Leonardo de Oliveira Maia

Melhorando a Usabilidade da Plataforma Peering: The BGP Testbed

Belo Horizonte
2025

Leonardo de Oliveira Maia

Melhorando a Usabilidade da Plataforma Peering: The BGP Testbed

Projeto de pesquisa apresentado ao Curso de Ciência da Computação da UFMG como requisito parcial para obtenção do grau de Bacharel.

Universidade Federal de Minas Gerais – UFMG
Departamento de Ciência da Computação – DCC

Orientador: Ítalo Fernando Scota Cunha

Belo Horizonte
2025

Resumo

Este trabalho apresenta o desenvolvimento de uma API Web integrada à plataforma Peering, com o objetivo de automatizar experimentos de roteamento BGP por meio do uso de prefixos públicos previamente alocados. A solução permite que usuários autorizados interajam com a infraestrutura da plataforma via interface RESTful, com controle de acesso, escalonamento de tarefas e execução remota de experimentos por meio de containers Docker. Testes realizados em ambiente de desenvolvimento validaram o funcionamento completo da solução, que visa aumentar a acessibilidade, agilidade e escalabilidade da plataforma.

Palavras-chave: BGP. Peering. Automação. API.

Abstract

This work presents the development of a Web API integrated into the PEERING platform, aiming to automate BGP routing experiments using pre-allocated public IP prefixes. The solution allows authorized users to interact with the platform's infrastructure through a RESTful interface, with access control, task scheduling, and remote execution of experiments using Docker containers. Tests carried out in a development environment validated the full functionality of the solution, which enhances the platform's accessibility, agility, and scalability for network experimentation.

Keywords: BGP. Peering. Automation. API.

Lista de ilustrações

Figura 1 – Diagrama conceitual do cliente tradicional 15
Figura 2 – Diagrama conceitual do cliente gerenciado pelo Peering 15
Figura 3 – Interface Rest API 16
Figura 4 – Containers gerenciados do Peering 20
Figura 5 – Interface Bird Looking Glass 21

Sumário

1 INTRODUÇÃO 7
1.1 Caracterização do Problema 7
1.2 Motivação 7
1.3 Objetivos 7
1.4 Estrutura do Trabalho 7

2 REFERENCIAL TEÓRICO 9

3 FERRAMENTAS UTILIZADAS 11
3.1 BIRD 11
3.2 BIRD-LG-Go 11
3.3 OpenVPN 11
3.4 Django 12
3.5 Docker 12
3.6 Vagrant 12

4 PEERING 13

5 METODOLOGIA 14
5.1 Funcionamento do Cliente Tradicional da Plataforma Peering 14
5.2 Client Gerenciado pela Plataforma Peering 14
5.2.1 Controle de Acesso via API 15
5.2.2 Criação do Client Gerenciado 15
5.2.3 Formas de Acesso pelo Usuário 16
5.2.4 Fila de Execução e Política de Escalonamento 16
5.2.4.1 Política de Fairness 17
5.2.4.2 Política de Backfill 18
5.2.5 Visualizador do Estado do Escalonador 19
5.2.6 Arquitetura Interna da Plataforma 20
5.3 Monitoramento das Sessões BGP com BIRD-LG-Go 20

6 RESULTADOS 22
6.1 Testes do `apiclient` 22
6.2 Testes do Escalonador e Fila de Prioridade 23
6.3 Testes do Módulo BIRD-LG-Go 23

7 CONCLUSÃO 24

8 PERSPECTIVAS FUTURAS 25

REFERÊNCIAS 26

1 Introdução

A crescente complexidade da Internet e, em especial, da infraestrutura de roteamento interdomínios, torna essencial a existência de plataformas que permitam a experimentação realista e controlada com o protocolo BGP (Border Gateway Protocol) (REKHTER; LI; HARES, 2006). Nesse contexto, a plataforma PEERING[1] (SCHLINKER et al., 2014; SCHLINKER et al., 2019) se destaca como uma ferramenta amplamente utilizada por pesquisadores, engenheiros e profissionais da área de redes para testar anúncios de prefixos reais em uma infraestrutura distribuída, conectada a provedores de trânsito globais.

1.1 Caracterização do Problema

Atualmente, a utilização da plataforma PEERING segue um fluxo consolidado, baseado no envio de propostas formais por parte dos usuários (SCHLINKER et al., 2019). Essas propostas são avaliadas por um comitê técnico, e, uma vez aprovadas, os prefixos necessários são alocados manualmente, com a supervisão da equipe responsável pela infraestrutura. Esse processo, embora seguro e funcional, apresenta limitações em termos de agilidade e escalabilidade, especialmente quando se trata de experimentos mais simples, automatizados ou de curta duração, que poderiam ser realizados com menor supervisão.

1.2 Motivação

Nesse cenário, surge a necessidade de modernizar o processo de experimentação, tornando-o mais acessível e dinâmico, sem comprometer a segurança e a estabilidade da plataforma.

1.3 Objetivos

O objetivo deste projeto é desenvolver e integrar uma API Web à plataforma PEERING, permitindo que usuários autorizados realizem experimentos com prefixos públicos de maneira automatizada.

1.4 Estrutura do Trabalho

O desenvolvimento deste trabalho foi conduzido de forma incremental, com as funcionalidades sendo implementadas à medida que as demandas surgiam. As decisões de

[1] <https://peering.ee.columbia.edu/>

projeto foram discutidas em reuniões online e, eventualmente, presenciais com o orientador, permitindo ajustes dinâmicos e alinhamento contínuo com os objetivos definidos.

A entrega principal foi a modernização da plataforma PEERING por meio da criação de uma API Web, desenvolvida com o framework Django, que automatiza a realização de experimentos com prefixos públicos por usuários autorizados. Essa solução visa eliminar a necessidade de solicitações formais ao comitê técnico para experimentos simples e de curta duração.

Além da API em si, o trabalho envolveu:

- A integração da solução aos servidores distribuídos da plataforma, viabilizando experimentação em múltiplas regiões geográficas.
- O desenvolvimento de um sistema de filas para gerenciamento eficiente, seguro e ordenado das requisições.
- A criação de uma interface, que permite aos usuários interagir com a API de forma simples e intuitiva.

2 Referencial Teórico

Com a crescente demanda por automação em sistemas computacionais, as APIs (Application Programming Interfaces) tornaram-se componentes centrais no desenvolvimento de plataformas modernas. Elas permitem que processos tradicionalmente manuais ou mediados por operadores humanos sejam expostos por meio de interfaces programáveis, promovendo ganhos significativos em eficiência, escalabilidade e reprodutibilidade.

Esse paradigma é amplamente adotado em áreas onde a agilidade e o controle sobre recursos são essenciais, como no gerenciamento de infraestrutura de rede, serviços em nuvem e ambientes de testes automatizados. As APIs viabilizam integrações com scripts, aplicações e sistemas externos, eliminando a necessidade de interação direta com interfaces gráficas ou solicitações manuais.

Para exemplificar a aplicação desse modelo em diferentes escopos de infraestrutura e serviços de rede, destacam-se:

- **Cloudflare API (Cloudflare, 2025)**: Permite o gerenciamento programático de zonas DNS, certificados SSL, políticas de firewall e outros recursos de rede. Usuários podem, por exemplo, automatizar a criação de domínios, atualizar regras de segurança ou monitorar tráfego de forma integrada com suas aplicações.

- **AWS EC2 API (Amazon Web Services) (Amazon Web Services, 2025)**: Um caso clássico de provisionamento de infraestrutura sob demanda. Por meio de chamadas REST, é possível instanciar máquinas virtuais, configurar redes privadas, volumes de armazenamento e políticas de segurança, o que permite a automação de tarefas que antes exigiam operações manuais via painel de controle.

- **RIPE Atlas API (KISTELEKI; ABEN, 2014)**: Embora voltada para o contexto de pesquisa e mensuração, essa API oferece acesso programático a sondas e medições ativas de rede, como *traceroutes* e *pings*. Usuários podem requisitar medições distribuídas e acompanhar os resultados em tempo real. Apesar de não atuar diretamente com anúncios BGP, serve como exemplo de como APIs podem facilitar experimentos em escala na infraestrutura da Internet.

As soluções citadas demonstram que a adoção de APIs pode transformar significativamente o modo como usuários interagem com sistemas complexos. No entanto, há um espaço ainda pouco explorado no que se refere à automação de experimentos BGP com anúncios de prefixos reais. A maioria das plataformas voltadas a esse tipo de experimento, como a própria Peering ou o GÉANT Testbed Service (GÉANT, 2025), ainda depende de fluxos manuais ou semi-automatizados para a aprovação e alocação de recursos.

Nesse contexto, o presente projeto se diferencia ao propor a construção de uma API voltada especificamente à automação do uso da plataforma PEERING, oferecendo um ponto de entrada padronizado, seguro e escalável para a configuração e execução de experimentos com prefixos públicos. A proposta visa eliminar a necessidade de envio de propostas formais para experimentos simples, ao mesmo tempo em que mantém o controle sobre o uso dos recursos por meio de autenticação, filas e monitoramento em tempo real (CASEY; HIGGINBOTHAM, 2016).

3 Ferramentas Utilizadas

Nas seções abaixo será explicado em mais detalhes as ferramentas utilizadas neste trabalho.

3.1 BIRD

Para a emulação dos roteadores virtuais e a execução dos protocolos de roteamento, utiliza-se o BIRD (*BIRD Internet Routing Daemon*) (BIRD Internet Routing Daemon Team, 2025). Este *daemon* é responsável por gerenciar as tabelas de roteamento do *kernel* do Linux e suporta protocolos essenciais para a infraestrutura da Internet, com destaque para o BGP (*Border Gateway Protocol*).

Sendo o BGP o protocolo padrão para o roteamento entre domínios e troca de tráfego entre Sistemas Autônomos, sua implementação através do BIRD na plataforma PEERING permite um ambiente de experimentação fiel. Isso possibilita que os usuários configurem sessões BGP reais, anunciem prefixos e observem o comportamento da rede de forma prática e controlada.

3.2 BIRD-LG-Go

O *BIRD-LG-Go* (XDDXDD, 2023) é uma ferramenta de *Looking Glass* moderna desenvolvida em Go, projetada para interagir diretamente com instâncias do daemon BIRD. Ela fornece uma interface web eficiente para consulta de informações de roteamento, como sessões BGP, tabelas de rotas, anúncios e estados de vizinhança, permitindo a inspeção em tempo real da operação dos roteadores. No contexto deste projeto, o *BIRD-LG-Go* foi implantado em todos os servidores que compõem a infraestrutura do PEERING, com o objetivo de coletar e disponibilizar informações detalhadas sobre as conexões BGP estabelecidas. Essa integração possibilitou que usuários da plataforma acessassem, por meio do site oficial do PEERING, dados atualizados sobre o estado das sessões e o comportamento das rotas, tornando o ambiente mais transparente e interativo para análise e acompanhamento dos experimentos.

3.3 OpenVPN

Para a conectividade segura entre a infraestrutura local do pesquisador e a rede distribuída do PEERING, utiliza-se o **OpenVPN** (OpenVPN Inc., 2025). Esta ferramenta

de código aberto implementa redes privadas virtuais (VPN) baseadas em SSL/TLS, garantindo autenticação robusta e criptografia de dados.

No contexto do cliente oficial da plataforma Peering (PEERING Testbed, 2024), o OpenVPN é responsável por estabelecer túneis autenticados por certificados digitais diretamente com os roteadores de borda (multiplexadores). É sobre essa infraestrutura de túneis que as sessões BGP são estabelecidas, permitindo que o roteador virtual local troque tráfego e anúncios de rotas com a rede global de forma transparente, como se estivesse fisicamente conectado ao ponto de presença (PoP).

3.4 Django

O Django (Django Software Foundation, 2025) é um framework web de alto nível, escrito em Python, que permite o desenvolvimento rápido de aplicações robustas, seguras e escaláveis. Para facilitar a criação e o consumo da API, foi utilizado o *Django REST Framework* (CHRISTIE, 2024), uma biblioteca poderosa que estende o Django com ferramentas específicas para construção de APIs Web.

3.5 Docker

Docker (Docker, Inc., 2025) é uma plataforma de virtualização baseada em containers que permite empacotar aplicações com todas as suas dependências, garantindo portabilidade, isolamento e reprodutibilidade. No contexto deste projeto, Docker foi utilizado para criar o container `apiclient`, responsável por simular o comportamento de um cliente tradicional da plataforma Peering.

3.6 Vagrant

O Vagrant (HashiCorp, 2025) é uma ferramenta de automação de ambientes virtuais, voltada para a criação e configuração reprodutível de máquinas virtuais. Neste projeto, Vagrant foi utilizado para simular o ecossistema completo da plataforma Peering, incluindo os servidores do site, cliente e roteadores upstream. Esse ambiente virtualizado permitiu realizar testes isolados e controlados da API, garantindo que as funcionalidades implementadas funcionassem corretamente antes de serem integradas ao ambiente de desenvolvimento oficial. Além disso, o uso do Vagrant facilitou a automação dos testes com múltiplos experimentos e sessões BGP.

4 Peering

O Peering é um sistema que oferece acesso seguro e fácil para pesquisadores e educadores ao sistema de roteamento BGP da Internet, possibilitando e inspirando pesquisas transformadoras. O sistema opera servidores físicos e máquinas virtuais que fornecem múltiplos experimentos paralelos com controle e visibilidade equivalentes à operação direta de um *autonomous system* (AS), uma rede ou grupo de redes muito grande com uma única política de roteamento. Assim, o Peering, fazendo uso de *BGP virtualization stack* (vBGP), design para virtualização dos planos de dados e controle de um roteador de borda BGP, simultaneamente com a imposição de políticas de segurança para evitar que os experimentos perturbem a Internet e uns aos outros, seus experimentos operam em um ambiente qualitativamente similar ao de um provedor de nuvem, e podem trocar rotas e tráfego com centenas de redes vizinhas e a Internet mais ampla em locais ao redor do mundo.

O Peering se conecta (via BGP) a redes reais em universidades e pontos de troca de Internet em todo o mundo. O Peering possui roteadores em universidades, instituições de pesquisa, *Internet Exchange Point* IXPs e no provedor de nuvem Vultr. Em vez de serem observadores do ecossistema da Internet, os pesquisadores tornam-se participantes, realizando experimentos que anunciam e selecionam rotas, enviam e recebem tráfego diretamente com essas redes.

5 Metodologia

5.1 Funcionamento do Cliente Tradicional da Plataforma Peering

Inicialmente, foi analisado como opera o cliente tradicional da plataforma Peering. Na prática atual, o usuário interessado em realizar experimentos na plataforma deve, primeiramente, se cadastrar por meio do portal oficial. Após o cadastro, o usuário deve preencher um formulario de submissão, disponibilizado pelo Peering, descrevendo os detalhes do experimento, incluindo o tipo de experimento que será realizado e os prefixos IP públicos que deseja anunciar durante os testes.

O comitê técnico da plataforma avalia a solicitação. Se houver inconsistências, o formulário é devolvido para revisão. Caso esteja tudo correto, o experimento é autorizado e são geradas as credenciais (certificados `.crt` e `.key`) necessárias para estabelecer uma conexão VPN segura com a infraestrutura da plataforma. Além disso, os prefixos IP solicitados são alocados ao usuário, permitindo o início do experimento com os anúncios BGP devidamente autorizados.

Após essa etapa, o usuário deve acessar o repositório público da plataforma Peering chamado `Client`, clonar o projeto e instalar as dependências necessárias em sua máquina local. Em seguida, ele configura os certificados recebidos e os prefixos IP alocados no arquivo de configuração do client. Com o ambiente local preparado, o client estabelece uma conexão VPN segura com a infraestrutura do Peering, utilizando um container `openvpn` presente nos roteadores da plataforma. Essa conexão permite a criação de uma sessão BGP entre a máquina do usuário e os roteadores do Peering. Esses roteadores executam o daemon BIRD em um container separado, com sessões BGP já estabelecidas com outras redes na Internet, como demonstra na figura 1. Assim, os prefixos anunciados pelo usuário são propagados de forma segura e controlada a partir da infraestrutura do Peering.

5.2 Client Gerenciado pela Plataforma Peering

O funcionamento do client gerenciado da plataforma pode ser visualizado de forma resumida na Figura 2. Nela, um usuário já autorizado no grupo `api_access` realiza uma requisição de experimento via API. Essa requisição é recebida pelo sistema e encaminhada para um escalonador, que controla a ordem de execução das tarefas. Quando os recursos estiverem disponíveis, o experimento é enviado ao container `apiclient`, que executa o anúncio BGP por meio de uma conexão VPN com os roteadores do Peering, propagando os prefixos IP solicitados até a Internet.

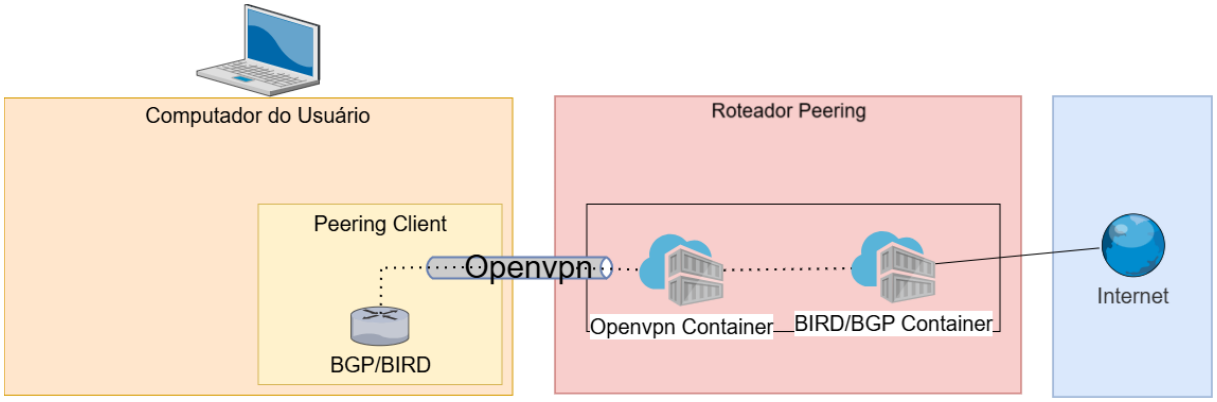

Figura 1 – Diagrama conceitual do cliente tradicional

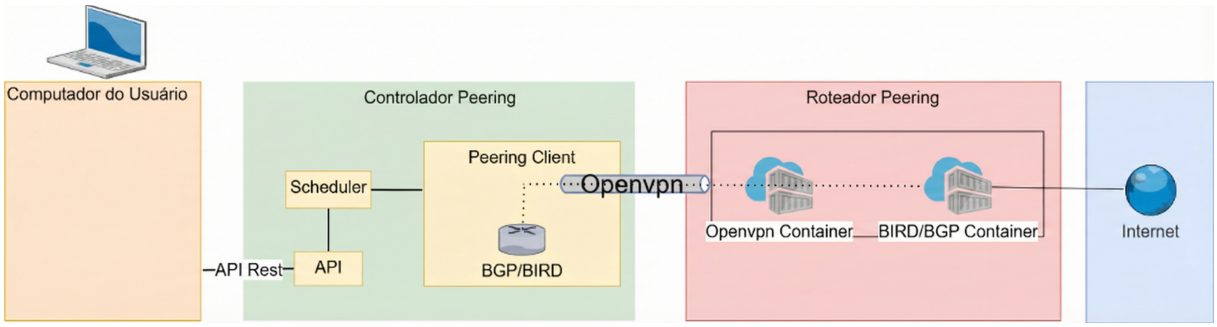

Figura 2 – Diagrama conceitual do cliente gerenciado pelo Peering

5.2.1 Controle de Acesso via API

Semelhante ao processo de uso do client tradicional, o usuário deve primeiramente se cadastrar no portal da plataforma. Após o cadastro, para utilizar o client gerenciado, ele precisa ser incluído manualmente em um grupo interno chamado `api_access`. Usuários nesse grupo recebem permissão para acessar a API da plataforma e têm um token exclusivo gerado automaticamente, visível em seu dashboard. Esse token pode ser utilizado em scripts para realizar requisições programáticas à API.

5.2.2 Criação do Client Gerenciado

Para permitir o funcionamento do client gerenciado, foi criado um usuário especial chamado `restapi`, com credenciais atualizadas para estabelecer sessões BGP com os roteadores da plataforma PEERING. A esse usuário são atribuídos prefixos IP públicos pré-alocados, que ficam disponíveis para qualquer membro do grupo `api_access`. Com isso, foi implementado um novo container, chamado `apiclient`, que executa um client tradicional do PEERING com todas as dependências instaladas, configurado com as credenciais do usuário `restapi` e com sessões BGP já estabelecidas.

5.2.3 Formas de Acesso pelo Usuário

O usuário com acesso à API pode utilizar o token gerado para realizar requisições via script, passando os parâmetros desejados para o experimento, ou pode utilizar uma interface gráfica desenvolvida com Django REST Framework. Essa interface, ilustrada na Figura 3, exibe parâmetros exigidos para a execução de um experimento e facilita o uso para usuários menos familiarizados com automações.

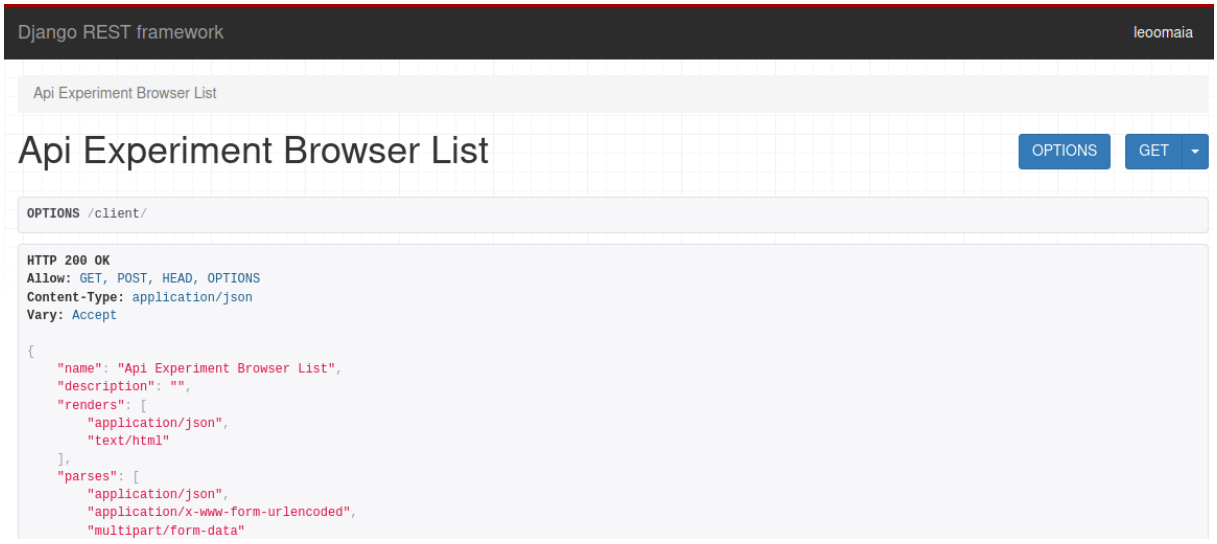

Figura 3 – Interface Rest API

5.2.4 Fila de Execução e Política de Escalonamento

Ao enviar uma requisição via API, o experimento do usuário é inserido em uma fila de execução compartilhada entre todos os participantes do grupo `api_access`. Essa fila é gerenciada por um escalonador implementado dentro do container Django, responsável por controlar a ordem de execução dos experimentos e a alocação dos recursos disponíveis.

O escalonador permite que cada usuário especifique manualmente os prefixos reais desejados ou utilize identificadores especiais (`A`, `B`, `C`, `D`). Durante a execução, cada identificador especial é automaticamente associado a um prefixo real disponível, garantindo flexibilidade na configuração dos experimentos e evitando conflitos de uso de recursos entre diferentes usuários.

O funcionamento do escalonador é estruturado em ciclos de execução, denominados *rodadas*, com duração de 90 minutos. Ao final de cada rodada, o sistema considera que uma iteração completa de experimentos foi realizada. Em seguida, o escalonador reavalia todos os experimentos pendentes e recalcula a fila de prioridade, determinando a nova ordem de execução com base em critérios de justiça (*fairness*) e disponibilidade de recursos.

A política de escalonamento adotada combina duas abordagens principais: *fairness* e *backfill*. Essa combinação busca garantir o uso equilibrado dos recursos pelos usuários e, ao mesmo tempo, otimizar a taxa de ocupação do sistema.

5.2.4.1 Política de Fairness

A política de *fairness* tem como objetivo assegurar que os recursos da plataforma sejam distribuídos de maneira justa entre os usuários, evitando que um único participante monopolize a infraestrutura. Para isso, a prioridade de cada experimento é calculada a partir de três fatores principais:

- **Histórico de uso:** quantidade de experimentos executados pelo usuário nos últimos sete dias;
- **Tempo estimado de execução:** experimentos com duração mais curta tendem a receber maior prioridade;
- **Tempo de espera:** quanto maior o tempo em fila, maior a prioridade atribuída ao experimento.

Esses fatores são ponderados por meio de uma fórmula que reflete diretamente o funcionamento do escalonador. Para cada experimento i, definimos inicialmente o custo de recurso:

$$\text{resource\_cost}_i = P_i \times R_i,$$

onde P_i representa o número de prefixos utilizados e R_i o número de rounds solicitados.

O histórico de uso de um usuário u dentro da janela dos últimos D dias é definido como:

$$\text{user\_cost}(u) = \sum_{i \in \mathcal{E}(u)} \text{resource\_cost}_i,$$

considerando apenas experimentos em estado *completed*, *round_ongoing* ou *changing_configuration*.

O tempo de espera do experimento é convertido em rounds de espera:

$$\text{waiting\_rounds} = \frac{t_{\text{agora}} - t_{\text{request}}}{T_{\text{round}}},$$

onde T_{round} corresponde à duração de um round (em segundos).

A prioridade final utilizada pelo escalonador é expressa como uma chave ordenável:

$$\text{priority\_key} = \left(\text{user\_cost}(u),\ -\text{waiting\_rounds}\right),$$

de forma que o escalonador sempre favorece usuários com menor custo histórico acumulado; em caso de empate, o desempate ocorre priorizando experimentos que estão há mais tempo na fila.

5.2.4.2 Política de Backfill

Após o cálculo da fila de prioridade, o escalonador aplica a política de *backfill*, cujo objetivo é maximizar a utilização dos recursos disponíveis, permitindo a execução de experimentos menores enquanto outros aguardam a liberação de prefixos específicos. O processo segue as seguintes etapas:

1. O primeiro experimento da fila tenta alocar todos os recursos solicitados (prefixos reais e/ou especiais). Caso os recursos estejam disponíveis, a execução é iniciada imediatamente.

2. Se algum recurso solicitado estiver ocupado, o escalonador reserva os recursos disponíveis e calcula o tempo de espera até que os recursos restantes sejam liberados. O cálculo ocorre em duas etapas:

    a) Para cada prefixo real solicitado que esteja em uso, é determinado o número de rodadas restantes até a liberação do prefixo com maior tempo de ocupação;

    b) Caso o experimento também dependa de prefixos especiais (ou seja, qualquer prefixo real disponível), o escalonador ordena os prefixos atualmente em uso de acordo com o tempo restante para liberação e reserva os prefixos com tempo de liberação mais tardios antecedentes ao tempo de liberação dos prefixos reais solicitados (calculado no item (a) acima); caso prefixos suficientes não sejam liberados antes do tempo de liberação dos prefixos reais solicitados (calculado no item (a)), o escalonador reserva os próximos prefixos liberados;

    c) O máximo entre os instantes de liberação calculados nos itens (a) e (b) determina o tempo de início do experimento.

3. Em seguida, os demais experimentos da fila tentam alocar recursos:

    - Se houver recursos disponíveis e não houver conflito com os recursos reservados pelo primeiro experimento, a execução é iniciada imediatamente;
    - Se houver conflito, o experimento poderá ser executado apenas se sua duração estimada for menor ou igual ao tempo de espera calculado para o experimento prioritário; caso contrário, permanece na fila.

Essa estratégia de escalonamento garante uma distribuição justa dos recursos e, ao mesmo tempo, aumenta a eficiência geral do sistema, permitindo que experimentos menores e independentes sejam executados enquanto outros aguardam recursos específicos. Com

isso, o tempo ocioso da infraestrutura é minimizado e a utilização global da plataforma torna-se mais equilibrada e previsível.

5.2.5 Visualizador do Estado do Escalonador

Com o objetivo de facilitar o monitoramento do sistema e fornecer transparência sobre o funcionamento interno do escalonador, foi desenvolvido um visualizador que disponibiliza, em tempo real, informações sobre o estado atual das filas de execução e dos experimentos. Essa ferramenta possibilita tanto aos administradores quanto aos usuários autorizados acompanhar o andamento dos experimentos submetidos e compreender a dinâmica de alocação de recursos dentro da plataforma.

O visualizador apresenta um resumo estruturado dos experimentos organizados em quatro categorias principais:

- **Fila pessoal (`my_queue`):** exibe os experimentos atualmente enfileirados pertencentes ao usuário autenticado, permitindo que ele verifique o status de submissões ainda não iniciadas e o momento em que foram solicitadas.
- **Execuções em andamento (`ongoing`):** mostra todos os experimentos que estão em execução no momento, independentemente do usuário responsável. Para cada experimento, são exibidas informações como o número total de rodadas, a rodada atual e os horários de início de cada uma delas.
- **Últimos experimentos concluídos (`completed_latest`):** apresenta um histórico resumido das cinco execuções mais recentes concluídas no sistema, fornecendo uma visão geral da atividade recente da plataforma.
- **Fila de maior prioridade (`top_priority_queue`):** lista os cinco experimentos com maior prioridade de execução entre todos os usuários autorizados da API. Essa visão global permite compreender o comportamento do algoritmo de escalonamento e verificar quais experimentos estão mais próximos de serem executados.

Essas informações são disponibilizadas tanto em formato de API (para integração com aplicações externas) quanto em uma interface navegável voltada para visualização direta no navegador. A estrutura foi projetada para operar sem efeitos colaterais, ou seja, as consultas realizadas ao visualizador não interferem no funcionamento do escalonador nem modificam o estado interno dos experimentos.

Com isso, o visualizador desempenha um papel importante na transparência e na auditabilidade do sistema, permitindo acompanhar a utilização dos recursos da plataforma e oferecendo subsídios para análise de desempenho e comportamento do escalonador ao longo do tempo.

5.2.6 Arquitetura Interna da Plataforma

A arquitetura da plataforma passou a contar com dois containers principais, conforme ilustrado na Figura 4:

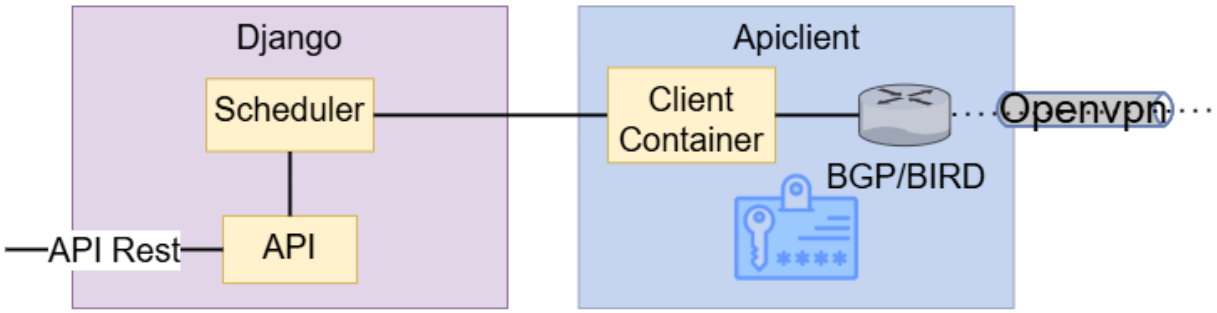

Figura 4 – Containers gerenciados do Peering

- **Container Django:** Responsável por servir o website da plataforma, incluindo a API e o módulo de escalonamento dos experimentos.
- **Container Apiclient:** Executa o client tradicional do PEERING com as credenciais do usuário `restapi`, mantendo sessões BGP ativas com os roteadores da plataforma. Esse container também contém o script responsável por executar os experimentos agendados pelo escalonador.

A combinação dessa arquitetura com controle de acesso e política de uso restritivo garante uma utilização equilibrada dos recursos compartilhados, mesmo com uma base limitada de prefixos disponíveis para o client gerenciado.

5.3 Monitoramento das Sessões BGP com BIRD-LG-Go

Com o objetivo de ampliar a visibilidade sobre o funcionamento das sessões BGP estabelecidas pelos experimentos, foi implementado um módulo de monitoramento utilizando a ferramenta *BIRD-LG-Go*. Essa ferramenta fornece uma interface web moderna para consulta de informações do daemon BIRD, permitindo visualizar de forma clara os estados das conexões BGP, os prefixos anunciados e as tabelas de roteamento ativas.

A implementação integra o monitoramento diretamente às sessões BGP criadas dinamicamente pelo `apiclient` durante a execução dos experimentos. Assim, em vez de depender

de conexões pré-existentes, o módulo utiliza exatamente as sessões que o `apiclient` estabelece com os roteadores do PEERING para ativar os anúncios e coletar métricas em tempo real.

Para isso, foram configurados dois containers principais na instância central do PEERING: um dedicado ao monitoramento das sessões BGP IPv4 e outro voltado às sessões IPv6. Ambos se comunicam com os roteadores através da mesma rede interna usada pelos experimentos, coletando periodicamente informações atualizadas sobre o estado das sessões, seus prefixos e atributos associados.

Do lado dos servidores do PEERING, também foram implantados dois containers adicionais por servidor—um para BGP IPv4 e outro para BGP IPv6 de modo a permitir a coleta distribuída das informações diretamente em cada ponto de conexão. Essa arquitetura garante que os dados de monitoramento reflitam com precisão o estado individual de cada roteador e suas respectivas sessões BGP.

O *BIRD-LG-Go* disponibiliza imagens de container prontas para uso, o que simplificou o processo de implantação e integração com a infraestrutura já existente. Foram configuradas interfaces web independentes para o monitoramento das sessões BGP IPv4 e IPv6, permitindo a visualização separada das informações de cada protocolo. Cada interface exibe detalhes sobre os vizinhos BGP ativos, as rotas anunciadas e recebidas, além de estatísticas de estabilidade e tempo de atividade das sessões. A Figura 5 ilustra a interface correspondente ao monitoramento das sessões IPv4 em um dos servidores administrados pela plataforma PEERING, apresentando a visualização consolidada dos dados coletados.

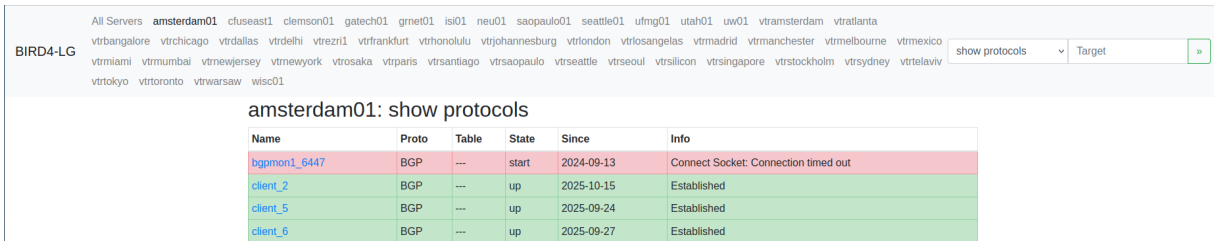

Figura 5 – Interface Bird Looking Glass

6 Resultados

Durante o desenvolvimento da API para experimentação automatizada na plataforma PEERING, foram realizados testes automatizados utilizando a biblioteca `<rest_framework.test.APITestCase>`, com o objetivo de validar o funcionamento completo do sistema, desde o controle de acesso até a alocação de recursos e integração com os componentes internos.

Os testes foram executados no ambiente de desenvolvimento oficial da plataforma (`monitor.peering.ee.columbia.edu`), que replica a estrutura do sistema de produção, garantindo validação realista. Instâncias de usuários foram criadas e adicionadas ao grupo `api_access`, garantindo que apenas usuários autorizados pudessem acessar a API. Cenários simulados incluíram múltiplos roteadores, uma organização fictícia, prefixos IP públicos alocados e experimentos aprovados, todos gerados dinamicamente em memória com a biblioteca `model_mommy`.

6.1 Testes do `apiclient`

Foram realizados testes específicos para o client gerenciado da plataforma:

- Autenticação com JWT: validação de usuários ativos e rejeição de usuários inativos;
- Reconhecimento e associação correta de prefixos alocados aos experimentos;
- Processamento da fila de execução com múltiplos experimentos simultâneos;
- Execução de anúncios BGP pelos containers `apiclient` utilizando os prefixos públicos disponíveis.

Complementarmente, foi criado um ambiente de simulação com `Vagrant`, replicando website, servidor, client e roteador *upstream*. Nesse ambiente, foram realizados:

- **Teste de conexão BGP:** acionamento do `apiclient` e verificação do estado `Established` para assegurar a correta criação da sessão BGP;
- **Testes funcionais de experimentos:** submissão sequencial de dois experimentos via API, cada um com múltiplas rodadas de ativação e desativação de anúncios BGP, confirmando a propagação correta dos prefixos no container cliente e no roteador *upstream*.

6.2 Testes do Escalonador e Fila de Prioridade

Para validar o comportamento do escalonador, foram criados experimentos fictícios com diferentes tempos de execução, quantidade de prefixos e histórico de uso, permitindo verificar a política combinada de *fairness* e *backfill*. Observou-se que:

- A prioridade dos experimentos refletiu corretamente o histórico de uso e o tempo de espera na fila;
- O recalculo da fila a cada rodada de 90 minutos ajustou dinamicamente a ordem de execução conforme novos experimentos e recursos liberados;
- O *backfill* aproveitou lacunas na utilização de recursos, executando experimentos menores enquanto outros aguardavam a liberação de prefixos;

6.3 Testes do Módulo BIRD-LG-Go

Foram realizados testes no módulo de visualização baseado na ferramenta *BIRD-LG-Go*, implantado nos servidores do PEERING, com containers independentes para BGP IPv4 e IPv6. Os testes demonstraram que:

- Conexões BGP estabelecidas pelos clientes da API são refletidas corretamente nas interfaces de monitoramento;
- Informações de vizinhança, rotas anunciadas e recebidas são coletadas continuamente pelos containers;
- Interfaces web IPv4 e IPv6 exibem dados atualizados de cada servidor administrado pela plataforma.

De forma geral, os testes demonstraram que a plataforma processa experimentos de maneira automatizada, segura e confiável, tanto no ambiente oficial quanto em ambientes isolados, garantindo robustez, escalonamento eficiente e correta propagação dos anúncios BGP.

7 Conclusão

Este trabalho teve como objetivo modernizar o processo de experimentação na plataforma PEERING, por meio do desenvolvimento de uma API Web que permite a execução automatizada de anúncios BGP. Foi implementada uma arquitetura baseada em containers, incluindo um client gerenciado com sessões BGP ativas e um escalonador capaz de distribuir os recursos de forma justa entre múltiplos usuários.

Os testes realizados, tanto no ambiente oficial de desenvolvimento (<monitor.peering.ee.columbia.edu>) quanto em uma infraestrutura simulada com Vagrant, demonstraram que a solução é funcional, segura e eficiente, garantindo a correta propagação dos prefixos e o gerenciamento adequado da fila de execução.

A proposta oferece uma alternativa ao fluxo tradicional da plataforma, tornando a experimentação mais ágil, transparente e acessível, especialmente em cenários que não exigem supervisão manual do comitê técnico.

8 Perspectivas Futuras

Como continuação deste trabalho, está prevista a implementação de um novo modelo de alocação de recursos que permitirá aos usuários utilizar seus próprios prefixos IP exclusivos, previamente aprovados e registrados pela plataforma. Essa evolução visa ampliar a flexibilidade de uso do *apiclient*, permitindo que esses prefixos sejam empregados em conjunto com os prefixos públicos já disponíveis, de forma a otimizar a utilização dos recursos da infraestrutura.

Além disso, o sistema permitirá que os usuários executem experimentos utilizando apenas seus prefixos exclusivos, tornando o processo de configuração mais intuitivo e proporcionando uma melhor integração com a interface da plataforma. Essa melhoria tem como objetivo aumentar a autonomia dos usuários e facilitar a condução de experimentos personalizados, mantendo a segurança e o controle da infraestrutura do PEERING.

Referências

Amazon Web Services. *Amazon EC2 API Reference*. 2025. <https://docs.aws.amazon.com/AWSEC2/latest/APIReference/>. Acessado em: 22 abr. 2025.

BIRD Internet Routing Daemon Team. *BIRD: The Internet Routing Daemon*. 2025. <https://bird.network.cz/>. Acessado em: 23 jun. 2025.

CASEY, K.; HIGGINBOTHAM, J. *A Practical Approach to API Design: From Principles to Practice*. Leanpub, 2016. Disponível em: <https://leanpub.com/practical-api-design>.

CHRISTIE, T. *Django REST Framework*. 2024. <https://www.django-rest-framework.org/>. Acessado em: 22 abr. 2025.

Cloudflare. *Cloudflare API Documentation*. 2025. <https://developers.cloudflare.com/api/>. Acessado em: 22 abr. 2025.

Django Software Foundation. *The Web Framework for Perfectionists with Deadlines*. 2025. <https://www.djangoproject.com/>. Acessado em: 22 abr. 2025.

Docker, Inc. *Docker: Empowering App Development for Developers*. 2025. <https://www.docker.com/>. Acessado em: 22 abr. 2025.

GÉANT. *GÉANT Testbed Service (GTS)*. 2025. <https://www.geant.org/Services/Advanced_Network_Services/Pages/GTS.aspx>. Acessado em: 22 abr. 2025.

HashiCorp. *Vagrant*. 2025. <https://www.vagrantup.com/>. Acessado em: 23 jun. 2025.

KISTELEKI, R.; ABEN, E. RIPE Atlas: A Global Internet Measurement Platform. *IEEE Communications Magazine*, v. 52, n. 6, 2014. Disponível em: <https://www.ripe.net/publications/docs/ripe-atlas-white-paper.pdf>.

OpenVPN Inc. *OpenVPN Community Edition*. 2025. <https://openvpn.net/community/>. Acessado em: 21 nov. 2025.

PEERING Testbed. *PEERING Client Controller*. 2024. <https://github.com/PEERINGTestbed/client>. Acessado em: 21 nov. 2025.

REKHTER, Y.; LI, T.; HARES, S. *A Border Gateway Protocol 4 (BGP-4)*. 2006. <https://datatracker.ietf.org/doc/html/rfc4271>. RFC 4271.

SCHLINKER, B. et al. PEERING: Virtualizing BGP at the Edge for Research. In: *Proc. ACM CoNEXT*. [S.l.: s.n.], 2019.

SCHLINKER, B. et al. PEERING: An AS for Us. In: *Proc. ACM HotNets*. [S.l.: s.n.], 2014.

XDDXDD. *BIRD-LG-Go: A Modern Looking Glass for BIRD*. 2023. <https://github.com/xddxdd/bird-lg-go>. Acessado em: nov. 2025.