TP2 – Monitorização Distribuída de Redes (Grupo 6)

André Carvalho[1[100818]], Flávio Sousa[1[100715]] and João Longo[1[95536]]

[1] Universidade do Minho, Braga, Portugal

Abstract. Este projeto propõe o desenvolvimento de um Sistema de Monitorização de Redes Distribuído (NMS), capaz de coletar métricas, identificar anomalias e gerar alertas em tempo real. A solução adota uma arquitetura cliente-servidor, onde o NMS_Agent coleta e reporta métricas para o NMS_Server, utilizando dois protocolos personalizados: NetTask (baseado em UDP) para a coleta de dados e AlertFlow (baseado em TCP) para a notificação de eventos críticos. O projeto aborda o desenvolvimento de mecanismos robustos de comunicação, incluindo retransmissão, controlo de fluxo e numeração de sequência para interações via UDP. Métricas como latência, largura de banda, perda de pacotes e jitter são obtidas com ferramentas como ping e iperf, enquanto que o uso de recursos dos dispositivos também é monitorizado. A configuração do sistema é gerida por arquivos JSON, permitindo flexibilidade na definição de tarefas e limites para alertas.

Keywords: Python, Redes de Computadores, Comunicação de Computadores, Core

1 Contextualização e Justificação

Este trabalho prático tem como objetivo o desenvolvimento de um Sistema de Monitorização de Redes (NMS) baseado numa arquitetura distribuída e resiliente. Utilizando protocolos personalizados, como o NetTask e o AlertFlow, implementados sobre as camadas de transporte UDP e TCP, respetivamente, fomos desafiados a criar uma solução capaz de coletar métricas de desempenho, identificar problemas e gerar alertas de forma eficiente. Além disso, a adoção de técnicas avançadas, como retransmissão e controlo de fluxo no protocolo UDP, fomenta o aprendizado de conceitos fundamentais de redes e comunicação distribuída.

A relevância deste trabalho reside não apenas na consolidação de conceitos teóricos, mas também na sua aplicabilidade prática. Ao implementar um sistema capaz de lidar com cenários reais de falhas e perdas de pacotes, somos preparados para enfrentar desafios comuns em ambientes complexos. A experiência proporcionada por este projeto é crucial para o desenvolvimento de habilidades técnicas e analíticas essenciais para a conceção de sistemas modernos de monitorização, alinhando-se às exigências do mercado e às necessidades atuais da área de redes. Assim, este trabalho integra inovação, aplicabilidade e aprendizado, proporcionando uma base sólida para o desenvolvimento de soluções tecnológicas avançadas.

2 Objetivos do Projeto

1. **Desenvolver um Sistema de Monitorização de Redes (NMS):** Implementar uma solução distribuída composta por um servidor central (NMS_Server) e múltiplos agentes (NMS_Agents) para monitorizar dispositivos e as respetivas rede, garantindo a recolha e o gerenciamento de métricas de desempenho.

2. Projetar e Implementar Protocolos Aplicacionais:

- Criar o protocolo NetTask (baseado em UDP) para a comunicação de tarefas e a recolha contínua de métricas, incorporando características como números de sequência, ACKs, retransmissões e controle de fluxo.
- Desenvolver o protocolo AlertFlow (baseado em TCP) para a notificação confiável de eventos críticos e alterações significativas no estado da rede.

3. Implementar Comunicação Resiliente:

- Garantir a confiabilidade e a ordem das mensagens no protocolo NetTask, mesmo em condições adversas de rede.
- Assegurar que a comunicação no protocolo AlertFlow seja robusta e livre de falhas.

4. Promover a Escalabilidade e Flexibilidade:

- Permitir a monitorização simultânea de múltiplos NMS_Agents reportando ao NMS_Server.

5. Simular Cenários Reais de Rede: Validar o sistema em um ambiente controlado utilizando o emulador CORE, incluindo a execução de testes com diferentes níveis de perda de pacotes e falhas nos links.

3 Requisitos do Sistema

3.1 Requisitos Funcionais

1. Coleta e Monitorização de Métricas: O sistema deve coletar métricas de rede (latência, jitter, perda de pacotes e largura de banda) e de dispositivos (uso de CPU, RAM e estatísticas de interfaces) sendo possível a respetiva configuração via arquivos JSON.
2. Comunicação Cliente-Servidor: Deve ser implementada a comunicação entre NMS_Agent e NMS_Server.
3. Registro e Gerenciamento de Agentes: O NMS_Agent deve registrar-se no NMS_Server ao iniciar e receber tarefas personalizadas.
4. Armazenamento de Dados: O servidor deve armazenar todas as métricas e informações recebidas para consulta e análise futura.
5. Geração de Alertas: O sistema deve enviar notificações ao NMS_Server sempre que valores críticos forem detectados, com base em limites pré-configurados.
6. Apresentação de Métricas: Deve ser disponibilizada uma interface no NMS_Server para visualizar métricas de forma clara e estruturada.

3.2 Requisitos Não Funcionais

1. Escalabilidade: O sistema deve suportar a comunicação simultânea de múltiplos NMS_Agents com um único NMS_Server.
2. Resiliência: O protocolo NetTask deve ser capaz de lidar com perdas de pacotes, utilizando retransmissão e mecanismos de controle de fluxo.
3. Desempenho: O sistema deve coletar e processar métricas de forma eficiente, com impacto mínimo no desempenho dos dispositivos monitorizados.
4. Confiabilidade: A comunicação via AlertFlow deve garantir a entrega confiável de mensagens, mesmo em cenários adversos.
5. Flexibilidade: A configuração das tarefas deve ser simples e extensível, utilizando arquivos JSON.
6. Compatibilidade: O sistema deve ser testado no emulador CORE para garantir que funcione em cenários de rede simulados.

4 Arquitetura do Sistema

O sistema desenvolvido apresenta uma arquitetura modular, projetada para realizar o monitoramento de recursos computacionais e o gerenciamento de tarefas em rede de forma eficiente e escalável. Cada componente desempenha um papel específico, contribuindo para o funcionamento harmonioso do todo. A seguir, detalhamos os principais módulos que compõem a solução:

1. Módulo NMS_SERVER
O módulo central da arquitetura, o NMS_SERVER, é responsável por coordenar as operações do sistema. Ele gerencia as conexões com os agentes distribuídos, organiza as tarefas enviadas aos clientes e garante a confiabilidade da comunicação, implementando mecanismos de retransmissão em caso de falhas. Adicionalmente, realiza verificações periódicas de inatividade para assegurar que todos os agentes estão operacionais.
2. Módulo NMS_AGENT
O NMS_AGENT opera como o elo entre os dispositivos monitorados e o servidor central. Este módulo é responsável por coletar informações de desempenho do hardware e executar comandos pré-definidos, como medições de conectividade e latência. Ele utiliza threads e sockets para processar múltiplas conexões simultâneas, garantindo a escalabilidade e a eficiência do sistema.
3. Módulo AlertFlow
Dedicado ao monitoramento de recursos críticos, o AlertFlow verifica continuamente o estado de uso da CPU, memória e interfaces de rede, utilizando a biblioteca *psutil*. Quando os limites estabelecidos são excedidos, o módulo gera alertas e transmite-os ao servidor via TCP.

5. Módulo NetTask

Desempenha um papel fundamental na arquitetura. Ele integra as operações de rede, como a criação e o gerenciamento de tarefas distribuídas, além de fornecer suporte a funcionalidades adicionais, como o balanceamento de carga entre agentes e a priorização de tarefas críticas.

4. Módulo execute_tasks

Especializado na execução de tarefas específicas, o execute_tasks é responsável por operações como a realização de testes de conectividade (e.g., comandos *ping*). Ele processa configurações recebidas e transmite os resultados ao servidor por meio de sockets UDP, complementando as funcionalidades de monitoramento.

4.1 Interconexão e Operação

A comunicação entre os módulos é realizada utilizando protocolos de rede (TCP e UDP), assegurando um fluxo contínuo de dados e o atendimento rápido a eventos críticos. A modularidade da solução facilita a manutenção e a expansão futura, permitindo que novos módulos ou funcionalidades sejam incorporados sem comprometer o funcionamento do sistema.

4.2 Fluxo de Comunicação

Na Figura 1, é ilustrado o fluxo detalhado de comunicação entre o cliente e o servidor, evidenciando os mecanismos de confiabilidade e controlo implementados no sistema. O processo inicia-se com o envio de um pedido de conexão pelo cliente, ao qual o servidor responde com uma mensagem de confirmação (ACK), estabelecendo o vínculo inicial entre as duas partes. Durante a comunicação, todas as mensagens enviadas pelo cliente são acompanhadas por um controle rigoroso de recebimento: para cada mensagem transmitida, o cliente espera um ACK de confirmação. Caso este não seja recebido após cinco tentativas de reenvio, o cliente assume que o servidor está inoperante e encerra a sua tentativa de comunicação.

Do lado do servidor, o controle de conectividade é mantido pelo envio periódico de mensagens de keep-alive por parte do cliente. Estas mensagens servem para indicar que o cliente permanece ativo na rede. Se o servidor não receber um keep-alive dentro do intervalo de tempo estipulado, ele presume que o cliente desconectou-se inesperadamente e atualiza o seu estado interno para refletir esta condição.

Além disso, no encerramento da comunicação, o cliente envia um pedido de desconexão, ao qual o servidor responde com um ACK* final, garantindo o término ordenado e seguro da sessão.

Outra característica essencial do sistema é a capacidade de realizar tarefas e medições de métricas de forma simultânea e eficiente. As tarefas são executadas em paralelo utilizando threads, enquanto os resultados são enviados ao servidor em conformidade com os protocolos definidos, assegurando a integridade e a sincronização dos dados durante todo o processo.

Este fluxo de comunicação demonstra a robustez do sistema, que não apenas garante a confiabilidade na troca de informações, mas também monitora continuamente o estado das conexões, permitindo uma operação resiliente em ambientes de rede dinâmicos.

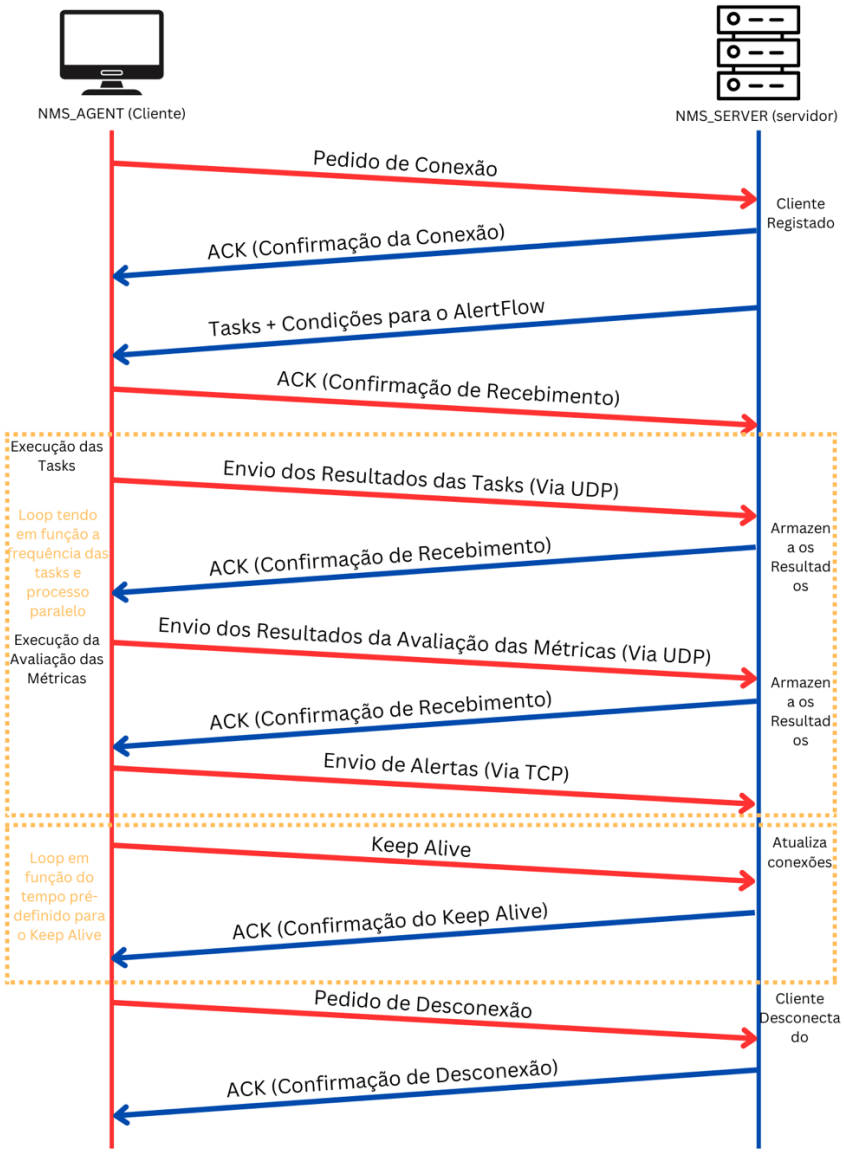
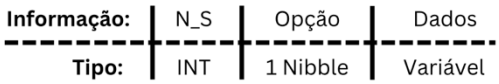


Figura 1 - Fluxo de Comunicação

5 Descrição do Protocolo Implementado

A informação base transportada em qualquer mensagem trasmitida via UDP toma por regra básica a seguinte estrutura:

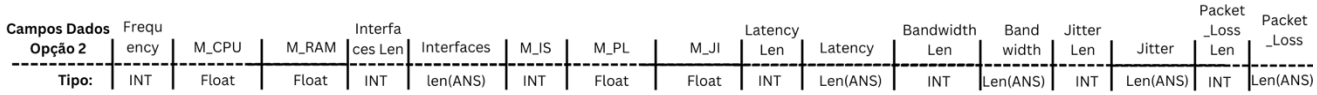


O campo N_Sé um inteiro auto-incrementado cuja principal função é garantir a ordenação das mensagens, tanto do lado do servidor quanto do cliente. Esse mecanismo assegura que as mensagens sejam processadas na sequência correta, evitando inconsistências na comunicação. O campo Opção consiste em um nibble (4 bits) destinado à qualificação do tipo de mensagem que está a ser transportada. Este campo é crucial para a identificação e o tratamento adequado das mensagens no protocolo. A Tabela 1 apresenta os diversos tipos de opções disponíveis no protocolo, detalhando as suas finalidades específicas. Por fim, o campo Dados é de comprimento variável, podendo conter ou não informações, dependendo do tipo de opção transportada. A interpretação e o processamento deste campo serão analisados detalhadamente de seguida.

Tabela 1 - Opções e Respetivas Descrições

Opção (Número em Inteiro)	Descrição
0000 (0)	Pedido de Conexão
0001 (1)	Resposta a uma Tarefa realizada
0010 (2)	Envio de Tarefas e Condições para o AlertFlow
0011 (3)	Resposta de Monitorização
0100 (4)	Resposta ACK
0101 (5)	Keep Alive
0111 (7)	Pedido de Disconexão

O campo Dados possui tamanho variável, sendo utilizado de forma condicional conforme o valor do campo Opção. Especificamente, quando o campo Opção assume os valores 0, 4, 5 ou 7, o campo Dados permanece vazio. Ele é preenchido exclusivamente nos casos em que o valor do campo Opção é 1 ou 2* de acordo com a lógica definida no protocolo.



Legenda: M_CPU (Máximo de CPU); M_RAM (Máximo de RAM); Len (Tamanho); M_IS (Máximo de Interfaces); M_PL (Máximo de Packet Loss); M_JI (Máximo de Jitter); Len(ANS) (Tamanho da célula anterior)

E para a opção 1 temos os seguintes campos:

Campos Dados	Resposta	
Opção 1	Len	Resposta
Tipo:	INT	Len(ANS)

Legenda: Len (Tamanho); Len(ANS) (Tamanho da célula anterior)

Dessa forma, é possível determinar com precisão os valores máximos e mínimos do tamanho do nosso protocolo. O valor máximo de tamanho é alcançado sempre que uma nova tarefa é enviada a um novo cliente, o que implica no preenchimento de T=41+$L1$+$L2$+$L3$+$L4$+$L5$ bytes, sendo que L_1 *a* L_5 corresponde aos campos de tamanho dependentes. Este cenário ocorre, por exemplo, quando há a necessidade de transmitir informações mais complexas ou detalhadas, como novas configurações de tarefas ou dados adicionais requeridos pelo sistema. Já o valor mínimo de tamanho é atingido na grande maioria das opções do protocolo, resultando em uma utilização de T=9+X bytes sendo X = *Tamanho da Resposta*. Este cenário ocorre quando as mensagens são mais simples, com menos informações a serem transmitidas, como em casos de controlo ou verificações periódicas.

Esta variação no tamanho das mensagens permite uma otimização eficiente da comunicação entre o cliente e o servidor. A flexibilidade no tamanho das mensagens, com valores máximos e mínimos bem definidos, proporciona um equilíbrio ideal entre o uso da largura de banda e a robustez do sistema. Desta maneira, o protocolo assegura um fluxo de comunicação estável e seguro, garantindo que, mesmo em situações de tráfego intenso ou complexidade aumentada, a troca de dados entre os dispositivos seja feita de maneira confiável, sem comprometer a integridade das informações ou a eficiência do sistema.

6 Funcionalidades Implementadas

O sistema desenvolvido conta com diversas funcionalidades que asseguram o seu desempenho, robustez e capacidade de monitoramento e gerenciamento num ambiente distribuído. Estas funcionalidades foram cuidadosamente projetadas e implementadas para atender aos requisitos do protocolo e às necessidades operacionais. Abaixo, destacamos as principais:

6.1 Estabelecimento e Gerenciamento de Conexões

- Pedido de Conexão com Confirmação (ACK): Permite que o cliente inicie uma conexão com o servidor de forma confiável, recebendo uma mensagem de confirmação para validar o vínculo estabelecido.

- Controle de Conexões Ativas com Keep-Alive: O cliente envia periodicamente mensagens de keep-alive para indicar a sua presença na rede. Caso o servidor não receba essas mensagens dentro do tempo estipulado, a conexão é considerada encerrada.

- Pedido de Desconexão com Confirmação (ACK): Garante o término ordenado da sessão entre cliente e servidor, com mensagens específicas para encerrar a comunicação de forma segura.

6.2 Execução de Tarefas e Monitoramento

- Execução Simultânea de Tarefas: Permite que tarefas, como envio de comandos e execução de testes, sejam realizadas paralelamente às medições de métricas, utilizando threads para melhorar a eficiência.

- Medição de Recursos do Cliente: O cliente realiza monitoramento contínuo de métricas, como uso de CPU, memória e estado das interfaces de rede, enviando os resultados ao servidor conforme necessário.

6.3 Tolerância a Falhas

- Reenvio de Mensagens: Em caso de falha de comunicação, o cliente realiza tentativas de reenvio com limitação configurada, aumentando a confiabilidade do sistema.

- Detecção de Inatividade: O servidor monitora ativamente as conexões por meio do recebimento de keep-alive. A ausência dessas mensagens resulta na remoção do cliente da lista de conexões ativas.

6.4 Suporte a Protocolos de Rede

- Comunicação via TCP e UDP: O sistema utiliza sockets para comunicação bidirecional eficiente, empregando TCP para confiabilidade e UDP para tarefas leves e de baixa latência.

7 Testes e Resultados

Os testes foram realizados num ambiente Core - Xubuntu, configurado numa máquina virtual alojada num sistema operativo Windows 11. O equipamento físico utilizado dispõe de um processador AMD Ryzen 5, 8 GB de memória RAM e 512 GB de armazenamento. A máquina virtual foi configurada com 2 GB de memória e o sistema operativo Xubuntu, garantindo um ambiente controlado e adequado para a execução dos testes.

Nas Figuras 2, 3, 4, 5 e 6 temos exemplos ilustrativos da interface e dos resultados obtidos. Podemos concluir que o programa executa de forma fiável todas as tarefas e responsabilidades.

Fig 2 - Menu Inicial

```
root@PC3:/tmp/pycore.35035/PC3.conf# cd /home/core/Desktop/Client/
root@PC3:/home/core/Desktop/Client# python3 NMS_AGENT.py
NMS_CLIENT Iniciado (UDP e TCP)
Digite 'q' para sair: 10.0.5.10:10:30
Calculando Latência para 10.0.5.10 com 10 pacotes...
Calculando Perda de Pacotes para 10.0.5.10 com 10 pacotes...
Calculando Jitter para 10.0.5.10 com 10 pacotes...
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size:  128 KByte (default)
------------------------------------------------------------
Calculando Jitter para 10.0.5.10 com 10 pacotes...
Calculando Perda de Pacotes para 10.0.5.10 com 10 pacotes...
Calculando Latência para 10.0.5.10 com 10 pacotes...
```

Fig. 4 - Execução Cliente

```
--- Lista de Conexões ---
-------------------------------------------------------------
| Número ||    IP     || Porta ||   Hora de Conexão
-------------------------------------------------------------
|   1    || 10.0.6.10 || 33326 || 2024-12-06 16:16:56
-------------------------------------------------------------
|   2    || 10.0.5.10 || 60467 || 2024-12-06 16:16:57
-------------------------------------------------------------
|   3    || 10.0.7.10 || 34138 || 2024-12-06 16:16:58
-------------------------------------------------------------
```

Fig. 3 - Conexões no Servidor

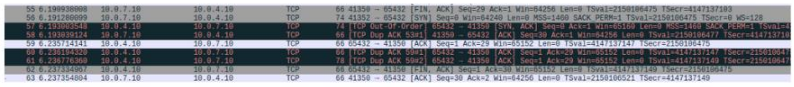


Fig. 7 - Retransmissões

```
2024-12-06 16:20:42
ALERT!!!: Packets in interface 'eth0': 2300

-------------------------
2024-12-06 16:20:42
CPU usage: 0.6%
RAM usage: 43.1%
Jitter: 1.058ms
Packet loss: 0%
Packets on eth0: 2300
Interface 'eth1' not found.
Interface 'eth2' not found.


-------------------------
2024-12-06 16:20:47
ALERT!!!: Packets in interface 'eth0': 2351

-------------------------
2024-12-06 16:20:47
ALERT!!!: CPU usage: 4.1%
```

Fig. 5 - Monitorizamento

```
------------------------------------------------------------
2024-12-06 16:41:29
Latência média para 10.0.5.10: 13.90 ms

------------------------------------------------------------
2024-12-06 16:41:30
Jitter para 10.0.6.10: 2.78 ms

------------------------------------------------------------
2024-12-06 16:41:31
Throughput de 10.0.5.10 para 10.0.6.10: 3.35 Mbits/sec

------------------------------------------------------------
2024-12-06 16:41:32
Latência média para 10.0.5.10: 39.26 ms

------------------------------------------------------------
2024-12-06 16:41:32
Perda de pacotes para 10.0.5.10: 0%
```

Fig. 6 - Tasks

8 Desafios Encontrados e Futuras Melhorias

Durante o desenvolvimento do sistema, enfrentamos diversos desafios técnicos e conceituais que exigiram adaptações e otimizações ao longo do processo. Além disso, identificamos oportunidades para melhorias futuras que podem ampliar a eficiência, escalabilidade e robustez do protocolo e do sistema como um todo. A seguir, destacamos os principais desafios enfrentados e as propostas de evolução:

8.1 Desafios Encontrados

1. Gerenciamento de Falhas de Comunicação: A ausência de ACK ou keep-alive em situações específicas exigiu a implementação de mecanismos robustos de tolerância a falhas, como retransmissões limitadas e monitoramento de inatividade. Configurar os tempos e os limites adequados foi uma tarefa que demandou testes extensivos.

2. Execução Simultânea de Tarefas e Métricas: A implementação de paralelismo por meio de threads trouxe desafios relacionados ao gerenciamento de recursos e à sincronização entre diferentes partes do sistema, principalmente em cenários de alta carga.

3. Otimização do Tamanho das Mensagens e Design do Protocolo: Encontrar o equilíbrio entre a quantidade de informações transmitidas e o tamanho das mensagens foi um ponto crítico, especialmente para evitar desperdício de largura de banda em redes com recursos limitados.

8.2 Futuras Melhorias

1. Criptografia e Segurança de Dados: Integrar um mecanismo de criptografia para as mensagens trocadas entre cliente e servidor, garantindo a confidencialidade e proteção contra ataques como intercepção e adulteração de pacotes.

2. Detecção e Recuperação Automática de Falhas: Desenvolver um sistema de recuperação automática para o cliente reconectar-se ao servidor após detectar uma falha, reduzindo o tempo de inatividade.

3. Adaptação Dinâmica dos Intervalos de Keep-Alive: Permitir que o intervalo de envio de mensagens keep-alive seja ajustado dinamicamente, baseado na estabilidade da rede e na carga do sistema.

4. Melhoramento do Protocolo: Desenvolver um protocolo com tamanho variável com envio do formato da mensagem.

Referências

1. Beazley, D. M., & Jones, B. K. (2013). *Python Cookbook: Recipes for Mastering Python 3*. O'Reilly Media.
2. Forouzan, B. A. (2017). *Data Communications and Networking* (5th ed.). McGraw-Hill Education.
3. Combs, G. (2004). *Wireshark Network Analysis: The Official Wireshark Certified Network Analyst Study Guide*. Riverbed Technology.