Operações de Segurança e Gestão de Incidentes - Projeto

Análise Comportamental de Tráfego com IA para Deteção de Ataques em Tempo Real

Introdução

No âmbito da unidade curricular “Operações de Segurança e Gestão de Incidente” foi proposto ao grupo a realização de um projeto focado nos temas lecionados ao longo do semestre. Desta forma, tendo em conta as novas tecnologias, em especial da Inteligência Artificial, o grupo optou por atender às necessidades das empresas quanto a ataques cibernéticos, através da utilização da Inteligência Artificial para a deteção e proteção contra estes mesmos ataques. No entanto, devido à alta variedade de ataques, o grupo decidiu apenas focar-se na deteção e proteção contra ataques de negação de serviço.

O presente Jupyter Notebook é referente ao tratamento dos dados e treino do algoritmo não-supervisionado para que posteriormente seja aplicado no "Servidor". O notebook está dividido de acordo com o seguinte indice:

Importação de Bibliotecas

In [ ]:
```
import json

# ------------------------------------------------------------- IMPORTS F

import cudf
from cuml.manifold.umap import UMAP as UMAP2
```

```
from cuml.preprocessing import MinMaxScaler as MinMaxScaler2
# ------------------------------------------------------------------ BIBLIOTEC


#Bibliotecas Standard
import pandas as pd
import numpy as np
import plotly.express as px
import matplotlib.pyplot as plt
import matplotlib.style as style
import statistics
import seaborn as sns

from matplotlib import cm
from matplotlib.colors import Normalize
from branca.colormap import LinearColormap

#Padronização dos dados
from sklearn.preprocessing import LabelEncoder, MinMaxScaler, StandardScaler
from sklearn.compose import ColumnTransformer

from sklearn.ensemble import IsolationForest
from sklearn.model_selection import train_test_split

palette_color = ['#167288', '#8cdaec', '#b45248', '#d48c84', '#a89a49', '#d6
```

1. Leitura de Dados

Na seguinte célula vai ser feita a leitura do ficheiro de logs "eve.json", onde se vai extrair as colunas timestamp, src_ip, dest_ip, proto, signature, sid

In [2]:
```
log_file = 'logs_demo/eve.json'
data = []

with open(log_file, 'r') as f:
    for line in f:
        try:
            event = json.loads(line)
            if event.get('event_type') == 'alert':
                alert = event.get('alert', {})
                data.append({
                    'timestamp': event.get('timestamp'),
                    'src_ip': event.get('src_ip'),
                    'dest_ip': event.get('dest_ip'),
                    'proto': event.get('proto'),
                    'signature': alert.get('signature'),
                    'sid': alert.get('signature_id'),
                })
        except json.JSONDecodeError:
            continue

# Converter em DataFrame
df = pd.DataFrame(data)
```

```
# Ver as primeiras entradas
df.head()
```

Out[2]:

	timestamp	src_ip	dest_ip	proto	signature	sid
0	2025-05-31T15:50:32.260025+0000	172.28.0.11	172.28.0.100	TCP	TCP packet detected	100001
1	2025-05-31T15:50:32.260025+0000	172.28.0.11	172.28.0.100	TCP	TCP SYN packet detected	100006
2	2025-05-31T15:50:32.356553+0000	172.28.0.12	172.28.0.100	UDP	UDP packet detected	100002
3	2025-05-31T15:50:32.526292+0000	172.28.0.11	172.28.0.100	TCP	TCP SYN packet detected	100006
4	2025-05-31T15:50:32.776781+0000	172.28.0.11	172.28.0.100	TCP	TCP SYN packet detected	100006

1.1 Descrição das Colunas

Para se perceber o significado de cada coluna foi criado a uma tabela com cada coluna e respetiva descrição. Além disso também foi criado uma tabela com a descrição para cada ID das assinaturas

In [10]:
```
descricoes = {
    "timestamp": "Data e Hora",
    "src_ip": "IP de Origem",
    "dest_ip": "IP de destino",
    "proto": "Protocolo do Pacote",
    "signature": "Tipo de pacote detetado",
    "sid": "ID da signature",
}
descricoes_sid = {
    "100001": "Pacote TCP",
    "100002": "Pacote UDP",
    "100004": "Pacote HTTP",
    "100005": "Pacote DNS",
    "100006": "Pacote TCP SYN",
    "100007": "Pacote TCP ACK",
}
colunas_sid = ["100001","100002","100004","100005","100006","100007",]
```

In [16]:
```
desc = pd.DataFrame({
    "Coluna": df.columns,
    "Descricao": [descricoes.get(col, "Descrição não definida") for col in c
})
```

```
desc_sid = pd.DataFrame({
    "ID": colunas_sid,
    "Descricao": [descricoes_sid.get(col, "Descrição não definida") for col
})
```

In [17]:
```
desc
```

Out[17]:

	Coluna	Descricao
0	timestamp	Data e Hora
1	src_ip	IP de Origem
2	dest_ip	IP de destino
3	proto	Protocolo do Pacote
4	signature	Tipo de pacote detetado
5	sid	ID da signature

In [18]:
```
desc_sid
```

Out[18]:

	ID	Descricao
0	100001	Pacote TCP
1	100002	Pacote UDP
2	100004	Pacote HTTP
3	100005	Pacote DNS
4	100006	Pacote TCP SYN
5	100007	Pacote TCP ACK

2. Data Wrangling

Neste tópico iremos analisar as colunas em questão, colocar os seus dados no seu tipo de dados correto e criar colunas que irão ser uteis não só para a análise dos dados, como também para a execução dos algoritmos.

In [34]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 91983 entries, 0 to 91982
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   timestamp  91983 non-null  object
 1   src_ip     91983 non-null  object
 2   dest_ip    91983 non-null  object
 3   proto      91983 non-null  object
 4   signature  91983 non-null  object
 5   sid        91983 non-null  int64 
dtypes: int64(1), object(5)
memory usage: 4.2+ MB
```

In [35]:
```
df['timestamp'] = pd.to_datetime(df['timestamp']) # Definir os dados da colu
```

In [36]:
```
df['hour'] = df['timestamp'].dt.hour # Nova coluna para a hora
df['minute'] = df['timestamp'].dt.minute # Nova coluna para os minutos
df['second'] = df['timestamp'].dt.second # Nova coluna para os segundos
df['millisecond'] = df['timestamp'].dt.microsecond // 1000 # Nova coluna par
```

In [37]:
```
df.shape #Tamanho do Dataset. Linhas x Colunas
```

Out[37]:
```
(91983, 10)
```

In [38]:
```
df['src_ip'].value_counts() # Diferentes IPs existentes nos dados e quantas
```

Out[38]:
```
src_ip
172.28.0.13    26472
172.28.0.11    26120
172.28.0.20    20000
172.28.0.15    13346
172.28.0.12     6045
Name: count, dtype: int64
```

In [39]:
```
df['sid'].value_counts() # Diferentes IDs de assinatura e quantas vezes apar
```

Out[39]:
```
sid
100006    34782
100001    16712
100002    12712
100007    10549
100004    10549
100005     6679
Name: count, dtype: int64
```

In [40]:
```
df = df.sort_values(by=['src_ip', 'timestamp']) #Ordenar as linhas do nosso

df['delta'] = df.groupby('src_ip')['timestamp'].diff().dt.total_seconds() #
```

In [41]:
```
df = df.dropna() # Retirar os pacotes com possíveis valores NaN
```

In [42]:
```
df
```

Out[42]:

	timestamp	src_ip	dest_ip	proto	signature	sid	hour
1	2025-05-31 15:50:32.260025+00:00	172.28.0.11	172.28.0.100	TCP	TCP SYN packet detected	100006	15
3	2025-05-31 15:50:32.526292+00:00	172.28.0.11	172.28.0.100	TCP	TCP SYN packet detected	100006	15
4	2025-05-31 15:50:32.776781+00:00	172.28.0.11	172.28.0.100	TCP	TCP SYN packet detected	100006	15
5	2025-05-31 15:50:33.033113+00:00	172.28.0.11	172.28.0.100	TCP	TCP packet detected	100001	15
6	2025-05-31 15:50:33.033113+00:00	172.28.0.11	172.28.0.100	TCP	TCP SYN packet detected	100006	15
...	...	...	...	...	...	...	...
56967	2025-05-31 16:28:07.909940+00:00	172.28.0.20	172.28.0.100	TCP	TCP SYN packet detected	100006	16
56968	2025-05-31 16:28:07.910021+00:00	172.28.0.20	172.28.0.100	TCP	TCP packet detected	100001	16
56969	2025-05-31 16:28:07.910021+00:00	172.28.0.20	172.28.0.100	TCP	TCP SYN packet detected	100006	16
56970	2025-05-31 16:28:07.910106+00:00	172.28.0.20	172.28.0.100	TCP	TCP packet detected	100001	16
56971	2025-05-31 16:28:07.910106+00:00	172.28.0.20	172.28.0.100	TCP	TCP SYN packet detected	100006	16

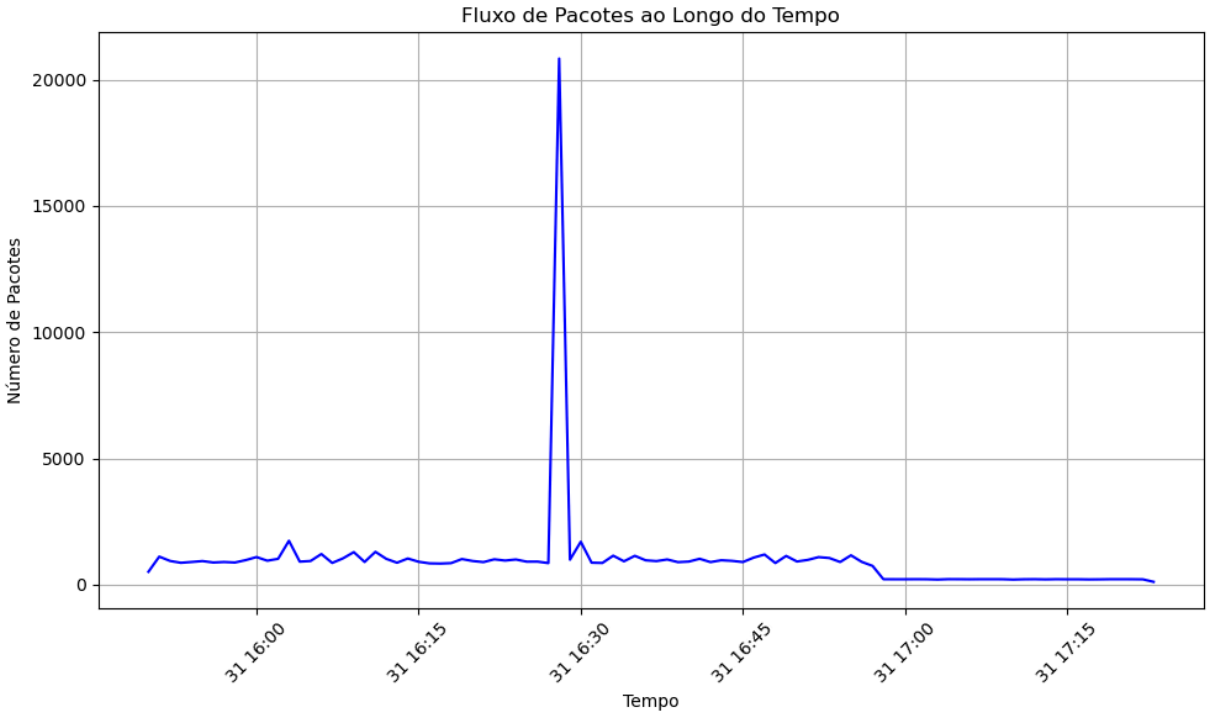
91978 rows × 11 columns

In [ ]:

In [43]:
```
# Agrupar os pacotes por segundo, minuto ou qualquer outro intervalo
df['time_interval'] = df['timestamp'].dt.floor('min')  # 'S' para agrupar po

# Contar o número de pacotes por intervalo de tempo
packets_per_interval = df['time_interval'].value_counts().sort_index()
```

```
# Adicionar essa contagem ao DataFrame como uma nova coluna
df['count_interval'] = df['time_interval'].map(packets_per_interval)
packets_per_interval = df.groupby('time_interval').size()

# Agrupa por src_ip e segundo
df['time_bucket'] = df['timestamp'].dt.floor('S')
packets_per_second = df.groupby(['src_ip', 'time_bucket']).size().reset_inde

# Junta ao df principal
df = df.merge(packets_per_second, on=['src_ip', 'time_bucket'], how='left')

# Plot do fluxo de pacotes ao longo do tempo
plt.figure(figsize=(10, 6))
plt.plot(packets_per_interval.index, packets_per_interval.values, label='Núm

# Adicionando rótulos e título
plt.title('Fluxo de Pacotes ao Longo do Tempo')
plt.xlabel('Tempo')
plt.ylabel('Número de Pacotes')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()

# Exibindo o gráfico
plt.show()
```

```
/tmp/ipykernel_3538/40460388.py:12: FutureWarning: 'S' is deprecated and wil
l be removed in a future version, please use 's' instead.
  df['time_bucket'] = df['timestamp'].dt.floor('S')
```



3. Feature Engennering

Neste tópico iremos preparar os nossos dados para a execução do algoritmo. Desta forma iremos usar o "labelEncoder" para transformar os diferentes valores do tipo texto para o tipo int. Por exemplo, tendo 3 diferentes valores de texto: Maça, Pera e Banana, o LabelEncoder irá transformar estes em 0, 1 e 2 onde 0 corresponde à Maçã, 1 à Perâ e 2 à Banana

In [44]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

In [45]:
```
df['src_ip'] = le.fit_transform(df['src_ip']) # Encondig dos IPs
```

In [47]:
```
mapping_srcip = dict(zip(le.classes_, range(len(le.classes_)))) # Mapeamento
print(mapping_srcip)
```

```
{'172.28.0.11': 0, '172.28.0.12': 1, '172.28.0.13': 2, '172.28.0.15': 3, '17
2.28.0.20': 4}
```

In [48]:
```
df['proto'] = le.fit_transform(df['proto']) # Encondig dos protocolos
```

In [49]:
```
mapping_proto = dict(zip(le.classes_, range(len(le.classes_)))) # Mapeamento
print(mapping_proto)
```

```
{'TCP': 0, 'UDP': 1}
```

In [50]:
```
df['sid'] = le.fit_transform(df['sid']) # Encondig dos ids de assinatura
```

In [51]:
```
mapping_sid = dict(zip(le.classes_, range(len(le.classes_)))) # Mapeamento d
print(mapping_sid)
```

```
{100001: 0, 100002: 1, 100004: 2, 100005: 3, 100006: 4, 100007: 5}
```

In [52]:
```
df['time_interval'] = le.fit_transform(df['time_interval']) # Encondig do in
```

In [53]:
```
mapping_timeinter = dict(zip(le.classes_, range(len(le.classes_)))) # Mapeam
print(mapping_timeinter)
```

{Timestamp('2025-05-31 15:50:00+0000', tz='UTC'): 0, Timestamp('2025-05-31 15:51:00+0000', tz='UTC'): 1, Timestamp('2025-05-31 15:52:00+0000', tz='UTC'): 2, Timestamp('2025-05-31 15:53:00+0000', tz='UTC'): 3, Timestamp('2025-05-31 15:54:00+0000', tz='UTC'): 4, Timestamp('2025-05-31 15:55:00+0000', tz='UTC'): 5, Timestamp('2025-05-31 15:56:00+0000', tz='UTC'): 6, Timestamp('2025-05-31 15:57:00+0000', tz='UTC'): 7, Timestamp('2025-05-31 15:58:00+0000', tz='UTC'): 8, Timestamp('2025-05-31 15:59:00+0000', tz='UTC'): 9, Timestamp('2025-05-31 16:00:00+0000', tz='UTC'): 10, Timestamp('2025-05-31 16:01:00+0000', tz='UTC'): 11, Timestamp('2025-05-31 16:02:00+0000', tz='UTC'): 12, Timestamp('2025-05-31 16:03:00+0000', tz='UTC'): 13, Timestamp('2025-05-31 16:04:00+0000', tz='UTC'): 14, Timestamp('2025-05-31 16:05:00+0000', tz='UTC'): 15, Timestamp('2025-05-31 16:06:00+0000', tz='UTC'): 16, Timestamp('2025-05-31 16:07:00+0000', tz='UTC'): 17, Timestamp('2025-05-31 16:08:00+0000', tz='UTC'): 18, Timestamp('2025-05-31 16:09:00+0000', tz='UTC'): 19, Timestamp('2025-05-31 16:10:00+0000', tz='UTC'): 20, Timestamp('2025-05-31 16:11:00+0000', tz='UTC'): 21, Timestamp('2025-05-31 16:12:00+0000', tz='UTC'): 22, Timestamp('2025-05-31 16:13:00+0000', tz='UTC'): 23, Timestamp('2025-05-31 16:14:00+0000', tz='UTC'): 24, Timestamp('2025-05-31 16:15:00+0000', tz='UTC'): 25, Timestamp('2025-05-31 16:16:00+0000', tz='UTC'): 26, Timestamp('2025-05-31 16:17:00+0000', tz='UTC'): 27, Timestamp('2025-05-31 16:18:00+0000', tz='UTC'): 28, Timestamp('2025-05-31 16:19:00+0000', tz='UTC'): 29, Timestamp('2025-05-31 16:20:00+0000', tz='UTC'): 30, Timestamp('2025-05-31 16:21:00+0000', tz='UTC'): 31, Timestamp('2025-05-31 16:22:00+0000', tz='UTC'): 32, Timestamp('2025-05-31 16:23:00+0000', tz='UTC'): 33, Timestamp('2025-05-31 16:24:00+0000', tz='UTC'): 34, Timestamp('2025-05-31 16:25:00+0000', tz='UTC'): 35, Timestamp('2025-05-31 16:26:00+0000', tz='UTC'): 36, Timestamp('2025-05-31 16:27:00+0000', tz='UTC'): 37, Timestamp('2025-05-31 16:28:00+0000', tz='UTC'): 38, Timestamp('2025-05-31 16:29:00+0000', tz='UTC'): 39, Timestamp('2025-05-31 16:30:00+0000', tz='UTC'): 40, Timestamp('2025-05-31 16:31:00+0000', tz='UTC'): 41, Timestamp('2025-05-31 16:32:00+0000', tz='UTC'): 42, Timestamp('2025-05-31 16:33:00+0000', tz='UTC'): 43, Timestamp('2025-05-31 16:34:00+0000', tz='UTC'): 44, Timestamp('2025-05-31 16:35:00+0000', tz='UTC'): 45, Timestamp('2025-05-31 16:36:00+0000', tz='UTC'): 46, Timestamp('2025-05-31 16:37:00+0000', tz='UTC'): 47, Timestamp('2025-05-31 16:38:00+0000', tz='UTC'): 48, Timestamp('2025-05-31 16:39:00+0000', tz='UTC'): 49, Timestamp('2025-05-31 16:40:00+0000', tz='UTC'): 50, Timestamp('2025-05-31 16:41:00+0000', tz='UTC'): 51, Timestamp('2025-05-31 16:42:00+0000', tz='UTC'): 52, Timestamp('2025-05-31 16:43:00+0000', tz='UTC'): 53, Timestamp('2025-05-31 16:44:00+0000', tz='UTC'): 54, Timestamp('2025-05-31 16:45:00+0000', tz='UTC'): 55, Timestamp('2025-05-31 16:46:00+0000', tz='UTC'): 56, Timestamp('2025-05-31 16:47:00+0000', tz='UTC'): 57, Timestamp('2025-05-31 16:48:00+0000', tz='UTC'): 58, Timestamp('2025-05-31 16:49:00+0000', tz='UTC'): 59, Timestamp('2025-05-31 16:50:00+0000', tz='UTC'): 60, Timestamp('2025-05-31 16:51:00+0000', tz='UTC'): 61, Timestamp('2025-05-31 16:52:00+0000', tz='UTC'): 62, Timestamp('2025-05-31 16:53:00+0000', tz='UTC'): 63, Timestamp('2025-05-31 16:54:00+0000', tz='UTC'): 64, Timestamp('2025-05-31 16:55:00+0000', tz='UTC'): 65, Timestamp('2025-05-31 16:56:00+0000', tz='UTC'): 66, Timestamp('2025-05-31 16:57:00+0000', tz='UTC'): 67, Timestamp('2025-05-31 16:58:00+0000', tz='UTC'): 68, Timestamp('2025-05-31 16:59:00+0000', tz='UTC'): 69, Timestamp('2025-05-31 17:00:00+0000', tz='UTC'): 70, Timestamp('2025-05-31 17:01:00+0000', tz='UTC'): 71, Timestamp('2025-05-31 17:02:00+0000', tz='UTC'): 72, Timestamp('2025-05-31 17:03:00+0000', tz='UTC'): 73, Timestamp('2025-05-31 17:04:00+0000', tz='UTC'): 74, Timestamp('2025-05-31 17:05:00+0000', tz='UTC'): 75, Timestamp('2025-05-31 17:06:00+0000', tz='UTC'): 76, Timestamp('2025-05-31 17:07:00+0000', tz='UTC'): 77, Timestamp('2025-05-31 17:08:00+0000', tz='UTC'): 78, Timestamp('2025-05-31 17:09:00+0000', tz='UTC'): 79, Timestam

```
p('2025-05-31 17:10:00+0000', tz='UTC'): 80, Timestamp('2025-05-31 17:11:00+
0000', tz='UTC'): 81, Timestamp('2025-05-31 17:12:00+0000', tz='UTC'): 82, T
imestamp('2025-05-31 17:13:00+0000', tz='UTC'): 83, Timestamp('2025-05-31 1
7:14:00+0000', tz='UTC'): 84, Timestamp('2025-05-31 17:15:00+0000', tz='UT
C'): 85, Timestamp('2025-05-31 17:16:00+0000', tz='UTC'): 86, Timestamp('202
5-05-31 17:17:00+0000', tz='UTC'): 87, Timestamp('2025-05-31 17:18:00+0000',
tz='UTC'): 88, Timestamp('2025-05-31 17:19:00+0000', tz='UTC'): 89, Timestam
p('2025-05-31 17:20:00+0000', tz='UTC'): 90, Timestamp('2025-05-31 17:21:00+
0000', tz='UTC'): 91, Timestamp('2025-05-31 17:22:00+0000', tz='UTC'): 92, T
imestamp('2025-05-31 17:23:00+0000', tz='UTC'): 93}
```

In [54]:
```
df
```

Out[54]:

	timestamp	src_ip	dest_ip	proto	signature	sid	hour	minute
0	2025-05-31 15:50:32.260025+00:00	0	172.28.0.100	0	TCP SYN packet detected	4	15	50
1	2025-05-31 15:50:32.526292+00:00	0	172.28.0.100	0	TCP SYN packet detected	4	15	50
2	2025-05-31 15:50:32.776781+00:00	0	172.28.0.100	0	TCP SYN packet detected	4	15	50
3	2025-05-31 15:50:33.033113+00:00	0	172.28.0.100	0	TCP packet detected	0	15	50
4	2025-05-31 15:50:33.033113+00:00	0	172.28.0.100	0	TCP SYN packet detected	4	15	50
...	...	...	...	...	...	...	...	...
91973	2025-05-31 16:28:07.909940+00:00	4	172.28.0.100	0	TCP SYN packet detected	4	16	28
91974	2025-05-31 16:28:07.910021+00:00	4	172.28.0.100	0	TCP packet detected	0	16	28
91975	2025-05-31 16:28:07.910021+00:00	4	172.28.0.100	0	TCP SYN packet detected	4	16	28
91976	2025-05-31 16:28:07.910106+00:00	4	172.28.0.100	0	TCP packet detected	0	16	28
91977	2025-05-31 16:28:07.910106+00:00	4	172.28.0.100	0	TCP SYN packet detected	4	16	28

91978 rows × 15 columns

4. Visualização dos Dados

Na próxima célula será feita a standardização dos dados para que possamos fazer a execução de um algoritmo não-supervisonado, que servirá apenas para visualização dos dados. A standardização dos dados corresponde ao processo de transformar os dados de maneira a que um tipo de dados não tenha mais significado que outro. Por

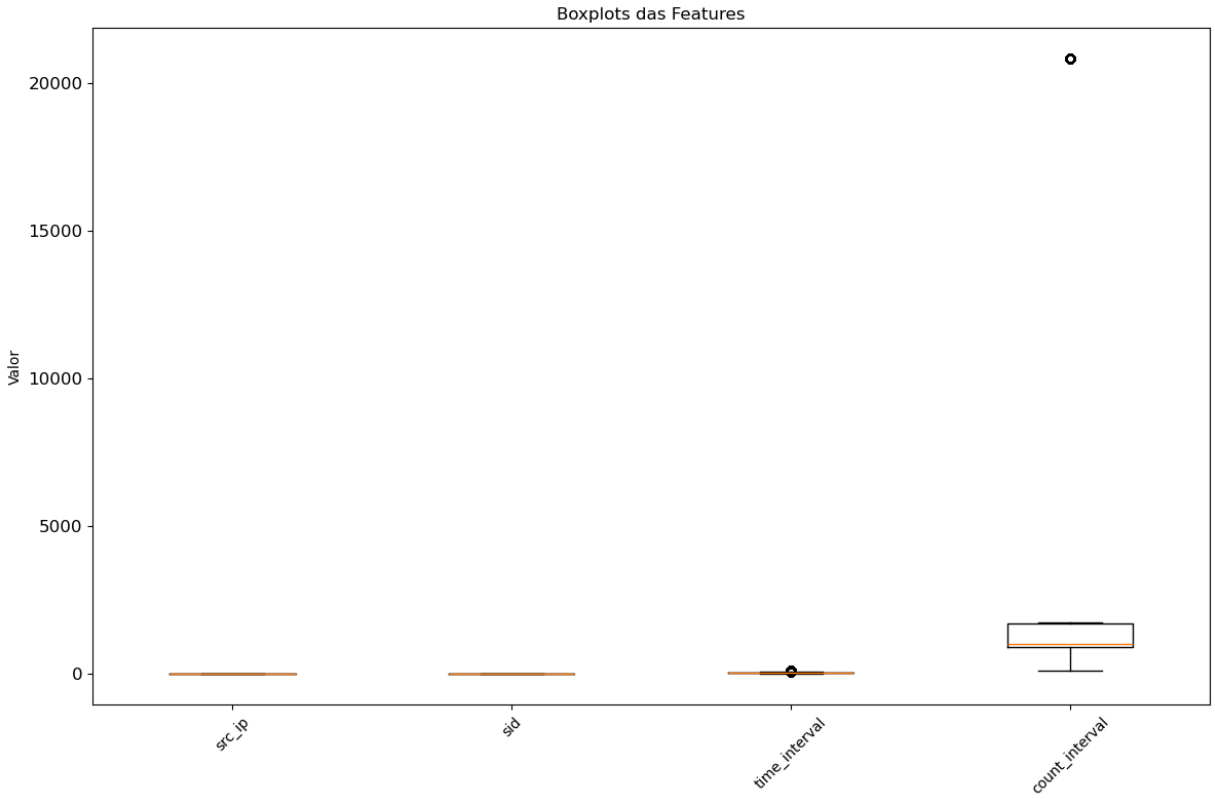
exemplo, podemos ter uma coluna com dados entre 5 e 6 e outra coluna com dados entre 100000 e 500000, e a standardização vai colocar ambos os dados entre 0 e 1 para que tenham a mesmo significado de valores. Desta forma não teremos nenhuma coluna com dados que influencie o algoritmo de forma mais significativa que outra.

In [25]:
```
scaler2 = MinMaxScaler2()
scaler = MinMaxScaler()
features = df[[ 'src_ip', 'sid', 'time_interval', 'count_interval']]
df_treino=df[[ 'src_ip', 'sid', 'time_interval','count_interval']]
X_scaled0 = scaler.fit_transform(features)
X_scaled = scaler2.fit_transform(features)
```

4.1 Visualização de Outliers

Na próxima célula encontra-se um gráfico de extremos e quartis para cada coluna, de forma a visualizarmos os outliers.

In [26]:
```
import matplotlib.pyplot as plt
features2 = [ 'src_ip', 'sid', 'time_interval', 'count_interval']

plt.figure(figsize=(12, 8))
plt.boxplot([df_treino[feature].dropna() for feature in features2], tick_lab
plt.xticks(rotation=45, fontsize=10)
plt.yticks(fontsize=12)
plt.ylabel('Valor')
plt.title('Boxplots das Features')
plt.tight_layout()
plt.show()
```



No gráfico acima observa-se que apenas as colunas de time_interval e count_interval contêm outliers. O outlier da coluna count_interval corresponde exatamente a um periodo de intervalo com uma contagem de pacotes que excede os restantes periodos de tempo

4.2 Algoritmo Não-Supervisionado UMAP

In [74]:
```
%%time
#import umap
# Umap sem gpu correu 1600 segundos ao qual acabei por cancelar
# Umap com gpu demorou 25 segundos


gdf = cudf.from_pandas(df_treino)
umap = UMAP2(random_state=42, n_neighbors=20, min_dist=0.2)
embedding = umap.fit_transform(gdf.values)
embedding.shape
```

```
CPU times: user 1.62 s, sys: 0 ns, total: 1.62 s
Wall time: 1.62 s
```

Out[74]: (91978, 2)

In [75]:
```
embedding_np = embedding.get()
# Converter embedding para DataFrame
embedding_df = pd.DataFrame(embedding_np, columns=['UMAP1', 'UMAP2'])

# Adicionar as features originais para o hover (supondo que X_scaled é um Da
for col in df_treino.columns:
```

```
    embedding_df[col] = df_treino[col].values

# Scatter plot interativo com hover das features
fig = px.scatter(
    embedding_df,
    x='UMAP1',
    y='UMAP2',
    hover_data=df_treino.columns,  # mostra as colunas das features no hover
    title='UMAP Dimensionality Reduction - Interactive',
    color=None  # ou podes colorir por alguma feature
)

fig.update_layout(
    xaxis_title='UMAP1',
    yaxis_title='UMAP2',
    hovermode='closest'
)

fig.show()
```

Na figura acima é observável a nossa distribuição de dados. No centro observa-se um aglormerado de dados bastante concentrado, enquando existem vários pontos à volta desse aglomerado. Ao passar o rato por cima deste conjunto de dados podemos

observar os valores do count_interval, e reparar que nos pontos à volta o valor é sempre 20833 enquanto no centro do aglomerado de dados, este valor é significativamente menor e varia de ponto para ponto.

5. Algoritmo Não-Supervisionado IsolationForest - Deteção de Outliers

5.1 Preparação

Antes de executar-mos o algoritmo, vamos criar uma nova tabela de dados, a partir dos dados originais, focada na contagem de pacotes por intervalo de tempo, agrupadas pelo IP. Para isso teremos a contagem de pacotes por id de assinatura, a média da contagem de pacotes por intervalo de tempo, a média de intervalo de tempo entre pacotes e o valor máximo da contagem de pacotes. Desta forma irá nos permitir mais facilmente perceber qual o IP que corresponde ao IP maligno.

In [22]:
```
agg_df = df.groupby('src_ip').agg({
    'sid': 'count',  # total de pacotes
    'count_interval': ['mean', 'max'],  # média e pico de pacotes por segund
    'delta': ['mean', 'std'],  # intervalo médio e variação
    'pps': 'max',
})

# Mudar os nomes das colunas
agg_df.columns = ['_'.join(col) for col in agg_df.columns]
agg_df.reset_index(inplace=True)
```

In [23]:
```
# Cria colunas de 0/1 para cada tipo de sid
for sid_code in df['sid'].unique():
    df[f'sid_{sid_code}'] = (df['sid'] == sid_code).astype(int)

# Agrega proporções por src_ip
sid_props = df.groupby('src_ip')[[f'sid_{sid}' for sid in df['sid'].unique()

# Junta ao agg_df
agg_df = agg_df.merge(sid_props, on='src_ip')
```

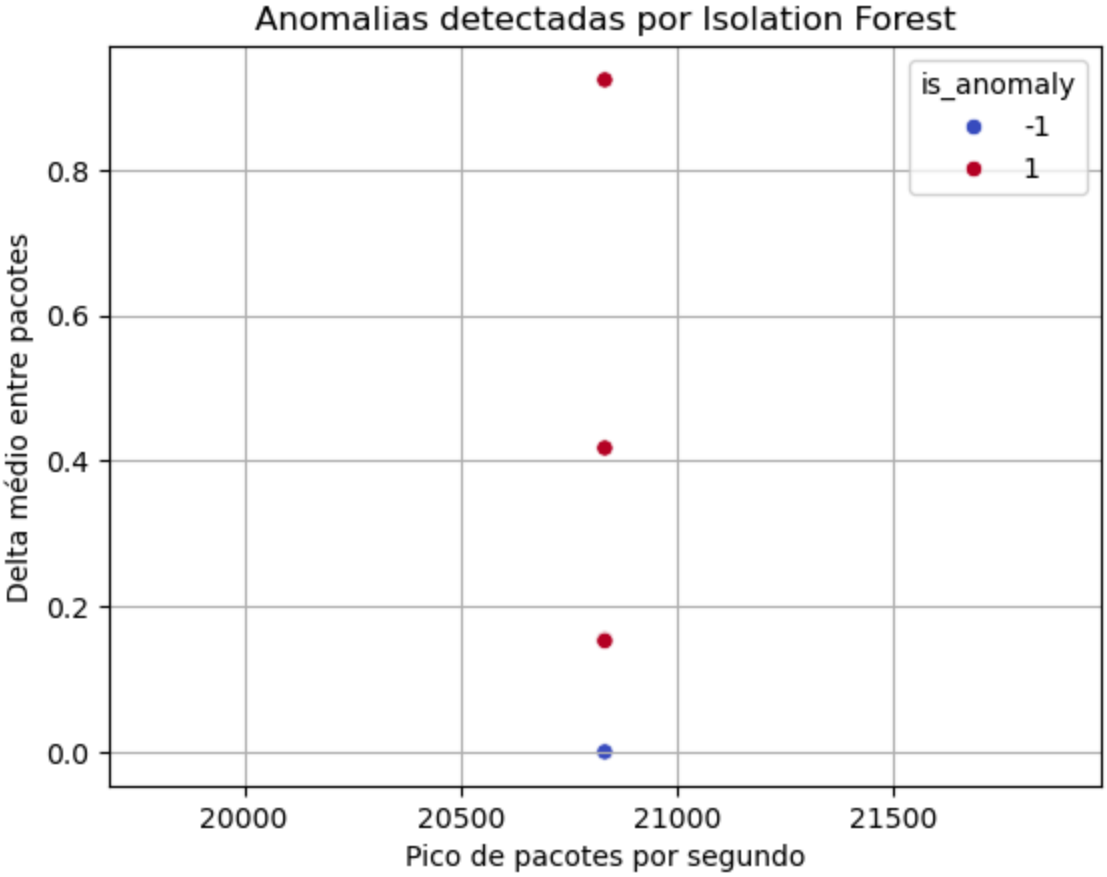
5.2 Execução Do Algoritmo

In [24]:
```
# Remove src_ip da normalização
X = agg_df.drop(columns=['src_ip'])

scaler = StandardScaler()
X_scaled3 = scaler.fit_transform(X)

clf = IsolationForest(contamination=0.1, random_state=42)
clf.fit(X_scaled3)
```

```
agg_df['anomaly_score'] = clf.decision_function(X_scaled3)
agg_df['is_anomaly'] = clf.predict(X_scaled3)  # -1 = anomalia, 1 = normal
```

5.3 Visualização de resultados

In [25]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.scatterplot(data=agg_df, x='count_interval_max', y='delta_mean', hue='is
plt.title('Anomalias detectadas por Isolation Forest')
plt.xlabel('Pico de pacotes por segundo')
plt.ylabel('Delta médio entre pacotes')
plt.grid(True)
plt.show()
```



Observa-se no gráfico em cima que os 4 pontos, correspondentes a cada IP, têm um pcio de pacotes por segundo semelhante, no entanto o ponto mais abaixo destaca-se por ter um delat médio entre pacotes significativamente menor. Assim, este ultimo com a cor azul, foi classificado como anomalia

In [26]:
```
agg_df
```

Out[26]:

	src_ip	sid_count	count_interval_mean	count_interval_max	delta_mean	delta_
0	172.28.0.11	26119	1263.285271	20833	0.154058	0.085
1	172.28.0.12	6044	971.726340	20833	0.923329	0.345
2	172.28.0.13	26471	1262.555891	20833	0.152593	0.615
3	172.28.0.15	13345	971.400375	20833	0.417556	1.378
4	172.28.0.20	19999	20833.000000	20833	0.000040	0.000

Na tabela acima pode-se observar os valores para cada coluna, onde o dispositivo com o IP 172.28.0.20 foi classificado como anomalia "-1".

In [ ]: