# Documentação NFV-TE

Felipe Ribeiro Quiles e João Vitor Moreira

- **Framework.py**
  - Ao se utilizar a interface para a montagem do arquivo JSON, o mesmo será salvo e o arquivo framework.py será executado, recebendo o JSON configurado e gerando o algoritmo de Engenharia de Tráfego correspondente já parametrizado.

- **Biblioteca**
  - **Network_Function_Resolver**
    - **class NetworkFunctionResolver**
      - **resolve(cls, network_function_name)** -> Função que recebe uma string com o nome da NF a ser gerada e retorna a classe correspondente
        - **network_function_name** -> nome do algoritmo de Engenharia de Tráfego que será gerado como NF

  - **Network_Functions**
    - **class NetworkFunction**
      - **SHAPING_CATEGORY** = 'shaping'
      - **POLICING_CATEGORY** = 'policing'
      - **getName(cls)** -> Função que retorna o nome do algoritmo de Engenharia de Tráfego a ser gerado
      - **getParameters(cls, network_category)** -> Função que retorna a lista de parâmetros do algoritmo de Engenharia de Tráfego a ser gerado
        - **network_category** -> categoria do algoritmo de Engenharia de Tráfego que será gerado como NF
      - **getScript(cls)** -> Função que retorna o nome do script do algoritmo de Engenharia de Tráfego a ser gerado
      - **raiseInvalidNetworkCategoryError(cls, network_category)** -> Função que retorna uma exceção se a categoria do algoritmo de Engenharia de Tráfego a ser gerado está incorreta
        - **network_category** -> categoria do algoritmo de Engenharia de Tráfego que será gerado como NF
    - **class LeakyBucket(NetworkFunction)** -> Classe do Leaky Bucket que implementa os métodos da classe NetworkFunction

        - **class TokenBucket(NetworkFunction)** -> Classe do Leaky Bucket que implementa os métodos da classe NetworkFunction
        - **class OneRateThreeColor(NetworkFunction)** -> Classe do Leaky Bucket que implementa os métodos da classe NetworkFunction
        - **class TwoRateThreeColor(NetworkFunction)** -> Classe do Leaky Bucket que implementa os métodos da classe NetworkFunction

    - **Network_Function_Script_Resolver**
        - **buildParametersDefinitionString(network_function, function_category, parameters)** -> Função que insere os parâmetros passados pelo arquivo JSON no script do algoritmo de Engenharia de Tráfego a ser gerado
            - **network_function** -> classe do algoritmo de Engenharia de Tráfego que será gerado como NF
            - **function_category** -> categoria do algoritmo de Engenharia de Tráfego que será gerado como NF
            - **parameters** -> lista de parâmetros necessários para gerar corretamente o algoritmo de Engenharia de Tráfego selecionado
        - **class NetworkFunctionScriptResolver**
            - **POSSIBLE_COLOR_AWARE = ["one-rate-three-color", "two-rate-three-color"]**
            - **resolve(cls, network_function, parameters)** -> Função que altera no script o campo “#__PARAMETERS__” para os parâmetros passados para o framework através do arquivo JSON
                - **network_function** -> classe do algoritmo de Engenharia de Tráfego que será gerado como NF
                - **parameters** -> lista de parâmetros necessários para gerar corretamente o algoritmo de Engenharia de Tráfego selecionado

    - **Network_Function_Validator**
        - **class NetworkFunctionValidator**
            - **POSSIBLE_COLOR_AWARE = ["one-rate-three-color", "two-rate-three-color"]**
            - **validate(cls, network_function, parameters)** -> Função que valida os parâmetros passados no arquivo JSON com os parâmetros esperados para o algoritmo de Engenharia de Tráfego selecionado

    - **Packet_Processing**

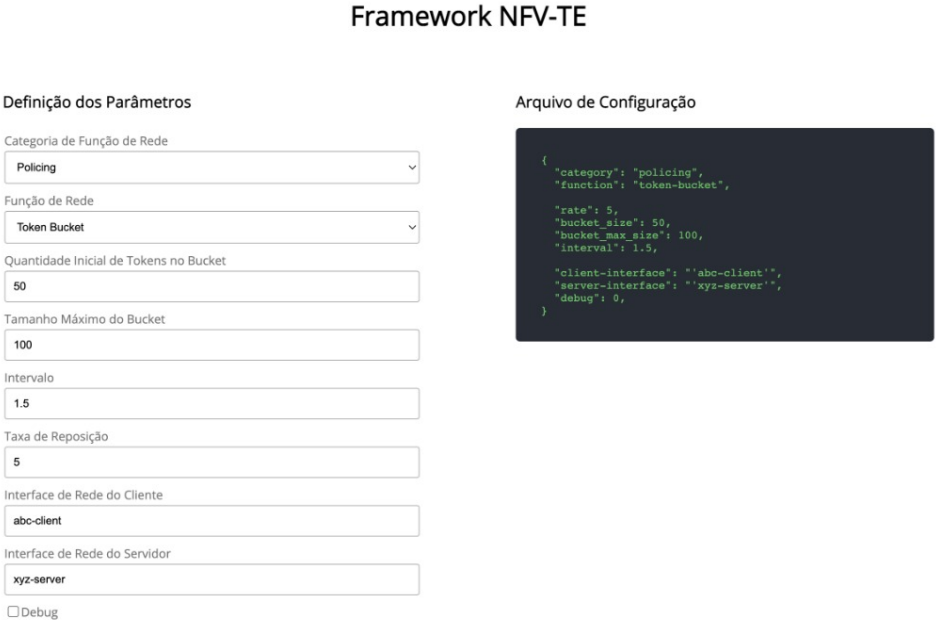
        - **unpackFrameEthernet(frame)** -> Função que desempacota o frame ethernet
            - **frame** -> pacote recebido
        - **bytesToHexa(bytes_address)** -> Função que transforma os endereços de bytes para hexadecimais
            - **bytes_address** -> endereço em bytes
        - **ipPacketData(data)** -> Função que desempacota o dados do pacote IP
            - **data** -> dados recebidos
        - **ipPacketSize(frame)** -> Função que recebe um frame e retorna o tamanho em bytes do mesmo
            - **frame** -> pacote recebido
        - **socketStart(net_interface)** -> Função que inicia o socket na interface de rede
            - **net_interface** -> interface de rede na qual o socket será iniciada
        - **packetAnalysis(data, serverSocket)** -> Função que recebe um conjunto de dados do pacote e analisa verificando se o pacote deve ou não ser transmitido para o servidor
            - **data** -> conjunto de dados do pacote
            - **serverSocket** -> socket do servidor
        - **packetDelay(last, now)** -> Função que calcula o delay no envio de um pacote da fila, verificando a diferença do último enviado para o que está sendo enviado no momento
            - **last** -> número do último pacote
            - **now** -> número do pacote enviado
        - **numberPacketsProcessed(n_transmitted, n_dropped, max_processed)** -> Função que verifica o número de pacotes enviados e descartados e compara o valor máximo de pacotes processados
            - **n_transmitted** -> número de pacotes transmitidos
            - **n_dropped** -> número de pacotes descartados
            - **max_processed** -> valor máximo de pacotes processados

- **Policers**
    - **Token-Bucket**
        - **saveInfos()** -> Função que salva as informações obtidas pelo algoritmo em um arquivo .csv de saída
        - **thread_Time(thread_name, interval)** -> Thread que adiciona tokens aos buckets a cada intervalo de tempo interval
            - **interval** -> intervalo de tempo para que sejam adicionados os tokens

        - **thread_TokenBucket()** ->Thread do TokenBucketPolicer que ao receber um pacote, decidindo se envia ou descarta o pacote de acordo com seus parâmetros
    - **One-Rate-Three-Color**
        - **saveInfosCA() ->** Função que salva as informações obtidas pelo algoritmo em um arquivo .csv de saída quando color-aware ativo
        - **saveInfos() ->** Função que salva as informações obtidas pelo algoritmo em um arquivo .csv de saída quando color-aware inativo
        - **colorAware(contentReceived, color)** -> Função de color-aware que recebe um pacote pré-colorido e verifica se essa coloração está correta de acordo com seus parâmetros
            - **contentReceived** -> conteúdo do pacote
            - **color** -> pré-coloração
        - **colorAwareBucketRate()** -> Função que adiciona tokens aos buckets relacionados ao color-aware mode
        - **thread_Time(thread_name, interval)** -> Thread que adiciona tokens aos buckets a cada intervalo de tempo
            - **interval** -> intervalo de tempo para que sejam adicionados os tokens
        - **thread_OneRateThreeColor()** -> Thread do OneRateThreeColor que ao receber um pacote, verifica qual será a coloração do mesmo de acordo com seus parâmetros, realizando a ação correspondente
    - **Two-Rate-Three-Color**
        - **saveInfosCA() ->** Função que salva as informações obtidas pelo algoritmo em um arquivo .csv de saída quando color-aware ativo
        - **saveInfos() ->** Função que salva as informações obtidas pelo algoritmo em um arquivo .csv de saída quando color-aware inativo
        - **colorAware(contentReceived, color)** -> Função de color-aware que recebe um pacote pré-colorido e verifica se essa coloração está correta de acordo com seus parâmetros
            - **contentReceived** -> conteúdo do pacote
            - **color** -> pré-coloração
        - **colorAwareBucketRate()** -> Função que adiciona tokens aos buckets relacionados ao color-aware mode
        - **thread_Time(thread_name, interval)** -> Thread que adiciona tokens aos buckets a cada intervalo de tempo
            - **interval** -> intervalo de tempo para que sejam adicionados os tokens

        - **thread_TwoRateThreeColor()** -> Thread do TwoRateThreeColor que ao receber um pacote, verifica qual será a coloração do mesmo de acordo com seus parâmetros, realizando a ação correspondente
- **Shapers**
    - **Token-Bucket**
        - **saveInfos()** -> Função que salva as informações obtidas pelo algoritmo em um arquivo .csv de saída
        - **consumeQueue()** -> Função que consome a fila de pacotes em espera quando há disponibilidade de tokens
        - **thread_Time(thread_name, interval)** -> Thread que adiciona tokens aos buckets a cada intervalo de tempo
            - **interval** -> intervalo de tempo para que sejam adicionados os tokens
        - **thread_TokenBucket()** -> Thread do TokenBucketShaper que ao receber um pacote enfileira, transmite ou descarta o pacote, de acordo com seus parâmetros
    - **Leaky-Bucket**
        - **saveInfos()** -> Função que salva as informações obtidas pelo algoritmo em um arquivo .csv de saída
        - **consumeBucket()** -> Função que consome o bucket de acordo com a quantidade de pacotes que são consumidos a cada intervalo de tempo
        - **thread_Time(thread_name, interval)** -> Thread que reseta o consumo do bucket a cada intervalo de tempo
            - **interval** -> intervalo de tempo para que sejam adicionados os tokens
        - **thread_LeakyBucket()** -> Thread do LeakyBucket que ao receber um pacote enfileira, transmite ou descarta o pacote, de acordo com seus parâmetros

- **Modo de Uso**
    - Utilize a interface para gerar o arquivo JSON e o executável cujo nome será:
        - nome_algoritmo-nf.py (EX: leaky-bucket-nf.py)




- Executar o arquivo com o seguinte comando:
  - python3 nome_algoritimo-nf.py
- Pronto, o algoritmo de Engenharia de Tráfego já estará executando na rede