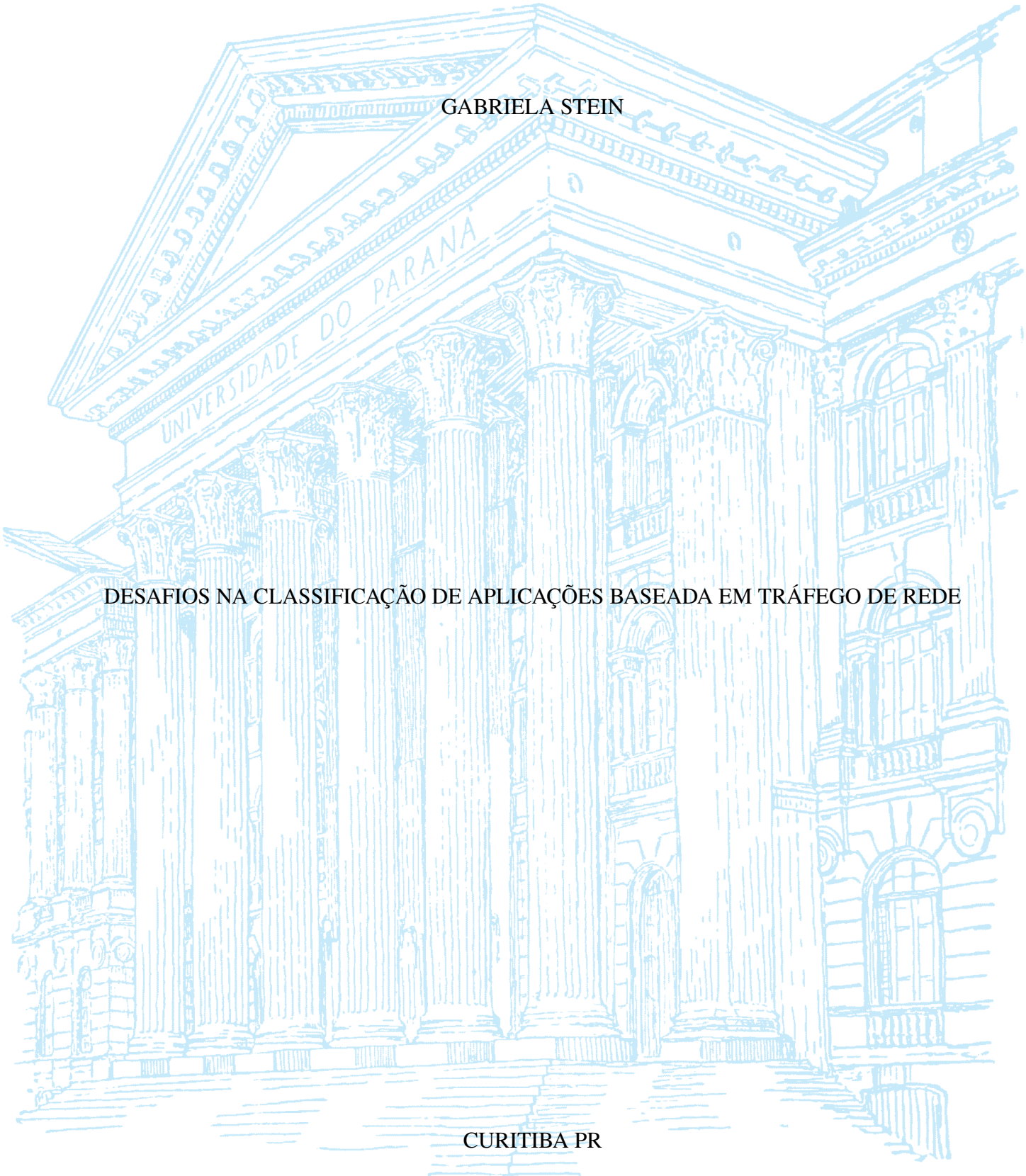
UNIVERSIDADE FEDERAL DO PARANÁ

GABRIELA STEIN



DESAFIOS NA CLASSIFICAÇÃO DE APLICAÇÕES BASEADA EM TRÁFEGO DE REDE

CURITIBA PR

2021

GABRIELA STEIN

DESAFIOS NA CLASSIFICAÇÃO DE APLICAÇÕES BASEADA EM TRÁFEGO DE REDE

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Prof. Dr. André R. A. Grégio.

CURITIBA PR

2021

A minha família.

AGRADECIMENTOS

Agradeço aqueles que estiveram comigo, família, amigos e meu amor. Sempre me apoiando e incentivando.

RESUMO

A análise e a classificação de tráfego de rede fomenta inúmeras pesquisas devido as suas aplicações em um amplo conjunto de problemas, já que podem auxiliar a prover uma comunicação de rede segura, confiável e de qualidade. Muitas soluções são desenvolvidas para resolver problemas de segurança e uso adequado de redes, como classificadores que independem de inspeção profunda de pacotes e que seja baseada apenas em informações de fluxo, evitando o acesso ao conteúdo de pacotes. Várias técnicas na literatura são propostas e experimentadas para analisar o tráfego de rede, incluindo técnicas baseadas em aprendizado de máquina.

Considerando os cenários citados e os desafios na resolução de problemas envolvendo aprendizado de máquina aplicado a tráfego de rede, bem como a reprodutibilidade de trabalhos da literatura, o presente trabalho tem por foco analisar e reproduzir os resultados de classificação de aplicações de tráfego de rede utilizados por Jan Pluskal, Ondrej Lichtner, Ondrej Rysavy no artigo *Traffic Classification and Application Identification in Network Forensics*, apresentado em 2018 na 14th IFIP International Conference on Digital Forensics, implementando um classificador baseado em *Naive Bayes* e outro em *Random Forest* com os parâmetros disponibilizados, e comparando-os com os resultados obtidos no artigo base e com o método ESPI nele apresentado. Além disso, a seleção de tráfego de rede TCP e treinamento de modelos será feita para verificar se os algoritmos são adequados para segregação de tráfego especializada por aplicação e quais as dificuldades encontradas durante o processo.

Palavras-chave: Análise de Tráfego de Rede. Classificação de Aplicações. Aprendizado de Máquina.

ABSTRACT

Network traffic analysis and classification fosters several research works aiming to solve a broad set of applied problems, since these tasks may help to provide a better, more secure, trusted and reliable communication. A plethora of solutions have been developed to solve issues regarding network security and proper usage (e.g., classifiers that do not need to rely on deep packet inspection, or flow-based classifiers that avoid needing access to packets' payload). Novel techniques have been proposed and tested to analyze network traffic using machine learning algorithms.

Considering the aforementioned scenarios and the current challenges on tackling problems related to the application of machine learning to network traffic, as well as the challenges on reproducing related works, in this work we focus on analyzing and replicating the experiments on Application Traffic Classification presented by Jan Pluskal, Ondrej Lichtner, and Ondrej Rysavy in the article "Traffic Classification and Application Identification in Network Forensics" (presented in 14th IFIP International Conference on Digital Forensics, 2018). The authors implemented classifiers based on Naive Bayes, Random Forest, and a custom one named ESPI, and we tested on the same dataset with our own parameters for Naive Bayes and Random Forest, and then compared with their results (including ESPI) to discuss. We also performed feature selection and filtering TCP-only traffic to verify whether those techniques are suitable to specialized traffic classification or not, and what are the challenges faced during this process.

Keywords: Network Traffic Analysis. Classification of Applications. Machine Learning.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

DINF	Departamento de Informática
PPGINF	Programa de Pós-Graduação em Informática
UFPR	Universidade Federal do Paraná
ISP	*Internet Service Provider* (Provedor de Serviço de Internet)
TIE	*Traffic Identification Engine*
VPN	*Virtual Private Networks*
KNN	*K-Nearest Neighbors Algorithm*
SVM	*Support Vector Machines*
SOM	*Self-Organizing Maps*
PCAP	*Packet Capture Data*

LISTA DE SÍMBOLOS

α	alfa, primeira letra do alfabeto grego
β	beta, segunda letra do alfabeto grego
γ	gama, terceira letra do alfabeto grego
ω	ômega, última letra do alfabeto grego
π	pi
τ	Tempo de resposta do sistema
θ	Ângulo de incidência do raio luminoso

SUMÁRIO

1 INTRODUÇÃO

Com o crescimento da Internet, a implementação de novos serviços de segurança e o surgimento constante de novas aplicações, a classificação de tráfego de rede se torna um desafio cada dia mais complexo, o que acarreta na pesquisa e subsequente implementação de diversas técnicas de classificação (Callado et al., 2009). Devido a extensão de formatos, protocolos e a complexidade dos dados que circulam na internet, é possível identificar que são muitas as abordagens de resolução desse problema. A classificação de tráfego também é utilizada no monitoramento de rede, análise de segurança (detecção de intrusão) e perícia forense computacional (linha do tempo de eventos de ataque, por exemplo) (Nguyen e Armitage, 2008).

A classificação de tráfego de rede é importante para auxiliar na identificação de tráfego de rede visando efetuar diversas tarefas de administração e segurança. Organizações, provedores de Internet (ISPs) e empresas/órgãos de segurança podem precisar saber informações sobre os protocolos de aplicação em tráfego, tais como qual aplicação está sendo mais utilizada, qual ocupa mais banda, qual deve ser bloqueada ou autorizada, entre outras. Portanto, faz-se necessária a aplicação de técnicas de classificação de forma automática e eficaz, seja para triagem de dados de forma a se diminuir o escopo de análise de uma massa grande de tráfego (por exemplo, uma perícia em Terabytes de pacotes pode se beneficiar da identificação de protocolos de aplicação de interesse para focar na investigação mais aprofundada dessas sessões de tráfego), ou para filtragem de tráfego (por exemplo, uma escola que necessita garantir que seus estudantes não estão fazendo acesso a *sites* maliciosos ou de conteúdo ilegal/prejudicial).

Para isso, tal classificação precisa considerar o direito à privacidade do usuário gerador do tráfego de rede, bem como potenciais restrições ao acesso ao conteúdo das mensagens de aplicação trafegadas. Portanto, as técnicas aplicadas podem observar a taxa de eficácia da classificação de tráfego usando somente informações dos cabeçalhos dos pacotes. O desafio então passa a ser separar sessões de tráfego de aplicações específicas e identificar os protocolos de aplicação sem informações do *payload*. Além da confidencialidade, limitar-se a utilização das informações constantes apenas nos cabeçalhos permite maior rapidez na tarefa de classificação, uma vez que o dispositivo de análise não precisa interpretar dados de camadas acima da camada de transporte.

Há diversos exemplos na literatura do uso de aprendizado de máquina para classificação de tráfego de rede (Ariu et al., 2011). Entretanto, os parâmetros utilizados nos algoritmos, os *datasets* e/ou as próprias taxas de resultados obtidas não são em geral mostrados por completo, corretamente interpretados ou suficientemente discutidos para que uma avaliação justa de sua aplicabilidade possa ser feita. Para mostrar o impacto do referido problema da replicação da literatura e avaliar o potencial do uso de técnicas de aprendizado de máquina para classificação de protocolos de aplicação utilizando exclusivamente informações do cabeçalho, escolheu-se reproduzir um artigo com esse foco (e em segurança, mais especificamente, em forense digital) (Pluskal et al., 2018). A motivação pela escolha do artigo é a disponibilização do tráfego de rede completo (em estado bruto) e do código-fonte usado nos classificadores, o que permitiu não só a re-extração de características para criação do *dataset*, mas a verificação dos parâmetros desses algoritmos e potencial extensão do trabalho citado.

Assim, no presente trabalho, propõe-se realizar um comparativo mais detalhado (no sentido de apresentação de taxas de resultados) entre os algoritmos tradicionais de classificação de tráfego de rede utilizados por Jan Pluskal, Ondrej Lichtner, Ondrej Rysavy no artigo *Traffic Classification and Application Identification in Network Forensics*, apresentado em 2018 na

14th IFIP International Conference on Digital Forensics. Pluskal et al. (Pluskal et al., 2018) apresentam um método de identificação de protocolo estatístico aprimorado (ESPI) e clamam superar classificadores tradicionais de aprendizado de máquina. Para tanto, além de descrever o seu método, os autores o comparam com os classificadores de rede Bayesiana (*Naive Bayes*) e *Random Forest*, discutindo seu uso para identificar os protocolos da camada de aplicação e até mesmo os aplicativos que fazem uso desses protocolos.

A partir disso e com base no banco de dados de 20 GB de tráfego de rede disponível em `https://github.com/pluskal/AppIdent`, este trabalho tem por objetivo a reprodução dos resultados de (Pluskal et al., 2018) via implementação de um classificador baseado em *Naive Bayes* e outro em *Random Forest* utilizando apenas os parâmetros disponibilizados, considerando que o código fonte providenciado pelos autores não pôde ser reaproveitado, e a comparação com os resultados obtidos no referido artigo, assim como com o método ESPI nele apresentado. Além disso, será feita a seleção de tráfego de rede TCP e treinamento de modelos para verificar se os algoritmos são adequados para segregação de tráfego especializada por aplicação. Os experimentos foram feitos com todas as características definidas no artigo de origem, bem como com *dataset* cujas características foram pré-selecionadas por Pluskal e com a aplicação de seleção de características própria feita no presente trabalho. Por fim, os resultados obtidos são comparados e discutidos, para verificação da viabilidade na aplicação dessas técnicas no escopo do problema levantado e tomada de conclusões sobre a reprodução do trabalho de outros pesquisadores.

Este documento está organizado da seguinte maneira: no Capítulo 2, apresenta-se os conceitos básicos utilizados neste trabalho, bem como a fundamentação teórica necessária e os trabalhos relacionados. O Capítulo 3 reúne detalhes sobre a proposta de trabalho, o projeto desenvolvido e a metodologia utilizada. Já o Capítulo 4 apresenta detalhes sobre a implementação, os experimentos e resultados. Finalmente, o Capítulo 5 traz as conclusões e considerações finais deste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Nesse capítulo são introduzidos os conceitos fundamentais relacionados a este trabalho, bem como são apresentados os trabalhos relacionados aos algoritmos e ferramentas de classificação e análise de tráfego que estão presentes na literatura.

2.1 ESTADO DA ARTE

A problemática da classificação do tráfego de rede tem crescido conforme a complexidade dessas redes e das aplicações que a utilizam também estão em crescente. A utilização de protocolos de segurança, criptografia dos dados e encapsulamento das aplicações dificultam a exploração deste tráfego de rede. Procurando identificar corretamente e lidar com esses obstáculos à análise e classificação do tráfego, a literatura traz abordagens de algoritmos e ferramentas, com diferentes metodologias.

Considerando o elevado volume de dados envolvido no tráfego de rede, Cunha et al. utilizam a ferramenta *TIE - Traffic Identification Engine*, que fornece uma plataforma de desenvolvimento e integração de técnicas de classificação de tráfego por meio de *plug-ins,* e têm o objetivo de explorar e comparar a acurácia de diferentes estratégias de classificação em conjunto com técnicas de amostragem, de modo a identificar se uma pequena proporção de tráfego fornece uma visão realista da rede (Cunha et al., 2015).

A evolução contínua da Internet e a geração de novos aplicativos e serviços, juntamente com a expansão das comunicações criptografadas torna a classificação de tráfego de rede uma tarefa difícil. *Virtual Private Networks* (VPNs) são um exemplo de serviço de comunicação criptografada que está se tornando popular, como método para contornar a censura, bem como acessar serviços que estão geograficamente bloqueados. Draper-Gil et al. focam na classificação e identificação de tráfegos encriptados e VPNs utilizando um método baseado em fluxo, com foco nas características temporais, caracterizando o tráfego criptografado em diferentes categorias, de acordo com o tipo de tráfego. Além disso, utilizam duas técnicas de aprendizado de máquina (C4.5 e KNN) para testar a precisão das características escolhidas pelo seu método e publicam um *dataset* de tráfego encriptado com 14 rótulos (Draper-Gil et al., 2016).

É possível identificar na literatura, diversas abordagens baseadas na inspeção do *payload* (carga útil) dos dados transmitidos na rede, como por exemplo, a procura de padrões dentro dos fluxos TCP. Todavia, essas técnicas podem ser lentas e seu funcionamento dificultado pelas técnicas de criptografia. Recentemente, o estudo de métodos estatísticos baseados no fluxo de informações para a classificação de tráfego em tempo real se tornou significativo. Gómez Sena e Belzarena apresentam um método de classificação que analisa o tamanho dos primeiros pacotes em ambas as direções de um fluxo e utiliza uma técnica de aprendizado de máquina supervisionado: *Support Vector Machines* (SVM) concentrando na classificação de tráfego *on-line* de modo a encontrar um método de análise estatística simples e rápido. Entretanto, essa abordagem necessita de um tráfego já pré-classificado para seu funcionamento (Gómez Sena e Belzarena, 2009).

Além da utilização de técnicas de criptografia, a utilização de protocolos de tunelamento dificulta a recuperação de informações adquiridas dentro de dados de rede. Quando a criptografia de ponta-a-ponta é utilizada, apesar de ser perdido o conteúdo, é possível identificar as conexões individuais. No tunelamento várias conexões são multiplexadas no túnel, interconectando redes através de uma possível rede com tecnologia incompatível. O conteúdo capturado do túnel pode

ser facilmente extraído e conexões individuais recuperadas. Atualmente, os protocolos modernos de tunelamento criptografam seus dados, de maneira que as informações só podem ser capturadas na saída do túnel. Pluskal et al. fornece um estudo de caso genérico de encapsulamento de fluxo e descreve como o investigador pode obter dados de aplicativos encapsulados de dentro do pacote, além de apresentar uma visão geral dos protocolos de tunelamento mais usados e seus recursos relativos à análise forense digital (Pluskal et al., 2019).

No contexto da utilização da classificação de tráfego de rede para a segurança computacional e análise forense de rede, na qual são adquiridas as informações de modo a litigar atividades criminosas e impedir incidentes de segurança, deve-se considerar a utilização de ferramentas de análise já consolidadas dentro da área, como *Wireshark*, *Network Miner*, *Xplico*, entre outras. Contudo, essas ferramentas têm suas limitações, como a falta de um processamento automático de dados ou a reconstrução bem-sucedida da comunicação quando os dados estão corrompidos. Pluskal et al. apresentam a ferramenta forense de rede *Netfox Detective* que tem sua aplicação no *reassembly* preciso de dados e seu fluxo, análise e reconstrução de aplicativos, descrevendo sua arquitetura e implementação. A ferramenta é projetada para reconstrução e análise de dados de rede e utiliza heurísticas e técnicas avançadas para *assembly* de dados, identificação de tráfego na camada de aplicação, reconstrução de conversas e apresentação (Pluskal et al., 2015). Levatay et al. trazem uma abordagem distribuída de ferramenta forense de análise de tráfego utilizando o *Network Traffic Processing & Analysis Cluster* (NTPAC) (Letavay et al., 2019).

É possível observar, a partir da leitura dos artigos citados acima, a imensidão de trabalhos que podem ser realizados dentro do campo da análise e classificação de tráfego de rede e as abordagens variam conforme o problema que o pesquisador planeja solucionar. O presente trabalho se coloca com o objetivo de realizar um comparativo dos algoritmos de classificação utilizados no trabalho base *Classification and Application Identification in Network Forensics* de PLuskal et al. (Pluskal et al., 2018). Ainda, por meio da replicação de um artigo da área, pode-se tirar conclusões referentes a viabilidade de reprodução das técnicas e dos experimentos descritos, além de constatar se os métodos aplicados são capazes de realizar a segregação de tráfego por aplicação.

2.2 CONCEITOS FUNDAMENTAIS

2.2.1 Aprendizado de Máquina

O aprendizado de máquinas é uma área da Inteligência Artificial na qual o maior objetivo é a construção e o desenvolvimento de técnicas computacionais com a habilidade de aprender por experiência (Monard e Baranauskas, 2003). Ou seja, constrói um algoritmo que é capaz de tomar decisões ou realizar classificações baseado em técnicas de reconhecimento de padrões, utilizando o conjunto de dados fornecido e as soluções anteriores ao problema dado.

Aplicações de aprendizado de máquinas podem ser úteis dentro das mais variadas áreas, como, por exemplo, no prognóstico de câncer, identificando a progressão e consequentemente o diagnóstico, assim auxiliando as decisões a serem tomadas pela equipe médica (Kourou et al., 2015). Além disso, pode ser aplicado na análise de conjuntos de dados de sequenciamento do genoma, incluindo a anotação de elementos de sequência e dados epigenéticos, proteômicos ou metabolômicos (Libbrecht e Noble, 2015).

Considerando a extensão das aplicabilidades da aprendizagem de máquina, um campo de pesquisa como a segurança de redes de computadores também traz inúmeras possibilidades para a utilização destes algoritmos como a análise forense de documentos textuais e e-mails, análise de dados e eventos, classificação de fragmentos de arquivo e obviamente, a análise do tráfego da rede (Ariu et al., 2011).

Algoritmos de aprendizado de máquina extraem informação a partir de um conjunto de dados de treinamento no qual procuram um padrão relacionando entradas e saídas. Dependendo da forma como esses dados são fornecidos, os algoritmos são divididos normalmente em duas categorias: aprendizagem supervisionada e aprendizagem não supervisionada, aprofundadas nas próximas subseções.

2.2.1.1 Aprendizagem supervisionada

Os algoritmos de aprendizagem supervisionada relacionam uma saída com uma entrada com base em dados rotulados. Para cada saída é atribuído um rótulo, que pode ser um valor numérico ou uma classe. O algoritmo determina uma forma de prever qual o rótulo de saída com base em uma entrada informada. Os algoritmos utilizados nesse trabalho são o *Naive Bayes* (Rish et al., 2001) e *Random Forest* (Breiman, 2001), descritos a seguir.

- **Random Forest**

  Random forest é uma coleção (*ensemble*) de classificadores baseados em árvores de decisão, as árvores são treinadas utilizando uma seleção aleatória de exemplos e características (*booting*). O resultado de cada árvore é um voto para uma classe, utilizado na votação da decisão final. A Figura 2.1 mostra um exemplo do funcionamento do *Random Forest.*

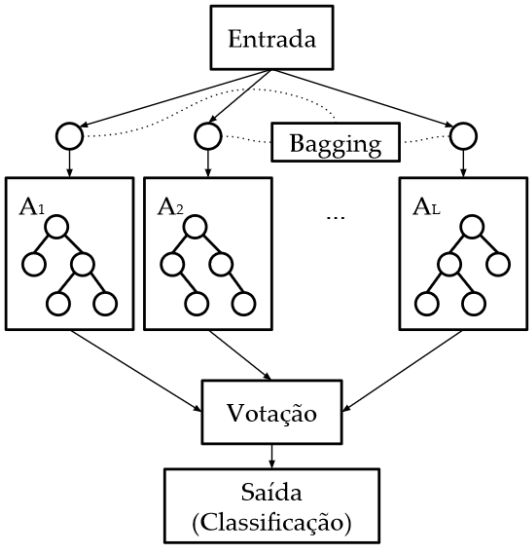


Figura 2.1: Exemplo de *Random Forest* (Ceschin et al., 2019).

- **Naive Bayes**

  Naive Bayes é um classificador de probabilidade. Nesse algoritmo está sendo procurada a hipótese h com maior probabilidade de acontecimento, baseando-se em um conjunto de dados d. É utilizado o teorema de Bayes, que permite calcular a probabilidade de um acontecimento, baseado em conhecimento pré-adquirido. Sua fórmula está descrita abaixo:

$$\Pr(h|d) = (\Pr(d|h) * \Pr(h))/\Pr(d) \tag{2.1}$$

  $\Pr(h|d)$ representa a probabilidade de acontecer a hipótese h, considerando d como verdade. $\Pr(d|h)$ é a probabilidade de d dado que h aconteceu, $\Pr(d)$ é a probabilidade

de d independente de h e $\Pr(h)$ é a probabilidade de h independente de d. Assim, calculamos a probabilidade de $\Pr(h|d)$ de acordo com os dados de entrada, para isso é utilizada uma tabela de frequência de dados, construindo uma tabela de probabilidades para cada entrada. Ao final, utilizando o teorema de Bayes, é calculada a probabilidade de cada evento e a com a maior probabilidade é escolhida.

2.2.1.2 Aprendizagem não supervisionada

No caso dos algoritmos de aprendizagem não supervisionada, não é atribuído um rótulo para os dados de saída. Com base em um número grande de dados, o algoritmo busca padrões e similaridades entre os dados, permitindo identificar grupos de itens similares ou similaridade de itens novos com grupos já definidos. Exemplos clássicos desse tipo de aprendizagem são os algoritmos *K-Means* (para problemas de *clustering*) (Sinaga e Yang, 2020) e *Self-Organizing Maps* (SOM) (Grégio e dos Santos, 2010), apresentados a seguir:

- **K-Means**

  Essa é uma técnica que consiste em classificar um conjunto de dados em um determinado número k de subconjuntos. Primeiro são definidos os k elementos centrais, que correspondem aos elementos característicos de um determinado *cluster*. Depois, cada item é avaliado e posicionado no *cluster* que contém o item mais semelhante a ele. O processo é repetido até que os elementos não troquem mais de lugar para assim definir os *clusters* finais.

- **Self-Organizing Maps**

  O objetivo principal de um SOM é transformar um padrão de sinal de entrada em um mapa discreto de uma ou duas dimensões, e realizar essa transformação de forma topologicamente ordenada (Bullinaria, 2004). O processo de auto-organização envolve quatro componentes principais:

  - Inicialização: Todos os pesos de conexão são inicializados com pequenos valores aleatórios.
  - Competição: Para cada padrão de entrada, os neurônios computam seus respectivos valores de uma função discriminante que fornece a base para a competição. O neurônio com o menor valor da função é o vencedor.
  - Cooperação: O neurônio vencedor determina a localização espacial de uma vizinhança topológica de neurônios ativados, fornecendo a base para cooperação entre neurônios vizinhos.
  - Adaptação: Os neurônios ativados diminuem seus valores individuais da função discriminante em relação ao padrão de entrada por meio de ajuste adequado dos pesos de conexão associados, de modo que a resposta do neurônio vencedor à aplicação subsequente de um padrão de entrada semelhante seja aprimorada.

  A partir disso, o método de *Self-Organizing Maps* cria uma rede que armazena informações de forma que qualquer relação topológica dentro dos dados de treinamento seja mantida.

3 METODOLOGIA

Neste capítulo é apresentada a proposta do trabalho que foi desenvolvido. A Seção 3.1 descreve o problema, relata a proposta do trabalho implementado e seu objetivo. Na Seção 3.2, o conjunto de dados utilizado é descrito.

3.1 PROPOSTA

O tráfego de rede pode ser analisado em diferentes níveis: no nível de pacote, nível de fluxo e nível de rede para gerenciamento de segurança. Pacote é a unidade que transita dentro de uma rede, a análise dele utiliza dados presentes no *payload* do pacote, ou seja é necessária uma inspeção profunda do conteúdo presente, enquanto a análise de fluxo pode utilizar somente os metadados presentes no cabeçalho para realizar a classificação de características do tráfego da rede. Uma estrutura genérica de análise envolve: pré-processamento dos dados, seguido por análises e observações para revelar padrões dos dados de rede (Joshi e Hadi, 2015). Uma estratégia para a classificação do tráfego de rede é a análise dos cabeçalhos ou da carga de pacotes, usada para identificar o tráfego associado a uma porta específica e, portanto, a um aplicativo específico (Callado et al., 2009).

Dentro das pesquisas que se dedicam a classificação do tráfego de rede, se têm o artigo *Traffic Classification and Application Identification in Network Forensics* de Jan Pluskal, Ondrej Lichtner, Ondrej Rysavy (Pluskal et al., 2018), que apresenta um método de identificação de protocolo estatístico aprimorado (ESPI) e o comparam com classificadores tradicionais, como o de rede Bayesiana (*Naive Bayes*) e *Random Forest*, discutindo seu uso para identificar os protocolos da camada de aplicação e os aplicativos que fazem uso desses protocolos.

De modo a verificar a aplicabilidade das técnicas aplicadas no artigo base além de discutir a reprodutividade de trabalhos de outros autores, foi proposta a implementação de um classificador baseado em *Naive Bayes* e outro em *Random Forest* com os parâmetros disponibilizados, além de realizar a comparação com os resultados obtidos no artigo de origem e com o método ESPI.

De forma a analisar se os algoritmos são adequados para segregação de tráfego especializada por aplicação será realizada a seleção de tráfego de rede TCP e treinamento de modelos. Os experimentos serão feitos com todas as características definidas no artigo de origem, bem como com *dataset* cujas características foram pré-selecionadas por Pluskal e com a aplicação de seleção de características própria feita no presente trabalho.

3.2 CONJUNTO DE DADOS

O banco de dados de 20 GB disponibilizado pelos autores no artigo base (Pluskal et al., 2018) foi escolhido para realizar todos os experimentos. A classificação de tráfego utiliza um arquivo de captura de tráfego de rede, comumente no formato PCAP. Neste *dataset*, o conjunto de dados foi capturado pelo *Microsoft Network Monitor*, que fornece rótulos de aplicativos para quase todas as conversas e contém tráfego de rede regular gerado por oito estações de trabalho de usuários executando o sistema operacional *Windows*. O arquivo final de captura tem essas características:

- **Tamanho do arquivo PCAP:** 19,5 GB.
- **Formato PCAP:** Microsoft NetMon 2.x.

- **Duração da captura:** 119 horas.
- **Número de pacotes:** 27.616.138.
- **Número de conversas da camada 7:** 269.459.
- **Número de Protocolos de Aplicação:** 58.
- **Número de aplicativos de comunicação:** 93

Os autores Pluskal et al., realizaram um pré-processamento após a captura dos dados, com o propósito de melhorar a extração de dados (Pluskal et al., 2018). Os passos utilizados no pré-processamento estão descritos em Matoušek et al. Netfox Detective (`github.com/nesfit/NetfoxDetective`), uma ferramenta criada de forma customizada para análise de tráfego, foi utilizada para reduzir o ruído de tráfego nas características extraídas. Além de abordar os problemas básicos relacionados ao processamento de conversas da camada de transporte, o *Netfox Detective* também permitiu que o conjunto de dados fosse processado para rastrear conversas da camada de aplicação. Isso aumentou a precisão da classificação ao identificar os padrões de comunicação dos aplicativos. Também foram eliminados os resquícios de fragmentação de pacotes de rede na camada da Internet e retransmissão TCP na camada de transporte.

Devido ao alto custo computacional, os passos de pré-processamento foram realizados somente uma vez e salvos em um arquivo binário. A partir deste arquivo, disponibilizado por Pluskal et al., foi extraído o *dataset* utilizado em todos os experimentos.

3.3 DISTRIBUIÇÃO DE CLASSES

A distribuição das classes presente no conjunto de dados com todas as características pode ser visualizada na Figura 3.1 que mostra a distribuição do *y_test*, os dados que serão utilizados para testar o classificador, enquanto na Figura 3.2 apresenta a distribuição das classes no *y_train* que serão utilizados para treinar o classificador. Todos os gráficos apresentados mostram a quantidade das 10 classes com maior quantidade no conjunto de dados, as tabelas com os dados inteiros estão disponíveis no Apêndice A. É possível identificar que temos um *dataset* muito desbalanceado, com algumas classes apresentando quantidade significativamente maior que outras. A divisão entre os dados foi 80% do *dataset* para treinamento e 20% para testes. As classes utilizadas foram as mesmas fornecidas pelos autores no próprio *dataset*, que utilizaram o *Network Monitor* para realizar a anotação das classes de acordo com um indicador da própria ferramenta, que traz a porta utilizada pelo protocolo de aplicação e o nome do protocolo. No eixo X dos gráficos abaixo temos a quantidade bruta de amostras pertencentes a determinada classe, representadas no eixo Y.

A Figura 3.3 mostra a distribuição do *y_test*, do *dataset* no qual o classificador utilizará apenas as características selecionadas pelos autores, enquanto na Figura 3.4 apresenta a distribuição das classes no *y_train*.

A Figura 3.5 mostra a distribuição do *y_test*, no classificador que utiliza todas as características iniciais, mas retira as amostras UDP, e na Figura 3.6 apresenta a distribuição das classes no *y_train*.

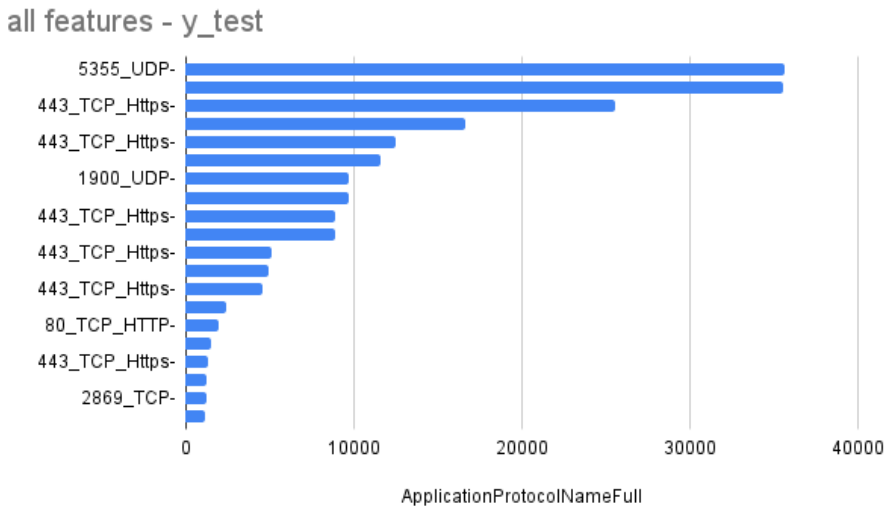


Figura 3.1: Gráfico todas as características - *y_test*

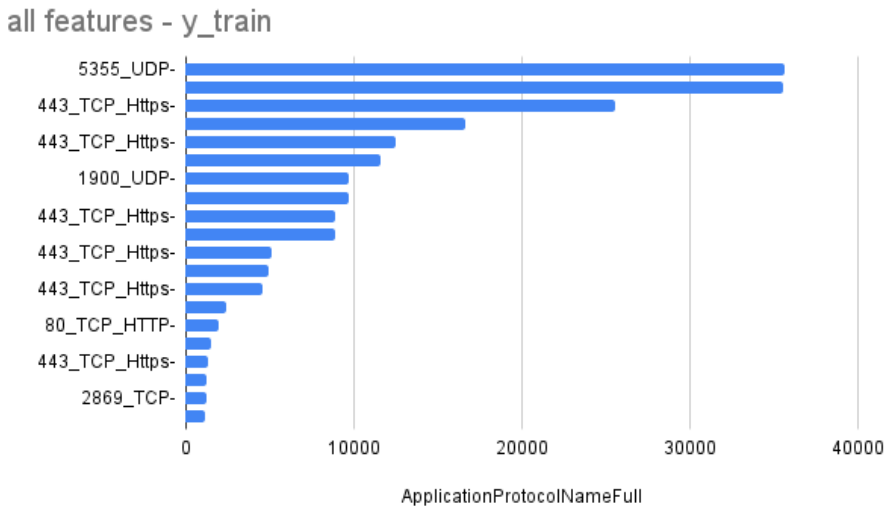


Figura 3.2: Gráfico todas as características - *y_train*

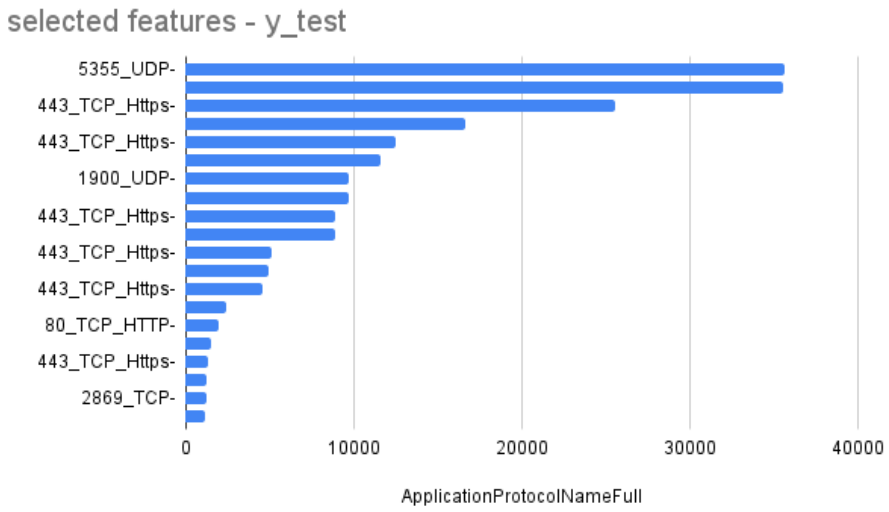


Figura 3.3: Gráfico com as 8 características - *y_test*

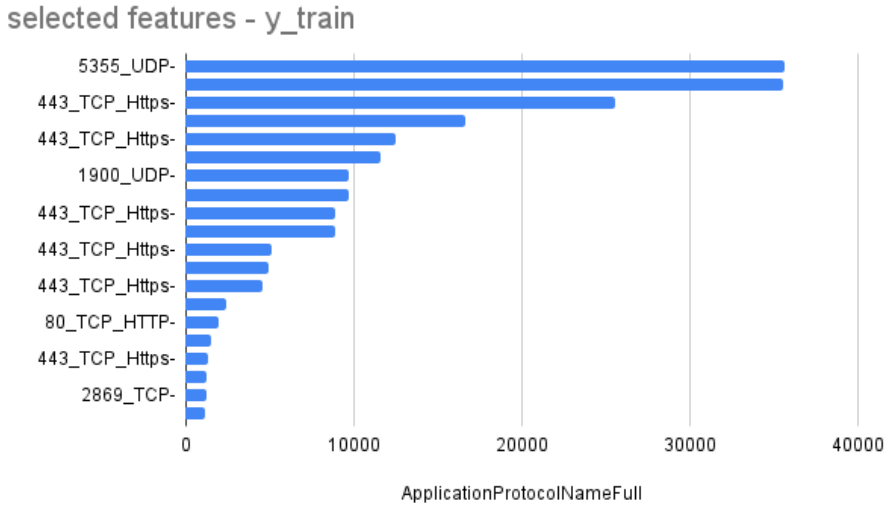


Figura 3.4: Gráfico com as 8 características - *y_train*

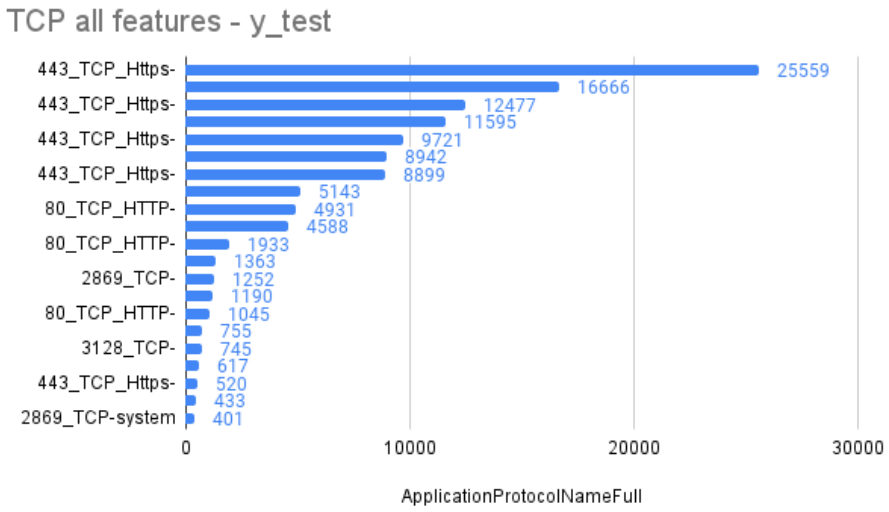


Figura 3.5: Gráfico com todas características do classificador TCP - *y_test*

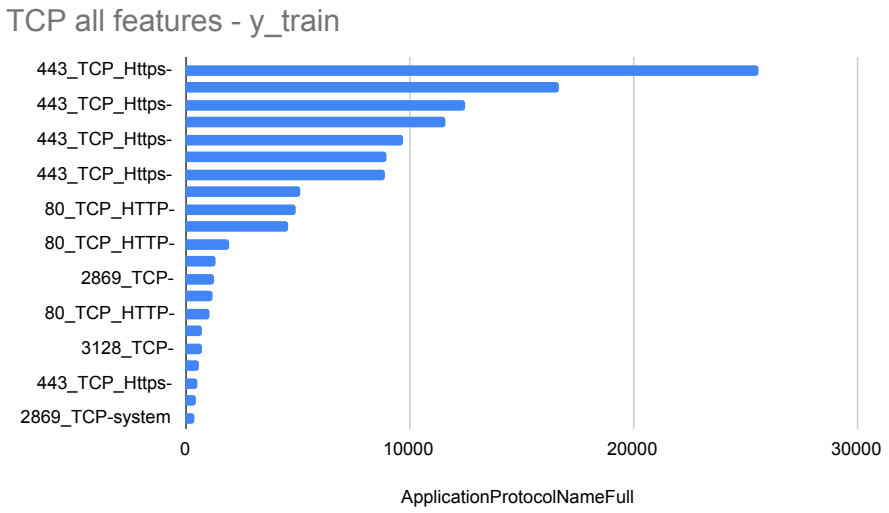


Figura 3.6: Gráfico com todas características do classificador TCP - *y_train*

A Figura 3.7 mostra a distribuição do *y_test*, no classificador que utiliza apenas características selecionadas pelos autores, mas contém apenas as amostras TCP, e na Figura 3.8 apresenta a distribuição das classes no *y_train*.

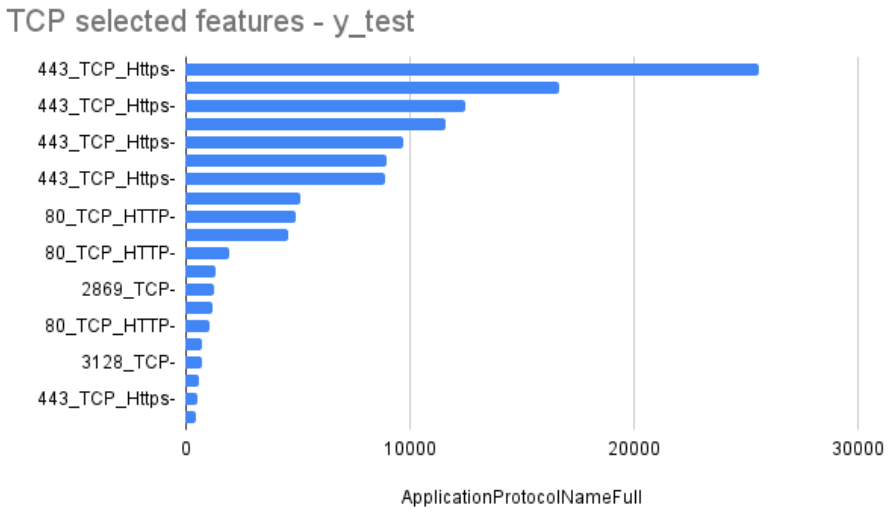


Figura 3.7: Gráfico com as 8 características do classificador TCP - *y_test*

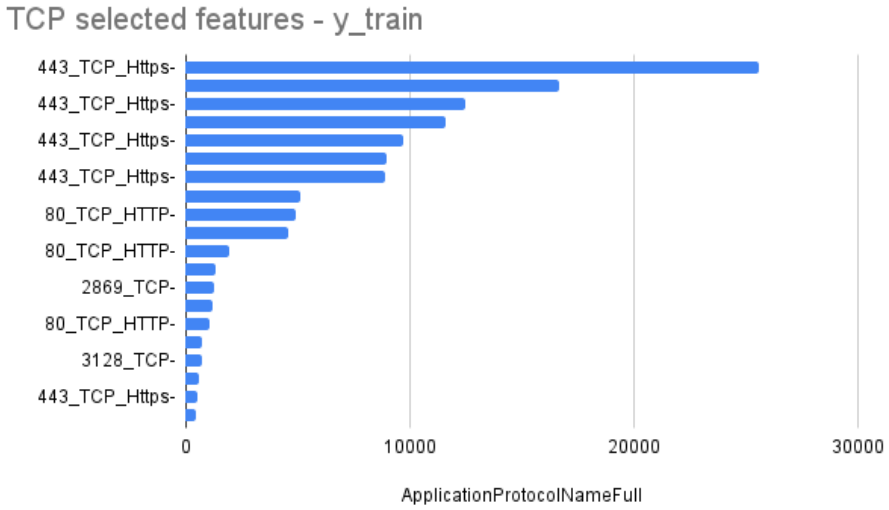


Figura 3.8: Gráfico com as 8 características do classificador TCP - *y_train*

4 RESULTADOS E DISCUSSÃO

Neste capítulo são apresentados os resultados do experimento, os quais são comparados com o trabalho base *"Traffic Classification and Application Identification in Network Forensics"* de modo a realizar as avaliações. O objetivo é reproduzir o trabalho de (Pluskal et al., 2018) e realizar testes adicionais para comparação de resultados e verificação de melhoria nas taxas inicialmente obtidas pelos autores do referido trabalho.

Os experimentos foram executados partir do *dataset* após o pré-processamento feito pelo *Netfox Detective*, que realizou o processamento do tráfego de rede capturado em conversas de aplicativo e mensagens. Os arquivos de disponibilizados pelos autores estavam no formato BIN, portanto foi realizada a conversão do formato para JSON e finalmente para um arquivo CSV, para poder mais facilmente processado pela biblioteca. A implementação foi feita utilizando a biblioteca *scikitlearn* disponível para a linguagem Python (Pedregosa et al., 2011).

Os parâmetros utilizados para a configuração do algoritmo *Random Forest* não foram disponibilizados, tendo em vista que Pluskal utilizou os parâmetros do classificador mais acurado. Considerando isso, o presente trabalho utilizou-se de uma das funções da biblioteca `GridSearchCV` que percorrer hiperparâmetros pré-definidos e ajusta seu estimador (modelo) em seu conjunto de treinamento. Assim, no final, os melhores parâmetros dos hiperparâmetros são listados e é possível selecionar o melhor. A seleção dos parâmetros foi feita para o *dataset* completo, sem nenhuma eliminação de características e foi o mesmo utilizado em todos os outros experimentos, tanto para o treino dos classificadores quanto para a execução de testes. Os parâmetros selecionados para o *dataset* completo foram os mesmos utilizados na execução dos experimentos com seleção de características e seleção de tráfego TCP.

A tabelas abaixo indicam os resultados dos classificadores rodados, demonstrando o F1-score de cada uma das classes. Essa foi a métrica utilizada pelos autores do trabalho base e decidiu-se usar a mesma de modo a fornecer dados acurados para a comparação. O F1-score permite visualizar as métricas *Precision* e *Recall* juntas, quanto mais próximo de 1, melhor o resultado. Os experimentos foram divididos em 2 categorias: o *dataset* completo com todas as características e utilizando a seleção de oito características disponibilizada pelos autores. Além disso, foi selecionado apenas o tráfego TCP de modo a identificar se os classificadores utilizados são um método viável para a realização da segregação de tráfego especializada por aplicação.

A Tabela 4.1 apresenta os resultados dos classificadores rodados, comparando os classificadores *Naive Bayes* e *Random Forest* para o *dataset* completo, demonstrando se a seleção de características realizadas pelos autores do trabalho base tiveram impacto nos resultados. A partir dele, podemos identificar que não houve melhora significativa no F1-score, mas é possível observar que o classificador *Random Forest* supera o resultado do *Naive Bayes*.

Tabela 4.1: F1-score para cada classe de tráfego geral, i.e., com todas as classes rotuladas TCP e UDP (**Protocolo de Aplicação**) com *Random Forest* (**RF**) e *Naive Bayes* (**NB**), considerando o *dataset* com todas as características (**completo**) e com as oito características previamente selecionadas (**seleção**).

Protocolo de Aplicação	**RF completo**	**RF seleção**	**NB completo**	**NB seleção**
Tempo de Treinamento (em segundos)	44,46	39,61	39,22	30,25
11000_UDP-	1,00	0,96	1,00	1,00
11888_UDP-	0,18	0,00	0,03	0,00
12350_TCP-skypeexe	1,00	0,96	0,79	0,28
137_UDP-	1,00	0,99	0,54	0,12

138_UDP-	1,00	1,00	1,00	1,00
139_TCP-	1,00	0,73	1,00	0,65
139_TCP-system	1,00	0,92	0,99	0,81
1900_UDP-	1,00	1,00	0,86	0,95
22_TCP_SSH-puttyexe	1,00	1,00	0,64	0,82
22_TCP_SSH-winscpexe	1,00	0,96	0,32	0,45
27017_UDP-	0,33	0,17	0,03	0,00
27018_UDP-	0,37	0,39	0,32	0,33
27019_UDP-	0,87	0,89	0,85	0,87
27020_UDP-	0,27	0,43	0,05	0,00
27036_UDP-	1,00	0,94	0,09	0,09
2869_TCP-	1,00	0,93	0,82	0,40
2869_TCP-system	0,99	0,79	0,70	0,51
3128_TCP-	0,83	0,69	0,16	0,17
3128_TCP-chromeexe	0,85	0,67	0,09	0,01
3128_TCP-spotifyexe	0,93	0,86	0,29	0,00
3128_TCP-steamwebhelperexe	0,94	0,90	0,38	0,25
3128_TCP-system	0,56	0,30	0,10	0,03
3478_UDP-	1,00	1,00	0,94	0,25
3702_UDP-	0,99	1,00	0,08	0,88
3800_UDP-	0,41	0,00	0,23	0,09
40003_UDP-	0,76	0,70	0,00	0,00
40005_UDP-	0,73	0,55	0,00	0,00
40018_UDP-	0,71	0,70	0,21	0,17
40020_UDP-	0,80	0,74	0,16	0,22
40023_UDP-	0,80	0,67	0,32	0,17
40024_UDP-	0,57	0,43	0,00	0,00
40025_UDP-	0,36	0,57	0,00	0,00
40027_UDP-	0,82	0,70	0,11	0,00
40029_TCP-skypeexe	1,00	1,00	0,60	0,25
40029_UDP-	0,64	0,73	0,00	0,18
40030_UDP-	0,57	0,62	0,00	0,00
4070_TCP-spotifyexe	0,98	0,94	0,48	0,84
43265_UDP-	0,69	0,59	0,10	0,00
443_TCP_Https-	0,79	0,67	0,13	0,01
443_TCP_Https-chromeexe	0,82	0,68	0,01	0,81
443_TCP_Https-firefoxexe	0,85	0,76	0,22	0,03
443_TCP_Https-googledrivesyncexe	0,67	0,50	0,02	0,00
443_TCP_Https-iexploreexe	0,80	0,69	0,07	0,00
443_TCP_Https-itunesexe	0,89	0,84	0,21	0,28
443_TCP_Https-msmpengexe	0,67	0,38	0,09	0,00
443_TCP_Https-onedriveexe	0,86	0,82	0,44	0,32
443_TCP_Https-skypebrowserhostexe	0,82	0,65	0,04	0,84
443_TCP_Https-skypeexe	0,89	0,85	0,01	0,85
443_TCP_Https-spotifyexe	0,77	0,71	0,01	0,11
443_TCP_Https-svchostexe	0,85	0,76	0,00	0,57

443_TCP_Https-system	0,76	0,61	0,50	0,22
443_TCP_Https-utorrentexe	0,97	0,93	0,77	0,00
443_UDP_Https-	1,00	1,00	0,93	0,59
45776_TCP-utorrentexe	0,94	0,89	0,84	0,38
465_TCP-thunderbirdexe	1,00	1,00	0,32	0,00
47993_UDP-	0,49	0,41	0,39	0,24
49001_UDP-	0,35	0,24	0,00	0,00
5004_UDP-	0,97	0,97	0,09	0,95
5005_UDP-	0,96	0,93	0,95	0,60
50321_UDP-	0,24	0,17	0,00	0,00
51413_UDP-	0,91	0,86	0,15	0,34
51654_UDP-	1,00	0,99	0,46	0,35
5222_TCP_Jabber-pidginexe	0,96	0,92	0,87	0,76
5223_TCP_JabberSsl-apsdaemonexe	0,95	0,99	0,12	0,77
53_TCP_Dns-system	1,00	1,00	1,00	0,80
53_UDP_Dns-	1,00	1,00	0,99	0,06
5351_UDP-	1,00	0,99	0,99	0,00
5353_UDP-	1,00	0,99	0,99	0,16
5355_UDP-	1,00	1,00	1,00	0,98
547_UDP_DhcPv6C-	1,00	0,94	0,91	0,82
57621_UDP-	1,00	1,00	1,00	0,86
5938_TCP-	0,90	0,86	0,22	0,10
5938_TCP-system	0,71	0,59	0,14	0,00
5938_TCP-teamviewer_serviceexe	0,77	0,72	0,33	0,50
5938_UDP-	1,00	1,00	1,00	0,98
67_UDP_DhcPc-	0,97	0,97	1,00	0,86
67_UDP_DhcPs-	0,99	0,99	1,00	0,99
6881_UDP-	0,74	0,71	0,02	0,06
6889_UDP-	0,76	0,75	0,00	0,09
6969_TCP-utorrentexe	1,00	0,98	0,98	0,00
80_TCP_HTTP-	0,88	0,82	0,02	0,04
80_TCP_HTTP-firefoxexe	0,63	0,40	0,09	0,09
80_TCP_HTTP-iexploreexe	0,63	0,44	0,07	0,00
80_TCP_HTTP-itunesexe	0,62	0,33	0,08	0,14
80_TCP_HTTP-onedriveexe	0,72	0,59	0,78	0,62
80_TCP_HTTP-spotifyexe	0,95	0,93	0,93	0,77
80_TCP_HTTP-steamexe	0,90	0,87	0,03	0,30
80_TCP_HTTP-system	0,73	0,39	0,03	0,15
80_TCP_HTTP-teamviewer_serviceexe	0,89	0,85	0,84	0,04
80_TCP_HTTP-utorrentexe	0,88	0,83	0,36	0,00
80_TCP_HTTP-utorrentieexe	0,67	0,48	0,03	0,00
8621_UDP-	0,15	0,13	0,00	0,38
9875_UDP-	1,00	1,00	0,93	0,93
995_TCP-thunderbirdexe	0,99	0,91	0,15	0,03

A Tabela 4.2 apresenta o F1-score apenas das amostras cujos protocolos de aplicação usam TCP, com cada uma das classes considerando o *dataset* completo com todas as características e com as oito características selecionadas anteriormente. Pode-se também observar que a aplicação do *Random Forest* supera o resultado dos outros classificadores, além de não revelar melhoria significativa entre os experimentos com todas as características e apenas as selecionadas.

Tabela 4.2: F1-score para cada classe de tráfego TCP (**Protocolo de Aplicação**) com *Random Forest* (**RF**) e *Naive Bayes* (**NB**), considerando o *dataset* com todas as características (**completo**) e com as oito características previamente selecionadas (**seleção**).

Protocolo de Aplicação	**RF completo**	**RF seleção**	**NB completo**	**NB seleção**
Tempo de Treinamento (em segundos)	29,55	26,76	29,02	26,08
12350_TCP-skypeexe	1,00	1,00	0,73	0,63
139_TCP-	1,00	0,61	1,00	0,63
139_TCP-system	1,00	0,88	1,00	0,74
22_TCP_SSH-puttyexe	1,00	1,00	0,59	0,44
22_TCP_SSH-winscpexe	0,96	0,96	0,16	0,50
2869_TCP-	1,00	0,93	0,80	0,25
2869_TCP-system	0,99	0,81	0,69	0,58
3128_TCP-	0,86	0,71	0,11	0,95
3128_TCP-chromeexe	0,85	0,59	0,06	0,93
3128_TCP-spotifyexe	0,81	0,81	0,28	0,13
3128_TCP-steamwebhelperexe	0,93	0,88	0,54	0,03
3128_TCP-system	0,55	0,22	0,07	0,07
40029_TCP-skypeexe	1,00	0,93	0,93	0,93
4070_TCP-spotifyexe	0,99	0,96	0,87	0,84
443_TCP_Https-	0,81	0,68	0,01	0,01
443_TCP_Https-chromeexe	0,81	0,68	0,01	0,09
443_TCP_Https-firefoxexe	0,86	0,75	0,11	0,02
443_TCP_Https-googledrivesyncexe	0,18	0,17	0,03	0,73
443_TCP_Https-iexploreexe	0,80	0,67	0,10	0,00
443_TCP_Https-itunesexe	0,89	0,83	0,21	0,70
443_TCP_Https-msmpengexe	0,86	0,73	0,10	0,91
443_TCP_Https-onedriveexe	0,86	0,81	0,55	0,47
443_TCP_Https-skypebrowserhostexe	0,74	0,62	0,01	0,01
443_TCP_Https-skypeexe	0,90	0,86	0,02	0,85
443_TCP_Https-spotifyexe	0,71	0,71	0,00	0,07
443_TCP_Https-svchostexe	0,84	0,80	0,18	0,58
443_TCP_Https-system	0,75	0,60	0,52	0,28
443_TCP_Https-utorrentexe	0,97	0,95	0,54	0,90
45776_TCP-utorrentexe	1,00	0,94	1,00	0,12
465_TCP-thunderbirdexe	1,00	0,97	0,15	0,10
5222_TCP_Jabber-pidginexe	1,00	0,96	0,93	0,77
5223_TCP_JabberSsl-apsdaemonexe	0,99	0,97	0,35	0,79
53_TCP_Dns-system	1,00	1,00	0,96	0,96
5938_TCP-	0,96	0,94	0,16	0,13
5938_TCP-system	0,77	0,77	0,18	0,14
5938_TCP-teamviewer_serviceexe	0,78	0,77	0,26	0,46

6969_TCP-utorrentexe	1,00	0,99	1,00	0,25
80_TCP_HTTP-	0,89	0,83	0,00	0,46
80_TCP_HTTP-firefoxexe	0,62	0,48	0,16	0,06
80_TCP_HTTP-iexploreexe	0,67	0,49	0,10	0,09
80_TCP_HTTP-itunesexe	0,55	0,48	0,06	0,07
80_TCP_HTTP-onedriveexe	0,62	0,59	0,75	0,73
80_TCP_HTTP-spotifyexe	0,96	0,94	0,93	0,84
80_TCP_HTTP-steamexe	0,87	0,79	0,44	0,04
80_TCP_HTTP-system	0,80	0,36	0,05	0,21
80_TCP_HTTP-teamviewer_serviceexe	0,93	0,81	0,70	0,56
80_TCP_HTTP-utorrentexe	0,90	0,85	0,38	0,06
80_TCP_HTTP-utorrentieexe	0,68	0,55	0,11	0,88
995_TCP-thunderbirdexe	1,00	0,98	0,96	0,03

A Tabela 4.3 mostra a quantidade de classes para cada valor de F1-score, de modo a considerar de forma agregada o resultado para todas as classes. É possível identificar que a quantidade de classes com um resultado superior a 0.9, o mais próximo do resultado ideal 1.0, é muito superior nos classificadores *Random Forest*. De modo a se assemelhar a filtragem de resultados realizada no trabalho base, foram retiradas todas as instâncias de classe cujo F1-score deu erro.

Tabela 4.3: Quantidade de classes por valor de F1-score para *Random Forest* (**RF**) e *Naive Bayes* (**NB**), considerando o dataset com todas as características (**completo**) e com as oito características previamente selecionadas (**seleção**).

F1 >=	**RF completo**	**RF seleção**	**NB completo**	**NB seleção**
0.0	0	2	10	30
0.1	2	3	22	19
0.2	2	1	10	8
0.3	4	6	7	7
0.4	2	5	8	4
0.5	3	6	5	4
0.6	8	9	2	3
0.7	12	13	3	2
0.8	19	12	7	9
0.9	42	37	20	8
Total	94	94	94	94

Na Tabela 4.4, pode-se notar que os resultados obtidos pelos experimentos realizados no presente trabalho, apresentados nas duas colunas mais à direita, alcançaram taxas de F1-score superiores aos obtidos por (Pluskal et al., 2018) na grande maioria dos casos. É interessante observar também que o algoritmo ESPI apresentado por Pluskal et al. em (Pluskal et al., 2018) não conseguiu superar os próprios experimentos do autor na maioria dos testes feitos com *Random Forest*. Os experimentos realizados pelos autores consideram apenas as oito características selecionadas, de modo a comparar qual a melhora dessa seleção, foi-se realizado o experimento com todas as características.

Os tempos de execução de cada algoritmo em **minutos** nos Experimentos B4, B5, B6, ESPI2, RF3 e RF4 realizados por Pluskal na Tabela 4.4 foram, respectivamente, 53, 63, 120, 71, 1213 e 1400. Todos os experimentos feitos neste trabalho obtiveram tempo de treinamento

Tabela 4.4: Comparação entre os resultados obtidos por Pluskal com Naive Bayes (B4, B5 e B6), o algoritmo proposto (ESPI2) e Random Forest (RF3 e RF4) e os experimentos realizados neste trabalho com Naive Bayes (NB) e Random Forest (RF) usando o *dataset* com todas as características (*) e as oito características selecionadas (8).

Protocolo de Aplicação	**B4/B5/B6**	**ESPI2**	**RF3/RF4**	**NB */8**	**RF */8**
tcp_smtptlsssl-thunderbirdexe	0/0/0	0,03	0,89/0,75	0,32/0,00	**1,00/1,00**
tcp_https-firefoxexe	0,88/**0,93**/0,91	0,41	0,71/0,77	0,22/0,03	0,85/0,76
tcp_https-svchostexe	0/0/0	0	0,71/0,77	0,00/0,57	**0,85**/0,76
tcp_http-steamwebhelperexe	0/0/0,38	0,52	0,72/0,79	0,38/0,25	**0,94**/0,90
tcp_icslap-system	0/0/0	0	0,7/0,81	0,70/0,51	**0,99**/0,79
tcp_https-onedriveexe	0/0,03/0,82	0	0,72/0,81	0,44/0,32	**0,86**/0,82
tcp_https-skypeexe	0,86/**0,99**/0,87	0,53	0,78/0,82	0,01/0,85	0,89/0,85
tcp_http-utorrentexe	0,01/0,11/0,32	0,01	0,84/0,83	0,36/0,00	**0,88**/0,83
tcp_http-teamviewer_serviceexe	0/0/0	0,87	**0,88**/0,86	0,33/0,50	0,77/0,72
tcp_skype-skypeexe	0,27/0,24/0	0,96	0,51/0,87	0,60/0,25	**1,00/1,00**
tcp_https-itunesexe	0,86/**0,89/0,89**	0,65	0,86/0,87	0,21/0,28	**0,89**/0,84
tcp_https-utorrentexe	0/0/0	0	0,92/0,89	0,77/0,00	**0,97**/0,93
tcp_dns-system	0/0/0	0,97	**1,00**/0,89	**1,00**/0,80	**1,00/1,00**
tcp_ssh-winscpexe	0/0/0	0,51	0,65/0,91	0,32/0,45	**1,00**/0,96
tcp_pop3tlsssl-thunderbirdexe	0/0/0	0	0,98/0,92	0,15/0,03	**0,99**/0,91
tcp_http-spotifyexe	0,93/0,91/0,93	0,9	0,93/0,93	0,01/0,11	0,77/0,71
tcp_tripe-spotifyexe	0/0/0,92	0,91	0,94/0,94	0,48/0,84	**0,98**/0,94
tcp_jabberssl-apsdaemonexe	0/0,72/0,81	0,91	0,94/0,95	0,12/0,77	0,95/**0,99**
tcp_jabber-pidginexe	0/0/0	**0,97**	0,94/**0,97**	0,87/0,76	0,96/0,92
tcp_netbiosss-system	0/0/0,9	0,44	0,98/0,99	0,99/0,81	**1,00**/0,92

inferior a um minuto (NB*, NB8, RF*, RF8 demoraram 39,22, 30,25, 44,46 e 39,61 segundos, respectivamente). A discrepância no tempo é indicada pelos autores em (Pluskal et al., 2018), uma vez que eles consideram o tempo necessário para completar todos os passos envolvidos nos experimentos, enquanto os tempos do presente trabalho fazem referência apenas a execução de cada um dos algoritmos de classificação. Além disso, conforme mencionado, o pré-processamento dos dados é uma operação com um alto tempo de processamento e os autores do artigo de referência optaram por fazer suas próprias implementações dos algoritmos, cujo código não compartimentaliza as etapas de pré-processamento, treinamento e teste.

Também, observando as colunas da Tabela 4.4 é possível identificar que para o classificador *Random Forest* implementado no presente trabalho, as características selecionadas não trouxeram melhora notável no desempenho. Considerando que em Pluskal et al. os experimentos todos foram realizados apenas com o *dataset* com as oito características selecionadas, é possível constatar a mesma situação.

5 CONCLUSÃO

Neste trabalho foi realizada a replicação dos experimentos descritos em *"Traffic Classification and Application Identification in Network Forensics"* (Pluskal et al., 2018). Os algoritmos foram implementadas utilizando a biblioteca *scikit-learn* para linguagem Python. Dessa forma, foi escolhida a implementação de dois classificadores, um deles baseado em *Naive Bayes* e outro em *Random Forest*. Os experimentos foram realizados com todas as características definidas no artigo de origem, bem como com *dataset* cujas características foram pré-selecionadas por Pluskal.

A partir dos resultados identificados é possível concluir que a seleção de características realizada no artigo de origem não demonstrou melhora nos resultados dos classificadores implementados, ao contrário do que se esperava. O classificador baseado em *Naive Bayes* apontou dados inferiores aos outros, e o *Random Forest* continuou mostrando os melhores resultados. Por fim, o algoritmo proposto pelos autores não conseguiu superar os resultados providos pelos algoritmos clássicos, seja nos experimentos realizados por Pluskal, ou nos experimentos realizados no presente trabalho.

Um dos maiores desafios na replicação do trabalho foi a falta de clareza na documentação, de organização no artigo do GitHub, do uso de ferramenta captura (NetMon[1]) que gera arquivos de tráfego não facilmente processáveis por ferramentas abertas (por exemplo, ferramentas como Wireshark[2] ou TCPDump[3] são capazes de ler arquivos CAP gravados pelo NetMon, mas não regravá-los após filtragem) ou código customizado e insuficientemente comentado dos autores que dificulta a comparação pareável com implementações da comunidade. Isto impactou não só as medidas de tempo, mas as comparações dos resultados, pois algumas tabelas não apresentavam informações completas para total entendimento do resultado ou discussão do experimento.

Além disso, foram realizados experimentos utilizando apenas as amostras TCP utilizando a mesma metodologia anterior, comparando todas as características com apenas as selecionadas e apresentando resultados similares aos realizados com o tráfego de rede em sua totalidade. Esses experimentos apontaram para a conclusão de que a utilização de algoritmos clássicos pode ser um método viável (rápido e efetivo) para segregação de tráfego especializada por aplicação, permitindo a criação de ferramentas práticas para filtragem de tráfego (por exemplo, identificação e bloqueio de certas aplicações em ambientes de laboratórios de informática). Entretanto, a análise das matrizes de confusão faz-se necessária para se entender as taxas de Falsos-Positivos em aplicações com objetivos similares, como navegadores, ou mensageiros. Esta tarefa foi deixada como trabalho futuro.

É possível apontar alguns problemas na reprodução de artigos acadêmicos a partir dessa experiência, e um deles é o desbalanceamento das classes, a rotulação confusa ou imprecisa e, principalmente, a falta de discussão desses aspectos por parte dos autores. O desbalanceamento de classes afeta negativamente o resultado devido a falta de filtragem, tendo em vista que muitas classes se referem a processos internos do sistema, que ao utilizar um protocolo de aplicação, gera um tráfego gigantesco de mensagens internas. Uma vez que são utilizados muitos dados e muita informação é gerada a partir dos experimentos, não é sempre fácil encontrar as informações necessárias para casos específicos, tais como parâmetros usados nos algoritmos, bem como as

[1] `https://docs.microsoft.com/en-us/windows/client-management/troubleshoot-tcpip-netmon`

[2] `https://www.wireshark.org/`

[3] `https://www.tcpdump.org/`

técnicas e configurações nas etapas de treinamento e validação. Além disso, caso a formatação dos resultados não esteja bem explícita, a comparação entre o artigo base e os testes realizados pelo autor que planeja reproduzir e validar os resultados vai ser dificultada, pois a interpretação ficará genérica e pode fugir do que era pretendido inicialmente. Um exemplo claro deste último problema é a versão de Pluskal para a Tabela 4.3, na qual, no artigo original, não fica claro o que foi filtrado (neste trabalho, removeu-se todas as instâncias de classes cujos resultados de F1-score deram erro) e como foi feita a conta das classes cujos valores de F1 estão dentro dos intervalos entre 0.0 e 0.9. Além disso, essa forma de apresentação de resultados não é usual, pois os autores poderiam mostrar as curvas por classes em modo *one versus all*, ou uma tabela com os valores de *Precisão*, *Recall* e *F1*, em vez de apresentar a contagem.

Trabalhos futuros sugeridos incluem, além do acima referido, mais experimentações com a seleção de características, objetivando melhorias na classificação, além de utilizar um *dataset* mais abrangente e coletado novamente em uma rede controlada.

REFERÊNCIAS

Ariu, D., Giacinto, G. e Roli, F. (2011). Machine learning in computer forensics (and the lessons learned from machine learning in computer security). Em *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, páginas 99–104.

Breiman, L. (2001). Random forests. *Machine Learning*, 45:5–32.

Bullinaria, J. A. (2004). Self organizing maps: Fundamentals. Em *Introduction to Neural Networks: Lecture 16*.

Callado, A., Kamienski, C., Szabó, G., Gero, B. P., Kelner, J., Fernandes, S. e Sadok, D. (2009). A survey on internet traffic identification. *IEEE communications surveys & tutorials*, 11(3):37–52.

Ceschin, F., Oliveira, L. S. e Grégio, A. R. A. (2019). Aprendizado de máquina para segurança: Algoritmos e aplicações. Em de Computação – SBC, S. B., editor, *Minicursos - XIX Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, páginas 41–90. Sociedade Brasileira de Computação – SBC.

Cunha, J. M. d. S., Silva, R. E. F. d., Silva, J. M. C. e Lima, S. (2015). Classificação multinível de tráfego de rede com base em amostragem.

Draper-Gil, G., Lashkari, A. H., Mamun, M. S. I. e Ghorbani, A. A. (2016). Characterization of encrypted and vpn traffic using time-related. Em *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*, páginas 407–414.

Gómez Sena, G. e Belzarena, P. (2009). Early traffic classification using support vector machines. Em *Proceedings of the 5th International Latin American Networking Conference*, páginas 60–66.

Grégio, A. R. A. e dos Santos, R. D. C. (2010). Análise e visualização de logs de segurança. Em da Rocha Fernandes; Elisangela Maschio de Miranda; Michelle Silva Wangham, A. M., editor, *Computer on the Beach*, páginas 85–107–.

Joshi, M. e Hadi, T. H. (2015). A review of network traffic analysis and prediction techniques. *arXiv preprint arXiv:1507.05722*.

Kourou, K., Exarchos, T. P., Exarchos, K. P., Karamouzis, M. V. e Fotiadis, D. I. (2015). Machine learning applications in cancer prognosis and prediction. *Computational and structural biotechnology journal*, 13:8–17.

Letavay, V., Pluskal, J. e Ryšavỳ, O. (2019). Network forensic analysis for lawful enforcement on steroids, distributed and scalable. Em *Proceedings of the 6th Conference on the Engineering of Computer Based Systems*, páginas 1–9.

Libbrecht, M. W. e Noble, W. S. (2015). Machine learning applications in genetics and genomics. *Nature Reviews Genetics*, 16(6):321–332.

Monard, M. C. e Baranauskas, J. A. (2003). Conceitos sobre aprendizado de máquina. *Sistemas inteligentes-Fundamentos e aplicações*, 1(1):32.

Nguyen, T. T. e Armitage, G. (2008). A survey of techniques for internet traffic classification using machine learning. *IEEE communications surveys & tutorials*, 10(4):56–76.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. e Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Pluskal, J., KOUTENSKÝ, M., VONDRÁČEK, M. e RYŠAVÝ, O. (2019). Network forensic investigations of tunneled traffic: A case study. *REVUE ROUMAINE DES SCIENCES TECHNIQUES-SERIE ELECTROTECHNIQUE ET ENERGETIQUE*, 64(4):429–434.

Pluskal, J., Lichtner, O. e Rysavy, O. (2018). Traffic classification and application identification in network forensics. Em *IFIP International Conference on Digital Forensics*, páginas 161–181. Springer.

Pluskal, J., Matoušek, P., Ryšavỳ, O., Kmet', M., Veselỳ, V., Karpíšek, F. e Vymlátil, M. (2015). Netfox detective: A tool for advanced network forensics analysis. *no. i*.

Rish, I. et al. (2001). An empirical study of the naive bayes classifier. Em *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, páginas 41–46.

Sinaga, K. P. e Yang, M.-S. (2020). Unsupervised k-means clustering algorithm. *IEEE access*, 8:80716–80727.

APÊNDICE A – TABELAS COMPLETAS

Aqui são expostas as tabelas completas com todos as classes e rótulos.

A.1 DISTRIBUIÇÃO DE CLASSES

Tabela A.1: Distribuição de Classes do *dataset* dividido para *y_test* com todas as classes

ApplicationProtocolNameFull	**Quantidade de amostras**
5355_UDP-	35656
53_UDP_Dns-	35577
443_TCP_Https-system	25559
443_TCP_Https-firefoxexe	16666
443_TCP_Https-	12478
80_TCP_HTTP-	11595
1900_UDP-	9732
443_TCP_Https-chromeexe	9722
443_TCP_Https-itunesexe	8942
443_TCP_Https-iexploreexe	8899
443_TCP_Https-spotifyexe	5143
80_TCP_HTTP-spotifyexe	4931
443_TCP_Https-skypeexe	4588
27019_UDP-	2421
80_TCP_HTTP-iexploreexe	1933
443_UDP_Https-	1534
443_TCP_Https-onedriveexe	1363
51654_UDP-	1254
2869_TCP-	1252
80_TCP_HTTP-firefoxexe	1190
80_TCP_HTTP-utorrentexe	1045
6881_UDP-	846
3128_TCP-chromeexe	755
3128_TCP-	745
443_TCP_Https-skypebrowserhostexe	617
51413_UDP-	578
443_TCP_Https-svchostexe	520
27018_UDP-	457
27017_UDP-	446
80_TCP_HTTP-steamexe	433
138_UDP-	428
2869_TCP-system	401
3128_TCP-steamwebhelperexe	391
137_UDP-	378
67_UDP_DhcPs-	374

27020_UDP-	348
80_TCP_HTTP-teamviewer_serviceexe	310
6889_UDP-	293
443_TCP_Https-utorrentexe	291
80_TCP_HTTP-utorrentieexe	290
3478_UDP-	224
3702_UDP-	224
3128_TCP-system	217
80_TCP_HTTP-itunesexe	214
5353_UDP-	210
6969_TCP-utorrentexe	200
995_TCP-thunderbirdexe	178
4070_TCP-spotifyexe	164
5351_UDP-	162
57621_UDP-	160
139_TCP-system	158
5223_TCP_JabberSsl-apsdaemonexe	149
43265_UDP-	120
50321_UDP-	115
5938_TCP-	100
80_TCP_HTTP-system	98
5938_TCP-teamviewer_serviceexe	96
5938_UDP-	90
47993_UDP-	78
465_TCP-thunderbirdexe	75
80_TCP_HTTP-onedriveexe	73
49001_UDP-	67
5938_TCP-system	62
3128_TCP-spotifyexe	61
67_UDP_DhcPc-	59
8621_UDP-	58
5004_UDP-	57
5005_UDP-	56
12350_TCP-skypeexe	55
443_TCP_Https-msmpengexe	54
3800_UDP-	53
40020_UDP-	50
22_TCP_SSH-winscpexe	50
5222_TCP_Jabber-pidginexe	49
11000_UDP-	46
40029_UDP-	45
40003_UDP-	42
139_TCP-	42
443_TCP_Https-googledrivesyncexe	41
53_TCP_Dns-system	38
27036_UDP-	38

547_UDP_DhcPv6C-	38
40027_UDP-	34
45776_TCP-utorrentexe	34
40024_UDP-	32
40018_UDP-	31
11888_UDP-	30
22_TCP_SSH-puttyexe	29
9875_UDP-	28
40029_TCP-skypeexe	28
40030_UDP-	28
40025_UDP-	25
40023_UDP-	25
40005_UDP-	25

Tabela A.2: Distribuição de Classes do *dataset* dividido para *y_train* com todas as classes

ApplicationProtocolNameFull	**Quantidade de amostras**
5355_UDP-	35656
53_UDP_Dns-	35577
443_TCP_Https-system	25559
443_TCP_Https-firefoxexe	16666
443_TCP_Https-	12478
80_TCP_HTTP-	11595
1900_UDP-	9732
443_TCP_Https-chromeexe	9722
443_TCP_Https-itunesexe	8942
443_TCP_Https-iexploreexe	8899
443_TCP_Https-spotifyexe	5143
80_TCP_HTTP-spotifyexe	4931
443_TCP_Https-skypeexe	4588
27019_UDP-	2421
80_TCP_HTTP-iexploreexe	1933
443_UDP_Https-	1534
443_TCP_Https-onedriveexe	1363
51654_UDP-	1254
2869_TCP-	1252
80_TCP_HTTP-firefoxexe	1190
80_TCP_HTTP-utorrentexe	1045
6881_UDP-	846
3128_TCP-chromeexe	755
3128_TCP-	745
443_TCP_Https-skypebrowserhostexe	617
51413_UDP-	578
443_TCP_Https-svchostexe	520
27018_UDP-	457
27017_UDP-	446

80_TCP_HTTP-steamexe	433
138_UDP-	428
2869_TCP-system	401
3128_TCP-steamwebhelperexe	391
137_UDP-	378
67_UDP_DhcPs-	374
27020_UDP-	348
80_TCP_HTTP-teamviewer_serviceexe	310
6889_UDP-	293
443_TCP_Https-utorrentexe	291
80_TCP_HTTP-utorrentieexe	290
3478_UDP-	224
3702_UDP-	224
3128_TCP-system	217
80_TCP_HTTP-itunesexe	214
5353_UDP-	210
6969_TCP-utorrentexe	200
995_TCP-thunderbirdexe	178
4070_TCP-spotifyexe	164
5351_UDP-	162
57621_UDP-	160
139_TCP-system	158
5223_TCP_JabberSsl-apsdaemonexe	149
43265_UDP-	120
50321_UDP-	115
5938_TCP-	100
80_TCP_HTTP-system	98
5938_TCP-teamviewer_serviceexe	96
5938_UDP-	90
47993_UDP-	78
465_TCP-thunderbirdexe	75
80_TCP_HTTP-onedriveexe	73
49001_UDP-	67
5938_TCP-system	62
3128_TCP-spotifyexe	61
67_UDP_DhcPc-	59
8621_UDP-	58
5004_UDP-	57
5005_UDP-	56
12350_TCP-skypeexe	55
443_TCP_Https-msmpengexe	54
3800_UDP-	53
40020_UDP-	50
22_TCP_SSH-winscpexe	50
5222_TCP_Jabber-pidginexe	49
11000_UDP-	46

40029_UDP-	45
40003_UDP-	42
139_TCP-	42
443_TCP_Https-googledrivesyncexe	41
53_TCP_Dns-system	38
27036_UDP-	38
547_UDP_DhcPv6C-	38
40027_UDP-	34
45776_TCP-utorrentexe	34
40024_UDP-	32
40018_UDP-	31
11888_UDP-	30
22_TCP_SSH-puttyexe	29
9875_UDP-	28
40029_TCP-skypeexe	28
40030_UDP-	28
40025_UDP-	25
40023_UDP-	25
40005_UDP-	25

Tabela A.3: Distribuição de Classes do *dataset* dividido para *y_test* com as classes selecionadas

ApplicationProtocolNameFull	**Quantidade de amostras**
5355_UDP-	35656
53_UDP_Dns-	35577
443_TCP_Https-system	25559
443_TCP_Https-firefoxexe	16666
443_TCP_Https-	12478
80_TCP_HTTP-	11595
1900_UDP-	9732
443_TCP_Https-chromeexe	9722
443_TCP_Https-itunesexe	8942
443_TCP_Https-iexploreexe	8899
443_TCP_Https-spotifyexe	5143
80_TCP_HTTP-spotifyexe	4931
443_TCP_Https-skypeexe	4588
27019_UDP-	2421
80_TCP_HTTP-iexploreexe	1933
443_UDP_Https-	1534
443_TCP_Https-onedriveexe	1363
51654_UDP-	1254
2869_TCP-	1252
80_TCP_HTTP-firefoxexe	1190
80_TCP_HTTP-utorrentexe	1045
6881_UDP-	846
3128_TCP-chromeexe	755

3128_TCP-	745
443_TCP_Https-skypebrowserhostexe	617
51413_UDP-	578
443_TCP_Https-svchostexe	520
27018_UDP-	457
27017_UDP-	446
80_TCP_HTTP-steamexe	433
138_UDP-	428
2869_TCP-system	401
3128_TCP-steamwebhelperexe	391
137_UDP-	378
67_UDP_DhcPs-	374
27020_UDP-	348
80_TCP_HTTP-teamviewer_serviceexe	310
6889_UDP-	293
443_TCP_Https-utorrentexe	291
80_TCP_HTTP-utorrentieexe	290
3478_UDP-	224
3702_UDP-	224
3128_TCP-system	217
80_TCP_HTTP-itunesexe	214
5353_UDP-	210
6969_TCP-utorrentexe	200
995_TCP-thunderbirdexe	178
4070_TCP-spotifyexe	164
5351_UDP-	162
57621_UDP-	160
139_TCP-system	158
5223_TCP_JabberSsl-apsdaemonexe	149
43265_UDP-	120
50321_UDP-	115
5938_TCP-	100
80_TCP_HTTP-system	98
5938_TCP-teamviewer_serviceexe	96
5938_UDP-	90
47993_UDP-	78
465_TCP-thunderbirdexe	75
80_TCP_HTTP-onedriveexe	73
49001_UDP-	67
5938_TCP-system	62
3128_TCP-spotifyexe	61
67_UDP_DhcPc-	59
8621_UDP-	58
5004_UDP-	57
5005_UDP-	56
12350_TCP-skypeexe	55

443_TCP_Https-msmpengexe	54
3800_UDP-	53
40020_UDP-	50
22_TCP_SSH-winscpexe	50
5222_TCP_Jabber-pidginexe	49
11000_UDP-	46
40029_UDP-	45
40003_UDP-	42
139_TCP-	42
443_TCP_Https-googledrivesyncexe	41
53_TCP_Dns-system	38
27036_UDP-	38
547_UDP_DhcPv6C-	38
40027_UDP-	34
45776_TCP-utorrentexe	34
40024_UDP-	32
40018_UDP-	31
11888_UDP-	30
22_TCP_SSH-puttyexe	29
9875_UDP-	28
40029_TCP-skypeexe	28
40030_UDP-	28
40025_UDP-	25
40023_UDP-	25
40005_UDP-	25

Tabela A.4: Distribuição de Classes do *dataset* dividido para *y_train* com as classes selecionadas

ApplicationProtocolNameFull	**Quantidade de amostras**
5355_UDP-	35656
53_UDP_Dns-	35577
443_TCP_Https-system	25559
443_TCP_Https-firefoxexe	16666
443_TCP_Https-	12478
80_TCP_HTTP-	11595
1900_UDP-	9732
443_TCP_Https-chromeexe	9722
443_TCP_Https-itunesexe	8942
443_TCP_Https-iexploreexe	8899
443_TCP_Https-spotifyexe	5143
80_TCP_HTTP-spotifyexe	4931
443_TCP_Https-skypeexe	4588
27019_UDP-	2421
80_TCP_HTTP-iexploreexe	1933
443_UDP_Https-	1534
443_TCP_Https-onedriveexe	1363

51654_UDP-	1254
2869_TCP-	1252
80_TCP_HTTP-firefoxexe	1190
80_TCP_HTTP-utorrentexe	1045
6881_UDP-	846
3128_TCP-chromeexe	755
3128_TCP-	745
443_TCP_Https-skypebrowserhostexe	617
51413_UDP-	578
443_TCP_Https-svchostexe	520
27018_UDP-	457
27017_UDP-	446
80_TCP_HTTP-steamexe	433
138_UDP-	428
2869_TCP-system	401
3128_TCP-steamwebhelperexe	391
137_UDP-	378
67_UDP_DhcPs-	374
27020_UDP-	348
80_TCP_HTTP-teamviewer_serviceexe	310
6889_UDP-	293
443_TCP_Https-utorrentexe	291
80_TCP_HTTP-utorrentieexe	290
3478_UDP-	224
3702_UDP-	224
3128_TCP-system	217
80_TCP_HTTP-itunesexe	214
5353_UDP-	210
6969_TCP-utorrentexe	200
995_TCP-thunderbirdexe	178
4070_TCP-spotifyexe	164
5351_UDP-	162
57621_UDP-	160
139_TCP-system	158
5223_TCP_JabberSsl-apsdaemonexe	149
43265_UDP-	120
50321_UDP-	115
5938_TCP-	100
80_TCP_HTTP-system	98
5938_TCP-teamviewer_serviceexe	96
5938_UDP-	90
47993_UDP-	78
465_TCP-thunderbirdexe	75
80_TCP_HTTP-onedriveexe	73
49001_UDP-	67
5938_TCP-system	62

3128_TCP-spotifyexe	61
67_UDP_DhcPc-	59
8621_UDP-	58
5004_UDP-	57
5005_UDP-	56
12350_TCP-skypeexe	55
443_TCP_Https-msmpengexe	54
3800_UDP-	53
40020_UDP-	50
22_TCP_SSH-winscpexe	50
5222_TCP_Jabber-pidginexe	49
11000_UDP-	46
40029_UDP-	45
40003_UDP-	42
139_TCP-	42
443_TCP_Https-googledrivesyncexe	41
53_TCP_Dns-system	38
27036_UDP-	38
547_UDP_DhcPv6C-	38
40027_UDP-	34
45776_TCP-utorrentexe	34
40024_UDP-	32
40018_UDP-	31
11888_UDP-	30
22_TCP_SSH-puttyexe	29
9875_UDP-	28
40029_TCP-skypeexe	28
40030_UDP-	28
40025_UDP-	25
40023_UDP-	25
40005_UDP-	25

Tabela A.5: Distribuição de Classes do *dataset* que contém apenas o tráfego TCP dividido para *y_test* com todas as classes

ApplicationProtocolNameFull	**Quantidade de amostras**
443_TCP_Https-system	25559
443_TCP_Https-firefoxexe	16666
443_TCP_Https-	12477
80_TCP_HTTP-	11595
443_TCP_Https-chromeexe	9721
443_TCP_Https-itunesexe	8942
443_TCP_Https-iexploreexe	8899
443_TCP_Https-spotifyexe	5143
80_TCP_HTTP-spotifyexe	4931
443_TCP_Https-skypeexe	4588

80_TCP_HTTP-iexploreexe	1933
443_TCP_Https-onedriveexe	1363
2869_TCP-	1252
80_TCP_HTTP-firefoxexe	1190
80_TCP_HTTP-utorrentexe	1045
3128_TCP-chromeexe	755
3128_TCP-	745
443_TCP_Https-skypebrowserhostexe	617
443_TCP_Https-svchostexe	520
80_TCP_HTTP-steamexe	433
2869_TCP-system	401
3128_TCP-steamwebhelperexe	391
80_TCP_HTTP-teamviewer_serviceexe	310
443_TCP_Https-utorrentexe	291
80_TCP_HTTP-utorrentieexe	290
3128_TCP-system	217
80_TCP_HTTP-itunesexe	214
6969_TCP-utorrentexe	20
995_TCP-thunderbirdexe	17
4070_TCP-spotifyexe	16
139_TCP-system	158
5223_TCP_JabberSsl-apsdaemonexe	150
5938_TCP-	100
80_TCP_HTTP-system	98
5938_TCP-teamviewer_serviceexe	96
465_TCP-thunderbirdexe	75
80_TCP_HTTP-onedriveexe	73
5938_TCP-system	62
3128_TCP-spotifyexe	61
12350_TCP-skypeexe	55
443_TCP_Https-msmpengexe	54
22_TCP_SSH-winscpexe	50
5222_TCP_Jabber-pidginexe	49
139_TCP-	42
443_TCP_Https-googledrivesyncexe	41
53_TCP_Dns-system	38
45776_TCP-utorrentexe	34
22_TCP_SSH-puttyexe	29
40029_TCP-skypeexe	28

Tabela A.6: Distribuição de Classes do *dataset* que contém apenas o tráfego TCP dividido para *y_train* com todas as classes

ApplicationProtocolNameFull	**Quantidade de amostras**
443_TCP_Https-system	25559
443_TCP_Https-firefoxexe	16666

443_TCP_Https-	12477
80_TCP_HTTP-	11595
443_TCP_Https-chromeexe	9721
443_TCP_Https-itunesexe	8942
443_TCP_Https-iexploreexe	8899
443_TCP_Https-spotifyexe	5143
80_TCP_HTTP-spotifyexe	4931
443_TCP_Https-skypeexe	4588
80_TCP_HTTP-iexploreexe	1933
443_TCP_Https-onedriveexe	1363
2869_TCP-	1252
80_TCP_HTTP-firefoxexe	1190
80_TCP_HTTP-utorrentexe	1045
3128_TCP-chromeexe	755
3128_TCP-	745
443_TCP_Https-skypebrowserhostexe	617
443_TCP_Https-svchostexe	520
80_TCP_HTTP-steamexe	433
2869_TCP-system	401
3128_TCP-steamwebhelperexe	391
80_TCP_HTTP-teamviewer_serviceexe	310
443_TCP_Https-utorrentexe	291
80_TCP_HTTP-utorrentieexe	290
3128_TCP-system	217
80_TCP_HTTP-itunesexe	214
6969_TCP-utorrentexe	200
995_TCP-thunderbirdexe	178
4070_TCP-spotifyexe	164
139_TCP-system	158
5223_TCP_JabberSsl-apsdaemonexe	150
5938_TCP-	100
80_TCP_HTTP-system	98
5938_TCP-teamviewer_serviceexe	96
465_TCP-thunderbirdexe	75
80_TCP_HTTP-onedriveexe	73
5938_TCP-system	62
3128_TCP-spotifyexe	61
12350_TCP-skypeexe	55
443_TCP_Https-msmpengexe	54
22_TCP_SSH-winscpexe	50
5222_TCP_Jabber-pidginexe	49
139_TCP-	42
443_TCP_Https-googledrivesyncexe	41
53_TCP_Dns-system	38
45776_TCP-utorrentexe	34
22_TCP_SSH-puttyexe	29

40029_TCP-skypeexe	28

Tabela A.7: Distribuição de Classes do *dataset* que contém apenas o tráfego TCP dividido para *y_test* com as classes selecionadas

ApplicationProtocolNameFull	**Quantidade de amostras**
443_TCP_Https-system	25559
443_TCP_Https-firefoxexe	16666
443_TCP_Https-	12477
80_TCP_HTTP-	11595
443_TCP_Https-chromeexe	9721
443_TCP_Https-itunesexe	8942
443_TCP_Https-iexploreexe	8899
443_TCP_Https-spotifyexe	5143
80_TCP_HTTP-spotifyexe	4931
443_TCP_Https-skypeexe	4588
80_TCP_HTTP-iexploreexe	1933
443_TCP_Https-onedriveexe	1363
2869_TCP-	1252
80_TCP_HTTP-firefoxexe	1190
80_TCP_HTTP-utorrentexe	1045
3128_TCP-chromeexe	755
3128_TCP-	745
443_TCP_Https-skypebrowserhostexe	617
443_TCP_Https-svchostexe	520
80_TCP_HTTP-steamexe	433
2869_TCP-system	401
3128_TCP-steamwebhelperexe	391
80_TCP_HTTP-teamviewer_serviceexe	310
443_TCP_Https-utorrentexe	291
80_TCP_HTTP-utorrentieexe	290
3128_TCP-system	217
80_TCP_HTTP-itunesexe	214
6969_TCP-utorrentexe	200
995_TCP-thunderbirdexe	178
4070_TCP-spotifyexe	164
139_TCP-system	158
5223_TCP_JabberSsl-apsdaemonexe	150
5938_TCP-	100
80_TCP_HTTP-system	98
5938_TCP-teamviewer_serviceexe	96
465_TCP-thunderbirdexe	75
80_TCP_HTTP-onedriveexe	73
5938_TCP-system	62
3128_TCP-spotifyexe	61
12350_TCP-skypeexe	55

443_TCP_Https-msmpengexe	54
22_TCP_SSH-winscpexe	50
5222_TCP_Jabber-pidginexe	49
139_TCP-	42
443_TCP_Https-googledrivesyncexe	41
53_TCP_Dns-system	38
45776_TCP-utorrentexe	34
22_TCP_SSH-puttyexe	29
40029_TCP-skypeexe	28

Tabela A.8: Distribuição de Classes do *dataset* que contém apenas o tráfego TCP dividido para *y_train* com as classes selecionadas

ApplicationProtocolNameFull	**Quantidade de amostras**
443_TCP_Https-system	25559
443_TCP_Https-firefoxexe	16666
443_TCP_Https-	12477
80_TCP_HTTP-	11595
443_TCP_Https-chromeexe	9721
443_TCP_Https-itunesexe	8942
443_TCP_Https-iexploreexe	8899
443_TCP_Https-spotifyexe	5143
80_TCP_HTTP-spotifyexe	4931
443_TCP_Https-skypeexe	4588
80_TCP_HTTP-iexploreexe	1933
443_TCP_Https-onedriveexe	1363
2869_TCP-	1252
80_TCP_HTTP-firefoxexe	1190
80_TCP_HTTP-utorrentexe	1045
3128_TCP-chromeexe	755
3128_TCP-	745
443_TCP_Https-skypebrowserhostexe	617
443_TCP_Https-svchostexe	520
80_TCP_HTTP-steamexe	433
2869_TCP-system	401
3128_TCP-steamwebhelperexe	391
80_TCP_HTTP-teamviewer_serviceexe	310
443_TCP_Https-utorrentexe	291
80_TCP_HTTP-utorrentieexe	290
3128_TCP-system	217
80_TCP_HTTP-itunesexe	214
6969_TCP-utorrentexe	200
995_TCP-thunderbirdexe	178
4070_TCP-spotifyexe	164
139_TCP-system	158
5223_TCP_JabberSsl-apsdaemonexe	150

5938_TCP-	100
80_TCP_HTTP-system	98
5938_TCP-teamviewer_serviceexe	96
465_TCP-thunderbirdexe	75
80_TCP_HTTP-onedriveexe	73
5938_TCP-system	62
3128_TCP-spotifyexe	61
12350_TCP-skypeexe	55
443_TCP_Https-msmpengexe	54
22_TCP_SSH-winscpexe	50
5222_TCP_Jabber-pidginexe	49
139_TCP-	42
443_TCP_Https-googledrivesyncexe	41
53_TCP_Dns-system	38
45776_TCP-utorrentexe	34
22_TCP_SSH-puttyexe	29
40029_TCP-skypeexe	28