Relatório Técnico: Sistema de Captura e Análise de Tráfego de Rede com Previsão de Demanda

Integrantes:

- Vinícius von Glehn Severo | 2312130010
- Lucas Montalvão Ramires | 2322130043
- Stefisson Rafael Honorato da Silva Junior | 2312130093
- Rafael Rodrigues Barbosa | 2212130042
- Davi Mendes Paraíso Carvalho | 2212130041

1. Introdução

Este relatório descreve o desenvolvimento e a arquitetura de um sistema projetado para a captura, armazenamento, análise e previsão de tráfego de pacotes de rede. A solução opera em tempo real, coletando dados em intervalos de 5 segundos, persistindo-os em um banco de dados PostgreSQL e disponibilizando as informações por meio de uma interface web interativa. O sistema se destaca pela capacidade de realizar previsões de demanda de tráfego, oferecendo uma ferramenta robusta para monitoramento e análise de redes.

A arquitetura do projeto é baseada em um backend desenvolvido em Python, utilizando o microframework Flask em conjunto com as bibliotecas Flask-SocketIO e Scapy, e estruturado no padrão MVC (Model-View-Controller). O frontend foi implementado com o framework Angular, empregando a suíte de componentes PrimeNG para a construção de dashboards e gráficos interativos.

2. Arquitetura do Sistema

A solução foi concebida de forma modular, separando as responsabilidades de captura de dados, lógica de negócios, persistência e apresentação, conforme detalhado a seguir.

2.1. Backend – Arquitetura MVC e Captura de Dados

O backend foi estruturado seguindo o padrão arquitetural Model-View-Controller (MVC) para garantir a organização e a escalabilidade do código:

- **Model**: Camada responsável pela representação das entidades de dados e pela interação com o banco de dados PostgreSQL. A principal entidade, `TrafficLog`, abstrai a estrutura dos registros de tráfego.
- **View**: Implementada de forma desacoplada, por meio de rotas de API RESTful que serializam os dados em formato JSON, servindo como a fonte de informações para o frontend.
- **Controller**: Contém a lógica de negócios da aplicação. Os controladores (`traffic_controller`) orquestram as operações de agregação de pacotes, gerenciamento das rotas da API e processamento das solicitações.

O processo de captura de pacotes é executado em intervalos de 5 segundos pela biblioteca Scapy, monitorando os protocolos TCP, UDP, ICMP, ARP e IPv6. Os pacotes capturados são agregados em janelas de tempo pré-definidas, permitindo a extração de métricas essenciais, como o volume de pacotes por protocolo, o tráfego por endereço IP e outras estatísticas temporais. Para garantir a integridade dos dados em um ambiente multithread, foi implementado o uso de `threading.Lock`, prevenindo condições de corrida durante os processos de coleta e agregação.

2.2. Persistência de Dados

Os dados agregados são persistidos em um banco de dados PostgreSQL. A estrutura da base é gerenciada por meio de *migrations*, o que assegura a consistência do esquema de dados entre diferentes ambientes de desenvolvimento e produção. Cada registro armazena informações detalhadas, incluindo: *timestamp* da captura, endereços IP de origem e destino, protocolo de transporte, métricas de tráfego e dados derivados da janela de agregação.

2.3. Módulo de Previsão de Tráfego

O sistema incorpora um módulo de previsão de demanda de tráfego, que utiliza um modelo de Regressão Linear da biblioteca `scikit-learn`. Este módulo opera da seguinte forma:

- Analisa os dados históricos agregados por janelas de tempo para identificar padrões.
- Permite a geração de estimativas sobre o volume futuro de pacotes, tanto por protocolo quanto por endereço IP.
- Integra-se ao frontend por meio de rotas de API específicas, viabilizando a exibição de gráficos preditivos em tempo real.

2.4. Frontend Interativo

A interface do usuário foi desenvolvida em Angular e proporciona uma experiência de visualização rica e interativa:

- **Dashboards Dinâmicos**: Apresentam gráficos e tabelas (desenvolvidos com PrimeNG) que exibem os dados de tráfego de forma intuitiva.
- **Atualização em Tempo Real**: A comunicação bidirecional com o backend é garantida pelo protocolo WebSockets, gerenciado pela biblioteca Socket.IO, permitindo que os dashboards sejam atualizados sem a necessidade de recarregar a página.
- **Análise Personalizada**: O usuário dispõe de filtros para segmentar os dados por período, protocolo ou endereço IP, facilitando análises específicas e detalhadas.

A comunicação entre o frontend e o backend segue um contrato de API bem definido, utilizando tanto o padrão REST para requisições pontuais quanto o streaming de dados via WebSockets para atualizações contínuas.

3. Desafios Técnicos e Soluções Adotadas

O desenvolvimento do projeto apresentou desafios técnicos que exigiram soluções específicas:

- **Sincronização de Threads**: A concorrência entre as threads de captura e agregação foi gerenciada com a implementação de *locks* para evitar inconsistências e garantir a integridade dos dados.
- **Agregação em Janelas de Tempo**: A solução adotada foi o uso de *buffers* temporários em memória para agregar os pacotes, com atualizações periódicas no banco de dados para minimizar a perda de dados e a sobrecarga de I/O.
- **Precisão dos Modelos de Previsão**: Foi necessário um trabalho de pré-processamento e normalização dos dados históricos para adequá-los aos modelos de Machine Learning, garantindo previsões mais acuradas.
- **Visualização de Grandes Volumes de Dados**: Para evitar a sobrecarga do navegador, foram implementadas estratégias de paginação e agregação de dados no backend antes do envio para o frontend.
- **Integração Backend-Frontend**: A consistência na comunicação foi mantida através da definição de um contrato claro para as estruturas de dados (JSON) e os eventos de WebSockets.

4. Conclusão

O projeto resultou em um sistema completo e robusto para o monitoramento de rede, integrando com sucesso as funcionalidades de captura de pacotes, persistência de dados, análise preditiva e visualização interativa em tempo real. A adoção da arquitetura MVC no backend conferiu modularidade e manutenibilidade ao sistema, enquanto a combinação de Angular e PrimeNG no frontend proporcionou uma interface de usuário dinâmica, responsiva e de alto valor analítico. O sistema demonstra grande potencial para ser estendido com novos modelos de previsão e funcionalidades de análise avançada.