Universidade Federal Fluminense

Relatório do Projeto Avaliação de Dados de Medições em Redes

Avaliação de Desempenho 2024.1

Alunos

Ramon Pedro Pereira Santos

Maria Julia Amancio Galiza

Professor

Antônio Augusto de Aragão Rocha

Niterói, 30 de junho de 2024

Introdução:

A RNP (Rede Nacional de Ensino e Pesquisa) disponibiliza dados de conectividade à internet, extraídos de pontos de presença (POPs) espalhados em cada um dos estados brasileiros. Esses dados podem ser coletados utilizando o Esmond, que é o componente de arquivamento do perfSONAR (uma coleção de softwares para trabalhar com medições de rede). Nesse contexto, a proposta deste trabalho é recuperar os dados do Esmond em diferentes contextos de testes e eventos, analisá-los e modelá-los em uma fila M/M/1, ou seja, com a taxa de entrada em um processo de Poisson, com o tempo de serviço em uma distribuição exponencial, e com apenas um servidor atendendo a fila.

Compreensão:

O repositório do código do projeto possui a seguinte estruturação:

- **main.py**: É o código que o usuário deverá executar para enviar as entradas de variáveis para o sistema e receber informações relevantes ao contexto da análise selecionada. Possui parâmetros pré selecionados, onde o usuário apenas poderá escolher, ao invés de digitar;
- **readme.md**: Possui informações relevantes acerca de como executar o main.py;
- **requirements.txt**: Relação de dependências de bibliotecas que o projeto possui;
- **módulos**: Pasta que agrupa arquivos relevantes à execução do projeto
    - **consultas.py**: encapsula todos os métodos necessários para fazer requisições GET ao archive;
    - **analise.py**: reúne todos os atributos e métodos necessários para receber um objeto de dados, processá-los para a extração e organização das informações, e modelá-los na fila M/M/1;

Análises:

Main.py:

Inicialmente, inicializa-se os parâmetros referentes ao menu de seleção do teste de redes:

```python
    def __init__(self):
        """
        Inicializa o menu de seleção de testes de rede.
        """
        self.loop = True
        self.test_types = self.get_tests()
        self.nodes_delay = self.get_nodes_delay()
        self.nodes_bandwidth = self.get_nodes_bandwidth()
        self.source = None
        self.test_type = None
        self.destination = None
        self.event_type = None
        self.run()
```

Em sequência, são definidos os tipos de teste disponíveis e seus eventos associados:

```python
def get_tests(self):
        """
        Define os tipos de testes disponíveis e seus eventos associados.
        """
        return {
            "Atraso e Perda de Pacotes": [
                "failures", "histogram-rtt", "histogram-ttl-reverse",
"packet-count-lost-bidir",
                "packet-count-sent", "packet-duplicates-bidir",
"packet-loss-rate-bidir",
                "packet-reorders-bidir"
            ],
            "Atraso Unidirecional": [
                "failures", "histogram-owdelay", "histogram-ttl",
"packet-count-lost",
                "packet-count-sent", "packet-duplicates", "packet-loss-rate",
"packet-reorders"
```

```
            ],
            "Banda (BBR)": [
                "failures", "packet-retransmits",
"packet-retransmits-subintervals",
                "throughput", "throughput-subintervals"
            ],
            "Banda (CUBIC)": [
                "failures", "packet-retransmits",
"packet-retransmits-subintervals",
                "throughput", "throughput-subintervals"
            ],
            "Traceroute": [
                "failures", "packet-trace", "path-mtu"
            ]
        }
```

Adiante, declara-se os nós de atraso e de banda disponíveis para serem selecionados nas análises do projeto. Dos 27 existentes, foram selecionados apenas 6 a fim de amostragem:

Python

```
    def get_nodes_delay(self):
        """
        Retorna os nós disponíveis para testes de atraso.
        """
        return [
            "monipe-rj-atraso.rnp.br", "monipe-sp-atraso.rnp.br",
"monipe-df-atraso.rnp.br",
            "monipe-ba-atraso.rnp.br", "monipe-es-atraso.rnp.br",
"monipe-rs-atraso.rnp.br"
        ]

    def get_nodes_bandwidth(self):
        """
        Retorna os nós disponíveis para testes de largura de banda.
        """
        return [
            "monipe-rj-banda.rnp.br", "monipe-sp-banda.rnp.br",
"monipe-df-banda.rnp.br",
            "monipe-ba-banda.rnp.br", "monipe-es-banda.rnp.br",
"monipe-rs-banda.rnp.br"
        ]
```

Por fim, organiza-se a lógica de exibição das opções em um menu interativo no terminal, onde podem ser selecionados parâmetros e, após a execução, exibição das informações modeladas.

Consulta.py:

Em seu construtor, declara-se os atributos necessários para resgatar o objeto de dados do archive Esmond, além de adicionar a largura de banda caso o teste não seja de atraso.

```python
    def __init__(self, event_type, test_type, source, destination, bandwidth):
        """
        Inicializa uma nova instância da classe Consulta.

        Args:
        - event_type (str): Tipo de evento a ser consultado.
        - test_type (str): Tipo de teste a ser realizado.
        - source (str): Nó de origem da consulta.
        - destination (str): Nó de destino da consulta.
        - bandwidth (int): Largura de banda para testes de Banda (BBR) ou Banda (CUBIC).
        """
        self.base_url = "http://monipe-central.rnp.br"
        self.url = f"{self.base_url}/esmond/perfsonar/archive/"
        self.test_type = test_type
        self.parametros = {
            'source': source,
            'destination': destination,
            'event-type': event_type,
            'time-range': 86400
        }
```

Após isso, o método get_data é responsável por realizar a requisição GET ao endpoint, utilizando a função get da biblioteca “requests”

```python
    def get_data(self):
        """
        Realiza a consulta aos dados de perfSONAR com base nos parâmetros definidos.
```

```
        Returns:
        - dict: Dados retornados pela consulta em formato JSON, ou None se
houver erro na consulta.
        """
        try:
            resposta = requests.get(self.url, params=self.parametros,
verify=False)
            resposta.raise_for_status()
            return resposta.json()
        except requests.RequestException as e:
            print(f"Erro na consulta: {e}")
            return None
```

Analise.py:

Inicialmente, inicia-se a classe declarando em seu construtor: a herança do construtor de Consulta, o tipo do teste a ser realizado, o tipo do evento a ser analisado, a fonte escolhida do teste, o destino escolhido do teste, e a largura de banda utilizada no teste.

Python

```
    def __init__(self, test_type, event_type, source, destination, bandwidth):
        """
        Inicializa a classe de análise com os parâmetros necessários.

        Args:
        - test_type: Tipo de teste a ser realizado.
        - event_type: Tipo de evento a ser analisado.
        - source: Fonte do teste.
        - destination: Destino do teste.
        - bandwidth: Largura de banda utilizada no teste.
        """
        super().__init__(event_type, test_type, source, destination, bandwidth)
        self.bandwidth = bandwidth
        self.dados = self.get_data()
        self.test_type = test_type
```

Em seguida, caso a requisição dos dados seja bem sucedida, a função analisar é chamada, e ela executa a análise específica conforme o tipo de evento. Caso contrário,

significa que a consulta ao archive retornou um objeto vazio. Nesse caso há dois eventos possíveis de terem gerado este resultado: o evento failures ou path-mtu. Como não foi possível encontrar uma combinação de source-destination que fornecesse um objeto não nulo para esses dois eventos, não foi realizado o processamento de dados e a modelagem deles.

Passado no teste acima, para cada item nos dados requisitados identifica o tipo do evento de cada um deles, e cada um deles é verificado para achar o correspondente ao escolhido para a análise que foi definido anteriormente.

```python
      for item in self.dados:
           event_types = item.get('event-types', [])
           for event in event_types:
               if event['event-type'] == self.parametros['event-type']:
                   self.escolher(event)
```

Posteriormente, ao identificar o evento a ser analisado, define-se o método de análise apropriado com base no tipo de evento, subdividindo em funções específicas a cada método.

Pode-se observar que existem 20 tipos de eventos porém somente 5 funções de análise. Entretanto, pela similaridade entre o tratamento desses eventos, foi possível fazer com que uma função atenda mais de um tipo de evento. Nesse caso as seguintes funções são responsáveis por tratar os seguintes eventos:

- **função analisar_ttl_reverse**: ‘packet-retransmits’, ’packet-reorders’, ‘packet-loss-rate’, ‘packet-duplicates’, ‘packet-reorders-bidir’, ‘packet-loss-rate-bidir’, ‘packet-duplicates-bidir’, ‘histogram-ttl-reverse’, ‘histogram-ttl’, ‘packet-retransmits-subintervals’, ‘throughput-subintervals’

- **função analisar_histogram**: ‘histogram-owdelay’, ‘histogram-rtt’

- **função analisar_packet_count**: ‘packet-count-sent’, ‘packet-count-lost’, ‘packet-count-lost-bidir’, ‘throughput’

- **função analisar_bidir**: ‘packet-reorders’, ‘packet-trace’

```python
   def escolher(self, event):
        if self.parametros['event-type'] in ['packet-retransmits',
'packet-reorders', 'packet-loss-rate', 'packet-duplicates', 'packet-reorders-bidir',
'packet-loss-rate-bidir', 'packet-duplicates-bidir', 'histogram-ttl-reverse',
'histogram-ttl', 'packet-retransmits-subintervals', 'throughput-subintervals']:
            self.analisar_ttl_reverse(event)
```

```
        elif self.parametros['event-type'] in ['histogram-owdelay',
'histogram-rtt']:
            self.analisar_histogram(event)
        elif self.parametros['event-type'] in ['packet-count-sent',
'packet-count-lost', 'packet-count-lost-bidir', 'throughput']:
            self.analisar_packet_count(event)
        elif self.parametros['event-type'] in ['packet-reorders', 'packet-trace']:
            self.analisar_bidir(event)
```

- Análise do tipo de evento histograma TTL reverso:

É retirada a base URI do dicionário contendo as informações do evento de histograma TTL reverso para posterior consulta dos dados dessa URI.

Essa consulta é realizada a partir de uma tentativa de capturar a resposta da requisição HTTP para o endereço formado pela junção de self.base_url + uri. Para isso, verifica se a solicitação foi bem sucedida, convertendo então a resposta em JSON e armazenando em uma variável. Caso o servidor retornar um código de status indicando erro, retorna uma mensagem de erro indicando qual URI causou o problema e a descrição da exceção capturada.

Python
```
        try:
            resposta = requests.get(self.base_url + uri, verify=False)
            resposta.raise_for_status()
            dados = resposta.json()
        except requests.RequestException as e:
            print(f"Erro na consulta do URI {uri}: {e}")
            return
```

Em seguida, inicia-se uma lista de valores para adicionar os dados e tratamos o formato no qual foi recebido (lista ou dicionário). No caso de ser uma lista, significa que se obteve também a duração de cada data, se houver. Por fim, esses valores são calculados (chama-se a função “calcular” que será mostrada posteriormente).

Python
```
        values = []
        for entry in dados:
            if 'val' in entry:
                if isinstance(entry['val'], dict):
                    values.extend(entry['val'].values())
```

```
            elif isinstance(entry['val'], list):
                for data in entry['val']:
                    if 'duration' in data:
                        values.append(data['duration'])
            else:
                values.append(entry['val'])
        self.calcular(values)
```

- Análise do tipo de evento de histograma:

Para tal análise, os sumários presentes no dicionário são recolhidos, contendo as informações do evento de histograma e, para cada um deles, obtém-se a janela de resposta que se refere ao time-range definido anteriormente. Nesse caso, obtém-se a sua URI, e a consulta e análise dos dados da URI são realizadas.

Python
```
        summaries = event.get('summaries', [])
        for summary in summaries:
            if int(summary['summary-window']) == self.parametros['time-range']:
                uri = summary['uri']
                self.analisar_dados_histograma(uri)
```

Essa consulta é feita da mesma forma que foi feita a consulta do evento histograma TTL reverso, tentando capturar a resposta da requisição HTTP e transformando num objeto Python se bem sucedida. Em seguida é feito uma lista de valores com os dados recebidos se possuírem os parâmetros ‘val’ e ‘mean’, e calculado os valores (função “calcular”).

Python
```
        values = [entry['val']['mean'] for entry in dados if 'val' in entry and
'mean' in entry['val']]
        self.calcular(values)
```

- Análise do tipo de evento de contagem de pacotes:

De forma similar à análise anterior, pegam-se os sumários para comparar cada um deles com o parâmetro do time-range definido anteriormente, que foi o escolhido, obteremos a URI do sumário desejado e faremos uma requisição HTTP assim como feita na análise dos dados do evento de histograma TTL reverso, associando os dados a uma lista de valores e calculando-os.

```python
summaries = event.get('summaries', [])
for summary in summaries:
    if int(summary['summary-window']) == self.parametros['time-range']:
        uri = summary['uri']
        self.analisar_dados_ttl_reverse(uri)
```

- Análise do tipo de evento de bidirecional:

Para essa análise é recolhida a URI base do dicionário contendo as informações do evento bidirecional, em seguida é feita uma requisição HTTP, a resposta é convertida em JSON e armazenada em uma variável. Caso o servidor retornar um código de status indicando erro, retorna uma mensagem de erro indicando qual URI causou o problema e a descrição da exceção capturada, assim como foi feito anteriormente.

A seguir, inicializa-se duas listas, uma para os valores TTL (tempo de vida) e outra para RTT (tempo de resposta). Assim, verifica-se se ‘val’ pertence à lista de dicionários e, para cada data presente em ‘val’, se em data existir um campo ‘TTL’, então é adicionada à lista de valores dos tempos de vida. O mesmo ocorre para a lista de valores do tempo de resposta. Ao final, esses valores são passados para a função “calcular”.

```python
values_ttl = []
values_rtt = []
for entry in dados:
    if 'val' in entry:
        if isinstance(entry['val'], list):
            for data in entry['val']:
                if 'ttl' in data:
                    values_ttl.append(data['ttl'])
                if 'rtt' in data:
                    values_rtt.append(data['rtt'])
if values_rtt:
    self.calcular(values_rtt)
if values_ttl:
    self.calcular(values_ttl)
```

Por fim, a função “calcular” realiza o cálculo estatístico dos valores fornecidos, modelados pelas funções acima. Dessa forma, os parâmetros estatísticos mostrados serão o

número de eventos, o menor valor obtido, o maior valor obtido, a média dos valores, a variância dos valores, e o desvio padrão dos valores.

```python
        if values:
            num_events = len(values)
            total_value = sum(values)
            mean_value = total_value / num_events
            variance_value = sum((x - mean_value) ** 2 for x in values) /
num_events
            std_dev_value = math.sqrt(variance_value)
            print(f"Total de entradas: {num_events}")
            print(f"\nMenor valor: {min(values)} ms")
            print(f"Maior valor: {max(values)} ms")
            print(f"Média dos valores: {mean_value} ms")
            print(f"Variância dos valores: {variance_value} ms²")
            print(f"Desvio padrão dos valores: {std_dev_value} ms")
```

Em seguida, é realizado o cálculo da modelagem de filas, mostrando o resultado dos parâmetros: taxa média de chegada em eventos por segundo, taxa média de serviço em eventos por segundo, utilização do sistema, tempo médio de espera na fila em segundos, tempo médio de espera no sistema em segundos e número médio de eventos no sistema. Caso a utilização do sistema for maior do que 1, será retornado que o sistema está saturado e não é possível calcular o tempo de espera na fila e no sistema de um sistema saturado. Há também o caso de não ter retornado a taxa média de serviço, então, será tratado este erro para que retorne ao usuário informando-o que a fila está inoperante. Por fim, se a lista de valores estiver vazia então não haverá dados a serem modelados em fila, informando ao usuário.

```python
        lambda_rate = num_events / (86400)  # taxa média de chegada em eventos
por segundo
        try:
                mu_rate = 1 / (mean_value / 1000)  # taxa média de serviço em
eventos por segundo
                rho = lambda_rate / mu_rate  # utilização do sistema
                if rho < 1:
                    Wq = rho / (mu_rate * (1 - rho))  # tempo médio de espera na
fila (segundos)
                    W = 1 / (mu_rate * (1 - rho))  # tempo médio de espera no
sistema (segundos)
                    Lq = (rho ** 2) / (1 - rho)  # número médio de eventos na
fila
```

```
                L = rho / (1 - rho)  # número médio de eventos no sistema
                print(f"\nTaxa média de chegada (λ): {lambda_rate:.6f}
eventos/segundo")
                print(f"Taxa média de serviço (μ): {mu_rate:.6f}
eventos/segundo")
                print(f"Utilização do sistema (ρ): {rho:.6f}")
                print(f"Tempo médio de espera na fila (Wq): {Wq:.6f}
segundos")
                print(f"Tempo médio de espera no sistema (W): {W:.6f}
segundos")
                print(f"Número médio de eventos na fila (Lq): {Lq:.6f}
eventos")
                print(f"Número médio de eventos no sistema (L): {L:.6f}
eventos")
            else:
                print("Utilização do sistema é 1 (ou maior), o que indica que
o sistema está saturado.")
                print("Não é possível calcular o tempo de espera na fila e no
sistema para um sistema saturado.")
        except Exception as e:
            print('Taxa média de serviço é zero, a fila está inoperante: ',
e)
```

Resultados:

Nesta seção os eventos serão demonstrados a partir de alguns exemplos que foram relatados no decorrer do projeto.

Resultado 1:

Teste selecionado: Atraso e Perda de Pacotes
Fonte selecionada: monipe-rj-atraso.rnp.br
Destino selecionado: monipe-sp-atraso.rnp.br
Evento selecionado: failures
A consulta para o archive retornou um objeto vazio. Isso está acontecendo em duas situações:
1. No evento 'failures'
2. No evento 'path-mtu'
Para esses eventos não foi possível encontrar uma combinação de source-destination que fornecesse um objeto não nulo, então não foi realizado o processamento de dados e a modelagem deles.

Resultado 2:

Teste selecionado: Atraso e Perda de Pacotes
Fonte selecionada: monipe-rj-atraso.rnp.br
Destino selecionado: monipe-sp-atraso.rnp.br
Evento selecionado: histogram-rtt

Total de entradas: 5

Menor valor: 7.4228 ms
Maior valor: 7.470477038091849 ms
Média dos valores: 7.452072760754694 ms
Variância dos valores: 0.0003615342710434535 ms²
Desvio padrão dos valores: 0.0190140545661217 ms

Taxa média de chegada (λ): 0.000058 eventos/segundo
Taxa média de serviço (μ): 134.190853 eventos/segundo
Utilização do sistema (ρ): 0.000000
Tempo médio de espera na fila (Wq): 0.000000 segundos
Tempo médio de espera no sistema (W): 0.007452 segundos
Número médio de eventos na fila (Lq): 0.000000 eventos
Número médio de eventos no sistema (L): 0.000000 eventos

Resultado 3:

Teste selecionado: Atraso Unidirecional
Fonte selecionada: monipe-ba-atraso.rnp.br
Destino selecionado: monipe-rs-atraso.rnp.br
Evento selecionado: histogram-owdelay

Nenhum dado disponível.
Total de entradas: 5

Menor valor: 21.841348277725846 ms
Maior valor: 23.125582174484 ms
Média dos valores: 22.566579364443633 ms
Variância dos valores: 0.1735218962242317 ms²
Desvio padrão dos valores: 0.41655959504521284 ms

Taxa média de chegada (λ): 0.000058 eventos/segundo
Taxa média de serviço (μ): 44.313318 eventos/segundo
Utilização do sistema (ρ): 0.000001
Tempo médio de espera na fila (Wq): 0.000000 segundos
Tempo médio de espera no sistema (W): 0.022567 segundos

Número médio de eventos na fila (Lq): 0.000000 eventos
Número médio de eventos no sistema (L): 0.000001 eventos

Resultado 4:

Teste selecionado: Atraso Unidirecional
Fonte selecionada: monipe-rj-atraso.rnp.br
Destino selecionado: monipe-sp-atraso.rnp.br
Evento selecionado: histogram-ttl
Total de entradas: 1500

Menor valor: 527 ms
Maior valor: 600 ms
Média dos valores: 599.886 ms
Variância dos valores: 3.623670666666896 ms²
Desvio padrão dos valores: 1.9035941444191553 ms

Taxa média de chegada (λ): 0.017361 eventos/segundo
Taxa média de serviço (μ): 1.666983 eventos/segundo
Utilização do sistema (ρ): 0.010415
Tempo médio de espera na fila (Wq): 0.006313 segundos
Tempo médio de espera no sistema (W): 0.606199 segundos
Número médio de eventos na fila (Lq): 0.000110 eventos
Número médio de eventos no sistema (L): 0.010524 eventos

Resultado 5:

Teste selecionado: Banda (BBR)
Fonte selecionada: monipe-ba-banda.rnp.br
Destino selecionado: monipe-rs-banda.rnp.br
Evento selecionado: throughput
Total de entradas: 3

Menor valor: 2813038089.0 ms
Maior valor: 3705239776.5 ms
Média dos valores: 3215754922.611111 ms
Variância dos valores: 1.3643498653443704e+17 ms²
Desvio padrão dos valores: 369371068.89202493 ms
Utilização do sistema é 1 (ou maior), o que indica que o sistema está saturado.
Não é possível calcular o tempo de espera na fila e no sistema para um sistema saturado.

Resultado 6:

Teste selecionado: Banda (BBR)

Fonte selecionada: monipe-ba-banda.rnp.br
Destino selecionado: monipe-rs-banda.rnp.br
Evento selecionado: packet-retransmits-subintervals
Total de entradas: 425

Menor valor: 0.9532850000000002 ms
Maior valor: 1.0385150000000003 ms
Média dos valores: 1.000010943529412 ms
Variância dos valores: 2.0688437027399132e-05 ms²
Desvio padrão dos valores: 0.004548454355866301 ms

Taxa média de chegada (λ): 0.004919 eventos/segundo
Taxa média de serviço (μ): 999.989057 eventos/segundo
Utilização do sistema (ρ): 0.000005
Tempo médio de espera na fila (Wq): 0.000000 segundos
Tempo médio de espera no sistema (W): 0.001000 segundos
Número médio de eventos na fila (Lq): 0.000000 eventos
Número médio de eventos no sistema (L): 0.000005 eventos

Resultado 7:

Teste selecionado: Banda (CUBIC)
Fonte selecionada: monipe-ba-banda.rnp.br
Destino selecionado: monipe-rs-banda.rnp.br
Evento selecionado: throughput-subintervals
Total de entradas: 475

Menor valor: 0.983078 ms
Maior valor: 1.0092299999999987 ms
Média dos valores: 0.9999845431578946 ms
Variância dos valores: 2.0517507955057887e-06 ms²
Desvio padrão dos valores: 0.0014323933801528785 ms

Taxa média de chegada (λ): 0.005498 eventos/segundo
Taxa média de serviço (μ): 1000.015457 eventos/segundo
Utilização do sistema (ρ): 0.000005
Tempo médio de espera na fila (Wq): 0.000000 segundos
Tempo médio de espera no sistema (W): 0.001000 segundos
Número médio de eventos na fila (Lq): 0.000000 eventos
Número médio de eventos no sistema (L): 0.000005 eventos

Resultado 8:

Teste selecionado: Atraso Unidirecional

Fonte selecionada: monipe-rj-atraso.rnp.br
Destino selecionado: monipe-sp-atraso.rnp.br
Evento selecionado: packet-reorders
Total de entradas: 1500

Menor valor: 0.0 ms
Maior valor: 0.0 ms
Média dos valores: 0.0 ms
Variância dos valores: 0.0 ms²
Desvio padrão dos valores: 0.0 ms
Taxa média de serviço é zero, a fila está inoperante: float division by zero

Taxa média de serviço (μ): 0.338735 eventos/segundo
Utilização do sistema (ρ): 0.000649
Tempo médio de espera na fila (Wq): 0.001918 segundos
Tempo médio de espera no sistema (W): 2.954076 segundos
Número médio de eventos na fila (Lq): 0.000000 eventos
Número médio de eventos no sistema (L): 0.000650 eventos

Resultado 9:

Teste selecionado: Traceroute
Fonte selecionada: monipe-rj-atraso.rnp.br
Destino selecionado: monipe-sp-atraso.rnp.br
Evento selecionado: packet-trace
Total de entradas para o RTT: 2604

Menor valor: 0.2 ms
Maior valor: 71.6 ms
Média dos valores: 5.275499231950798 ms
Variância dos valores: 46.11014472004561 ms²
Desvio padrão dos valores: 6.790445104707468 ms

Taxa média de chegada (λ): 0.030139 eventos/segundo
Taxa média de serviço (μ): 189.555520 eventos/segundo
Utilização do sistema (ρ): 0.000159
Tempo médio de espera na fila (Wq): 0.000001 segundos
Tempo médio de espera no sistema (W): 0.005276 segundos
Número médio de eventos na fila (Lq): 0.000000 eventos
Número médio de eventos no sistema (L): 0.000159 eventos

Total de entradas para o TTL: 2604

Menor valor: 1 ms
Maior valor: 6 ms
Média dos valores: 3.5 ms
Variância dos valores: 2.9166666666666665 ms²
Desvio padrão dos valores: 1.707825127659933 ms

Taxa média de chegada (λ): 0.030139 eventos/segundo
Taxa média de serviço (μ): 285.714286 eventos/segundo
Utilização do sistema (ρ): 0.000105
Tempo médio de espera na fila (Wq): 0.000000 segundos
Tempo médio de espera no sistema (W): 0.003500 segundos
Número médio de eventos na fila (Lq): 0.000000 eventos
Número médio de eventos no sistema (L): 0.000105 eventos

Conclusões:

Pode-se concluir que:

1. Para alguns eventos são sempre retornados objetos vazios, como o failures e path-mtu, em todas as fontes e destinos, e para algumas localidades específicas, todas as requisições retornam objetos nulos, tornando assim impossível a modelagem deles na fila proposta.
2. A utilização do sistema quando este é modelável, na maioria dos eventos, é sempre muito baixa, demonstrando que há um atraso maior para a chegada dos dados do que para o processamento desses. Dessa forma, conclui-se que a estruturação da RNP para processamento de dados é eficiente, sendo eficiente em monitorar a maioria dos dados de conectividade à internet de todo o Brasil.
3. Alguns poucos eventos apresentaram taxa de ocupação maior do que 1 quando processados, demonstrando assim inviabilidade de modelagem. Nesse sentido, conclui-se que alguns eventos podem não estar sendo processados de forma adequada pela RNP.
4. Certos eventos não retornaram objetos vazios, porém retornaram dados com o valor igual a zero. Dessa forma, foi impossível modelar em uma fila, pois a taxa média de serviço seria zero e isso implicaria em divisões por zero.