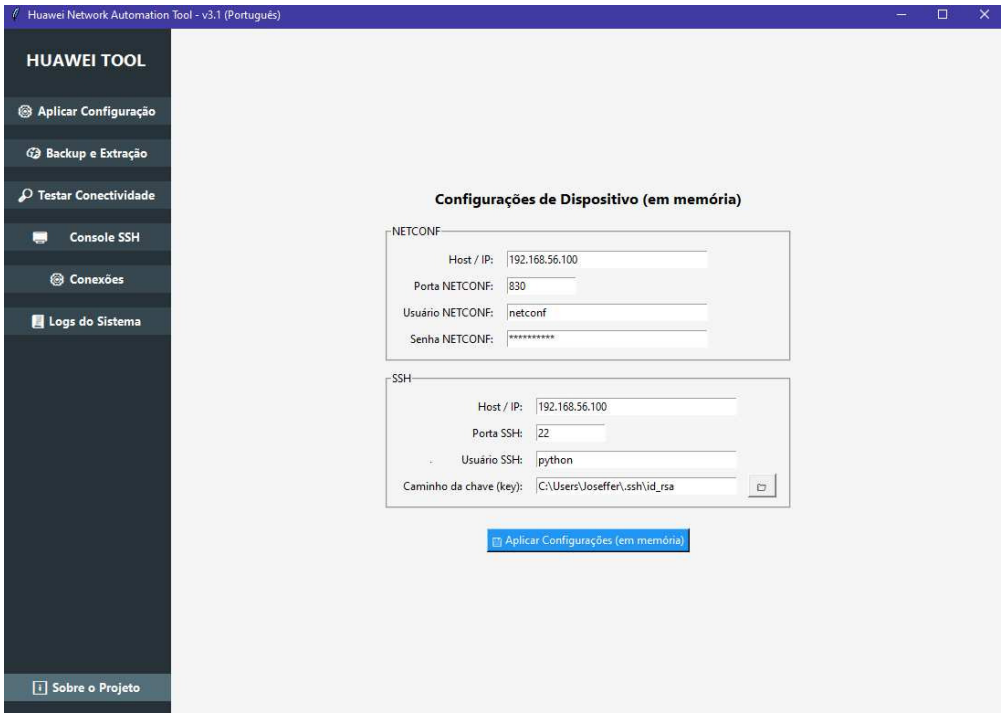
O módulo **Console SSH Interativo** é essencial para a administração e o controle dos dispositivos, pois possibilita a intervenção direta do administrador quando necessário, complementando os módulos automatizados de configuração e *backup*. Além disso, serve como ferramenta de verificação imediata, permitindo confirmar visualmente as alterações realizadas nos equipamentos. Essa característica reforça sua importância como um recurso central da aplicação, combinando automação e controle manual em um único ambiente de gestão.

3.5 Módulo de Conexões

O módulo **Conexões** é responsável pelo gerenciamento centralizado dos parâmetros de comunicação utilizados pelos demais componentes da aplicação. Por meio dessa interface, o administrador define e ajusta as informações necessárias para o estabelecimento de conexões seguras via protocolos *NETCONF* e *SSH*. Esses parâmetros são aplicados diretamente na memória do sistema, garantindo agilidade na configuração sem a necessidade de reinicialização da ferramenta.

A Figura 3.13 apresenta a interface do módulo, composta por duas seções principais: uma destinada ao protocolo *NETCONF* e outra ao *SSH*. Em cada seção, o usuário deve inserir o endereço *IP* do dispositivo, a porta de comunicação, as credenciais de acesso e, no caso do *SSH*, o caminho da chave privada *RSA* gerada previamente no ambiente *Git Bash*.

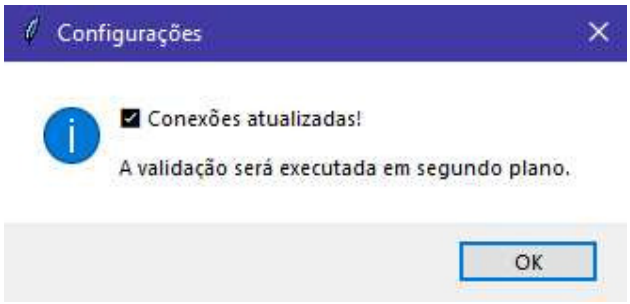
Figura 3.13: *Interface do módulo Conexões.*



Fonte: elaborado pelo autor, 2025.

Após preencher os campos, o usuário deve clicar no botão **Aplicar Configurações (em memória)** para confirmar as alterações. Ao realizar essa ação, a aplicação exibe uma janela de notificação indicando que as conexões foram atualizadas, conforme ilustrado na Figura 3.14. Esse mecanismo garante que as novas definições sejam carregadas instantaneamente, permitindo que os módulos de automação utilizem os dados mais recentes durante suas operações.

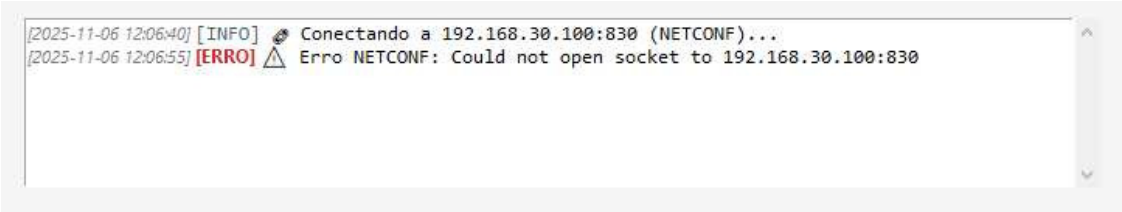
Figura 3.14: *Confirmação de atualização das conexões.*



Fonte: elaborado pelo autor, 2025.

É importante destacar que qualquer modificação incorreta nesses parâmetros impacta diretamente o funcionamento das demais funcionalidades da aplicação. Por exemplo, a alteração do endereço *IP* utilizado pelo *NETCONF* de `192.168.56.100` para `192.168.30.100` resulta em falha de conexão no módulo de Aplicar Configuração, conforme demonstrado na Figura 3.15.

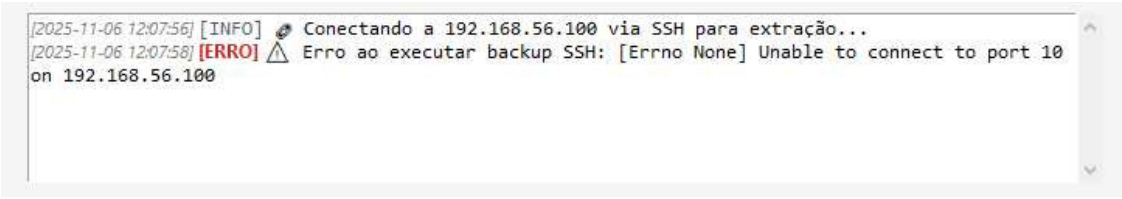
Figura 3.15: *Falha de comunicação no protocolo NETCONF.*



Fonte: elaborado pelo autor, 2025.

Situação semelhante ocorre quando há inconsistência nas informações do *SSH*. Nesse caso, qualquer alteração incorreta da porta de comunicação, do usuário ou do caminho da chave privada resulta em falhas durante a tentativa de conexão com o dispositivo. A Figura 3.16 apresenta um exemplo prático desse comportamento, em que a aplicação tenta realizar uma extração de *backup* via *SSH*, mas não obtém sucesso devido à modificação indevida da porta de comunicação.

Figura 3.16: *Erro de conexão durante a extração de backup via SSH.*



Fonte: elaborado pelo autor, 2025.

O mesmo comportamento é observado no módulo de *Console SSH Interativo*, mostrado na Figura 3.17. Nesse teste, a porta padrão `22` foi alterada para `10`, ocasionando falha

imediata ao tentar estabelecer a sessão com o equipamento. A ferramenta exibe mensagens informativas que auxiliam na identificação do problema, permitindo ao administrador corrigir rapidamente o parâmetro incorreto.

Figura 3.17: *Falha ao conectar-se ao equipamento no Console SSH Interativo.*

Fonte: elaborado pelo autor, 2025.

Esses testes demonstram a interdependência entre os módulos e reforçam a necessidade de uma configuração precisa das credenciais e parâmetros de acesso. Cada campo inserido no módulo de **Conexões** impacta diretamente o funcionamento de operações críticas, como o envio de *scripts XML*, a extração de configurações, a criação de *backups* e a execução de comandos remotos via *SSH*.

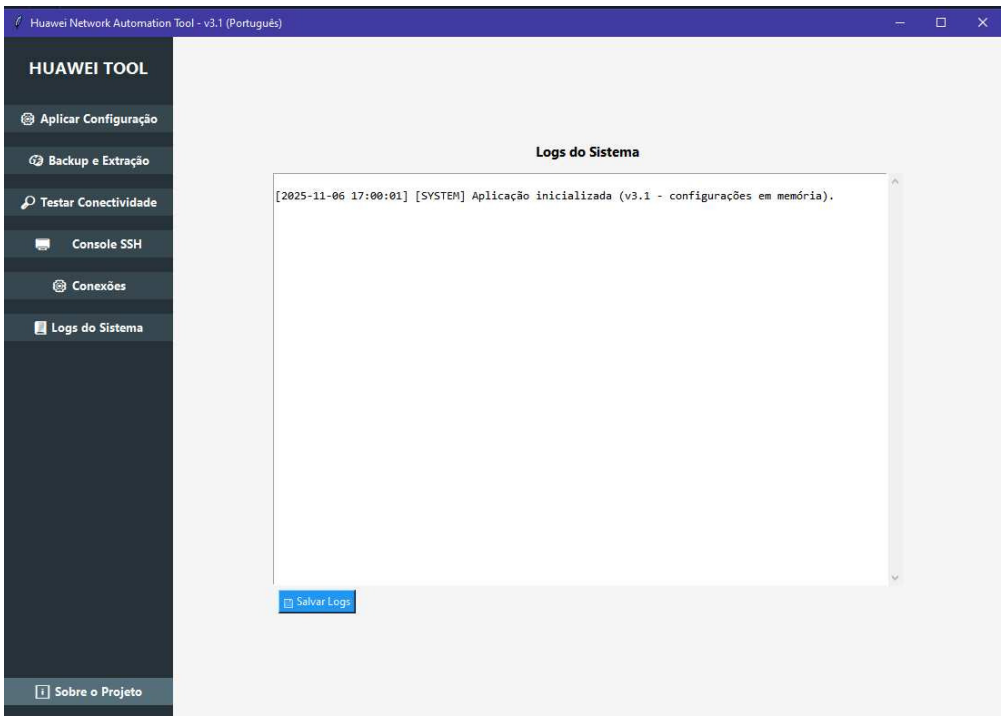
Dessa forma, o módulo **Conexões** consolida-se como a base de comunicação da aplicação, garantindo que todos os processos subsequentes sejam realizados com segurança, integridade e compatibilidade. A validação automática em segundo plano assegura o funcionamento adequado das interfaces e reduz o risco de falhas operacionais decorrentes de erros de configuração.

3.6 Módulo de *Logs* do Sistema

O módulo **Logs do Sistema** tem como finalidade registrar e apresentar, em tempo real, todas as ações executadas dentro da aplicação. Cada evento, seja ele relacionado à aplicação de configurações, extração de dados, execução de *backups*, testes de conectividade ou falhas de conexão, é documentado automaticamente com data, hora e segundo exatos. Esse recurso garante rastreabilidade total das atividades, tornando-se essencial para a análise de desempenho, auditoria de processos e identificação de possíveis falhas operacionais.

A Figura 3.18 apresenta a interface principal do módulo, onde são exibidas as mensagens de log de maneira sequencial e categorizada. Cada linha é precedida por um marcador que identifica o tipo de evento facilitando a interpretação por parte do administrador. Todas as mensagens são organizadas em ordem cronológica, possibilitando a visualização detalhada de cada etapa executada na aplicação.

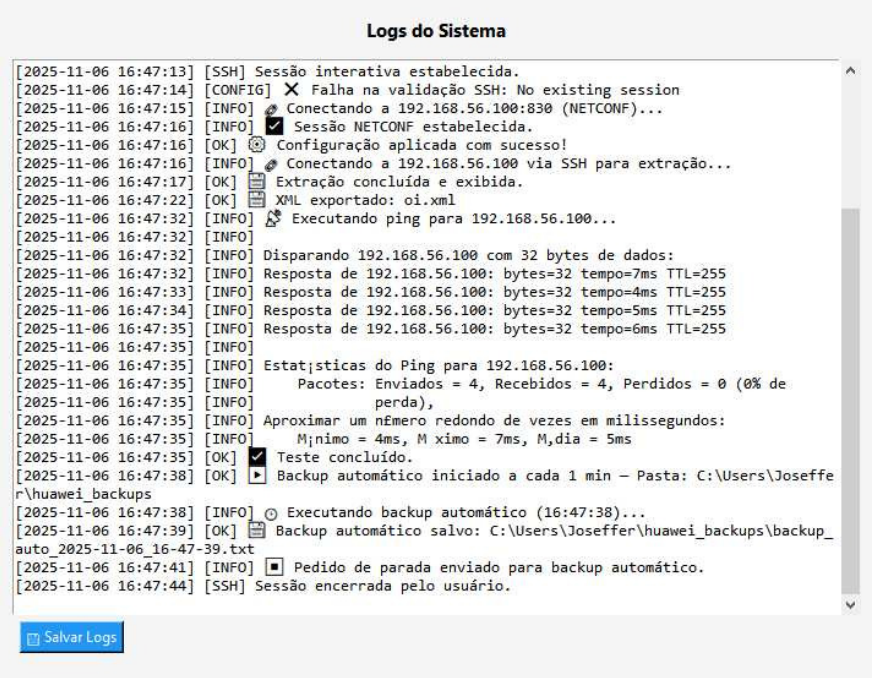
Figura 3.18: *Interface do módulo Logs do Sistema.*



Fonte: elaborado pelo autor, 2025.

Durante os testes realizados, observou-se que o módulo registra de forma precisa todas as interações com o sistema. Como exemplo, são exibidas mensagens de conexão estabelecida via *SSH*, falhas de autenticação, execução de comandos *NETCONF*, extrações de configurações e processos de *backup* automáticos. O formato detalhado de cada registro permite ao administrador compreender não apenas o que ocorreu, mas também o momento exato de cada ação e seu resultado, conforme ilustrado na Figura 3.19.

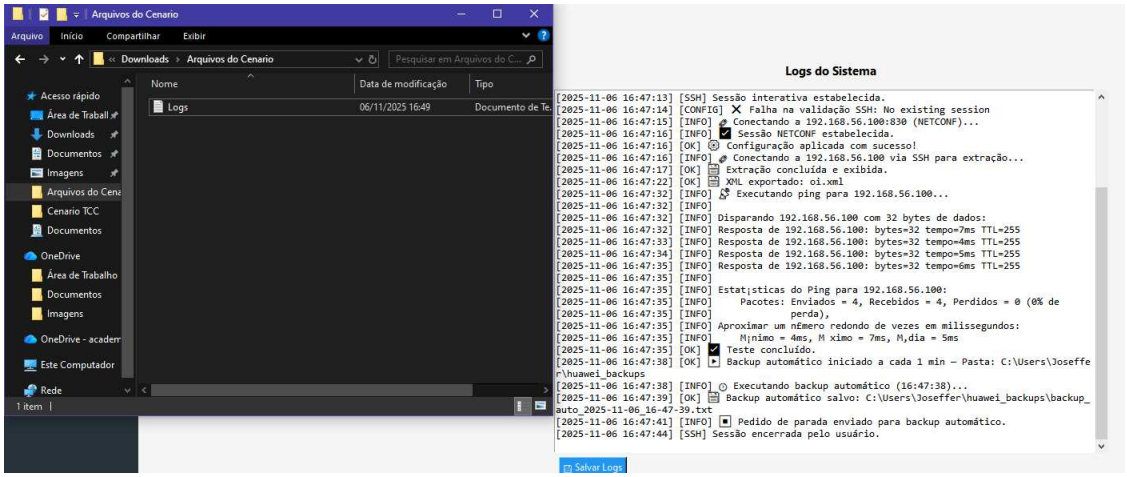
Figura 3.19: *Exemplo de registros detalhados no módulo Logs do Sistema.*



Fonte: elaborado pelo autor, 2025.

Além da visualização direta na interface, o módulo oferece a opção **Salvar Logs**, que permite exportar o conteúdo exibido para um arquivo de texto. Essa funcionalidade é especialmente útil para fins de auditoria, armazenamento histórico e diagnóstico posterior. A Figura 3.20 mostra um exemplo de arquivo de *log* salvo localmente pelo usuário, contendo registros de eventos capturados durante as operações do sistema.

Figura 3.20: *Exportação dos registros do sistema em arquivo de log.*



Fonte: elaborado pelo autor, 2025.

O módulo **Logs do Sistema** é, portanto, uma ferramenta fundamental para o controle e a manutenção da aplicação, servindo como uma camada de transparência e suporte à

análise de incidentes. Em ambientes de rede, onde a confiabilidade e a rastreabilidade são fatores críticos, o log desempenha papel primordial, permitindo ao administrador identificar rapidamente a origem de um erro, acompanhar a execução de tarefas e garantir a integridade operacional de todo o sistema.

3.7 Módulo Informativo “Sobre o Projeto”

O sistema desenvolvido inclui um módulo informativo denominado “**Sobre o Projeto**”, cuja finalidade é apresentar ao usuário a natureza da ferramenta e esclarecer seu propósito geral. Esse módulo não executa qualquer operação de automação, atuando exclusivamente como um painel descritivo que reúne informações básicas sobre o software.

A interface exibida na Figura 3.21 possui elementos como a identificação do autor, a finalidade educacional do sistema, a versão atual do software e uma observação destacando que o nome “Huawei” é utilizado apenas como referência técnica, sem representar vínculo, parceria ou autorização institucional. Também há um botão destinado ao fechamento da janela, permitindo o retorno direto à tela principal.

Figura 3.21: *Janela informativa do módulo “Sobre o Projeto”.*

Fonte: elaborado pelo autor, 2025.

O módulo tem caráter exclusivamente explicativo e foi inserido na ferramenta para fornecer clareza ao usuário, apresentando informações essenciais sem envolver qualquer funcionalidade relacionada à automação de tarefas ou à comunicação com dispositivos de rede.

Capítulo 4

Considerações Finais e Sugestões para Trabalhos Futuros

O desenvolvimento da ferramenta *Huawei Network Automation Tool* evidenciou, ao longo deste trabalho, que a integração entre programação e redes de computadores é não apenas possível, mas necessária para o avanço das práticas de administração de infraestruturas de comunicação. O estudo demonstrou que a aplicação de conceitos de lógica de programação, aliada ao uso de protocolos padronizados, permite criar soluções capazes de automatizar tarefas complexas, reduzir falhas humanas e otimizar significativamente o tempo de configuração e gerenciamento de dispositivos.

A ferramenta proposta mostrou-se funcional e eficiente, reunindo em uma única interface operações essenciais, como aplicação de configurações, execução de comandos, cópias de segurança (*backup*), verificação de conectividade, gerenciamento de conexões e registro de logs do sistema. Essa centralização de processos representa um avanço importante, pois oferece ao administrador de rede um ambiente unificado, seguro e acessível para realizar tarefas que tradicionalmente demandariam múltiplos sistemas ou interfaces.

A *Huawei Network Automation Tool* também reforça o papel da programação como um recurso indispensável na automação de redes. O uso da linguagem *Python* e das bibliotecas *Paramiko* e *ncclient* demonstrou que é possível desenvolver aplicações práticas, seguras e adaptáveis às demandas de ambientes reais, mesmo com recursos limitados. A automação apresentada comprova que, em um cenário tecnológico cada vez mais dinâmico, programação e redes devem andar lado a lado, promovendo soluções inteligentes e escaláveis para o gerenciamento de sistemas de comunicação.

Constatou-se, portanto, que a automação não apenas otimiza processos, mas também amplia a segurança e a padronização das operações. Ferramentas como a *Huawei Network Automation Tool* podem ser amplamente aplicadas em empresas e instituições que ainda utilizam métodos manuais de configuração, possibilitando a modernização de práticas e a redução do tempo gasto em tarefas repetitivas. A eficiência obtida pela aplicação prática deste trabalho evidencia que a automação é um caminho viável e essencial para o futuro da

Engenharia de Redes.

Além disso, os conceitos estudados e aplicados ao longo deste trabalho como protocolos de comunicação, arquiteturas de rede, programação e automação demonstram estar cada vez mais presentes no cotidiano das organizações e da sociedade. Com o conhecimento adequado, planejamento estruturado e domínio das ferramentas corretas, é possível empregar a programação não apenas como um meio técnico, mas como uma estratégia para aumentar a produtividade, reforçar a segurança e ampliar a escalabilidade das operações de rede. Dessa forma, o uso da automação torna-se um diferencial para o administrador de redes moderno, que passa a atuar de maneira mais proativa, eficiente e preparada para os desafios das infraestruturas digitais contemporâneas.

4.1 Sugestões para Trabalhos Futuros

A partir dos resultados alcançados, diversas melhorias e expansões podem ser exploradas em trabalhos futuros, tanto no campo técnico quanto na interface da ferramenta. Dentre as principais sugestões, destacam-se:

- Implementar novos módulos na *Huawei Network Automation Tool*, como a verificação automática de interfaces ativas, exibição da versão do sistema operacional dos dispositivos e monitoramento do tempo de atividade (*uptime*);
- Adicionar mecanismos avançados de segurança, incluindo a criptografia dos arquivos de *backup* e dos registros de log, a fim de aumentar a confidencialidade e a integridade das informações geradas;
- Criar um sistema de notificações e janelas interativas (*pop-ups*) que orientem o usuário sobre o funcionamento de cada módulo e sobre possíveis falhas durante a execução das operações;
- Ampliar a interface gráfica da ferramenta, tornando-a mais intuitiva e didática, para facilitar seu uso em ambientes corporativos e acadêmicos;
- Realizar testes de desempenho com múltiplos dispositivos simultâneos, de modo a avaliar a escalabilidade e a estabilidade da aplicação em redes de maior porte;
- Implementar melhorias de desempenho e otimização de código, com foco na redução do consumo de recursos e no aumento da velocidade das operações automatizadas;
- Corrigir o comportamento anômalo identificado no módulo de console interativo *SSH*, onde a utilização das teclas de seta ou o ato de apagar caracteres resulta na exibição incorreta de símbolos na interface. Essa limitação será ajustada em uma versão posterior da ferramenta, visto que, devido ao tempo disponível para o desenvolvimento, não foi possível realizar a correção nesta versão.

A continuidade deste projeto poderá resultar em uma versão ainda mais robusta e segura da *Huawei Network Automation Tool*, consolidando-a como uma solução acessível e eficiente para o gerenciamento de redes. Dessa forma, este trabalho reafirma a importância da integração entre programação e redes como pilares complementares de uma infraestrutura moderna, automatizada e confiável.

Apêndice A

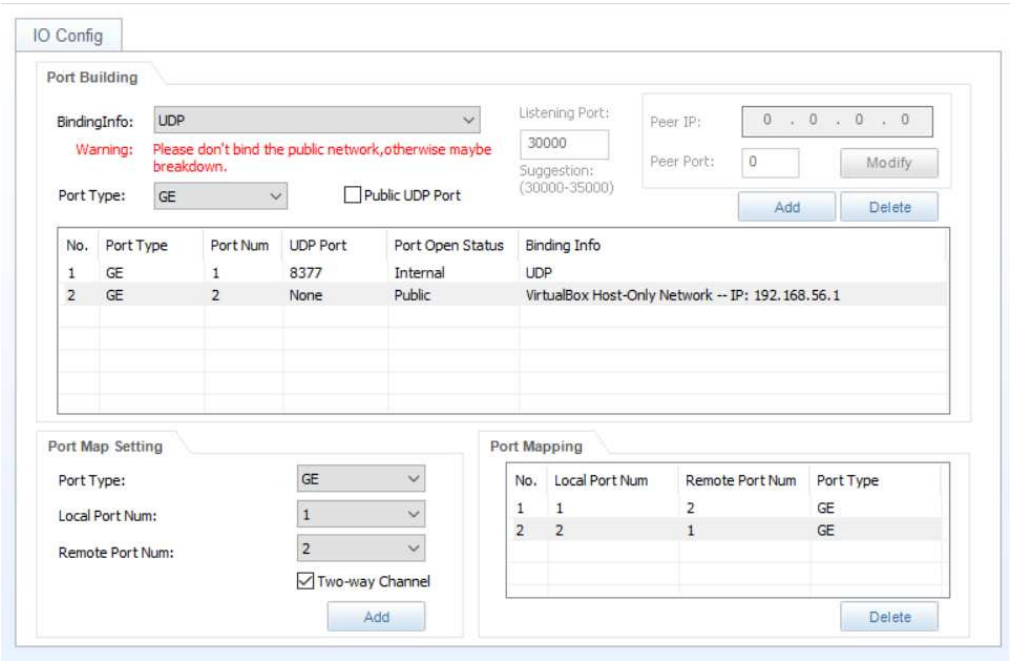
Base de Dados

Este apêndice reúne as configurações práticas utilizadas durante o desenvolvimento e teste da ferramenta *Huawei Network Automation Tool*. São apresentadas as configurações aplicadas no simulador *eNSP* e os comandos executados no ambiente *Git Bash* para geração e autenticação da chave *RSA*. O código-fonte completo da ferramenta está disponível em repositório público no *GitHub*.

Configuração da Nuvem no eNSP

A Figura A.1 exibe a configuração da nuvem (*Cloud*) no simulador *eNSP*, que realiza a integração entre o ambiente virtual e o adaptador *VirtualBox Host-Only Network*, possibilitando a comunicação entre o simulador e a aplicação Python.

Figura A.1: *Configuração da Nuvem no simulador eNSP.*



Fonte: elaborado pelo autor, 2025.

Configurações no eNSP

```
1 # NETCONF e usuario de autenticacao
2 snetconf server enable
3 ssh user netconf
4 ssh user netconf authentication-type password
5 ssh user netconf service-type snetconf
6 netconf
7  protocol inbound ssh port 830
8  quit
9 aaa
10  local-user netconf password irreversible-cipher Huawei12#$
11  local-user netconf service-type ssh
12  local-user netconf level 3
13  quit
14
15 # SSH, Paramiko e chave RSA
16 stelnet server enable
17 user-interface vty 0 4
18  authentication-mode aaa
19  protocol inbound ssh
20  user privilege level 3
21  quit
22 aaa
23  local-user python password irreversible-cipher Huawei12#$
24  local-user python user-group manage-ug
25  local-user python service-type ssh
26  quit
27 ssh user python
28 ssh user python authentication-type rsa
29 ssh user python service-type stelnet
30 rsa peer-public-key rsa01 encoding-type openssh
31  public-key-code begin
32   # Insira sua chave publica aqui #
33  public-key-code end
34  peer-public-key end
35 ssh user python assign rsa-key rsa01
36
37 # Interface de gerenciamento
38 interface Vlanif1
39  ip address 192.168.56.100 255.255.255.0
40  undo shutdown
41  quit
42
43 # Interface fisica
44 interface GigabitEthernet1/0/0
45  undo shutdown
46  quit
```

Listing A.1: *Configurações aplicadas no ambiente eNSP*

Geração e Visualização da Chave RSA no Git Bash

```
1 # Geracao das chaves publica e privada RSA
2 ssh-keygen -t rsa
3
4 # Visualizacao da chave publica para insercao no eNSP
5 cat /c/Users/Joseffer/.ssh/id_rsa.pub
```

Listing A.2: *Geração e exibição da chave RSA no ambiente Git Bash*

Repositório Oficial

<https://github.com/joseffermax/Huawei-Network-Automation-Tool>

Referências Bibliográficas

[Alfaresa, Arifwidodo e Khair 2023] ALFARESA, Y.; ARIFWIDODO, B.; KHAIR, F. Automate igp and egp routing protocol configuration using a network automation library. *Jurnal Online Informatika*, v. 8, n. 2, p. 222–231, 2023. 12

[Alharbi *et al.* 2016] ALHARBI, T. *et al.* Securing arp in software defined networks. In: *2016 IEEE 41st Conference on Local Computer Networks (LCN)*. [S.l.: s.n.], 2016. p. 523–526. 25

[Álvarez 2021] ÁLVAREZ, J. A. Evaluación de las herramientas gns3 y ensp para laboratorios virtuales. 2021. 14, 15

[Andrade 2024] ANDRADE, G. A. Implementação de uma ferramenta web para a automação de redes ip utilizando python. Universidade Federal de Uberlândia, 2024. 11

[Baarsen 2014] BAARSEN, J. V. *GitLab Cookbook*. [S.l.]: Packt Publishing Ltd, 2014. 18

[Baltatu *et al.* 2000] BALTATU, M. *et al.* Security issues in control, management and routing protocols. *Computer Networks*, v. 34, n. 6, p. 881–894, 2000. ISSN 1389-1286. Pioneering Tomorrow's Internet: Selected papers from the TARENA Networking Conference 2000. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1389128600001596>. 26

[Bellovin 1989] BELLOVIN, S. M. Security problems in the tcp/ip protocol suite. *ACM SIGCOMM Computer Communication Review*, v. 19, 1989. ISSN 0146-4833. 22

[Bhushan, Tran e Schönwälder 2009] BHUSHAN, S.; TRAN, H. M.; SCHÖNWÄLDER, J. Ncclient: A python library for netconf client applications. In: NUNZI, G.; SCOGLIO, C.; LI, X. (Ed.). *IP Operations and Management*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 143–154. ISBN 978-3-642-04968-2. 30

[Bider e Baushke 2012] BIDER, D.; BAUSHKE, M. *Sha-2 data integrity verification for the secure shell (ssh) transport layer protocol*. [S.l.], 2012. 13

[Bouju *et al.* 1999] BOUJU, A. *et al.* Client-server architecture for accessing multimedia and geographic databases within embedded systems. In: IEEE. *Proceedings. Tenth International Workshop on Database and Expert Systems Applications. DEXA 99*. [S.l.], 1999. p. 760–764. 28

[Bowen 2018] BOWEN, J. P. The impact of alan turing: Formal methods and beyond. In: *International Summer School on Engineering Trustworthy Software Systems*. [S.l.]: Springer, 2018. p. 202–235. 6, 8

[Brandão 2019] BRANDÃO, P. R. Alan turing: da necessidade do cálculo, a máquina de turing até à computação. 2019. 7, 8

[Campos e Paixo 2019] CAMPOS, G. D. S.; PAIXO, F. C. D. Desenvolvimento de ferramenta de gerenciamento de funcionalidades de switches, roteadores e cme da cisco para otimizao de processo de configurao para garantia de qualidade com uso da linguagem python. *Revista Computao Aplicada - UNG-Ser*, v. 7, p. 27, 10 2019. ISSN 2316-7394. 11

[Cañas 2025] CAñAS, D. E. M. Automatización de infraestructura de redes: eficiencia de tareas concurrentes utilizando lenguaje de programación python. *Realidad y Reflexión*, p. 124–137, 7 2025. ISSN 2520-9299. 10

[Chacon e Straub 2010] CHACON, S.; STRAUB, B. Pro git. *Recuperado de: http://labs. kernelconcepts. de/downloads/books/Pro% 20Git*, 2010. 16, 17

[Chacon e Straub 2014] CHACON, S.; STRAUB, B. *Pro git*. [S.l.]: Springer Nature, 2014. 16, 19

[Chappell 2013] CHAPPELL, C. Creating the programmable network: The business case for netconf/yang in network devices. *Whitepaper on Behalf of Tail-f*, p. 5, 2013. 28, 29

[Chen 2001] CHEN, T. M. Increasing the observability of internet behavior. *Communications of the ACM*, ACM New York, NY, USA, v. 44, n. 1, p. 93–98, 2001. 26

[Choi 2024] CHOI, B. *Introduction to Python Network Automation Volume II*. [S.l.]: Springer, 2024. 12

[Chun 2001] CHUN, W. *Core python programming*. [S.l.]: Prentice Hall Professional, 2001. v. 1. 8, 9

[CINGOLANI 2019] CINGOLANI, C. Simulazione di una architettura di rete enterprise basata sul software huawei ensp. 2019. 14, 15

[Class *et al.* 1996] CLASS, B. *et al.* Understanding ip addressing: Everything you ever wanted to know. 1996. 23

[Cohen-Almagor 2013] COHEN-ALMAGOR, R. Internet history. In: *Moral, ethical, and social dilemmas in the age of technology: Theories and practice*. [S.l.]: IGI Global, 2013. p. 19–39. 20

[Curran 2012] CURRAN, J. Rethinking internet history: James curran. In: *Misunderstanding the internet*. [S.l.]: Routledge, 2012. p. 40–71. 20

[Delaval, Martins e Melo 2024] DELAVAL, R. C.; MARTINS, R. H.; MELO, S. G. Análise e monitoramento de redes de computadores com zabbix®: Estudo de caso em empresa no interior de são paulo. *Revista Interciência e Sociedade*, v. 9, n. 2, 2024. 25

[DeWitt *et al.* 1994] DEWITT, D. J. *et al.* Client-server paradise. In: CITESEER. *VLDB*. [S.l.], 1994. v. 94, p. 558–569. 28

[Hedstrom, Watwe e Sakthidharan 2011] HEDSTROM, B.; WATWE, A.; SAKTHIDHARAN, S. Protocol efficiencies of netconf versus snmp for configuration management functions. *University of Colorado, Master Thesis*, 2011. 28, 29

[Khoa 2023] KHOA, D. N. A. Network automation with python. 2023. 10, 11, 12

[Kumar *et al.* 2010] KUMAR, M. A. *et al.* A new way towards security in tcp/ip protocol suite. In: CITESEER. *Proceedings of the 14th WSEAS international conference on Computers: part of the 14th WSEAS CSCC multiconference*. [S.l.], 2010. v. 1. 26

[Kumar e Shinde 2016] KUMAR, R.; SHINDE, P. R. Computer network-ip address & subnetting. *International Journal of Engineering and Advanced Technology (IJEAT)*, v. 5, n. 4, p. 242–246, 2016. 23

[Kumar 2019] KUMAR, S. A review on client-server based applications and research opportunity. *International Journal of Recent Scientific Research*, v. 10, n. 7, p. 33857–3386, 2019. 27, 28

[Leiner *et al.* 1997] LEINER, B. M. *et al.* The past and future history of the internet. *Communications of the ACM*, ACM New York, NY, USA, v. 40, n. 2, p. 102–108, 1997. 20

[Maldonado 2022] MALDONADO, I. A. Z. *Implementación de prácticas de enrutamiento con equipos virtualizados de networking de huawei en GNS3 y ENSP.*. Dissertação (B.S. thesis) — Quito, 2022, 2022. 16

[Menezes 2010] MENEZES, N. N. C. Introdução à programação com python. *São Paulo: Novatec*, p. 34, 2010. 8, 9

[Mihăilă *et al.* 2017] MIHĂILĂ, P. *et al.* Network automation and abstraction using python programming methods. *MACRo 2015*, Walter de Gruyter GmbH, v. 2, n. 1, p. 95–103, 2017. 10, 12

[Milanesio 2013] MILANESIO, L. *Learning Gerrit Code Review*. [S.l.]: Packt Publishing, 2013. v. 144. 18, 19

[Nyabuto 2023] NYABUTO, G. Client-server architecture, a review. *International Journal of Advanced Science and Computer Applications*, v. 3, 12 2023. ISSN 2809-7467. 27

[Oluwatosin 2014] OLUWATOSIN, H. S. Client-server model. *IOSR Journal of Computer Engineering*, v. 16, n. 1, p. 67–71, 2014. 27

[Osei-Wusu *et al.* 2025] OSEI-WUSU, F. *et al. A Scheme for Network Programmability and Backup Automation Using Python Netmiko Library on Cisco; the Case Study of the Komfo Anokye Teaching Hospital Local Area Network*. 2025. 11

[Parameswari e SURESH 2009] PARAMESWARI, D.; SURESH, D. Arp protocol sequence analysis for intrusion detection system. *Research Scholar, Mother Teresa Women s University, Kodaikanal-624*, Citeseer, v. 101, 2009. 24

[Platform 2020] PLATFORM, K. A. Internet. *Zurich, Switzerland: KNIME AG*, 2020. 20

[Prabowo e Ariyanto 2025] PRABOWO, W.; ARIYANTO, R. Implementation of python-based network automation technology for computer infrastructure optimization. *Impact: Multidisciplinary Journal*, v. 1, n. 01, p. 34–40, 2025. 10

[Rossum e Jr 2017] ROSSUM, G. V.; JR, F. L. D. *El tutorial de Python*. [S.l.]: Python Software Foundation, 2017. 9

[Saputro 2015] SAPUTRO, F. Icmp (internet control message protocol). 2015. 25, 26

[Schmitzhaus 2024] SCHMITZHAUS, C. Usabilidade em interfaces de linha de comando. Universidade Federal de Santa Maria, 2024. 16, 17, 18

[Schönwälder, Björklund e Shafer 2010] SCHöNWäLDER, J.; BJöRKLUND, M.; SHAFER, P. Network configuration management using netconf and yang. *IEEE Communications Magazine*, v. 48, n. 9, p. 166–173, 2010. 29

[Silva *et al.* 2024] SILVA, D. M. da *et al.* History and legacy of alan turing for computer science. *International Journal of Scientific Research and Management (IJSRM)*, v. 12, p. 1047–1056, 2 2024. ISSN 2321-3418. 6, 7

[Singh 2015] SINGH, D. A. K. Internet protocol (ip) address–subnetting and supernetting. *Int. J. Emerg. Trends Technol. Comput. Sci*, v. 4, p. 87–90, 2015. 23

[Smaldone 2004] SMALDONE, J. *Introducción a Secure Shell*. [S.l.]: Obtenido de http://es. tldp. org/Tutoriales/doc-ssh-intro/introduccion_ssh-0 . . . , 2004. 12, 13, 14

[Stancu, Shevrikuko e Rueda 2019] STANCU, S. N.; SHEVRIKUKO, A.; RUEDA, D. G. Evolving cern's network configuration management system. *EPJ Web of Conferences*, v. 214, p. 08015, 9 2019. ISSN 2100-014X. 31

[Sukkar *et al.* 2016] SUKKAR, G. A. *et al.* Address resolution protocol (arp): Spoofing attack and proposed defense. Scientific Research Publishing, 2016. 24

[Tang, Liu e Zhu 2004] TANG, X. H.; LIU, Z. L.; ZHU, M. L. Tcp-rab: A receiver advertisement based tcp protocol. *Journal of Zhejiang University: Science*, v. 5, 2004. ISSN 10093095. 21

[Thomas 2000] THOMAS, G. Introduction to subnetting. *The extension A Technical Supplement to Control network*, v. 1, n. 8, 2000. 23

[Trabelsi e Shuaib 2008] TRABELSI, Z.; SHUAIB, K. Spoofed arp packets detection in switched lan networks. In: *Communications in Computer and Information Science*. [S.l.: s.n.], 2008. v. 9. ISSN 18650937. 25

[Tran, Tumar e Schönwälder 2009] TRAN, H. M.; TUMAR, I.; SCHÖNWÄLDER, J. Netconf interoperability testing. In: SADRE, R.; PRAS, A. (Ed.). *Scalability of Networks and Services*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 83–94. ISBN 978-3-642-02627-0. 29

[Đumić e Lubura 2021] ĐUMIĆ, D.; LUBURA, S. Leveraging raspberry pi as a server for the integration of the netconf protocol within iot systems based on yang. *Journal of Engineering and Natural Sciences*, v. 3, 2021. 30

[Vidya, Gowri e Bhaskaran] VIDYA, S.; GOWRI, N.; BHASKARAN, R. Arp traffic and network vulnerability. *proceedings of INDIACOM-2011, conducted by BVICAM, New Delhi, India, page–619 and in CD*, Citeseer. 24

[Wallin e Wikström 2011] WALLIN, S.; WIKSTRÖM, C. Automating network and service configuration using {NETCONF} and {YANG}. In: *25th Large Installation System Administration Conference (LISA 11)*. [S.l.: s.n.], 2011. 29

[Wegner e Rockell 2000] Chapter 1 - addressing and subnetting basics. In: WEGNER, J.; ROCKELL, R. (Ed.). *IP Addressing Subnetting INC IPV6*. Rockland: Syngress, 2000. p. 1–38. ISBN 978-1-928994-01-5. Disponível em: <https://www.sciencedirect.com/science/article/pii/B9781928994015500048>. 22

[Wegner e Rockell 2000] Chapter 2 - creating an addressing plan for fixed-length mask networks. In: WEGNER, J.; ROCKELL, R. (Ed.). *IP Addressing Subnetting INC IPV6*. Rockland: Syngress, 2000. p. 39–86. ISBN 978-1-928994-01-5. Disponível em: <https://www.sciencedirect.com/science/article/pii/B978192899401550005X>. 23

[Winston 2002] WINSTON, B. *Media,Technology and Society*. [S.l.]: Routledge, 2002. ISBN 9780203024379. 19

[Xi 2017] XI, H. Research and application of arp protocol vulnerability attack and defense technology based on trusted network. *AIP Conference Proceedings*, v. 1820, n. 1, p. 090019, 03 2017. ISSN 0094-243X. Disponível em: <https://doi.org/10.1063/1.4977403>. 24

[Yang 2014] YANG, R. Study on arp protocol and network security in digital manufacturing. In: *Green Power, Materials and Manufacturing Technology and Applications III*. [S.l.]: Trans Tech Publications Ltd, 2014. (Applied Mechanics and Materials, v. 484), p. 191–195. 25

[Ylonen 2006] YLONEN, T. *Rfc 4251: The secure shell (ssh) protocol architecture*. [S.l.]: RFC Editor, 2006. 12, 13

[Ylonen e Lonvick 2006] YLONEN, T.; LONVICK, C. *The secure shell (SSH) transport layer protocol*. [S.l.], 2006. 13

[Zadka 2019] ZADKA, M. Paramiko. In: _____. *DevOps in Python: Infrastructure as Python*. Berkeley, CA: Apress, 2019. p. 111–119. ISBN 978-1-4842-4433-3. Disponível em: <https://doi.org/10.1007/978-1-4842-4433-3_9>. 11

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA
Campus Campina Grande - Código INEP: 25137409
R. Tranqüilino Coelho Lemos, 671, Dinamérica, CEP 58432-300, Campina Grande (PB)
CNPJ: 10.783.898/0003-37 - Telefone: (83) 2102.6200

Documento Digitalizado Restrito

Entrega de Trabalho de TCC em formato PDF a

Assunto:	Entrega de Trabalho de TCC em formato PDF a
Assinado por:	Joseffer Merces
Tipo do Documento:	Dissertação
Situação:	Finalizado
Nível de Acesso:	Restrito
Hipótese Legal:	Informação Pessoal (Art. 31 da Lei no 12.527/2011)
Tipo do Conferência:	Cópia Simples

Documento assinado eletronicamente por:

- **Joseffer Maxwel Oliveira das Merces**, **DISCENTE (202311210030) DE TECNOLOGIA EM TELEMÁTICA - CAMPINA GRANDE**, em 23/02/2026 10:29:15.

Este documento foi armazenado no SUAP em 27/02/2026. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifpb.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 1782372
Código de Autenticação: 2a8c308fbb