Relatório Técnico: Dashboard de Monitoramento de Tráfego de Servidor em Tempo Real

1. Introdução

O presente projeto teve como objetivo principal o desenvolvimento de um **Dashboard de Monitoramento de Tráfego de Rede em Tempo Real**. A solução foi projetada para capturar e analisar todo o tráfego de entrada e saída de um servidor-alvo específico, agregando os dados em janelas de tempo discretas de 5 segundos. O sistema implementa uma visualização interativa, permitindo a análise do volume de tráfego por cliente (endereço IP) e oferecendo a funcionalidade de *drill-down* para detalhar o uso de protocolos (HTTP, FTP, SSH, etc.).

Este relatório detalha a arquitetura, as tecnologias e a metodologia de desenvolvimento, com ênfase nos objetivos de aprendizagem e nos desafios técnicos superados.

2. Arquitetura do Sistema (Três Camadas)

A solução é dividida em três componentes principais, trabalhando de forma assíncrona para garantir a fluidez da coleta e da apresentação dos dados:

2.1. Camada de Captura e Processamento (Backend)

- **Responsabilidade:** Capturar pacotes brutos da interface de rede (porta espelhada ou loopback) e realizar a agregação de dados.
- **Lógica de Agregação:** Para cada pacote capturado, o sistema determina o cliente de origem, o sentido do tráfego (in ou out), e extrai o protocolo baseado nas portas de transporte (Ex: 80, 21, 22, 25, 53). Os dados são somados em uma estrutura de memória (a lista windows) que é rotacionada a cada 5 segundos.

2.2. Camada de API RESTful

- **Tecnologia: FastAPI** (Python) rodando via Uvicorn na porta **8000**.
- **Função:** Atua como a ponte de dados, expondo *endpoints* RESTful que o frontend consome.
    - /api/windows/latest: Retorna os dados da janela de 5 segundos mais recente.
    - /api/drilldown: Retorna a quebra de protocolos de um cliente específico para uma determinada janela (ativado pelo clique no gráfico).
- **Comunicação:** Utiliza **CORS Middleware** para permitir que o frontend (rodando em localhost:5173) possa acessar os dados do backend (localhost:8000), garantindo a segurança e a interoperabilidade.

2.3. Camada de Apresentação (Frontend)

- **Tecnologia: React** (Vite) estilizado com **Tailwind CSS**.
- **Função:** Apresentação dinâmica e interativa dos dados em gráficos. O frontend faz *polling* da API a cada 5 segundos para manter a visualização em tempo real.

3. Tecnologias Utilizadas

Componente	Tecnologia	Uso no Projeto
Captura	**Scapy** & **Npcap**	Captura e parsing de pacotes IP, TCP/UDP e protocolos de aplicação.
Backend/API	**FastAPI** e **Uvicorn**	Framework Python moderno para construir a API assíncrona e servidor de alta performance.
Frontend	**React**	Biblioteca JavaScript para construção da interface de usuário dinâmica.
Gráficos	**Chart.js** & **react-chartjs-2**	Renderização dos gráficos de barras (clientes, drill-down) e gráficos de rosca (resumo).
Estilo/UX	**Tailwind CSS**	Framework utilitário para design responsivo, profissional e moderno.

4. Implementação e Desafios Superados

O desenvolvimento exigiu a superação de desafios técnicos cruciais, garantindo a estabilidade e a fluidez do sistema:

4.1. Configuração e Estabilidade do Sniffer (Scapy/Npcap)

A Scapy, rodando em ambiente Windows, requer o driver **Npcap** instalado no modo de compatibilidade com WinPcap. A falha na instalação resultava em erros de RuntimeError, impedindo a captura de pacotes. A solução envolveu a validação da instalação do driver e a

execução do backend com privilégios adequados (sudo/administrador).

4.2. Gerenciamento de Conexão Frontend-Backend (Erros de Fetch)

O principal desafio de integração foi o erro TypeError: Failed to fetch, que surgia quando o frontend não conseguia estabelecer a conexão com a API. As causas e soluções foram:

1. **Conflito de Portas/Domínios:** Solucionado configurando o middleware **CORS** no FastAPI, permitindo requisições da porta do Vite (5173) para a porta do Uvicorn (8000).
2. **Falha de Inicialização:** Implementação de uma **lógica de retentativa (retry logic)** usando setTimeout nos *hooks* iniciais do React. Isso assegurou que o dashboard continuasse tentando conectar ao backend a cada 2 segundos até ter sucesso, aumentando a resiliência e fluidez da inicialização.

4.3. Gerenciamento de Estado e UX (Fluidez)

O objetivo de fluidez foi atingido através das seguintes práticas:

- **Hooks Otimizados:** Utilização de useEffect para gerenciar ciclos de vida da busca de dados (5s) e do relógio de UI (1s) separadamente.
- **Performance:** Uso de **useMemo** para calcular as estruturas de dados dos gráficos (KPIs, chartData, protocolsSummary) apenas quando o estado de latest data era atualizado, prevenindo recálculos desnecessários.
- **Design Otimizado:** Introdução de **KPIs** (Key Performance Indicators) no topo para fornecer contexto imediato e a função **formatBytes** para tornar todos os volumes de tráfego legíveis (ex: de bytes para KB, MB, etc.) nos gráficos e *tooltips*.

5. Estratégia de Testes

O sistema foi validado com sucesso nas seguintes etapas:

1. **Teste de Loopback (127.0.0.1):** Confirmou a funcionalidade da Scapy e da lógica de agregação do Python.
2. **Teste de Multi-Protocolo:** Configuração de serviços **HTTP (80), FTP (21), SSH (22), SMTP (25) e DNS (53)** no servidor-alvo (192.168.0.22).
3. **Simulação Multi-Cliente:** Utilização de múltiplas máquinas (ou VMs) para gerar tráfego concorrente, garantindo que o dashboard exibisse corretamente a separação de tráfego por diferentes endereços IP de origem.
4. **Validação do Drill Down:** Confirmação de que o clique em uma barra de IP abre o modal e apresenta a distribuição percentual do tráfego para os protocolos mapeados (HTTP, FTP, etc.).

6. Conclusão e Objetivos de Aprendizagem

O projeto resultou em um dashboard de monitoramento robusto e profissional, atendendo plenamente aos requisitos iniciais. Os principais objetivos de aprendizagem atingidos incluem:

- **Sistema de Captura de Pacotes:** Implementação e configuração bem-sucedida da

Scapy para monitoramento específico.

- **Estruturas de Dados e Agregação:** Desenvolvimento de lógica eficaz para janelas de tempo discretas (5s).
- **API RESTful:** Criação de uma API assíncrona com FastAPI para servir dados de série temporal de forma eficiente.
- **Interface Dinâmica e Interativa:** Construção de um frontend em React que utiliza Chart.js e Tailwind CSS para visualizações complexas (gráficos empilhados, KPIs, Donut Charts) e um *drill-down* funcional, demonstrando proficiência no **gerenciamento de estado** para uma experiência de usuário fluida e intuitiva.