HEITOR SCALCO NETO

SISTEMA DE DETECÇÃO DE INTRUSÃO EM REDES DE COMPUTADORES COM TÉCNICAS DE INTELIGÊNCIA COMPUTACIONAL

LAVRAS – MG

2017

HEITOR SCALCO NETO

SISTEMA DE DETECÇÃO DE INTRUSÃO EM REDES DE COMPUTADORES COM TÉCNICAS DE INTELIGÊNCIA COMPUTACIONAL

Dissertação apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, área de concentração em Ciência da Computação, para a obtenção do título de Mestre.

Prof. Dr. Wilian Soares Lacerda
Orientador

LAVRAS – MG

2017

Ficha catalográfica elaborada pelo Sistema de Geração de Ficha Catalográfica da Biblioteca Universitária da UFLA, com dados informados pelo(a) próprio(a) autor(a).

Scalco Neto, Heitor .
Sistema de Detecção de Intrusão em Redes de Computadores com Técnicas de Inteligência Computacional / Heitor Scalco Neto. – Lavras : UFLA, 2017.
152 p. : il.

Dissertação (mestrado acadêmico)–Universidade Federal de Lavras, 2016.
Orientador(a): Wilian Soares Lacerda.
Bibliografia.

1. Sistema de Detecção de Intrusão em Redes de Computadores. 2. Redes Neurais Artificiais. 3. Florestas Aleatórias. 4. Máquinas de Vetores de Suporte. I. Universidade Federal de Lavras. II. Título.

HEITOR SCALCO NETO

SISTEMA DE DETECÇÃO DE INTRUSÃO EM REDES DE COMPUTADORES COM TÉCNICAS DE INTELIGÊNCIA COMPUTACIONAL

Dissertação apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, área de concentração em Ciência da Computação, para a obtenção do título de Mestre.

APROVADA em 04 de Novembro de 2016.

Prof. Dr. Luiz Henrique Andrade Correia	UFLA
Prof. Dr. Cristiano Leite de Castro	UFMG

Prof. Dr. Wilian Soares Lacerda
Orientador

LAVRAS – MG
2017

Aos meus pais, José Scalco Neto e Ana Regina Pinto Scalco

e ao meu irmão, Henrique Scalco, pelo incentivo e apoio em todos os momentos.

Dedico

AGRADECIMENTOS

Agradeço, primeiramente, a Deus, por tudo.

Aos meus pais, **José Scalco Neto** e **Ana Regina Pinto Scalco**, por serem pais exemplares, pelo apoio em todas as horas e também por acreditarem em mim.

Ao meu irmão, **Henrique Scalco**, pela amizade e companheirismo.

À **Nicole Alessandra Engel**, por todo o apoio, paciência, amizade, coragem e companheirismo durante todo o tempo do mestrado.

Ao **Dr. Wilian Soares Lacerda**, pela excelente orientação, amizade, compreensão e disponibilidade durante o mestrado.

Ao **Dr. Luiz Henrique Andrade Correia**, pelas excelentes aulas de Arquitetura de Computadores e toda a ajuda que tive no decorrer do curso.

Aos amigos da "Padaria", **Victor Grudtner**, **Vancley Simão**, **Matheus Nogueira (Barba)**, **Francisco Magalhães (Chicão)** e **Cristino Souza Junior**, pelo companheirismo e pela ajuda no decorrer deste trabalho.

Aos amigos **Wesley Natanael Gallo**, **Claudiane Oliveira**, **Rodrigo Amador**, **Celso Ávila**, **João Antônio Silva**, **Camila Bastos**, **Gustavo Figueiredo Araújo** e **Mariana Bernardes** pelos vários cafés, pães de queijo, amizade e aprendizado repassado.

À minha tia, **Maria Cristina Scalco Dani**, pela atenção e por toda a ajuda.

Ao **Dr. Cristiano Leite de Castro**, pela disponibilidade para a participação em minha banca.

A todos aqueles que, de alguma forma, colaboraram para que eu conseguisse concluir o mestrado.

“Uns sonham com o sucesso,

nós acordamos cedo e trabalhamos duro para consegui-lo”.

(Abilio Diniz)

RESUMO

Os Sistemas de Detecção de Intrusão em Redes de Computadores (NIDS - *Network Intrusion Detection Systems*) têm importância fundamental para garantir a confiabilidade e disponibilidade em uma rede de computadores. Desta forma, esta dissertação de mestrado propõe uma metodologia para o desenvolvimento de um NIDS, por anomalias, Open-Source, com as seguintes técnicas de Inteligência Computacional (I.C): Redes Neurais Artificiais, Máquinas de Vetores de Suporte e Florestas Aleatórias.

As técnicas de I.C são aplicadas e comparadas, a fim de avaliar os métodos para a detecção de intrusão nos ambientes computacionais. Para que o NIDS seja capaz de operar em ambiente real, fez-se necessário o desenvolvimento de uma API que tem como objetivo capturar o tráfego de rede e realizar o pré-processamento da informação para as técnicas de I.C. Desta forma, é possível realizar testes com diferentes infraestruturas de rede e, também, em ambiente real. O treinamento dessas técnicas foi realizado com a base de dados de tráfego de rede ISCX 2012, a qual é composta por tipos de tráfego variados.

A partir da API desenvolvida, criou-se uma base de dados auxiliar, para testes, abordando tipos de tráfego um pouco alternativos ao encontrado na ISCX 2012, porém com uma rede em menor escala, com diferentes sistemas operacionais e ferramentas. Esta base de dados permite que testes de eficácia das técnicas de I.C sejam realizados em diferentes infraestruturas e modos de utilização.

Esta dissertação de mestrado tem como principais contribuições os seguintes tópicos: (i) desenvolvimento de uma API, *Open-Source*, para captura de pacotes, pré-processamento e integração com as técnicas de Inteligência Computacional; (ii) avaliação das técnicas de Inteligência Computacional para o problema de detecção de intrusão em redes de computadores; (iii) utilização de características independentes de *softwares* e/ou *hosts*.

Os resultados obtidos com a base de dados ISCX 2012 e as técnicas de I.C apresentam médias de acerto em torno de 95%. Já, com a base de testes, obtiveram-se médias de acerto em torno de 97% afirmando, assim, a viabilidade da utilização de técnicas de I.C para a resolução de problemas de reconhecimento de intrusão em redes de computadores — Cabe ressaltar que a base de testes não foi utilizada para realizar o treinamento das técnicas de I.C, apenas para a validação dos mesmos.

Palavras-chave: Sistema de Detecção de Intrusão em Redes de Computadores. Redes Neurais Artificiais. Máquinas de Vetores de Suporte. Florestas Aleatórias.

ABSTRACT

The Network Intrusion Detection Systems - NIDS have great importance in guaranteeing the reliability and availability of computer networks. Therefore, this thesis proposes a methodology for developing an anomaly based and Open-Source NIDS, using the following Computational Intelligence Techniques (CI): Artificial Neural Networks, Support Vector Machines and Random Forests. The CI techniques are applied and compared in order to evaluate the intrusion detection methods for computing environments. In order for the NIDS to operate in real environment, it was necessary to develop an API, with the objective of capturing the network traffic and preprocess the information for the CI techniques. Thus, it was possible to perform the tests in different network infrastructures and in real environment. The training of these techniques was done using the ISCX 2012 network traffic database, comprised by varied types of traffic. Using the developed API, we created an auxiliary database for tests, approaching traffic types alternative to that found with the ISCX 2012, however with network in smaller scale and with different operational systems and tools. This database allows the efficacy tests of the CI techniques to be performed in different infrastructures and modes of use. This thesis had the main contributions in the following topics: (i) development of an API, Open-Source, for capturing packages, preprocessing and integrating with the Computacional Intelligence techniques; (ii) evaluation of the Computacional Intelligence techniques for the network intrusion detection issue; (iii) use of independent software and/or host features . The results obtained with the ISCX 2012 database and CI techniques presented adjustment averages close to 95%. With the test database, the adjustment averages were of close to 97%, affirming the feasibility of the use of CI techniques for resolving network intrusion reconnaissance issues. It is worth mentioning that the test database was not used to train the CI techniques, only to validate the same.

Keywords: Network Intrusion Detection System. Artificial Neural Networks. Support Vector Machines. Random Forests.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A comodidade e, em contrapartida, a dependência que a utilização da internet nos proporciona, tais como a facilidade, para encontrar informações, as redes sociais e sistemas de gestão pessoal/empresarial online, faz com que sua utilização continue crescendo, exponencialmente, ao passar dos anos. A grande maioria da população mundial hoje, de alguma forma, depende da internet, tanto no ambiente empresarial ou acadêmico quanto doméstico (CUNHA NETO, 2011).

Anomalias de tráfego ocasionando falhas, principalmente negação de serviço, são a cada dia mais comuns. A capacidade de identificar, diagnosticar e tratar essas anomalias é de extrema importância para manter uma rede operando (BARFORD et al., 2002). Assim, é necessária a utilização de um sistema que possa auxiliar na segurança desse volume de dados. Em diferentes áreas de utilização da internet, são utilizados programas e sistemas operacionais que podem apresentar problemas de segurança. Para detectar e prevenir os eventuais usos destas vulnerabilidades, nos ambientes computacionais, é necessário que um Sistema de Detecção de Intrusão em Redes (*NIDS - Network Intrusion Detection System*) seja utilizado (MAFRA et al., 2008). Um NIDS é um conjunto de ferramentas de *software* que permite a análise e detecção de intrusões em redes de dados (AZEVEDO, 2012).

Os sistemas detectores de intrusão em redes têm como característica oferecer um método, para reduzir a possibilidade de intrusão, pela antecipação e acompanhamento dos ataques (HEINEN; OSÓRIO, 2006). O principal objetivo de um Sistema de Detecção de Intrusão é ser capaz de alcançar uma alta taxa de acertos e uma baixa taxa de alarmes falsos (SOUZA; MONTEIRO, 2008). Sistemas de Detecção de Intrusão podem ser classificados de três diferentes formas: baseados em *hosts*, baseados em redes e híbrido, os quais são apresentados na Seção 2.2. Também é possível classificá-los pela metodologia de detecção, por anomalias ou por assinaturas (WANG, 2009).

Diversas propostas apresentam o problema de detecção de intrusão em redes de computadores com a utilização de técnicas de Inteligência Computacional, porém não são aplicadas em ambiente real. A aplicação de um NIDS, com técnicas de Inteligência Computacional, em ambiente real, pressupõe diversos desafios, tais como o pré-processamento dos dados *online*. Esta dissertação limita-se ao desenvolvimento de um Sistema de Detecção de Intrusão em Redes (*NIDS - Network Intrusion Detection System*), por anomalias, em ambiente real, utilizando

e avaliando as seguintes técnicas de Inteligência Computacional: Redes Neurais Artificiais, Máquinas de Vetores de Suporte e Florestas Aleatórias.

1.1 Objetivos

Esta seção descreve o objetivo geral deste trabalho e seu desmembramento em objetivos específicos.

1.1.1 Objetivo Geral

Este trabalho objetiva, principalmente, a investigação de técnicas de detecção de intrusão em redes de computadores, utilizando métodos baseados em Inteligência Computacional, em ambiente real. A partir de uma base de dados com informações de conexões, obtidas pelo tráfego de redes de computadores, as técnicas de Redes Neurais Artificiais, Máquinas de Vetores de Suporte e Florestas Aleatórias serão implementadas, testadas e avaliadas. Com o intuito de realizar testes em ambiente real, faz-se necessário propor uma nova base de dados, contendo uma porção do tráfego de uma rede real atual. Por fim, propor uma API (*Application Programming Interface*), para o desenvolvimento de um NIDS compatível a essas e outras técnicas de Inteligência Computacional e com a capacidade de criar novas bases de dados de tráfego de rede.

1.1.2 Objetivos Específicos

Os objetivos específicos estão divididos nos tópicos a seguir.

1. Estudar as técnicas de Redes Neurais Artificias, Máquinas de Vetores de Suporte e Florestas Aleatórias, suas bibliotecas e/ou ferramentas.

2. Desenvolver uma API para capturar e preprocessar o tráfego de rede.

3. Pré-processar a base de dados ISCX [1] 2012 com a API.

4. Utilizar as bibliotecas e/ou ferramentas das técnicas de Inteligência Computacional, aplicando a base de dados ISCX 2012.

5. Criar uma base de dados, em ambiente real, para testes.

[1] Information Security Centre of Excellence - University of New Brunswick

6. Unir a API ao módulo de Inteligência Computacional (via *socket*), formando assim um NIDS.

7. Avaliar os resultados de cada técnica.

1.2 Justificativa

De acordo com Uchoa (2009), vulnerabilidades em aplicações são descobertas com frequência, portanto é possível afirmar, com absoluta tranquilidade, que não existem servidores totalmente seguros. Visto que são necessários equipamentos de segurança (*firewalls*) para manter uma rede segura, também, é necessário que um Sistema de Detecção de Intrusão em Redes seja utilizado em conjunto com essas ferramentas.

Um Sistema de Detecção de Intrusão em Redes (NIDS) é capaz de alertar o administrador de redes sobre um possível ataque que não foi detectado pelo *firewall* ou outros meios de segurança na rede. Sendo assim, é possível que o administrador da rede analise os alarmes gerados pelo NIDS e reconfigure os equipamentos de segurança. Portanto fica evidente a oportunidade de uma ampla área de pesquisa e desenvolvimento que é a segurança da informação.

Alguns trabalhos relacionados propõem metodologias de Sistemas de Detecção de Intrusão (IDS) com técnicas de Inteligência Computacional (SILVA, 2011), porém não são aplicados em ambiente real e não utilizam apenas as características de NIDS — contando com várias informações obtidas de *hosts* específicos, e não apenas do tráfego de rede. Desta forma, não é possível verificar a efetividade do método para diferentes tipos de infraestrutura e tráfegos, que não os da base de dados utilizada para o treinamento dessas técnicas. Ao se desenvolver uma API que realiza o pré-processamento dos dados *online*, é possível realizar experimentos em ambiente real e com diversas infraestruturas de rede.

1.3 Estrutura do trabalho

Esta dissertação de mestrado está organizada da seguinte forma: no capítulo 2, é apresentada uma revisão bibliográfica sobre os termos e técnicas utilizadas na pesquisa. Em seguida, no capítulo 3, são apresentados os materiais e métodos necessários para que a dissertação seja executada. Logo após, nos capítulos 4 e 5, são apresentados os resultados preliminares e as conclusões oriundas desta dissertação. Por fim, a seção 5.1 apresenta as propostas de continuidade para trabalhos futuros.

2 REVISÃO BIBLIOGRÁFICA

Esta seção apresenta uma breve revisão bibliográfica sobre os assuntos que possuem relação com a pesquisa proposta.

2.1 Redes de computadores

Redes de computadores são a cada dia mais presentes e importantes na sociedade atual. Praticamente todos os profissionais, estudantes e pessoas comuns utilizam, de alguma forma, esse recurso. Como exemplo, pode-se citar a utilização de um sistema bancário que permite que transações bancárias possam ser realizadas fora da agência de origem com relativa facilidade (UCHOA, 2009).

Em virtude da grande expansão de redes de computadores e da necessidade de comunicação de dados padronizada, criaram-se modelos de padronização para que todos os dispositivos possam comunicar-se. De acordo com Tanenbaum (2011), os principais modelos adotados são o modelo OSI e o TCP/IP que serão detalhados, embasados, principalmente, na referência bibliográfica (TANENBAUM, 2011), nas próximas seções.

2.1.1 Modelo OSI

Este modelo baseia-se em uma proposta desenvolvida pela ISO (*Internation Standards Organization*) como um primeiro passo em direção à padronização internacional dos protocolos empregados nas diversas camadas. De acordo com Uchoa (2009), esse modelo é pouco adotado em redes reais, mas muitos dos seus princípios influenciam os especialistas na área. A Tabela 2.1 apresenta o modelo, que foi dividido em 7 diferentes camadas, cada uma com uma função específica.

2.1.2 Modelo TCP/IP

Este modelo começou a ser utilizado na ARPANET (rede de computadores geográfica, destinada à pesquisa). De acordo com Uchoa (2009), a maioria das implementações atuais utilizam-se do Modelo TCP/IP, também chamado de Arquitetura Internet. Diferentemente do modelo OSI, esse modelo conta apenas com 4 camadas definidas, as quais serão apresentadas a seguir (TANENBAUM, 2011).

Tabela 2.1 – Camadas do Modelo OSI e suas funções

Camada	**Principal Função**	**Unidade**
Aplicação	Fornecer a interface de aplicação do usuário com o protocolo de comunicação	Dados
Apresentação	Representação da codificação dos dados	Dados
Sessão	Estabelecimento de sessões entre hosts da rede	Dados
Transporte	Conexão origem - destino	Segmentos
Rede	Determinação da rota e endereçamento lógico	Pacotes
Enlace	Endereçamento físico	Quadros (frames)
Física	Transmissão do sinal por meio	Bits

Fonte: Modificado de Uchoa (2009)

1. **Camada *host*/Rede:** o primeiro nível do modelo TCP/IP é composto pela camada de host/rede. Essa camada tem como função encarregar-se da conexão do *host* à rede, utilizando algum protocolo para que seja possível enviar pacotes IP na camada superior.

2. **Camada Inter-Redes ou Internet**: essa camada tem como objetivo permitir que os *hosts* injetem pacotes em qualquer rede e garantir que esses pacotes trafegarão, independentemente, até o destino. Dessa forma, os pacotes podem chegar a uma ordem diferente daquela na qual foram disparados, obrigando as camadas superiores a reorganizá-los, caso a entrega em ordem seja necessária. A camada de inter-redes também define um formato de pacote oficial e um protocolo chamado IP (*Internet Protocol*). O roteamento de pacotes é uma questão de grande importância nessa camada, assim como a necessidade de evitar o congestionamento. Por esses motivos, pode-se afirmar que a função da camada inter-redes do TCP/IP é muito parecida com a camada de rede do modelo OSI.

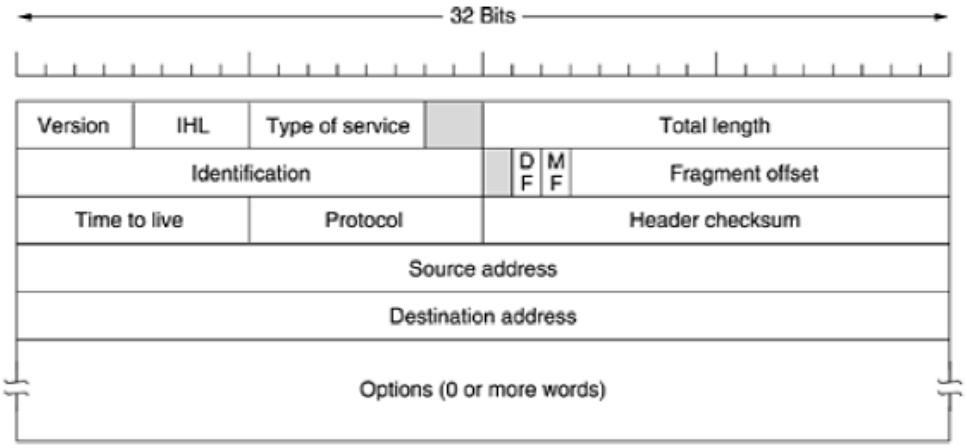
3. **Camada de Transporte:** a camada de transporte é localizada em um nível acima da camada inter-redes. O objetivo dessa camada é permitir que o *host* de origem e de destino possam comunicar-se (essa camada é idêntica à camada de transporte apresentada pelo modelo OSI). Neste nível foram definidos dois protocolos fim a fim. São eles: o TCP (*Transmission Control Protocol*) que é orientado à conexão, o qual permite permite a entrega sem erros de um fluxo de bytes originado de uma máquina com destino a outra, e o protocolo UDP, que não é orientado a conexão e não possui confirmação de recebimento.

4. **Camada de aplicação:** o modelo TCP/IP não traz consigo as camadas de sessão e apresentação, pois notou-se que a utilização dessas duas camadas é restrita apenas a algumas aplicações. A camada de aplicação contém todos os protocolos de nível mais alto. Dentre eles estão os protocolos: TELNET, FTP, HTTP, SMTP, POP3, NNTP e outros. Essa é a camada na qual a maioria dos usuários, geralmente, tem mais conhecimento.

2.1.3 Protocolo IP

Este protocolo está incluso na camada de rede do modelo OSI e na camada Inter-redes no modelo TCP/IP. O pacote do protocolo IP consiste em uma parte de cabeçalho e uma parte de dados propriamente ditos. Esse cabeçalho é dividido em duas partes, uma fixa e outra variável. O formato do cabeçalho é apresentado na Figura 2.1. Também vale ressaltar que ele é transmitido em uma ordem *big endian*, ou seja, da esquerda para a direita, com o bit de mais alta ordem do campo *Version* aparecendo primeiro. Nas máquinas *litte endian*, é necessária a conversão em *software* para transmissão e recepção (TANENBAUM, 2011).

Figura 2.1 – Cabeçalho do pacote IP



Fonte: Tanenbaum (2011)

Praticamente todos os protocolos de transporte da *Internet* fazem uso do protocolo IP (camada de inter-redes) para transferir dados de um *host* de origem a um *host* de destino. Esse protocolo não é orientado a conexão e não possui garantia de entrega de pacotes, o que é feito na camada superior (transporte) (UCHOA, 2009).

2.1.4 Protocolo TCP

De acordo com Tanenbaum (2011), o TCP (*Transmission Control Protocol*) foi projetado, especificamente, para oferecer um tráfego de dados confiável em uma arquitetura de rede não confiável. A camada de rede não oferece garantia de que os datagramas enviados chegarão ao destino ou serão entregues, de forma apropriada, a partir disto, a camada de transporte (com protocolo TCP) tem a função de administrar o tráfego. Também cabe ao protocolo TCP reorganizar os datagramas que chegam fora de ordem. Enfim, o protocolo TCP tem a função de fornecer a confiabilidade de que a maioria dos usuários precisa, mas que o protocolo IP não oferece.

Uma conexão TCP é estabelecida por meio de *sockets* no *host* de origem e no *host* de destino. Um *socket* é composto por um endereço IP e um número de 16 *bits* que define a porta. Um *socket* é utilizado para um *host* ouvir ou transmitir, em uma determinada porta, sendo que as portas abaixo de 1024 são destinadas a serviços padrões e as portas acima são livres para utilização (TANENBAUM, 2011).

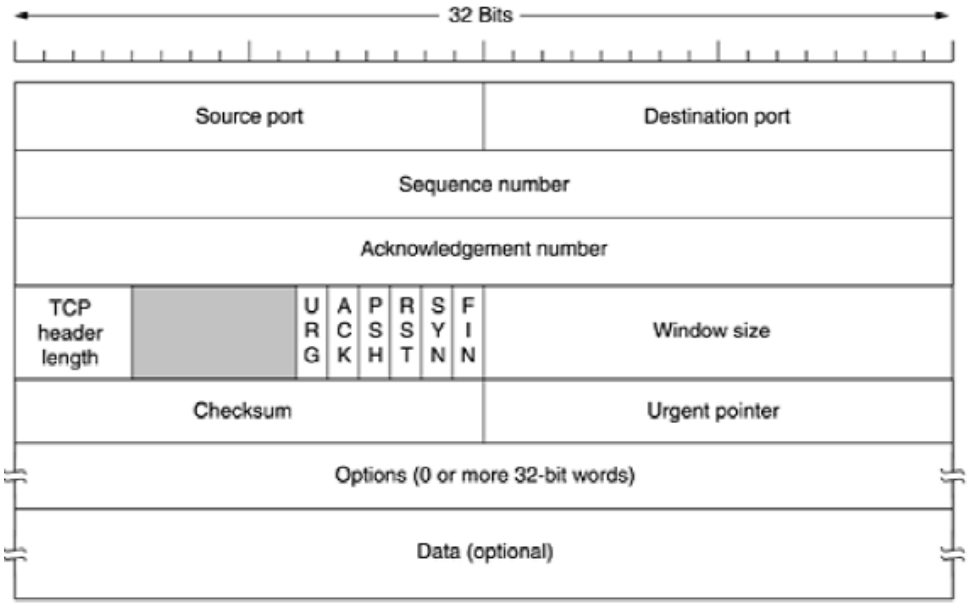
O protocolo TCP tem como característica a numeração para cada conexão que é composta por 32 *bits*. O emissor e receptor TCP trocam dados em forma de segmentos. Um segmento TCP é constituído por um cabeçalho fixo de 20 *bytes*, seguido por zero ou mais bytes de dados. O *software* do *socket* define qual será o tamanho dos segmentos. Portanto, é necessário respeitar os limites da camada mais baixa (Rede), que utiliza o protocolo IP e tem uma limitação, na carga do pacote, que é de 65.515 *bytes*. Outro ponto importante a ser respeitado é o MTU (*Maximum Transfer Unit*), em que cada segmento deve caber no tamanho de MTU definido pelo administrador da rede. Esse valor, geralmente, é de 1500 *bytes* (TANENBAUM, 2011).

O cabeçalho de segmento do TCP começa com um tamanho fixo de 20 *bytes* e é apresentado na Figura 2.2. De acordo com Tanenbaum (2011), cada parte do cabeçalho tem uma função específica que é explicada a seguir:

1. ***Source port e Destination port*:** porta de origem e porta de destino da conexão.

2. ***Sequence Number e Acknowledgment Number*:** ambos possuem 32 bits. O *Acknowledgment Number* especifica o próximo byte esperado (não o último *byte* recebido corretamente).

Figura 2.2 – Cabeçalho do TCP



Fonte: Tanenbaum (2011)

3. ***TCP Header length*:** traz consigo quantas palavras de 32 bits existem no cabeçalho TCP, ou seja, o tamanho do cabeçalho em palavras.

4. ***URG, ACK, PSH, RST, SYN, FIN*:** são *flags* de 1 bit. O valor 1 é definido no URG se o *Urgent Pointer* estiver ativado. No ACK, se o *Acknowledgment Number* estiver ativado. No PSH, a *flag PUSH* está ativada. No RST, a conexão é reinicializada. No SYN, é utilizado para estabelecer conexões. No FYN, é utilizado para encerrar uma conexão.

5. ***Window Size*:** este campo é a chamada "Janela Deslizante", ou seja, indica a quantidade de *bytes* que podem ser enviados a partir do *byte* confirmado.

6. ***Checksum*:** é fornecido para aumentar a confiabilidade, por meio do algoritmo de *Checksum*.

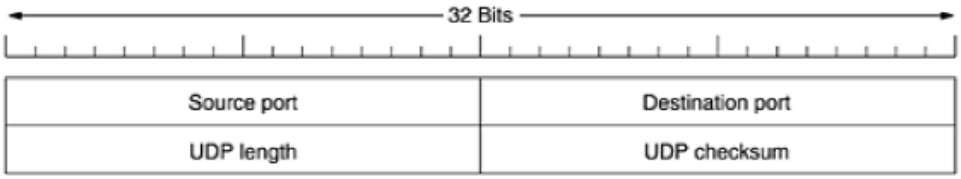
7. ***Options*:** este campo tem como função oferecer recursos extras, ou seja, recursos que não foram projetados pelo cabeçalho comum.

2.1.5 Protocolo UDP

O protocolo UDP (*User Datagram Protocol*) caracteriza-se por ser um protocolo de transporte sem conexões. O cabeçalho do UDP (Figura 2.3) possui apenas 8 *bytes* (enquanto

o TCP precisa de 20), mais os dados. As portas de origem e destino servem para endereçar o pacote, o campo *UDP length* inclui o cabeçalho de 8 bytes e os dados, e o campo UDP *checksum* é opcional e pode ser armazenado com 0 se não for calculado (Faz uma checagem superficial do conteúdo dos pacotes (UCHOA, 2009)). Desativar o campo *checksum* não é aconselhável, a menos que a qualidade dos dados não tenha importância. Serviços como DNS utilizam o protocolo UDP (TANENBAUM, 2011).

Figura 2.3 – Cabeçalho UDP



Fonte: Tanenbaum (2011)

2.1.6 Protocolo ICMP

O protocolo ICMP *Internet Control Message Protocol* tem sua definição na camada de rede do modelo TCP/IP. Este protocolo tem a função de "mensageiro", essas mensagens são encapsuladas em pacotes IP. De acordo com Filippetti (2011), apesar deste protocolo fazer parte da camada de rede e ser encapsulado por um cabeçalho IP, o tratamento é totalmente diferenciado, isso porque faz-se necessário abrir o conteúdo do pacote ICMP para identificar qual o tratamento adequado.

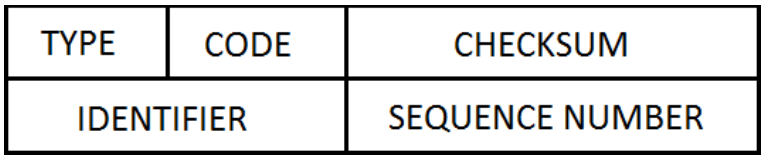
De acordo com Filippetti (2011), os tipos de mensagens mais comuns do ICMP são:

1. Tipo 0: *Echo Replay* (resposta a um ping).
2. Tipo 3: *Destination Unreachable* (destino inalcançável).
3. Tipo 8: *Echo Request* (solicitação ping).
4. Tipo 11: *Time Exceed* (TTL excedido).
5. Tipo 30: *Traceroute*.

A Figura 2.4 representa o cabeçalho ICMP, de acordo com Postel et al. (1981). O campo *Type* identifica o tipo da mensagem e o campo *Code* identifica a mensagem. Já o *checksum* permite realizar a verificação de erros no pacote. Por fim, o campo *identifier* permite identificar um

tráfego ICMP entre dois *hosts* e o campo *Sequence Number* é utilizado para definir a sequência das mensagens ICMP.

Figura 2.4 – Cabeçalho do Protocolo ICMP



Fonte: Postel et al. (1981)

2.2 Sistemas de detecção de intrusão

A implementação de tecnologias antimaliciosas nas redes de computadores é muito importante para a manutenção da segurança dessas redes. A utilização de Sistemas de Detecção de Intrusão é bem-vinda, para que seja possível realizar o monitoramento e análise (pela implementação de *honeypots* [1]) do tráfego, que ingressa na rede, passando pelo *firewall* [2] (WANG, 2009). Este processo é de extrema importância, pois permite que o administrador da rede consiga estabelecer regras mais concisas nos ativos de segurança.

O termo Detecção de Intrusão define-se como o processo de monitorar eventos que estão ocorrendo em um sistema computacional ou em uma rede de computadores, buscando tráfego intrusivo. Incidentes de segurança possuem várias causas, como a propagação de um *malware* (Ex.: *worms, spyware, trojan*), atacantes tentando elevar privilégios para acessar sistemas não autorizados, entre outros (SCARFONE, 2007). Dessa forma, uma tecnologia tornou-se aliada aos administradores de rede e segurança, são os Sistemas de Detecção de Intrusão (IDS), cuja função é reconhecer um comportamento anômalo ou uma ação intrusiva e reportar ao administrador da rede para tomar as medidas necessárias (LAUREANO; MAZIERO; JAMHOUR, 2003).

2.2.1 Métodos de detecção de intrusão

A forma pela qual os Sistemas de Detecção de Intrusão (IDS) reconhecem uma ação intrusiva pode ser baseada em assinaturas ou em anomalias. Sistemas de detecção, baseados em assinaturas identificam ataques por meio da análise de assinaturas, previamente configuradas,

[1] Sistema para capturar metodologias de ataque.
[2] Dispositivo para bloquear/liberar determinado tipo de tráfego.

sobre o comportamento padrão de algum tipo de ataque. No caso do IDS, baseado em anomalias, é analisado o tráfego da rede, classificando o tráfego em anômalo ou normal (AZEVEDO, 2012). A principal vantagem de um IDS, baseado em anomalia, é que é possível detectar ataques desconhecidos, o que não acontece na detecção por assinatura (WANG, 2009). Os modos de reconhecimento de intrusão serão apresentados mais detalhadamente a seguir:

1. **Sistemas de Detecção, baseados em assinaturas ou abuso**, são bastante utilizados em virtude do baixo custo computacional exigido. A detecção de intrusões é feita por meio de regras bem definidas, geralmente, obtidas com a utilização de *honeypots*, em que o tráfego é analisado e comparado a um ataque. Dessa forma, o IDS é bastante eficiente e rápido, porém, é incapaz de identificar novos ataques (SILVA, 2011; WANG, 2009).

2. **Sistemas de Detecção, baseados em anomalia,** fazem a supervisão do comportamento do sistema, no qual assume-se que atividades anormais são classificadas como intrusões (SILVA, 2011). Este método pode ser implementado com várias técnicas, entre elas Inteligência Computacional (SEN; SEN; CHATTOPADHYAY, 2014), Estatística (RAJAGOPAL; THILAKAVALLI; FATHIMA, 2015), Transformada de *Wavelet* (AZEVEDO, 2012) e outras.

2.2.2 Classificação dos sistemas de detecção de intrusão

De acordo com IATAC (2009) e Wang (2009), Sistemas de Detecção de Intrusão podem ser classificados em três diferentes categorias: IDS baseado em *hosts* (*HIDS - Host Based Intrusion Detection System*), IDS baseado em redes (*NIDS - Network Intrusion Detection System*) e, ainda, IDS híbrido. Um HIDS analisa o comportamento de um *host*, em específico, geralmente, pelas informações obtidas por SNMP [3]. Um NIDS analisa somente o tráfego de rede (é o modelo mais utilizado). Por fim, o IDS híbrido faz a união das características do HIDS e NIDS (AZEVEDO, 2012).

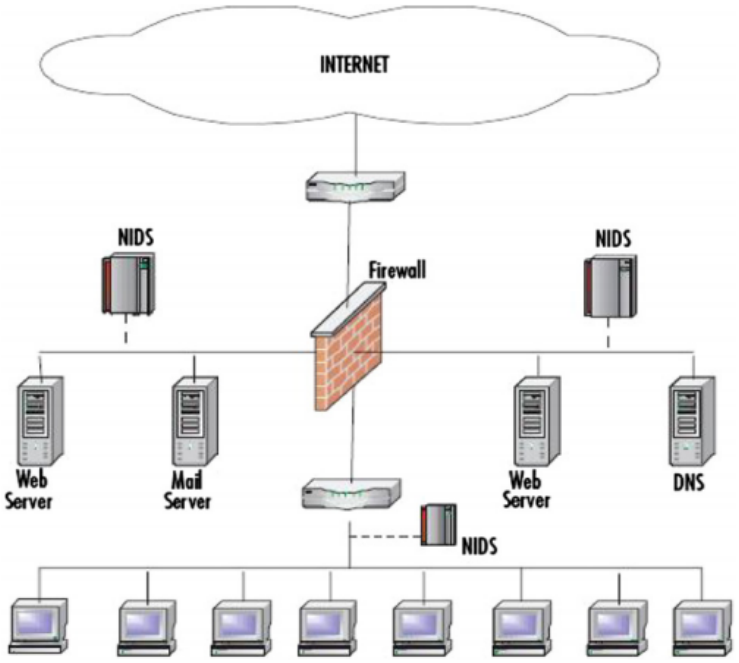
Detecções de intrusões baseadas em *host* (HIDS), são caracterizadas por possuírem um agente de monitoramento instalado no *host* alvo. Esse agente monitora os *logs* de eventos do sistema e alerta a estação de monitoramento caso alguma ação maliciosa seja detectada. As ações monitoradas pelo agente podem ser desde manipulações de arquivos até estabelecimento de novas conexões (WANG, 2009).

[3] Protocolo de aplicação para gerenciamento de redes de computadores

Podem-se citar algumas vantagens, na utilização da técnica HIDS, para detecção de intrusão, são elas: podem detectar ataques criptografados, pois serão descriptografados quando chegarem ao *host*, podem detectar ataques com pacotes fragmentados e não precisam de *hardware* especial. Contudo, também, é possível citar algumas desvantagens em relação a essa técnica: consomem recursos extras nos *hosts* hospedeiros (CPU, armazenamento, memória RAM), ataques que afetam essas máquinas, também, podem afetar o funcionamento do HIDS, não podem ser instalados em conjunto com roteadores e/ou *switches* (WANG, 2009).

Por outro lado, um Sistema de Detecção de Intrusão em Redes (NIDS) trabalha analisando o tráfego de rede, geralmente, utilizando uma interface em modo promíscuo, funcionando como um *sniffer* [4] (UCHOA, 2009). Esse tipo de sistema, geralmente, atua com um ou mais sensores na rede e uma estação de monitoramento. Quando um sensor detecta uma atividade anormal na rede, um alerta é transmitido à estação de monitoramento que informará ao administrador da rede sobre a situação. A Figura 2.5 apresenta como deve ser realizada a implementação de um NIDS na rede (LABS, 1999).

Figura 2.5 – Posicionamento de um NIDS na rede



Fonte: Maltainfosec (2011)

As principais vantagens da utilização de um NIDS são: baixo custo, possui boa taxa de acertos para uma série de ataques, não requer alterações em servidores e/ou *hosts* em produção,

[4] Ferramenta de captura de tráfego de rede

não causa gargalo ou indisponibilidade na rede (pois é um equipamento que funciona de forma passiva). Algumas desvantagens, também, podem ser citadas: incapacidade de detectar ataques oriundos de protocolos criptografados (SSL, IPsec, SSH) e ataques fragmentados (LABS, 1999; WANG, 2009).

Como exemplo de NIDS, utilizando tanto o método por assinaturas quanto por anomalias, é possível citar o *Snort* (HIDS e NIDS) (SNORT IDS, 2015).

2.2.3 Métodos de análise de eficácia

Algumas políticas de detecção de intrusão são utilizadas para identificar atividades intrusivas. Essas políticas, geralmente, especificam qual o tipo de atividade será considerada como intrusiva e, também, como o sistema deverá responder a esta atividade. Políticas de detecção ideais devem ser simples, eficazes e fáceis de implementar, mas também preocupando-se em gerar o mínimo de falsos alarmes (WANG, 2009).

Para realizar a análise de eficácia de qualquer tipo de IDS, deve-se levar em consideração a análise quantitativa dos seguintes pontos (WANG, 2009):

1. **Falsos Positivos (FP):** é a situação em que um tráfego normal é analisado, mas o IDS acusa como sendo parte de um tráfego intrusivo.

2. **Falsos Negativos (FN):** ocorre quando um tráfego malicioso é analisado e o IDS classifica-o como normal.

3. **Verdadeiros Positivos (VP):** ocorre quando um tráfego intrusivo é analisado e o IDS classifica-o como intrusivo, em outras palavras, quando o IDS acerta a classificação intrusiva.

4. **Verdadeiros Negativos (VN):** ocorre quando um tráfego normal é analisado e o IDS classifica-o como normal, em outras palavras, quando o IDS acerta a classificação de dados normais.

Com base nesses parâmetros, é possível apresentar uma matriz de confusão (Tabela 2.2), a qual permite analisar a eficácia do IDS.

Outra técnica, para classificar a eficácia de um IDS, é a utilização do coeficiente Kappa. O coeficiente Kappa é uma métrica de concordância, aplicada para medir o nível de concordância ou discordância de classificadores, observando um mesmo fenômeno (ARAÚJO et al., 2013; COHEN et al., 1960).

Tabela 2.2 – Matriz de Confusão

	Positivo	**Negativo**	**Total**
Positivo	VP	FN	VP+FN
Negativo	FP	VN	FP+VN
Total	VP+FP	VN+FN	VN+VP+FP+FN

Fonte: Modificado de Wu e Banzhaf (2010)

De acordo com Araújo et al. (2013), o coeficiente Kappa, para problemas de detecção de Intrusão, mede a proporção de concordância observada (Po) entre as diferentes classes (real e prevista) sobre a proporção de concordância devido ao acaso (Pa). O cálculo do coeficiente é apresentado pelas Equações 2.1, 2.2, 2.3.

$$k = \frac{(Po - Pa)}{(1 - Pa)} \tag{2.1}$$

$$Po = VP + VN \tag{2.2}$$

$$Pa = (VP + FN) * (VP + FP) + (FN + VN) * (FP + VN) \tag{2.3}$$

O valor resultante do coeficiente Kappa representa a consistência dos resultados obtidos, onde valores aproximados a zero descrevem que os resultados apresentados foram obtidos por mero acaso, por outro lado, valores aproximados a 1 representam alta confiabilidade dos dados obtidos (Tabela 2.3) (COHEN et al., 1960).

Tabela 2.3 – Valores Kappa

Coeficiente Kappa	**Consistência dos Resultados**
0,40	Pobre
0,40 e <= 0,75	Satisfatório a Bom
0,75	Excelente

Fonte: Modificado de Fonseca, Silva e Silva (2013)

Por fim, outra técnica que pode ser utilizada, para a avaliação da eficácia de um classificador, são os gráficos ROC. De acordo com Prati, Batista e Monard (2008), o gráfico ROC é baseado na probabilidade de detecção, ou taxa de verdadeiros positivos, e na probabilidade de

falsos alarmes (ou falsos positivos). A construção de um gráfico ROC dá-se pela disposição da taxa de falsos positivos no eixo X e pela taxa de verdadeiros positivos no eixo Y. A interpretação do gráfico ROC é feita, baseando-se na Figura 2.6 e seguindo as seguintes diretrizes, de acordo com Prati, Batista e Monard (2008):

1. Ponto (0,0): o modelo assume a estratégia de nunca classificar um exemplo como positivo, ou seja, não apresenta falsos positivos, mas também não classificam nenhum verdadeiro positivo.

2. Ponto (1,1): assume a estratégia inversa do Ponto (0,0), ou seja, sempre classifica um exemplo como positivo.

3. Ponto (0,1): representa o modelo de classificação perfeito, todos os exemplos positivos e negativos são, corretamente, classificados.

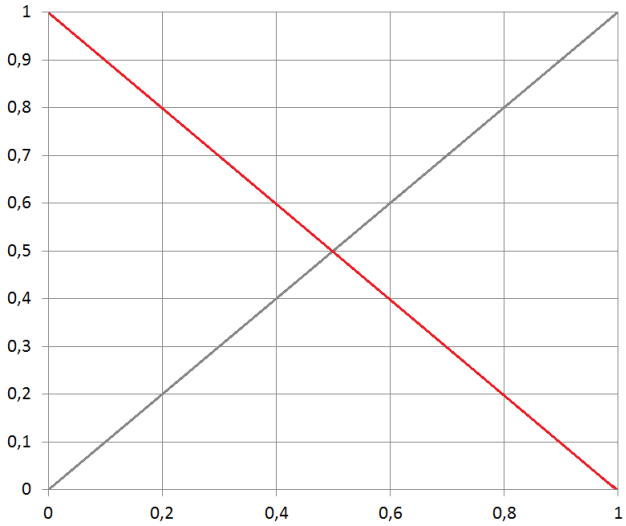
4. Ponto (1,0): representa o inverso do ponto (0,1), sempre faz as predições de forma incorreta.

Desta forma, analisando a Figura 2.6, percebe-se que os modelos mais próximos ao canto inferior esquerdo fazem uma classificação positiva somente se têm grande segurança na classificação, esses modelos podem ser considerados “conservadores”. Modelos desse tipo, geralmente, apresentam poucos falsos positivos, mas têm baixas taxas de verdadeiros positivos. Por outro lado, modelos próximos ao canto superior direito são considerados “liberais”, pois classificam a classe positiva com maior frequência, de tal maneira que realizam a classificação das amostras positivas, corretamente, porém com altas taxas de falsos positivos (PRATI; BATISTA; MONARD, 2008).

De acordo com Prati, Batista e Monard (2008), tomando como base a Figura 2.6 e a linha diagonal ascendente (Pontos (0,0) e (1,1)), é possível determinar que os pontos pertencentes ao triângulo superior esquerdo a essa diagonal representam modelos que trazem resultados melhores que o aleatório entretanto, no triângulo inferior direito, o resultado é o inverso. Já na diagonal descendente (Pontos (0,1) e (1,0) — diagonal em vermelho) são representados os modelos de classificação que classificam, com eficácia equivalente, ambas as classes. À esquerda dessa diagonal são representados os modelos que desempenham melhor para a classe negativa, e à direita, os modelos que desempenham melhor para a classe positiva.

A utilização destes métodos permite a avaliação de diferentes classificadores. Um método para o reconhecimento de intrusão em redes de computadores, pode ser a utilização de

Figura 2.6 – Exemplo de Gráfico ROC



Fonte: Modificado de Prati, Batista e Monard (2008)

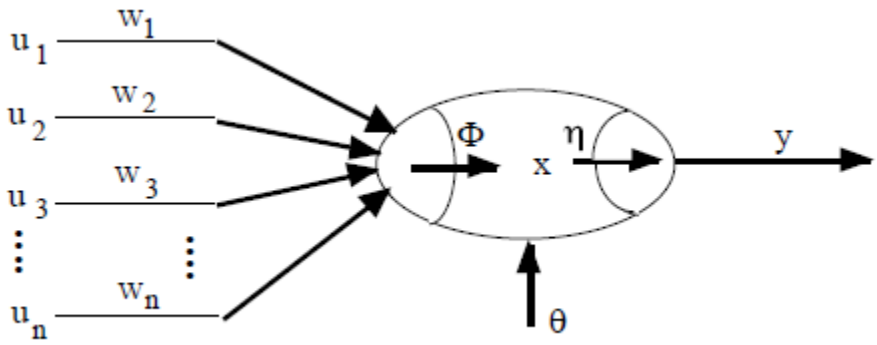
técnicas de Inteligência Computacional, como Redes Neurais Artificiais, Máquinas de Vetores de Suporte e Florestas Aleatórias. Estas técnicas serão detalhadas a seguir, nas seções 2.3, 2.4 e 2.5.

2.3 Redes neurais artificiais

Em 1943, *Walter Pitts* e *Warren McCulloch* propuseram o primeiro modelo computacional de um neurônio biológico. Grande parte do trabalho, envolvendo redes neurais artificiais, baseia-se em aprimorar métodos de aprendizado. Este tema veio a ser objeto de estudo somente após alguns anos da proposta de *Pitts* e *McCulloch* (BRAGA; FERREIRA; LUDERMIR, 2007).

Redes neurais artificiais buscam imitar o funcionamento do cérebro humano, o qual é formado por cerca de 10^{11} neurônios. O neurônio do tipo *perceptron*, proposto por *Frank Rosenblatt* (1958), é o mais utilizado atualmente. O perceptron possui três estágios: o primeiro recebe os dados de entrada, o segundo recebe os impulsos oriundos do primeiro estágio e multiplica pelo peso de cada entrada, e o terceiro apresenta os resultados. Um perceptron (Figura 2.7) é constituído por corpo da célula, dendritos e o axônio (BRAGA; FERREIRA; LUDERMIR, 2007). Os dendritos são responsáveis por conduzir os sinais de entrada para o corpo celular. O corpo celular é responsável por fazer a soma (processamento) dos dados. As terminações do axônio são conectadas, diretamente, a dendritos de outros neurônios, formando, assim, uma rede de neurônios (BARRETO, 2002).

Figura 2.7 – Neurônio Artificial (*Perceptron*)

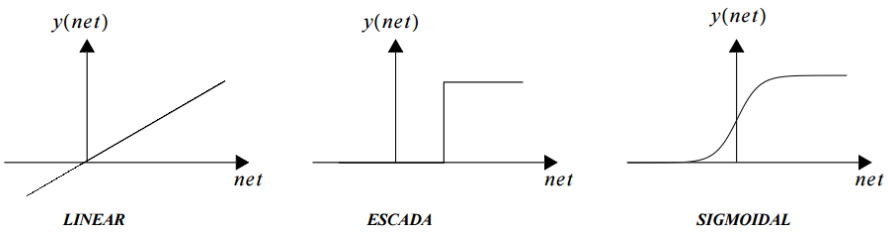


Fonte: Barreto (2002)

O principal atrativo de Redes Neurais Artificiais é, sem dúvida, a capacidade de generalização (BRAGA; FERREIRA; LUDERMIR, 2007). Generalização refere-se à capacidade de uma rede neural produzir saídas adequadas, para entradas que não estavam presentes, durante os dados de treinamento (etapa de aprendizagem) (SIMON, 2001). As RNAs, também, são capazes de realizar a extração de informações, que não foram apresentadas, de forma explícita, por meio dos exemplos (BRAGA; FERREIRA; LUDERMIR, 2007).

De acordo com o modelo proposto por *Walter Pitts* e *Warren McCulloch* em 1943, foram derivados outros modelos que fazem com que as saídas não sejam exclusivamente, 0 ou 1, mas sim com diferentes funções de ativação. A Figura 2.8 apresenta algumas funções de ativação que são aplicadas na saída da rede neural para "formatar" os dados de saída. A partir da Figura 2.8, pode-se perceber a representação de uma função de ativação linear, outra degrau ou escada (0 ou 1) e a mais utilizada, atualmente, que é a *Sigmoidal* (abrange valores de 0 a 1) (BRAGA; FERREIRA; LUDERMIR, 2007).

Figura 2.8 – Funções de ativação



Fonte: Modificado de Rauber (2005)

O aprendizado em Redes Neurais Artificiais é, sem dúvida, o fator mais importante para que haja sucesso na aplicação dessa técnica. Uma RNA tem a capacidade de aprender

pelos exemplos e, ainda, fazer interpolações e extrapolações do que aprendeu. Um algoritmo de aprendizado é definido como um conjunto de procedimentos, com a finalidade de adaptar os parâmetros de uma RNA, para que ela possa aprender uma determinada função. Existem vários algoritmos de aprendizado, cada um diferencia-se pelo modo de como o ajuste de pesos é realizado (BRAGA; FERREIRA; LUDERMIR, 2007). De acordo com Braga, Ferreira e Ludermir (2007), a etapa de aprendizagem consiste em um processo iterativo de ajuste dos pesos da rede neural.

Vários métodos de aprendizado foram propostos, podendo ser agrupados em duas categorias: aprendizado supervisionado e não supervisionado (BRAGA; FERREIRA; LUDERMIR, 2007). Estes dois métodos de treinamento serão detalhados a seguir:

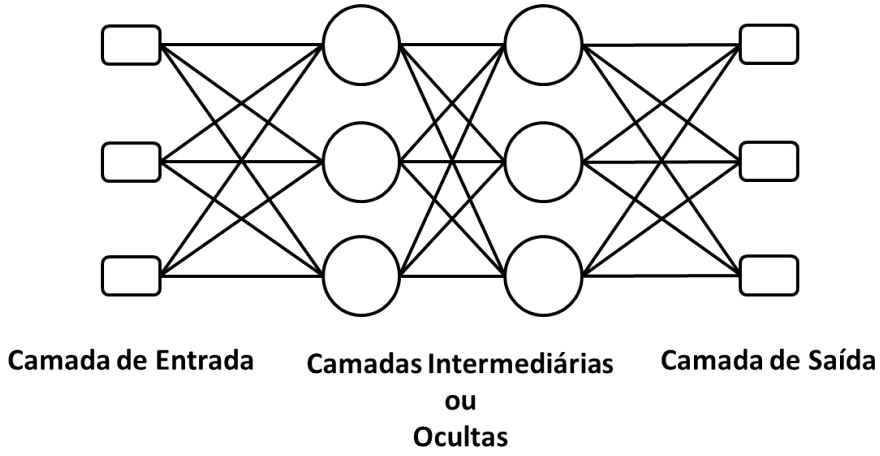
1. **Aprendizado Supervisionado:** nesse método, há um "professor" que controla o treinamento da rede, ou seja, a entrada e saída desejadas são fornecidas pelo "professor" (BARRETO, 2002). O "professor" indica um comportamento bom ou ruim para a rede, com objetivo de direcionar o processo de treinamento. A saída da rede é comparada com a saída desejada, recebendo informações do professor sobre o erro obtido. Os exemplos mais utilizados de algoritmos de aprendizado supervisionado são a regra *delta* e o algoritmo *backpropagation* que será detalhado posteriormente (BRAGA; FERREIRA; LUDERMIR, 2007).

2. **Aprendizado Não-Supervisionado:** nesse método não há um supervisor ou professor para acompanhar o processo de aprendizado (BRAGA; FERREIRA; LUDERMIR, 2007). Em outras palavras, é quando o valor desejado não é conhecido e não é utilizado durante o processo de treinamento (BARRETO, 2002).

De acordo com Bishop (1995), técnicas de Inteligência Computacional, tais como, aprendizado de máquina por redes neurais artificiais, têm apresentado avanços em vários campos da ciência. A maioria desses resultados veio de aplicações nas quais envolveu reconhecimento de padrões e fez uso de arquiteturas de redes neurais como: *Multi-layer perceptron - MLP* e redes RBF (*Radial based Function*). Podem-se citar alguns problemas envolvendo reconhecimento de padrões, os quais podem ser resolvidos a partir da aplicação de redes neurais artificiais: reconhecimento de caracteres, reconhecimento de intrusão em redes de computadores, previsões de dados financeiros, entre outros (BISHOP, 1995).

Uma nomenclatura utilizada em RNA é chamada de época. Uma época define-se por um ciclo em que todos os dados de entrada passam pelos pesos e, em seguida, os pesos são reajustados (NISSEN; NEMERSON, 2000).

Figura 2.9 – Arquitetura Rede Neural MLP



Fonte: Do Autor (2016)

2.3.1 Redes neurais MLP

De acordo com Braga, Ferreira e Ludermir (2007), redes neurais de uma só camada resolvem apenas problemas linearmente separáveis, diferentemente da maioria dos problemas reais que são não linearmente separáveis. O uso de duas camadas intermediárias permite a resolução de qualquer problema linearmente ou não-linearmente separável (CYBENKO, 1988).

A arquitetura de uma rede neural de múltiplas camadas (MLP) é composta por (Figura 2.9):

1. **Camada de Entrada:** em que os dados de entrada são apresentados. Não ocorre nenhum tipo de processamento nessa camada.

2. **Camadas Intermediárias:** são responsáveis pela precisão dos resultados obtidos pela rede. Todo o treinamento é realizado nas camadas intermediárias.

3. **Camada de saída:** local em que são apresentados os dados resultantes do treinamento da rede neural.

De acordo com Duda, Hart e Stork (2012), existem alguns fatores a serem definidos que influenciam no rendimento da rede neural. São eles:

1. **Número de camadas escondidas:** o número de camadas escondidas define a complexidade da fronteira de decisão (*decision boundary*), que é, em palavras simples, a linha traçada para separar dois ou mais tipos de dados. A definição da quantidade de camadas escondidas depende, diretamente, da complexidade do problema (se ele é linearmente separável ou não).

2. **Inicialização dos pesos:** é necessário inicializar os pesos das entradas de modo aleatório, com valores que vão, geralmente, de -1 a 1.

3. **Eliminação dos pesos:** essa regra consiste na eliminação de alguns pesos dos neurônios que não influenciam no resultado.

4. **Taxa de aprendizagem:** a taxa de aprendizagem pode interferir na velocidade em que a rede neural será treinada. A definição desse parâmetro varia conforme o tipo de problema a ser resolvido pela rede neural. Quando o valor da taxa de aprendizado é maior, o treinamento converge mais rapidamente, porém, quando este valor é muito elevado, o treinamento não converge.

5. **Taxa de Momentum:** a taxa de momentum é um fator utilizado para aumentar a velocidade do treinamento da rede. Esta taxa fornece uma "prévia" da mudança dos pesos comparados com os pesos atuais. Geralmente a taxa de momento é definida entre 0,3 a 0,5, mas existem casos em que esta taxa chega até a 0.9 (ATTOH-OKINE, 1999).

6. **Parada do treinamento:** para que não ocorra o *overfitting* (processo em que a rede decora os dados de entrada), é necessário parar o treinamento antecipadamente (*Early-Stopping*).

Ao notar-se o conceito intuitivo de que "quanto maior o número de camadas intermediárias, mais a rede conseguirá aprender", é preciso ter cautela, pois a utilização de muitas camadas intermediárias não é recomendada. A utilização de muitas camadas intermediárias faz com que o erro medido, durante o processo de treinamento, não seja, corretamente, ajustado, nas primeiras camadas (em virtude da propagação do erro), portanto o treinamento se torna menos útil ou preciso (BRAGA; FERREIRA; LUDERMIR, 2007).

2.3.2 Algoritmo *backpropagation*

Um dos algoritmos de aprendizado de RNA mais conhecidos e utilizados é o *backpropagation*. A proposta deste algoritmo foi responsável pelo ressurgimento do interesse na área de Redes Neurais Artificias. Este algoritmo faz o treinamento da rede de maneira supervisionada, ou seja, compara a saída obtida com a desejada e, em seguida, faz o ajuste dos pesos. O treinamento ocorre em duas fases, a fase de *forward* e *backward*. A primeira fase (*forward*) é responsável pelo processamento dos dados de entrada e por fornecer a saída da rede neural. A segunda fase (*backward*) é responsável por comparar os resultados obtidos com o desejado e, em seguida, voltar à arquitetura da rede para ajustar os pesos das entradas dos neurônios. A Equação 2.4 apresenta a regra de ajuste de pesos utilizada pelo algoritmo *Backpropagation*. A Figura 2.10 apresenta um esboço do funcionamento do *backpropagation* (BRAGA; FERREIRA; LUDERMIR, 2007).

$$w_{ji}^{(l)}(n+1) = w_{ji}^{(l)}(n) + \alpha[w_{ji}^{(l)}(n-1)] + \eta\delta_{j}^{(l)}(n)y_{i}^{(l-1)}(n) \tag{2.4}$$

Fonte: (SIMON, 2001)

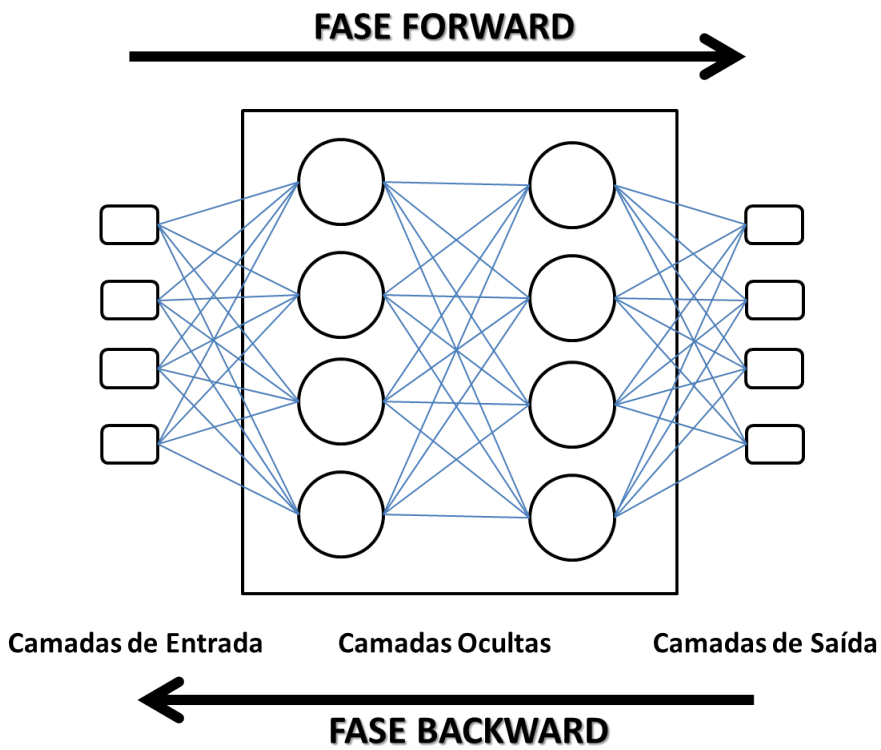
2.3.3 Biblioteca Pybrain

A biblioteca *Pybrain* trabalha em conjunto com a biblioteca científica *SciPy*, a aplicação da *Pybrain* restringe-se ao treinamento supervisionado, não supervisionado e por reforço de técnicas de Inteligência Computacional, tais como Redes Neurais Artificiais. Entre os algoritmos de aprendizagem supervisionados suportados pela biblioteca, encontra-se o *backpropagation* (SCHAUL et al., 2010). A qualidade da documentação da biblioteca foi um dos fatores pelo qual se escolheu utilizá-la.

Alguns autores citam a utilização da biblioteca em suas publicações, entre eles: Rückstieß, Felder e Schmidhuber (2008), Scalco (2015), Schaul e Schmidhuber (2009), Sehnke et al. (2008).

O *Pybrain* possui alguns requisitos, são eles: Scipy *(http://www.scipy.org)* e o Python 2.5 (ou versões mais atuais) (SCHAUL et al., 2010).

Figura 2.10 – Fases do Algoritmo *Backpropagation*.



Fonte: Do Autor (2016)

2.4 Máquinas de vetores de suporte

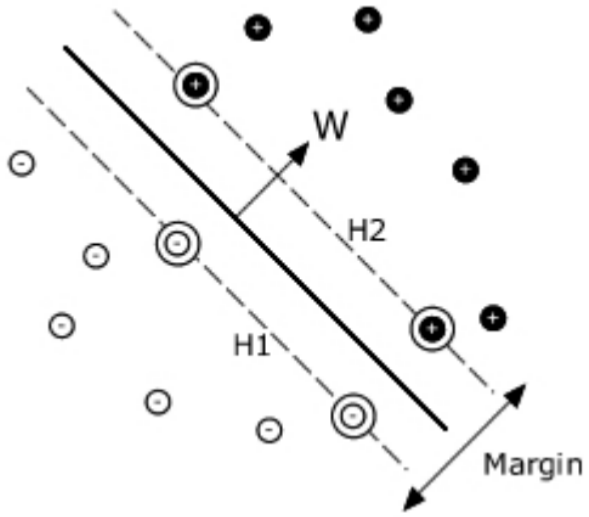
De acordo com Burges (1998) e Rodrigues et al. (2007), *Support Vector Machines* (SVM) ou Máquinas de Vetores de Suporte, é uma técnica utilizada, para o treinamento de classificadores, baseado no conceito da minimização do risco estrutural. Essa técnica foi proposta por Vladimir Vapnik, em 1979 (VAPNIK, 1995), por meio de fundamentos na teoria de aprendizagem estatística (LIMA, 2014). Nota-se um crescente interesse por pesquisadores, desde a década de 90, em problemas de reconhecimento de padrões (RODRIGUES et al., 2007), tais como detecção de intrusão, como se percebe em Mukkamala, Janoski e Sung (2002) e Deng, Zeng e Agrawal (2003).

Máquinas de Vetores de Suporte destacam-se por duas características: possuem uma fundamentação teórica plausível e podem alcançar alto desempenho em certas aplicações (SANTOS, 2002), também têm um grande poder de generalização (RODRIGUES et al., 2007). De acordo com Jalil, Kamarudin e Masrek (2010) e Shalev-Shwartz et al. (2011), em aplicações nas quais se exija precisão em classificação ou regressão, SVM é um método eficaz para tarefas de aprendizagem de máquina.

Esta técnica geralmente é utilizada para tratar dados de grande dimensão, nos quais outras técnicas de aprendizado obtém classificadores com pouco ou sobre ajuste. Uma característica atrativa é a convexidade do problema de otimização do treinamento, o qual faz com que exista apenas um único mínimo global. Essa é a principal vantagem de SVM em relação a Redes Neurais Artificiais (RNAs) (LORENA; CARVALHO, 2007).

De acordo com Rodrigues et al. (2007), SVMs são classificadores lineares que tem como objetivo separar os dados em duas ou mais classes, através da criação de um hiperplano de separação. Um hiperplano ótimo separa os dados de forma que a margem seja a maior possível, sendo que a mesma é definida através da soma das distâncias entre os pontos positivos e negativos mais próximos do hiperplano, o qual é criado através de um conjunto finito de dados de treinamento. Esses pontos compõe os vetores de suporte (pontos demarcados na Figura 2.11) (RODRIGUES et al., 2007).

Figura 2.11 – Classificação de um conjunto de dados utilizando SVM linear



Fonte: Rodrigues et al. (2007)

Existem alguns tipos de SVM, cada qual com uma finalidade específica, os mais populares são (CHANG; LIN, 2011):

1. SVC (*Support Vector Classification*): realiza o processo de classificação em duas ou mais classes.

2. One-Class: é um método que usa uma estratégia de classificação chamada *one-class* (SANTOS, 2002).

3. SVR (*Support Vector Regression*): realiza o processo de Regressão.

Algumas propostas, utilizando Máquinas de Vetores de Suporte em Sistemas de Detecção de Intrusão, são encontradas na literatura, tais como: Ambwani (2003), Chen, Hsu e Shen (2005), Li et al. (2003) e Rao, Dong e Yang (2003). Estas propostas afirmam a viabilidade da implementação de SVM, para solucionar problemas de detecção de intrusão, porém não são aplicadas a um NIDS.

2.4.1 Tipos de kernel

De acordo com Ben-Hur e Weston (2010), um *kernel* é um algoritmo que depende do produto escalar de um determinado conjunto de dados. Em outras palavras, um *kernel* é uma função de similaridade que é utilizada, dependendo do conjunto de dados, juntamente com o algoritmo de aprendizado.

Em muitos casos classificadores (*kernels*) não lineares proporcionam melhores taxas de acertos. Porém, conforme apresentado em Hsu, Chang e Lin (2003), em alguns casos um classificador linear pode se sobressair, são eles:

1. Número de amostras $<<$ Número de Características.
2. Número de amostras e Número de Características muito grandes.
3. Número de amostras $>>$ Número de Características.

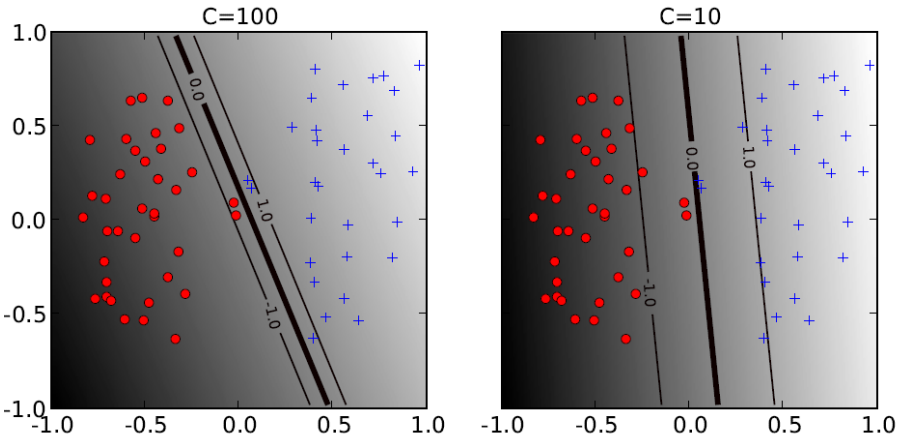
Os tipos de *kernel* mais comumente utilizados, de acordo com Hsu, Chang e Lin (2003), são o linear, polinomial, RBF e sigmoide. Hsu, Chang e Lin (2003) recomenda fortemente que o treinamento inicial do SVM seja realizado com *kernel* linear, o qual pode trazer algumas vantagens, tais como a economia de tempo e simplicidade do modelo. Caso o aprendizado, utilizando *kernel* linear, não resulte em um classificador eficaz, aconselha-se a tentativa com outros modelos de *kernel*.

2.4.2 Parâmetros de treinamento

Assim como outras técnicas de Inteligência Computacional, tais como RNA (apresentada na seção 2.3), é necessário realizar, manualmente, o ajuste de alguns parâmetros. A quantidade de parâmetros varia, conforme o *kernel* utilizado, portanto os parâmetros, frequentemente, utilizados serão apresentados, sucintamente, no decorrer desta seção. Conforme já discutido,

o SVM cria um hiperplano de separação entre duas ou mais classes, esse hiperplano pode ser "moldado" com os parâmetros *gamma* e custo.

Figura 2.12 – Resultado de diferentes valores de custo na classificação



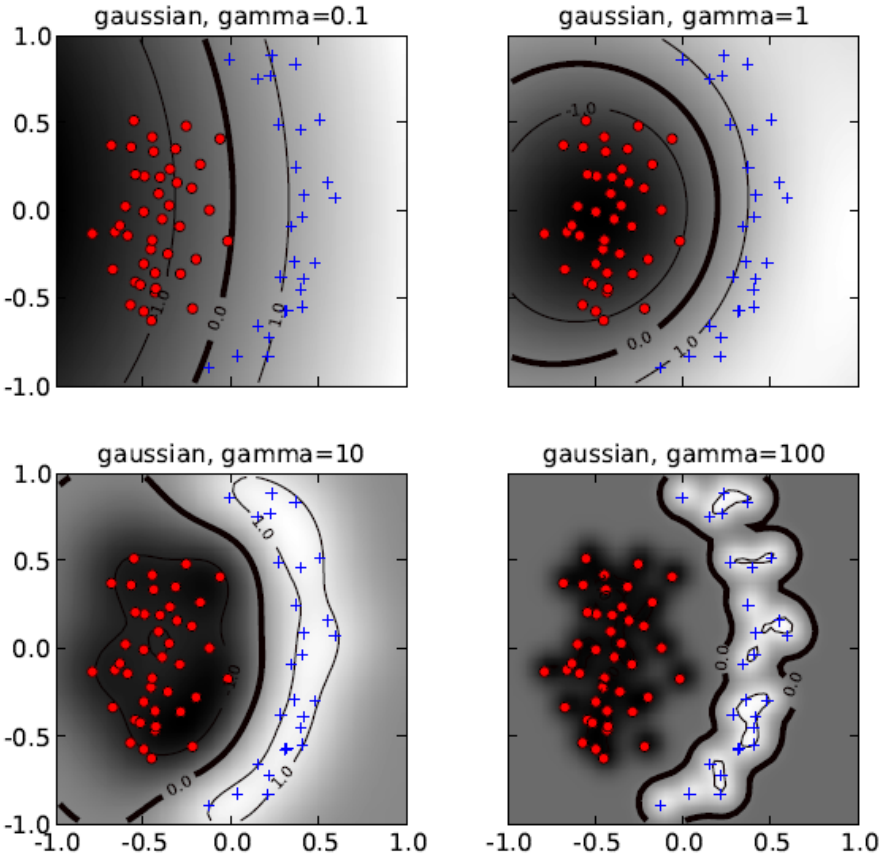
Fonte: Ben-Hur e Weston (2010)

O parâmetro custo (Figura 2.12) é utilizado, para atribuir valores de penalidade para erros no treinamento. Um valor de penalidade alto (Conforme Figura 2.12 - Esquerda) faz com que os dois pontos próximos ao hiperplano afetem sua orientação, resultando em um hiperplano de separação que é muito próximo ao de outras classes de dados. Por outro lado, um valor de custo baixo (Conforme Figura 2.12 - Direita) faz com que a orientação do hiperplano não seja tão afetada por erros de treinamento, sendo assim, é possível conseguir uma margem maior de separação.

Já o parâmetro *gamma* define a "flexibilidade" da fronteira de decisão (BEN-HUR; WESTON, 2010). À medida que o *gamma* aumenta, a expansão da localidade do vetor de suporte, também, aumenta, com tendência para uma maior curvatura da fronteira de decisão (dependendo do conjunto de dados). A Figura 2.13 representa a utilização de variados valores de *gamma*. Percebe-se que, de acordo com a Figura 2.13, a utilização de um valor de *gamma* muito alto pode causar *overfitting* no treinamento. O parâmetro *gamma* só é empregado com a utilização de *kernels* não lineares (SHALEV-SHWARTZ et al., 2011).

Desta forma, os parâmetros custo e *gamma* precisam formar uma combinação que permita ao SVM realizar a classificação de forma eficaz. A Figura 2.14 apresenta a relação do custo e *gamma* com diversas combinações.

Figura 2.13 – Diferentes Valores de *gamma* para função Gaussiana



Fonte: Ben-Hur e Weston (2010)

2.4.3 Plataforma *Microsoft Azure Machine Learning Studio*

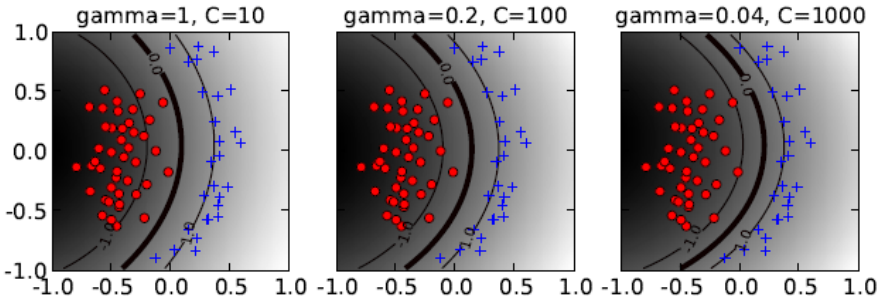
Em virtude do alto custo computacional da técnica de Máquinas de Vetores de Suporte, faz-se necessária a utilização de uma solução que proporcione resultados mais rapidamente. Sendo assim, a plataforma *Microsoft Azure Machine Learning Studio* (MICROSOFT, 2016) atende aos requisitos, a qual conta com uma infraestrutura própria e a implementação de SVM com kernel Linear, entre outros.

A plataforma demonstrou-se estável, ágil e dinâmica, sendo possível executar *scripts*, em *Python*, *R* e realizar todas as tarefas necessárias para que o treinamento da técnica funcionasse devidamente, além de ser gratuita — até certo ponto de utilização.

2.5 Florestas aleatórias

Para que seja possível apresentar a técnica de Florestas Aleatórias, faz-se necessária, primeiramente, a compreensão do conceito de Árvores de Decisão. De acordo com Oshiro

Figura 2.14 – Fronteiras de decisão similares com combinações de custo e *gamma*

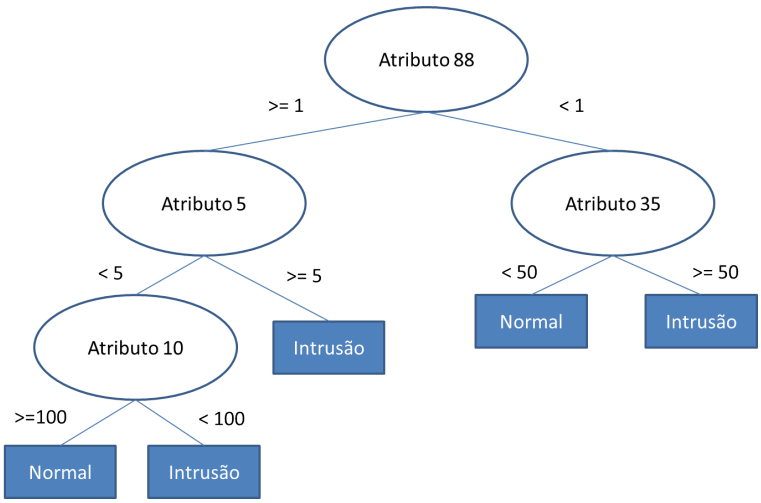


Fonte: Ben-Hur e Weston (2010)

(2013), as Árvores de Decisão utilizam a estratégia de dividir para conquistar, fazendo com que um problema complexo seja decomposto em vários subproblemas simples, de forma recursiva. O processo de construção de uma Árvore de Decisão é realizado com a seleção de um atributo que será um divisor do conjunto de dados.

Na Figura 2.15 é representada a estrutura de uma Árvore de Decisão, para realizar a classificação de tráfego de redes de computadores, em que o resultado pode ser Normal ou Intrusão. Na raiz da árvore, encontra-se o "Atributo 88". Se o valor de "Atributo 88" $>= 1$, então o algoritmo segue para o lado esquerdo da árvore, chegando ao "Atributo 5" o qual realiza a mesma comparação até chegar em um nó folha com uma classe associada (como é o caso dos quadrados em azul).

Figura 2.15 – Árvore de Decisão



Fonte: Do Autor (2016)

Uma Árvore de Decisão pode ser utilizada, para classificar outros exemplos, além daqueles utilizados no treinamento. Desta forma, ao inserir um novo exemplo, percorre-se a Árvore de Decisão, partindo-se da raiz e desvia-se, em cada nó de decisão, até chegar em um nó folha, com a classe correspondente (OSHIRO, 2013).

A técnica de Florestas Aleatórias utiliza Árvores de Decisão para constituir a sua "floresta" de classificação. Sa (2011) afirma que as Árvores de Decisão são excelentes preditores, porém nem sempre conseguem uma boa capacidade de generalização. Por outro lado, a técnica de Florestas Aleatórias apresenta excelentes características de precisão e alta capacidade de generalização.

De acordo com Breiman e Cutler (2016) e Boldt et al. (2014), a técnica de Florestas Aleatórias é um algoritmo de particionamento recursivo que combina predições feitas por um conjunto de Árvores de Decisão. Este algoritmo utiliza o mesmo método das Árvores de Decisão, no qual os dados são divididos, recursivamente, em nós de classificação.

O treinamento desta técnica funciona com a criação de centenas (ou até milhares) de Árvores de Decisão. Cada árvore é criada baseando-se em um pequeno conjunto (ou apenas uma) amostra dos dados. As árvores são criadas sem um limite máximo de profundidade dos nós (exceto na utilização de "*prunning*" [5]), sendo que as florestas são formadas pela agregação destas árvores (BOLDT et al., 2014).

Outra característica muito interessante é a capacidade do método em indicar a importância de cada variável de entrada na saída estimada Sa (2011).

2.5.1 Parâmetros de treinamento

O treinamento de Florestas Aleatórias, assim como em Redes Neurais Artificiais e Máquinas de Vetores de Suporte, requer a definição de alguns parâmetros. A utilização do algoritmo de treinamento de Florestas Aleatórias, utilizando a biblioteca Scikit-Learn (próxima Seção 2.5.2), permite definir os seguintes parâmetros (SCIKIT, 2016):

1. *max_depth*: profundidade Máxima da Árvore.
2. *max_features*: número Máximo de Características utilizadas.
3. *n_estimators*: número Máximo de Árvores na Floresta.

[5] Em português, "Poda", é o processo realizado para limitar a profundidade das árvores.

4. *min_samples_split*: número Mínimo de Amostras para criar um nó.

5. *min_samples_leaf*: número Mínimo de Amostras para criar uma folha.

Com a realização de testes, por tentativa e erro, os parâmetros podem ser melhores ajustados, tendo em vista limitações da profundidade das árvores e outros aspectos, evitando, assim, a ocorrência de *overfitting* (SCIKIT, 2016). A próxima seção apresenta a biblioteca Scikit-Learn.

2.5.2 Biblioteca Scikit-Learn

A biblioteca Scikit-learn é um módulo, em *Python*, que reúne grande parte dos algoritmos de aprendizado de máquina para treinamento supervisionado e não supervisionado. O desenvolvimento da biblioteca teve como principal foco disponibilizar uma plataforma de fácil utilização, para não especialistas em determinados algoritmos. O principal atrativo da biblioteca é a vasta documentação, além de ter o código-fonte disponível (*Open-source*) (PEDREGOSA et al., 2011).

2.6 Trabalhos relacionados

Existem algumas propostas de Sistemas de Detecção de Intrusão do tipo NIDS e HIDS onde são utilizados módulos de redes neurais artificiais para classificação de tráfego malicioso. O sistema proposto por Jing-xin, Zhi-ying e Kui (2004) foi desenvolvido de forma modularizada, contendo os seguintes módulos: monitor de pacotes, extrator de características do tráfego, algoritmo de classificação, mensageiro e uma base de exemplos.

A proposta de Al-Janabi e Saeed (2011) utiliza a base de dados da KDDCup'99, para realizar o treinamento da rede neural, porém são utilizados apenas 22 entradas da base de dados (que contém 41), em razão da dificuldade de se obter alguns detalhes característicos da KDDCup'99. Deste modo, obteve-se uma taxa de detecção de, aproximadamente, 90% com índices aceitáveis de falsos alarmes. A proposta de Han et al. (2011), também, faz uso da base de dados da KDDCup'99 e obteve-se uma taxa de detecção de 80.5% e cerca de 7.4% de falsos positivos. Kukielka e Kotulski (2008) apresentam um estudo comparativo de arquiteturas de redes neurais para problemas de reconhecimento de intrusão. É possível perceber que a aplicação do algoritmo *backpropagation* é plausível para resolução de problemas, que envolvem classificação de tráfego de redes.

A proposta de Yassin et al. (2013) utiliza a base de dados ISCX 2012 com a técnica K-Means. Nesta proposta obteve-se uma taxa de acertos de 98%, com apenas 2.2% de taxa de falsos positivos. Outras produções científicas de NIDS ou HIDS, com diferentes técnicas de Inteligência Computacional, também, foram propostas. Uma técnica geralmente bem-sucedida (para problemas de detecção de intrusão) são as Máquinas de Vetores de Suporte (SVM). Li et al. (2012) propuseram um IDS utilizando a base de dados da KDDCup'99 e um classificador SVM. Também apresentam um algoritmo para redução de características da base. Como resultado, há um modelo de IDS utilizando 19 características da base de dados com uma taxa de acerto de até 98% para tráfego normal.

Zhang e Zulkernine (2005) apresentam o estudo de uma proposta de NIDS, utilizando Florestas Aleatórias, com a base de dados KDDCup'99 em conjunto com um *framework* para realizar a leitura dos pacotes. Ao final, são apresentadas algumas comparações e afirma-se a viabilidade de utilizar Florestas Aleatórias para resolução de problemas de detecção de intrusão em redes de computadores.

2.7 Considerações finais

Neste capítulo foram apresentados, sucintamente, os conteúdos necessários para a compreensão da metodologia utilizada nesta dissertação de mestrado.

Pode-se perceber que segurança da informação é uma área que precisa de atenção, por isso, diversos pesquisadores propuseram ferramentas que auxiliam o dia a dia de um administrador de redes e/ou profissional de segurança da informação.

Na última década, Sistemas de Detecção de Intrusão em Redes de Computadores por anomalias passaram a ser propostos, geralmente, em conjunto com técnicas de Inteligência Computacional, tais como: Redes Neurais Artificiais, Florestas Aleatórias, Máquinas de Vetores de Suporte, entre outras. Os resultados obtidos com o uso destas técnicas motivaram pesquisadores para o contínuo desenvolvimento desta área.

Uma forma de avaliar a eficácia de um NIDS, pode ser pelas seguintes métricas de desempenho: Verdadeiro Positivo (VP), Verdadeiro Negativo (VN), Falso Positivo (FP), Falso Negativo (FN). Também é importante que o coeficiente Kappa e o gráfico ROC sejam apresentados.

Mesmo com o grande número de propostas de NIDS, utilizando técnicas de Inteligência Computacional, poucos autores implementaram o sistema em uma rede com tráfego real e rea-

lizaram testes com outros dados (diferentes dos encontrados na base de dados de treinamento). Visto isso, esta dissertação de mestrado propõe um NIDS com técnicas de Inteligência Computacional, que pode ser utilizado de forma *online* ou *offline*. Em conjunto com o NIDS, serão avaliadas as técnicas de Inteligência Computacional utilizadas.

3 MATERIAIS E MÉTODOS

Esta seção descreve a metodologia utilizada para a realização desta dissertação de mestrado, abordando as técnicas e equipamentos necessários para o desenvolvimento de um Sistema de Detecção de Intrusão em Redes de Computadores (NIDS), utilizando algumas técnicas de Inteligência Computacional que foram apresentadas na revisão bibliográfica. A seção foi segmentada para descrever, detalhadamente, cada processo do desenvolvimento.

3.1 Base de dados ISCX 2012

Durante as últimas décadas, detecção de intrusão por anomalias atraiu a atenção de muitos pesquisadores. A base de dados KDDCup'99 (STOLFO et al., 2000) é, atualmente, uma das bases de dados mais utilizadas para avaliação e treinamento de sistemas detectores de intrusão (SALEM; REISSMANN; BUEHLER, 2014), porém, de acordo com Uchoa (2009) e Tavallaee et al. (2009) ela não reflete mais a infraestrutura, serviços e tráfego de uma rede de computadores atual. Desta forma, faz-se necessária a utilização de alguma base de dados que tenha a capacidade de representar (o mais próximo o possível) o tráfego real de uma rede de computadores contemporânea. Para isso, a base de dados ISCX [1] 2012 (SHIRAVI et al., 2012) foi utilizada para a metodologia deste trabalho.

A proposta da base de dados ISCX 2012 foi suprir algumas deficiências encontradas nas bases, anteriormente, propostas (CAIDA, DARPA e KDD) (SHIRAVI et al., 2012). No caso da CAIDA, foram retirados todos os *payloads* dos pacotes e não há registros sobre quais conexões são normais ou intrusivas. Nas bases da DARPA e KDD, conforme já citado por Uchoa (2009), não é possível reproduzir um ambiente de rede atual. Contudo a base de dados ISCX 2012 oferece os registros das conexões (normais ou intrusivas), todo o tráfego capturado (sem remoção dos *payloads*), além da captura ser realizada em ambiente real. Um resumo das características de cada base de dados é apresentada na Tabela 3.1.

A capacidade de reprodução dos pacotes coletados pelos autores da ISCX 2012 motivou a escolha desta base de dados para a proposta deste trabalho. Este fator permite a extração de características, oriundas do tráfego de rede, a critério dos pesquisadores (assunto abordado nas próximas seções).

[1] Information Security Centre of Excellence - University of New Brunswick

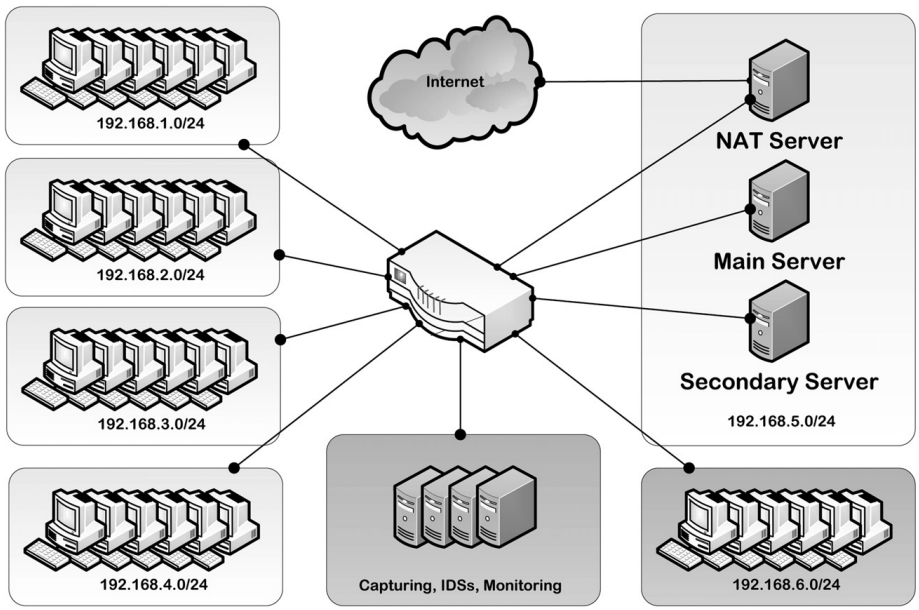
Tabela 3.1 – Comparação das Características das Bases de Dados

	Configuração realista da Rede	Tráfego Realístico	Classificação das Conexões	Arq. de Captura Completa	Diversos cenários de Ataque
CAIDA	Sim	Sim	Não	Não	Não
I.T.A	Sim	Sim	Não	Não	Não
LBNL	Sim	Sim	Não	Não	Não
DARPA-99	Sim	Não	Sim	Sim	Sim
KDD-99	Sim	Não	Sim	Sim	Sim
DEFCON	Não	Não	Não	Sim	Sim
ISCX 2012	**Sim**	**Sim**	**Sim**	**Sim**	**Sim**

Fonte: Modificado de Shiravi et al. (2012)

A base de dados conta com o tráfego capturado, durante 1 semana completa, totalizando 2.450.324 conexões. Durante a captura do tráfego, diversos serviços foram inicializados, tais como: FTP, HTTP, HTTPS, DNS, Netbios, POP3, SMTP, SNMP, SSH, Messenger e outros. A disposição da infraestrutura de rede pode ser analisada na Figura 3.1.

Figura 3.1 – Ambiente de Rede ISCX 2012



Fonte: Shiravi et al. (2012)

Para representar os ataques foram criados 4 cenários, os quais foram reproduzidos separadamente, são eles (SHIRAVI et al., 2012):

1. **Exploração de dentro da rede:** consiste em explorar vulnerabilidades em alvos que estão na mesma rede e que estão executando alguma aplicação em específico.

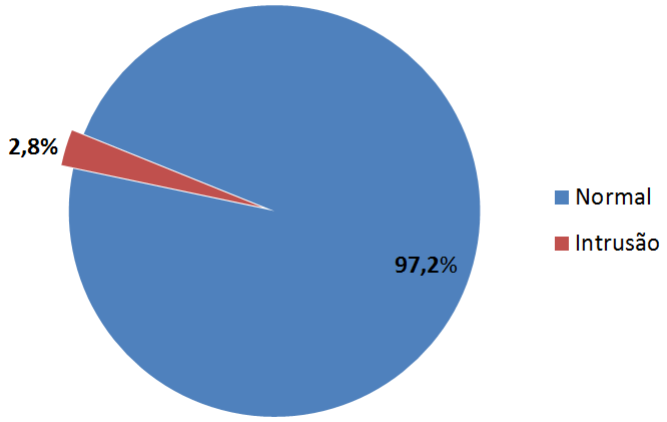
2. **Negação de Serviço HTTP (DoS):** consiste na inundação de pacotes dos mais diversos tipos, com objetivo de prejudicar o correto funcionamento de serviços como o HTTP (ou qualquer outro).

3. **Negação de Serviço usando Botnet (DDoS):** também consiste em inundação de pacotes, porém de forma distribuída (utilizando uma *Botnet* [2])).

4. **Força Bruta SSH:** ataque que visa à descoberta da senha do serviço de SSH por meio de força bruta.

Com intuito de aplicar a base de dados, em técnicas de Inteligência Computacional, é importante avaliar a disposição dos dados das classes (normal ou intrusão). Com base na Figura 3.2 é possível perceber um alto grau de desbalanceamento das amostras de duas classes. Desta forma, faz-se necessário o balanceamento dos dados, o qual será discutido na próxima seção.

Figura 3.2 – Proporções da base de dados ISCX 2012



Fonte: Do Autor (2016)

3.1.1 Balanceamento da base de dados

De acordo com Batista (2003), vários pesquisadores trabalharam, para a analisar o problema do aprendizado em técnicas de Inteligência Computacional, com conjuntos de dados com

[2] Vários computadores *zumbis* realizando uma tarefa específica

classes desbalanceadas. Alguns métodos de pré-processamento de dados tiveram destaque, são eles (BATISTA, 2003):

1. **Under-sampling:** visa balancear o conjunto de dados por meio da eliminação de amostras da classe majoritária.

2. **Over-sampling:** realiza o balanceamento do conjunto de dados por meio da replicação de amostras de exemplos da classe minoritária.

Para realizar o balanceamento da base de dados, foi utilizado o método *Over-sampling*, pois, de acordo com Prati et al. (2003), ao utilizar métodos de remoção (como o *Under-sampling*) é possível eliminar exemplos, potencialmente, úteis do conjunto de dados. Na próxima seção, é apresentada a proposta de uma API, para extrair as características do tráfego de rede da base de dados e, também, de qualquer rede de computadores.

3.2 API de captura e tratamento de pacotes e conexões

Para que seja possível a extração de características de pacotes de rede brutos, assim como são apresentados na base de dados ISCX 2012 (SHIRAVI et al., 2012), faz-se necessário o desenvolvimento de uma aplicação que capture os pacotes e realize o pré-processamento. Desta forma, uma API, *Open-source*, foi proposta.

De acordo com Salem, Reissmann e Buehler (2014), uma metodologia eficaz para o pré-processamento de uma base de dados de pacotes de rede, com objetivo de detectar ataques, principalmente, de negação de serviço, é utilizando vetores de conexão em vez de analisar apenas pacotes individualmente. Sendo assim, é possível criar uma representação do tráfego de uma janela de tempo (conexão) e analisá-la como um conjunto. Contudo, a API foi desenvolvida com intuito de transformar os pacotes que entram em uma determinada interface de rede, em vetores de conexão. Cabe ressaltar que a união da API proposta, o motor de classificação (Técnicas de Inteligência Computacional) e o módulo de notificação, tem como objetivo formar um NIDS.

O desenvolvimento de uma API torna possível o pré-processamento de bases de dados existentes, a criação de novas bases de dados e a detecção *online* de intrusões. A API será disponibilizada [3] para a comunidade (*Open-source*), com o objetivo de permitir que outros autores consigam gerar as suas próprias bases de dados, de forma simplificada e, principalmente,

[3] Disponível em: <https://github.com/heitorscalco/NIDSProject>

com a capacidade de alterar os parâmetros e a extração de características dos pacotes a qualquer momento. Algumas propostas semelhantes foram analisadas (SALEM; BUEHLER, 2013; SALEM; REISSMANN; BUEHLER, 2014; ZHANG; ZULKERNINE, 2005), porém a necessidade de alterações constantes e a diferenciação do método motivaram o desenvolvimento de uma API deste tipo.

Conforme já discutido, a API tem como principal objetivo capturar os pacotes da rede (com a definição de um filtro) e processá-los, de forma que sejam formadas diversas "conexões", oriundas de diversos pacotes. O conceito de "conexão" utilizado nesse *software* é definido por um *Unique_id* — Esta definição não tem relação com o conceito de conexão do protocolo TCP — sendo assim, o *Unique_id* é composto por protocolo, endereço IP de origem, porta de origem, endereço IP de destino e porta de destino. Alguns exemplos da formatação do *Unique_id* são mostrados a seguir:

1. *TCP-177.105.60.1:5800-177.60.20.30:80*.
2. *UDP-177.105.60.1:44000-177.60.23.31:6505*.

Desta forma, todos os pacotes que tiverem as combinações apresentadas nos tópicos a seguir, serão adicionados em suas respectivas conexões. Caso, ainda, não exista um *Unique_id*, para a conexão, uma chave é criada.

1. *{PROTOCOLO - IP_DE_ORIGEM:PORTA_DE_ORIGEM - IP_DE_DESTINO:PORTA_DE_DESTINO}*.
2. *{PROTOCOLO - IP_DE_DESTINO:PORTA_DE_DESTINO - IP_DE_ORIGEM:PORTA_DE_ORIGEM}*.
3. *{ICMP (PROTOCOLO) - IP_DE_DESTINO - IP_DE_ORIGEM - ICMP_ID}*.

Cabe ressaltar que apenas alguns campos do cabeçalho do pacote são armazenados na conexão, isso evita a sobrecarga do sistema. No caso do protocolo ICMP, onde o cabeçalho não faz a utilização de portas, os campos *PORTA_DE_ORIGEM* e *PORTA_DE_DESTINO* foram substituídos pelo campo ICMP_ID, presente no cabeçalho ICMP e que possibilita a identificação de um determinado tráfego ICMP. Caso algum pacote ICMP não contenha o campo ICMP_ID, o mesmo é substituído por -1.

Cada conexão é tratada de forma independente e é importante resolver alguns impasses, para que seja possível estabelecer, corretamente, características como, por exemplo, a quantidade de *bytes* trafegados da origem para o destino e vice-versa. Caso a direção da conexão não for definida de forma correta, os valores podem ficar invertidos, prejudicando a qualidade dos dados. A Tabela 3.2 apresenta um método, para a definição do sentido da conexão, baseado no primeiro pacote recebido em relação a cada protocolo.

Tabela 3.2 – Definição do sentido da Conexão

Primeiro Pacote Recebido	**Ação**
Flag SYN Ativada	O host que enviou o pacote é a Origem
Flag SYN + ACK Ativada	O host que enviou o pacote é o Destino
Flag ACK Ativada e Payload vazio	Último pacote do 3-way-handshake, a origem do pacote é a Origem
UDP ou ICMP	O host que enviou o pacote é a Origem
Nenhuma das situações, com Payload	Maior número de porta é a Origem

Fonte: Modificado de Salem e Buehler (2013)

Outro fator é a definição das *flags* do estado da conexão TCP. Este é um parâmetro importante, para identificar, por exemplo, um ataque de inundação de pacotes com *flag* SYN (*SYN Flood*), em que o estado da conexão será definido como *handshake*. Os estados de conexão foram definidos, de forma simplificada, em relação à apresentada em Bing, Xiaosu e Ning (2009), na qual são definidos, de forma ordenada, 5 estados de conexão: *Handshake*, *Established*, *Termination*, *Closed* e *Unknown*. Cabe ressaltar que o estado de conexão segue um fluxo contínuo, ou seja, uma conexão nunca passará de *Established* para *Handshake*, por exemplo. No caso dos protocolos UDP e ICMP, os estados de conexão iniciam e terminam como *Closed* (já que não são protocolos orientados à conexão). As *Flags* dos estados de conexão são apresentadas na Tabela 3.3.

Para que uma conexão, que não foi terminada, possa ser analisada pelo NIDS, faz-se necessário o estabelecimento de *timeouts*. Dessa forma, conexões que foram perdidas, ao longo do tráfego, ou até mesmo conexões não finalizadas propositalmente [4], poderão ser analisadas em um tempo pré-definido. Os *timeouts* foram definidos, quanto ao último pacote de cada conexão e, também, ao estado em que a conexão se encontra. Os valores são apresentados na

[4] Geralmente ataques de negação de serviço, tais como *SYN Flood*, esgotam os recursos de um alvo por meio da abertura de diversas conexões, sem finalizá-las

Tabela 3.3 – *Flags* do estado da conexão

Flag	**Situação**
S0	Tentativa de Conexão, porém ainda sem resposta (SYN)
S1	Conexão estabelecida, ainda ativa
S2	Conexão estabelecida e requisição para fechamento pela Origem, porém ainda sem resposta do Destino
S3	Conexão estabelecida e requisição para fechamento pelo Destino, porém ainda sem resposta da Origem
SF	Conexão normalmente estabelecida e fechada
RSTO	Conexão estabelecida, porém a Origem fechou a conexão enviando um RST
RSTR	Conexão estabelecida, porém o Destino fechou a conexão enviando um RST
RSTOS0	Origem enviou um SYN seguido de um RST, nunca recebeu um SYN+ACK do Destino
RSTRH	Destino enviou um SYN+ACK seguido de um RST
SH	Origem enviou um SYN seguido de um FIN, nunca recebeu um SYN+ACK do Destino
SHR	Destino enviou um SYN+ACK seguido de um FIN
OTH	Outro tipo de tráfego

Fonte: Modificado de Bing, Xiaosu e Ning (2009)

Tabela 3.4. A implementação de valores de *timeouts* altos (tais como os apresentados na Tabela 3.4) faz com que o NIDS torne-se menos eficiente, tendo como resultado um atraso de alguns segundos na classificação. É importante afirmar que os valores foram escolhidos, em virtude do embasamento teórico apresentado por NIDS já conhecidos e, também, utilizados nas propostas de Salem e Buehler (2013) e Salem, Reissmann e Buehler (2014), ficando a critério do usuário da API a definição de outros valores.

Grande parte das características utilizadas, para o treinamento das Técnicas de Inteligência Computacional, foram definidas, baseando-se nas propostas de Moustafa e Slay (2016), Salem, Reissmann e Buehler (2014), Zhang e Zulkernine (2005) e Tavallaee et al. (2009). O conjunto de características de conexão, utilizadas nos vetores de conexão, pode ser dividido em 3 tipos, são eles: Características da Conexão (Tabela 3.5), Características do *Buffer* de Tempo (Tabela 3.6) e Características do *Buffer* de Conexões (Tabela 3.7).

Tabela 3.4 – *Timeouts*

Protocolo	**Estado**	**Timeout (em segundos)**
UDP	-	180
ICMP	-	180
TCP	*Handshake*	20
TCP	*Established*	720
TCP	*Termination*	675
TCP	*Closed*	240

Fonte: Modificado de Salem e Buehler (2013)

Tabela 3.5 – Características da Conexão

#	**Característica**	**Descrição**	**Tipo**
1	*Duração*	Tempo (em segundos) da conexão	contínuo
2	*Protocolo*	tipo do protocolo (Ex.: TCP, UDP, ICMP)	discreto
3	*Serviço*	Serviço utilizado (determinado pela porta)	discreto
4	*Flag da Conexão*	Estado da Conexão (Ex.: *Handshake, Established, Termination, Closed*)	discreto
5	*SourceToDest*	Quantidade de *bytes* enviados da Origem para o Destino	contínuo
6	*DestToSource*	Quantidade de *bytes* enviados do Destino para a Origem	contínuo
7	*Land*	1 se a conexão é de/para o mesmo destino/porta, 0 o inverso	discreto
8	*Wrong*	Número de pacotes com erro de *checksum*	contínuo
9	*Urgent*	Número de pacotes TCP com a *Flag* Urgent	contínuo
10	*STTL*	TTL do primeiro pacote da Origem	contínuo
11	*DTTL*	TTL do primeiro pacote do Destino	contínuo
12	*SourceToDestPkts*	Contador dos pacotes enviados da Origem para o Destino	contínuo
13	*DestToSourcePkts*	Contador dos pacotes enviados do Destino para a Origem	contínuo

Fonte: Do Autor (2016)

Tabela 3.6 – Características do *Buffer* de Tempo (Padrão: 2 segundos)

#	**Característica**	**Descrição**	Tipo
14	*CountSameHost*	Contador de conexões para o mesmo *host*	contínuo
15	*CountSameService*	Contador de conexões com o mesmo serviço	contínuo
16	*Serror_rate*	% de conexões para o mesmo *host*, com erros de *SYN* — Aplicável apenas ao TCP	contínuo
17	*Srv_serror_rate*	% de conexões para o mesmo serviço, com erros de *SYN* — Aplicável apenas ao TCP	contínuo
18	*Same_srv_rate*	% de conexões para o mesmo serviço	contínuo
19	*Diff_srv_rate*	% de conexões para serviços diferentes	contínuo
20	*Srv_diff_host_rate*	% de conexões para o mesmo serviço com *host* diferente	contínuo

Fonte: Do Autor (2016)

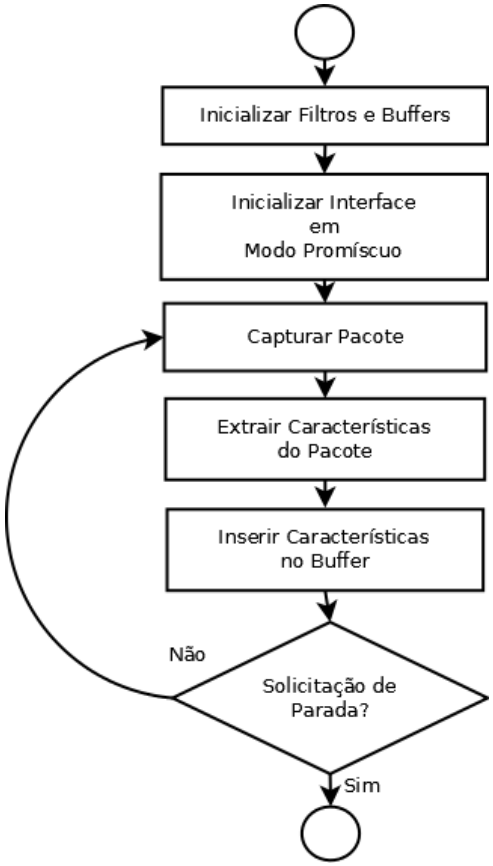
Tabela 3.7 – Características do *Buffer* de Conexões (Padrão: 100 conexões)

#	**Característica**	**Descrição**	**Tipo**
21	*Count*	Contador de conexões para o mesmo *host*	contínuo
22	*Srv_count*	Contador de conexões com o mesmo serviço	contínuo
23	*Same_srv_rate*	% de conexões para o mesmo *host*, com o mesmo serviço	contínuo
24	*Diff_srv_rate*	% de conexões para o mesmo *host*, com serviços diferentes	contínuo
25	*Same_src_port_rate*	% de conexões com a mesma porta de origem	contínuo
26	*Srv_diff_host_rate*	% de conexões para o mesmo serviço, com *host* diferente	contínuo
27	*Serror_rate*	% de conexões para o mesmo *host*, com erro de SYN — Aplicável apenas ao TCP	contínuo
28	*Srv_serror_rate*	% de conexões para o mesmo serviço, com erro de SYN — Aplicável apenas ao TCP	contínuo

Fonte: Do Autor (2016)

O funcionamento do sistema consiste em 5 módulos, um módulo de Captura de Pacotes (Figura 3.3), um módulo de Gerenciamento de Conexões (Figura 3.4), um módulo de Monitoramento de *timeouts* (Figura 3.5), um módulo de Classificação (Figura 3.6) e um módulo de Notificação (Figura 3.7), todos operando de forma concorrente. A definição de cada módulo é apresentada nos tópicos a seguir:

Figura 3.3 – Fluxograma do Módulo de Captura

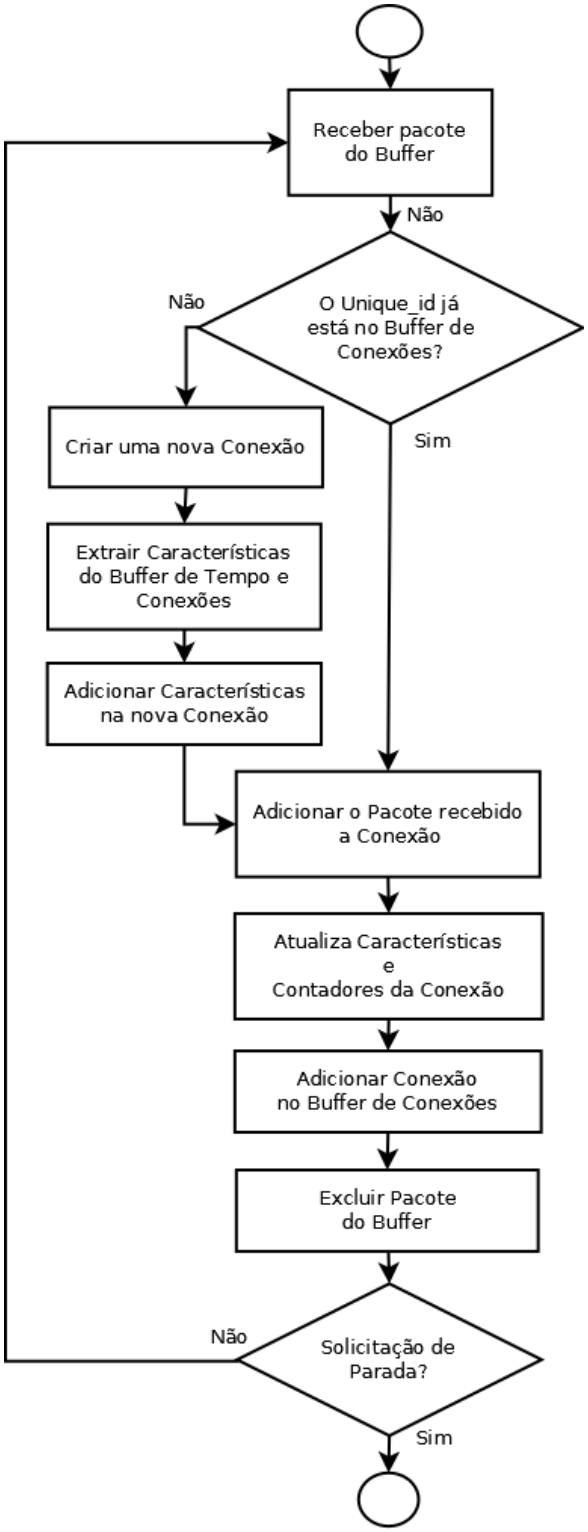


Fonte: Do Autor (2016)

1. **Módulo de Captura de Pacotes:** é responsável por deixar a interface de rede em modo promíscuo e, utilizando um filtro, capturar todos os pacotes que entram ou saem nessa interface. Esse módulo é, também, responsável por obter as características necessárias do cabeçalho dos pacotes e despachá-las para o módulo de Gerenciamento de Conexões.

2. **Módulo de Gerenciamento de Conexões:** tem como principal função determinar o destino de cada pacote, baseando-se no *Unique_id*. Também é responsável por realizar todo o tipo de atualização e obtenção das características da conexão, tais como: quantidade de dados da origem para o destino, número de *flags* do TCP, número de erros de

Figura 3.4 – Fluxograma do Módulo de Gerenciamento



Fonte: Do Autor (2016)

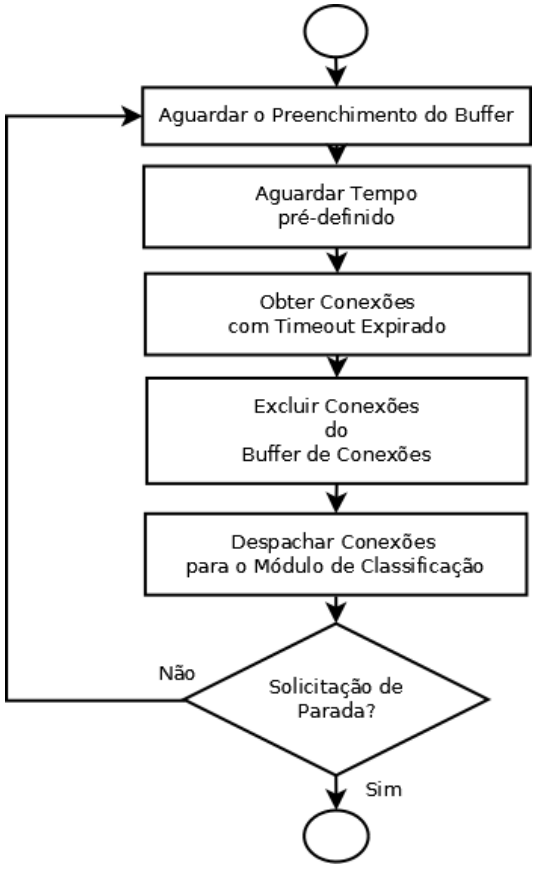
checksum, atualização da *Flag* de conexão, atualização do estado da conexão, ajuste do *timeout*, entre outros.

3. **Módulo de Monitoramento de *Timeouts*:** analisa, continuamente, as conexões que estão em *buffer*, verificando se o *timeout* de cada conexão expirou. Caso afirmativo, faz uma cópia da conexão e despacha para o Módulo de Classificação, utilizando *socket*. Logo após, a conexão é excluída do *buffer*.

4. **Módulo de Classificação:** é um módulo à parte da API, é constituído pelas técnicas de Inteligência Computacional, recebe os dados utilizando *sockets* e realiza a classificação. Caso uma conexão seja classificada como Intrusão, os dados são despachados para o módulo de notificação. Cabe ressaltar que a utilização de *sockets*, para se obter os vetores de conexão, permite que outros autores possam implementar, juntamente com essa aplicação, outras técnicas de classificação, inclusive, com outras linguagens de programação, sem que seja necessário realizar alterações na API.

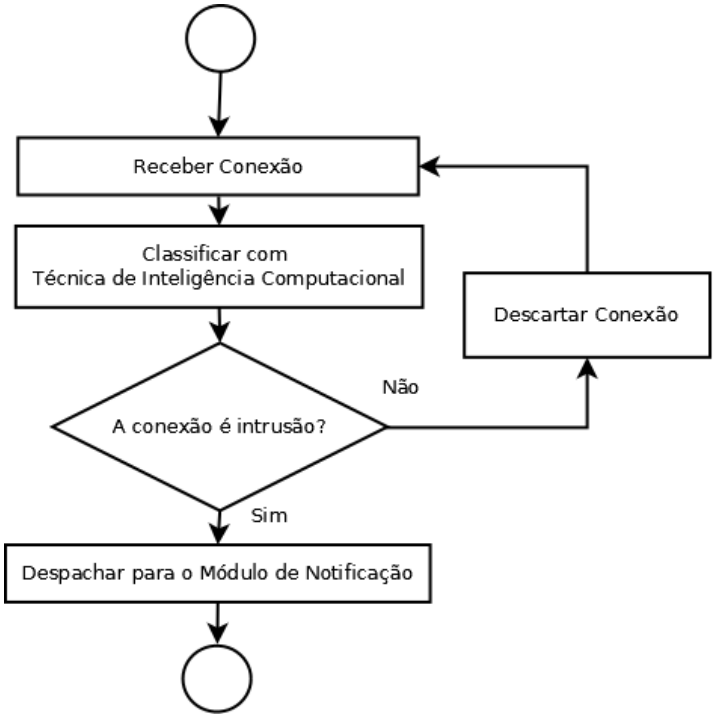
5. **Módulo de Notificação:** caso o sistema esteja funcionando *Online*, um *log* será gerado a cada intrusão detectada, contendo todas as informações da conexão capturada.

Figura 3.5 – Fluxograma do Módulo de Monitoramento



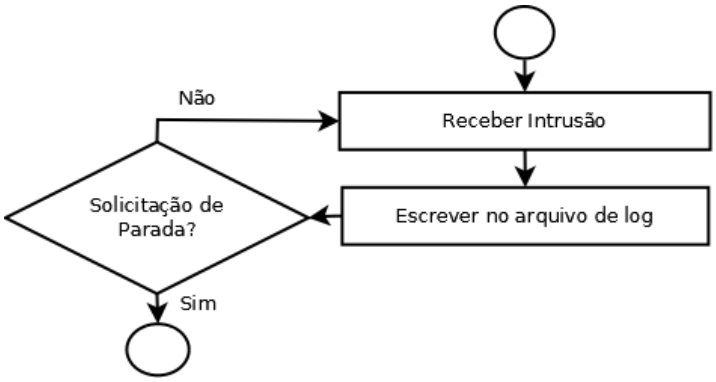
Fonte: Do Autor (2016)

Figura 3.6 – Fluxograma do Módulo de Classificação



Fonte: Do Autor (2016)

Figura 3.7 – Fluxograma do Módulo de Notificação



Fonte: Do Autor (2016)

A API foi desenvolvida em Java — portanto não foi projetada para ser eficiente, mesmo com boas práticas de desenvolvimento — utilizando o ambiente de programação *Eclipse Mars*. Para fazer o uso da aplicação e/ou modificá-la existem alguns pré-requisitos, são eles:

1. Open JDK 8: kit de desenvolvimento Java.
2. Libpcap 0.8: biblioteca de captura de pacotes em C.
3. JnetPcap 1.4: biblioteca de captura de pacotes que utiliza a *Libpcap*, porém em Java.

Por fim, o formato dos dados de saída da aplicação, que são disponibilizados para as técnicas de Inteligência Computacional, via *socket*, são apresentados seguindo o modelo dos tópicos abaixo, os quais representam os vetores de conexão:

1. 63.0,TCP,HTTP,S0,280,[...],30,100.0,0.0,0.0,93.0,100.0,6.0,normal.
2. 0.0,UDP,DNS,SF,16,[...],27,53.0,46.0,17.0,74.0,0.0,0.0,normal.
3. 0.0,ICMP,ICMP,SF,16,0,[...],1,0.0,100.0,1.0,100.0,0.0,0.0,normal.
4. 0.0,TCP,SSH,S0,[...],30100.0,34,1,0.0,100.0,5.0,100.0,0.0,100.0,ataque.
5. 0.0,UDP,DNS,SF,16,[...],100.0,0.0,4.0,66.0,0.0,0.0,ataque.

Conforme já discutido, a API pode funcionar de diversas formas, as quais serão apresentadas na próxima seção.

3.2.1 Modos de operação

É possível utilizar a API de três maneiras distintas, são elas:

1. **Online:** em conjunto com o módulo de Inteligência Computacional, esse modo de operação permite que a API opere como um NIDS, classificando o tráfego *Online* na rede de computadores. Os dados pré-processados são disponibilizados para o motor de classificação utilizando *sockets* ou arquivo texto.
2. **Base de dados:** este modo de operação consiste em realizar a leitura de uma base de dados e seus respectivos *labels*, permitido, assim, a extração de características de pacotes brutos. Ao final do processamento, o *Unique_id*, juntamente dos *timestamps* das conexões são comparados aos *labels* e definidos como Normal ou Intrusão (Caso o *label* não seja encontrado, a conexão é descartada). Cabe ressaltar que este modo pode ser utilizado para realizar a leitura de qualquer base de dados que esteja no formato *pcap*. Os dados pré-processados são gravados em um arquivo texto. Esse modo foi utilizado para pré-processar a base de dados ISCX 2012.
3. **Construção de Base de Dados:** este modo de operação consiste em realizar a construção de uma nova base de dados. Os pacotes são pré-processados (*online* ou *offline*), extraindo todas as características necessárias, sendo classificados por uma *tag default*

(Normal ou Intrusão). Portanto é inevitável que algumas poucas conexões sejam classificadas de modo incorreto – em virtude de ruídos trafegando na rede. Cabe ao usuário realizar o isolamento da rede para capturar tráfegos normais e intrusivos. Esse modo de operação foi utilizado, para criar uma base de dados de testes, para a metodologia proposta.

Os diferentes modos de operação permitem aos pesquisadores realizar diversos experimentos, sem precisar alterar a API. Cabe ressaltar que, havendo a necessidade de um novo modo de operação, é possível adicionar novos recursos, de forma simplificada, a API.

3.2.2 Processo de construção da base de dados de testes

Uma das propostas desse trabalho é implementar um NIDS em um ambiente real. Entretanto, para que seja possível realizar algumas análises estatísticas e até mesmo testes com as técnicas de Inteligência Computacional, decidiu-se capturar o tráfego de um ambiente real e gravá-lo em um arquivo com extensão *"pcap"* [5]. Desta forma, é possível reproduzir todo o tráfego na API — no modo de operação Base de dados / Construção de Base de dados, quantas vezes for necessárias.

A base de dados da ISCX 2012 já disponibiliza dados de uma rede de computadores em um ambiente real. Porém, faz-se necessária a construção de uma nova base de dados, com uma disposição de infraestrutura (Computadores, Roteadores, *Switches*, *Smartphones*) diferente da apresentada pela ISCX 2012. A aplicação de uma outra base de dados permite a comprovação da eficácia do método proposto para diversas infraestruturas de rede.

O processo de construção da base de dados proposta foi subdividido nas seções 3.2.2.1, 3.2.2.2 e 3.2.2.3, conforme apresentado a seguir.

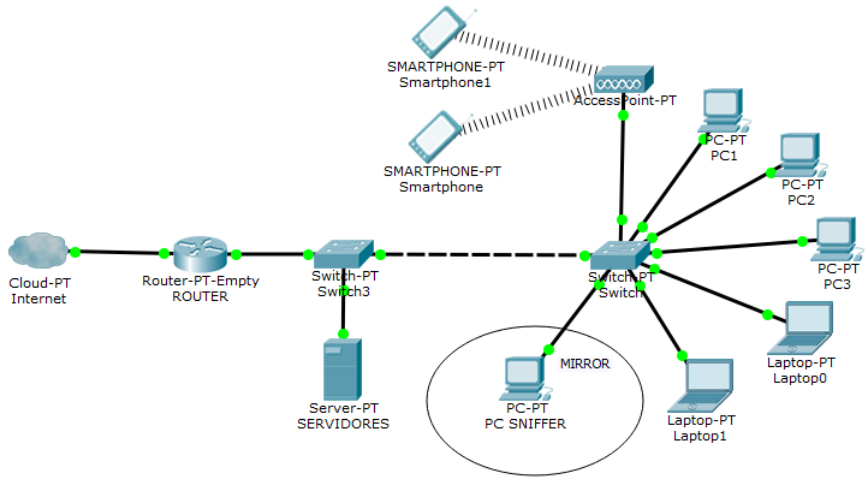
3.2.2.1 Preparação da infraestrutura de rede

A infraestrutura de rede, sem dúvida, é um ponto importante, para a criação de uma base de dados, portanto é necessário ter cautela para diversos fatores, tais como: topologia, serviços, segurança e disponibilidade dos ativos de rede. A infraestrutura utilizada contou com 8 computadores, sendo 1 utilizado para o *sniffer* [6] , 1 para virtualizar os servidores e outros 6, para gerar tráfego de rede e, também, 2 *Smartphones* para realizar a utilização de aplicativos.

[5] Extensão utilizada por *sniffers* de rede, tais como: Wireshark e TCPDUMP
[6] Programa de captura de tráfego de rede

Fez-se necessária a configuração do espelhamento das portas do *switch* (*Mirroring*) para que todo o tráfego pudesse ser capturado (representado pelo *"Mirror"* da Figura 3.8). A Figura 3.8 representa um esboço do ambiente de rede utilizado para a criação da base de dados.

Figura 3.8 – Ambiente de Rede



Fonte: Do Autor (2016)

O *Switch*, *Access Point* e o Roteador (Figura 3.8) foram utilizados, sem qualquer configuração de segurança, o que garante a livre transmissão dos pacotes de rede (maliciosos ou não). Os endereços MAC dos computadores e *Smartphones* foram cadastrados no servidor DHCP, para que os endereços IP não sofressem alterações, durante o processo de criação da base de dados.

3.2.2.2 Serviços inclusos

Para que seja possível representar o uso normal e anômalo de uma rede de computadores, é necessário que vários serviços do dia a dia sejam executados durante a construção da base de dados. Desta forma, diversas aplicações foram testadas, durante o processo de captura de pacotes, conforme é apresentado na Tabela 3.8;

Também é importante ressaltar que a rede conta com computadores com sistemas Microsoft Windows 8, Microsoft Windows 10, Linux Ubuntu 14.04 LTS, Linux Debian 8, Linux Kali e, também, com *smartphones* com sistema Android 5 e Android 6. Dessa forma é possível capturar diversos tipos de tráfego, tais como: atualizações de sistema, programas e quaisquer tipos de tráfego que possam variar de um sistema operacional para outro.

Tabela 3.8 – Serviços/Protocolos utilizados durante a construção da base de dados

Ferramenta	**Protocolo**
Navegador Web (*Google Chrome e Mozilla Firefox*)	HTTP, HTTPS
Spotify	Streaming de Áudio (UDP e TCP)
Hangouts, Skype e *Facebook Call*	VOIP (UDP e TCP)
Cliente FTP (*Filezilla*)	FTP
Cliente de Email (*Outlook, Thunderbird, Gmail (Android)*)	POP3, IMAP, SMTP
Ferramenta de Monitoramento de Redes (*Cacti*)	SNMP
Cliente SSH (*Shell Linux, Putty, WinSCP*)	SSH
Cliente e Servidor DNS (*Bind9 - Linux*)	DNS
Cliente e Servidor DHCP (*ISC DHCP- Linux*)	DHCP
Sincronização de Horário com Servidor	NTP
Descoberta de Rede Microsoft	NETBIOS
Cliente *MEGA* e *Dropbox*	TCP, HTTP, HTTPS

Fonte: Do Autor (2016)

3.2.2.3 Captura dos dados

A captura dos dados foi realizada no *sniffer*, representado na Figura 3.8. A ferramenta de captura utilizada foi o *tcpdump*, um *software* de captura de pacotes bastante conhecido e difundido no mercado (utiliza a biblioteca *libpcap*). Os parâmetros foram definidos conforme apresentado a seguir:

```
tcpdump -i INTERFACE -nN -w ARQUIVO.pcap -B 10000 'tcp or udp or icmp'
```

O parâmetro *-i* indica qual interface entrará em modo promíscuo, seguido dos parâmetros *-nN* que têm como função não converter endereços para nomes, *-w* que é responsável por salvar os pacotes em um arquivo e, por fim, o *-B* que aumenta o *buffer* do sistema operacional (TCPDUMP, 2015).

A captura dos pacotes de tráfego normal foi realizada em uma janela de tempo de 10 horas. Já a captura dos pacotes com tráfego intrusivo foi realizada em uma janela de tempo de 6 horas. Ambas as janelas de tempo demonstraram-se suficientes, para realizar os mais diversos tipos de acessos a aplicações, páginas web e sistemas, e também para realizar os testes com tráfego malicioso, principalmente, ataques de negação de serviço.

O tráfego intrusivo foi gerado com auxílio das ferramentas *t50* (T50, 2016), *Nmap* (NMAP, 2016) e *Hping3* (HPING3, 2016). Estas ferramentas são muito utilizadas por *pen-testers* ou criminosos cibernéticos para realizar ataques de negação de serviço e varredura de redes. Os ataques contabilizados e suas devidas ferramentas estão representados no Apêndice A.

Os dados coletados, no arquivo com extensão "pcap", foram, posteriormente, reproduzidos na API para realizar o tratamento e organização de pacotes em vetores de conexões. Cabe lembrar que a decisão de utilizar a ferramenta *tcpdump* (TCPDUMP, 2015) foi tomada, para gravar o tráfego em um arquivo, o tráfego poderia ser capturado com a API no modo de operação "Construção de Base de Dados".

Após a construção dos vetores de conexão, os dados são aplicados aos classificadores de Inteligência Computacional (para o treinamento e testes), conforme serão descritos, nas próximas seções.

3.2.3 Pré-processamento comum dos dados

Os processos de normalização e pré-processamento são, sem dúvida, fases muito trabalhosas e importantes para garantir a eficácia do treinamento das técnicas de Inteligência Computacional. Visto a aplicação dos mesmos conjuntos de dados, para variadas técnicas de Inteligência Computacional, algumas etapas de pré-processamento são comuns a todas essas as técnicas.

Como as bases de dados utilizadas possuem algumas entradas não numéricas, é necessário realizar o pré-processamento para transformá-las em entradas numéricas. Alguns fatores precisam ter uma atenção especial, para o tratamento dos dados, por exemplo, os protocolos, as *Flags* de conexões e os nomes dos serviços. Como a classificação de vários protocolos, *Flags* e/ou serviços não são uma grandeza, em que é possível definir valores de distância entre si, é necessário implementar um algoritmo que adicione uma entrada diferente para cada protocolo. Por exemplo, a entrada do protocolo do dado corrente é preenchido com 1, enquanto as outras entradas de protocolo são preenchidas com o valor 0. De forma que (n é o número de Protocolos/*Flags*/Serviços distintos encontrados na base):

1. http = [1 0 0 0 ... n].
2. smtp = [0 1 0 0 ... n].

3. pop3 = [0 0 1 0 ... *n*].

4. https = [0 0 0 1 ... *n*].

Dessa forma, os dados obtidos pelas bases de dados, que têm 28 variáveis de entrada, passam a ter 96 variáveis de entrada, em virtude do grande número de serviços contidos na base de dados e, ainda, outras informações que precisam ser partilhadas em entradas (Protocolos e *Flags*).

3.3 Redes neurais artificiais

Conforme apresentada na revisão bibliográfica, a aplicação de Redes Neurais Artificiais em problemas de reconhecimento de intrusão, em redes de computadores, tem proporcionado avanços à área de segurança da informação. Uma biblioteca que aborda, de forma eficaz, redes neurais MLP é a *Pybrain* (SCHAUL et al., 2010) e será utilizada para realizar a classificação do tráfego de rede.

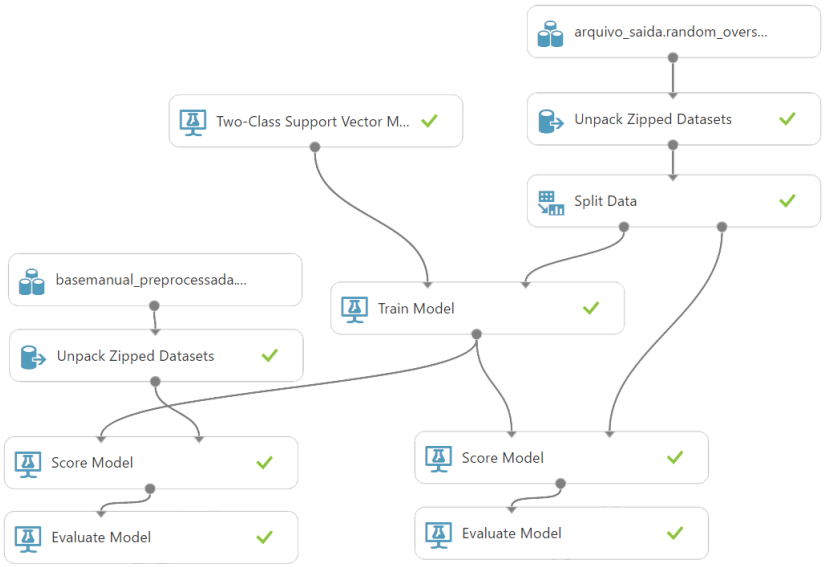
A arquitetura da rede neural será definida por tentativa e erro, iniciando com 2 camadas intermediárias (recomendado por Cybenko (1988)) e variando o número de neurônios em cada camada. Após encontrar uma arquitetura para a rede que atenda as necessidades de classificação, o treinamento da rede neural, visto a disponibilidade de tempo e a duração do treinamento, será repetido por 10 vezes, com intuito de calcular a média de acertos obtida por esta técnica. Desta forma é possível descartar questionamentos em relação a aleatoriedade dos pesos da rede, onde, em alguns casos, um valor de peso aleatório pode favorecer mais o treinamento da rede do que outro.

O pré-processamento comum a todas as técnicas já supre as necessidades de organização dos dados para realizar o treinamento da Rede Neural Artificial. Todavia, é importante que a normalização dos dados seja realizada para evitar situações como, no caso da implementação de Redes Neurais Artificiais, a saturação da função de ativação *Sigmoidal* (Figura 2.8). Como os resultados são caracterizados como normal ou intrusão, o valor resultante para normal será 0,1 e para intrusão será 0,9.

3.4 Máquinas de vetores de suporte

Em razão do alto custo computacional desta técnica, fez-se necessária a utilização de uma plataforma de alto desempenho, como é o caso da *Microsoft Azure Machine Learning*

Figura 3.9 – Modelo Lógico da Plataforma Microsoft Azure



Fonte: Do Autor (2016)

Studio, conforme Figura 3.9. Com a aplicação dessa técnica, espera-se a obtenção de índices de acerto acima de 90%, assim como em Li et al. (2012) — proposta que utiliza outra base de dados. Os parâmetros de treinamento serão definidos por uma função de busca de parâmetros nativa da plataforma *Microsoft Azure* (MICROSOFT, 2016), utilizando cerca de 10% dos dados de treinamento.

Assim como na aplicação de Redes Neurais Artificiais, o pré-processamento comum dos dados já supre as necessidades para aplicação em SVM. As classes normal e intrusão foram definidas como 0 e 1, respectivamente. Ao contrário de outras bibliotecas (como é o caso da LibSVM e LibLinear), o *Microsoft Azure Machine Learning Studio* aceita diversos formatos de bases de dados.

3.5 Florestas aleatórias

Para a técnica de Florestas Aleatórias, a biblioteca Scikit-Learn será utilizada. A base de dados não precisa de alterações, em relação ao pré-processamento comum a todas as técnicas. As classes normal e intrusão, também, foram definidas como 0 e 1, respectivamente. Os parâmetros serão definidos por tentativa e erro, com atenção para a ocorrência de *overfitting*, comum nas Florestas Aleatórias. Como resultado, espera-se a obtenção de um percentual de

acertos acima de 95%, assim como em Zhang e Zulkernine (2005) — proposta que utiliza outra base de dados.

3.6 Métodos de comparação

Após realizar a implementação das técnicas de Inteligência Computacional propostas, um estudo comparativo será realizado, para selecionar a técnica, que mais se adapta ao problema de reconhecimento de intrusão em redes de computadores. A metodologia utilizada, para realizar a seleção da melhor técnica, será baseada em números de Falsos Positivos, Falsos Negativos, Verdadeiros Positivos e Verdadeiros Negativos. Por fim, para afirmar a consistência dos resultados, serão apresentados os índices Kappa e o gráfico ROC de cada resultado. Conforme citado na revisão bibliográfica, acredita-se que esses parâmetros sejam satisfatórios para apresentar a eficácia de um Sistema de Detecção de Intrusão em Redes (NIDS).

4 RESULTADOS E DISCUSSÕES

Com base na aplicação da metodologia proposta neste trabalho, utilizando a base de dados ISCX 2012 (SHIRAVI et al., 2012) e a base de dados criada, a partir da API, foi possível obter diversos resultados. Esta seção foi dividida para apresentar, de forma detalhada, os resultados obtidos em cada uma das técnicas de Inteligência Computacional e o comparativo final.

4.1 Redes neurais artificiais

Após realizar o pré-processamento dos dados, construção e treinamento da rede neural, pôde-se perceber a eficácia do método proposto para a solução do problema de reconhecimento de intrusão em redes de computadores. Os erros de validação foram obtidos, baseando-se nos testes da função *trainUntilConvergence* da biblioteca *Pybrain* (SCHAUL et al., 2010), com 10% das amostras de treinamento (valor padrão da biblioteca). Os resultados desta técnica são apresentados nas Tabelas 4.1, 4.2, 4.3.

Em decorrência da elevada quantidade de amostras da base de dados, o treinamento teve uma duração média de 17,5 horas (plataforma não distribuída). Após a realização de treinamentos experimentais, percebeu-se que, após 15 épocas, não se obteve mais convergência significativa, desta forma, para tornar o experimento mais eficiente, o número máximo de épocas foi limitado em 25. Os parâmetros utilizados foram:

1. Taxa de aprendizagem: 0.01.
2. Função de ativação: *Sigmoidal*.
3. Número máximo de épocas: 25.
4. *continueEpochs*: 10.
5. Taxa de Momentum: 0.9.
6. Camadas ocultas/escondidas: 2.
7. Quantidade de neurônios em cada camada: 20 e 20.
8. Porção para efetuar o treinamento: 80%.
9. Porção para validação dos dados: 10% dos dados de treinamento.

10. Porção para testar o treinamento: 20%.

Como a inicialização dos pesos das entradas da rede é definida de forma aleatória, é possível contestar a taxa de acertos apresentada. Portanto, para comprovar a real eficácia da rede, cada treinamento, assim como seus respectivos testes, foram repetidos 10 vezes (Tabela 4.1), com os mesmos parâmetros. Foram utilizados 20% dos dados da base para teste (cerca de 805.164 amostras). O conjunto de dados foi misturado e separado, em cada treinamento, de forma randômica, visando ao aprendizado uniforme da rede neural. A Tabela 4.1, também, apresenta os resultados obtidos com testes em ambiente real, a terceira coluna apresenta a taxa de acertos utilizando a base de dados criada com a API como dados de teste — perceba que a base de dados criada com a API não foi utilizada para treinamento, apenas para dados de teste.

Conforme apresentado na Figura 4.1, o Erro Médio Quadrático no final do treinamento foi muito pequeno (próximo a zero). Percebe-se também um grau de oscilação, o qual é decorrente da taxa de aprendizado e da escala do gráfico. Testes com taxas de aprendizado e momentum menores foram realizados, porém não notou-se diferenças significativas nos resultados. Desta forma, mantiveram-se os padrões selecionados. As Tabelas 4.2 e 4.3 apresentam a matriz de confusão das duas bases de dados testadas.

Tabela 4.1 – Percentual de acertos utilizando a Base de Dados ISCX 2012 com RNA

Treinamento	**Taxa de Acerto com os dados de Teste**	**Taxa de Acerto com a Base da API**	**Duração**
1	89.63%	97.42%	17h21min
2	90.14%	99.01%	17h52min
3	90.04%	99.07%	17h17min
4	89.78%	97.69%	17h36min
5	89.97%	98.58%	17h41min
6	90.17%	98.35%	17h59min
7	89.59%	98.75%	18h01min
8	90.01%	98.94%	17h53min
9	90.44%	99.29%	17h42min
10	89.90%	98.72%	18h03min
Média	**89.96%**	**98.58%**	-
Desvio Padrão	**0.2576**	**0.6053**	-

Fonte: Do Autor (2016)

Tabela 4.2 – Matriz de Confusão do NIDS com a Base de Dados ISCX 2012 (Primeiro Treinamento) com RNA

	Positivo	**Negativo**	**Total**
Positivo	342333 (42.51%)	25081 (3.11%)	367414
Negativo	58362 (7.24%)	379388 (47.11%)	437750
Total	400695	404469	805164

Fonte: Do Autor (2016)

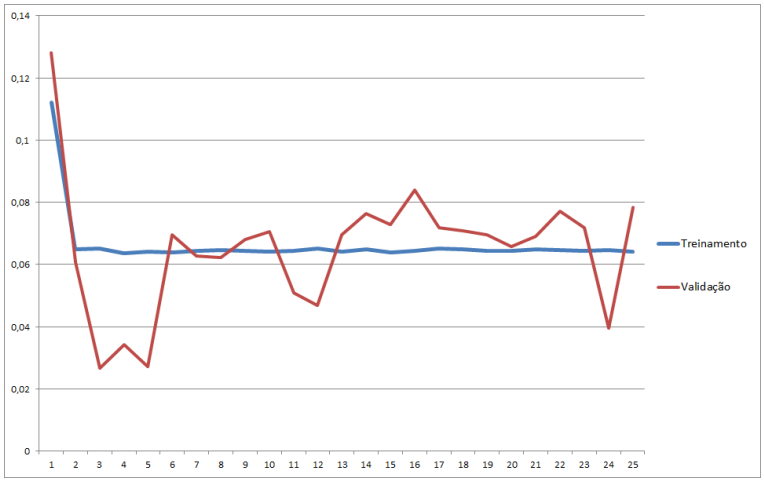
Tabela 4.3 – Matriz de Confusão do NIDS com a Base de Dados da API (Primeiro Treinamento) com RNA

	Positivo	**Negativo**	**Total**
Positivo	62799 (54.59%)	2637 (2.29%)	65436
Negativo	326 (0.28%)	49268 (42.83%)	49594
Total	63125	51905	115030

Fonte: Do Autor (2016)

Na próxima seção, são apresentados os resultados obtidos com a técnica de Máquinas de Vetores de Suporte.

Figura 4.1 – Gráfico do Erro Médio Quadrático por Época (Primeiro Treinamento) com RNA



Fonte: Do Autor (2016)

4.2 Máquinas de vetores de suporte

A utilização da plataforma *Microsoft Azure* (MICROSOFT, 2016) demonstrou-se eficaz para a classificação de tráfego de rede de computadores. Uma função para mistura (de forma randômica) e divisão dos dados de teste, também, foram utilizadas, possibilitando, assim, um treinamento mais eficaz e, por fim, a validação da técnica através das repetições dos treinamentos. A duração média dos treinamentos, dada a infraestrutura de alta performance da plataforma, foi de 12 minutos e 48 segundos.

Os itens a seguir apresentam os parâmetros utilizados para realizar o treinamento da técnica de Máquinas de Vetores de Suporte (SVM):

1. Tipo de SVM: C-SVC.
2. *Kernel:* Linear.
3. Custo: 0,09.
4. Número de Iterações: 100.
5. Porção para realizar o treinamento: 80%.
6. Porção para testar o treinamento: 20%.

A Tabela 4.4 apresenta as taxas de acerto com os dados de teste e, também, com os dados da base de dados da API. O treinamento foi repetido por 10 vezes, com variação dos dados de treinamento e teste a cada novo treinamento. Percebe-se um desvio padrão mínimo nos resultados, o que demonstra, em conjunto com as taxas de acerto apresentadas, a real eficácia do método para reconhecimento de intrusão em redes de computadores.

As Tabelas 4.5 e 4.6 apresentam as matrizes de confusão, percebe-se uma diferença significativa, em ambos os casos, no número de falsos positivos em relação aos falsos negativos. Já a Figura 4.2 apresenta a análise da importância das características da base de dados, em relação ao primeiro treinamento. Essa informação foi obtida utilizando uma função (*Permutation Feature Importance*) nativa da plataforma, a qual recebe como parâmetro o modelo treinado e a base de dados.

Na próxima seção, serão apresentados os resultados obtidos com a técnica de Florestas Aleatórias.

Tabela 4.4 – Percentual de Acertos utilizando a Base de Dados ISCX 2012 com SVM

Treinamento	Taxa de Acerto com os dados de Teste	Taxa de Acerto com a Base de dados da API	Duração
1	95.03%	95.90%	11min e 55s
2	95.06%	95.91%	13min e 05s
3	95.02%	95.90%	13min e 34s
4	95.01%	95.90%	12min e 40s
5	95.04%	95.89%	12min e 01s
6	95.04%	95.90%	11min e 57s
7	95.05%	95.90%	13min e 25s
8	95.09%	96.33%	13min e 48s
9	95.07%	95.89%	12min e 03s
10	95.02%	96.05%	13min e 40s
Média	**95.04%**	**95.95%**	-
Desvio Padrão	**0.02497**	**0.13953**	-

Fonte: Do Autor (2016)

Tabela 4.5 – Matriz de Confusão do NIDS com a Base de Dados ISCX 2012 (Primeiro Treinamento) com SVM

	Positivo	Negativo	Total
Positivo	391389 (48.60%)	10112 (1.25%)	401501
Negativo	29890 (3.71%)	373773 (46.42%)	403663
Total	421279	383885	805164

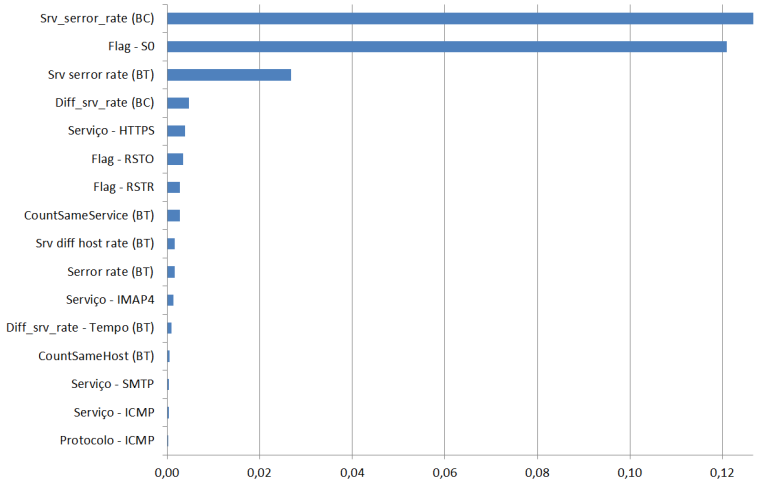
Fonte: Do Autor (2016)

Tabela 4.6 – Matriz de Confusão do NIDS com a Base de Dados da API (Primeiro Treinamento) com SVM

	Positivo	Negativo	Total
Positivo	62719 (54.52%)	406 (0.35%)	63125
Negativo	4315 (3.75%)	47590 (41.37%)	51905
Total	67034	47996	115030

Fonte: Do Autor (2016)

Figura 4.2 – Gráfico da importância das principais características da Base de Dados ISCX 2012, utilizando SVM (Primeiro Treinamento)



Fonte: Do Autor (2016)

4.3 Florestas aleatórias

A utilização da biblioteca Scikit-Learn, para o treinamento da técnica de Florestas Aleatórias demonstrou-se eficaz e eficiente. Na Tabela 4.7, são apresentados os percentuais de acerto dos testes com as duas bases de dados, é possível perceber o pequeno desvio padrão dos resultados. Cabe ressaltar que, assim como nos métodos anteriores, os dados de treinamento e teste foram misturados e separados, aleatoriamente, a cada novo treinamento. É importante, também, evidenciar que a base de dados da API foi utilizada apenas como dados de teste, sem que os dados interferissem no treinamento.

Após exaustivas tentativas de definição de parâmetros, obteve-se a seguinte combinação:

1. Profundidade Máxima: 3.
2. Máximo de Características: 25.
3. Número de Árvores na Floresta: 1000.
4. Número mínimo de amostras para criar um nó: 2.
5. Número mínimo de amostras para criar um nó folha: 5.

As Tabelas 4.8 e 4.9 apresentam as matrizes de confusão para os resultados da base de dados ISCX 2012 e a base de dados da API, respectivamente. Na base de dados ISCX 2012 (Tabela 4.8) percebe-se um equilíbrio entre os números de falsos positivos e falsos negativos, o que ocorre em menor grau na base de dados da API (Tabela 4.9).

Conforme citado no capítulo de Revisão Bibliográfica, a técnica de Florestas Aleatórias disponibiliza, de forma nativa, os valores de importância das principais características utilizadas para realizar a classificação. A Figura 4.3 apresenta um gráfico das principais características utilizadas. Os demais valores foram excluídos do gráfico, por não apresentarem importância significativa (muito próximo a zero ou igual a zero). As abreviações “BC” e “BT” significam a origem das características, as quais são *Buffer* de Conexões e *Buffer* de Tempo, respectivamente.

A próxima seção apresenta um estudo comparativo dos resultados das técnicas apresentadas. Entre os métodos de comparação estão: análise de falsos positivos, falsos negativos, acurácia média, coeficiente Kappa e gráfico ROC.

Tabela 4.7 – Percentual de acertos utilizando a Base de Dados ISCX 2012 com Florestas Aleatórias

Treinamento	Taxa de Acerto com os dados de Teste	Taxa de Acerto com a Base de dados da API	Duração
1	98.71%	96.08%	6h14min
2	98.75%	96.36%	6h15min
3	98.72%	95.44%	6h17min
4	98.73%	95.93%	6h12min
5	98.75%	96.19%	6h26min
6	98.79%	96.34%	6h17min
7	98.69%	95.67%	6h20min
8	98.85%	96.71%	6h11min
9	98.70%	96.26%	6h22min
10	98.76%	96.41%	6h06min
Média	**98.74%**	**96.14%**	-
Desvio Padrão	**0.07913**	**0.37442**	-

Fonte: Do Autor (2016)

4.4 Comparação entre as técnicas

Dado os resultados apresentados nas Seções 4.1, 4.2 e 4.3, é possível apresentar um estudo avaliativo e comparativo das técnicas de Redes Neurais Artificias, Máquinas de Vetores de Suporte e Florestas Aleatórias. Os métodos escolhidos para a comparação são os seguintes: análise de falsos positivos, falsos negativos, verdadeiros positivos, verdadeiros negativos, acurácia média, coeficiente Kappa e gráfico ROC. Acredita-se que os métodos comparativos são suficientes, para apresentar as vantagens e desvantagens de cada técnica, assim como seu nível de eficácia para o reconhecimento de intrusão em redes de computadores.

As Figuras 4.4 e 4.6 apresentam os gráficos ROC com a utilização da base de dados ISCX 2012 e a base de dados da API, respectivamente. De acordo com Prati, Batista e Monard (2008), o ponto (0,1) [1] do gráfico ROC apresenta um classificador perfeito. Desta forma, ao analisar as Figuras 4.4 e 4.6, percebe-se a alta eficácia dos classificadores apresentados nesta dissertação de mestrado.

[1] O numeral 0 representa o eixo das ordenadas — eixo X — e 1 representa o eixo das abscissas — eixo Y.

Tabela 4.8 – Matriz de Confusão do NIDS com a Base de Dados ISCX 2012 (Primeiro Treinamento) com Florestas Aleatórias

	Positivo	**Negativo**	**Total**
Positivo	394702 (49.02%)	5749 (0.71%)	400451
Negativo	4587 (0.56%)	400126 (49.69%)	404713
Total	399289	405875	805164

Fonte: Do Autor (2016)

Tabela 4.9 – Matriz de Confusão do NIDS com a Base de Dados da API (Primeiro Treinamento) com Florestas Aleatórias

	Positivo	**Negativo**	**Total**
Positivo	62429 (54.27%)	696 (0.60%)	63125
Negativo	3802 (3.30%)	48103 (41.81%)	51905
Total	66231	48799	115030

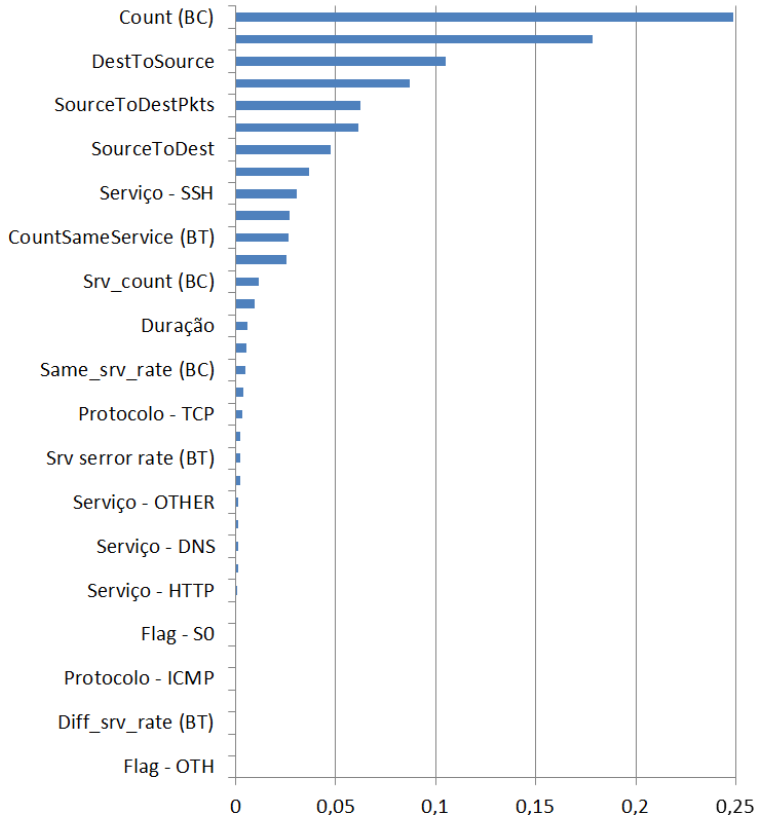
Fonte: Do Autor (2016)

Na Figura 4.4, de acordo com os tópicos apresentados na Seção 2.2.3, observa-se que as três técnicas possuem uma característica de "conservadorismo" na classificação. Neste caso, a técnica de Florestas Aleatórias é claramente mais eficaz, de acordo com Fawcett (2006), pois está acima e à esquerda dos demais pontos, tendo como consequência uma maior taxa de verdadeiros positivos e uma menor taxa de falsos positivos, parâmetros que são comprovados a partir da Figura 4.5.

Já na Figura 4.6, é possível perceber um equilíbrio entre as técnicas de Máquinas de Vetores de Suporte (SVM) e Florestas Aleatórias. A técnica de RNA apresenta-se mais à esquerda, porém abaixo das outras técnicas, isso significa que o número de falsos positivos, em relação às outras técnicas, será reduzido (Figura 4.7). Entretanto, a eficácia na classificação de verdadeiros positivos fica prejudicada, mesmo assim, com o alto poder de generalização da técnica de Redes Neurais Artificiais, a classificação de Verdadeiros Positivos e Negativos permaneceu equilibrada, se comparada às outras técnicas (Figura 4.8).

O cálculo do coeficiente Kappa foi realizado, para ambas as bases de dados, porém, mesmo com os bons resultados obtidos, não é possível distinguir a melhor técnica. O gráfico dos valores Kappa é apresentado na Figura 4.9. Pode-se perceber, em todas as técnicas e em ambas as bases de dados que os valores resultantes são considerados excelentes, de acordo com a Tabela 2.3, apresentada na Seção 2.2.3.

Figura 4.3 – Gráfico da importância das principais características da Base de Dados ISCX 2012, com Florestas Aleatórias (Primeiro Treinamento)



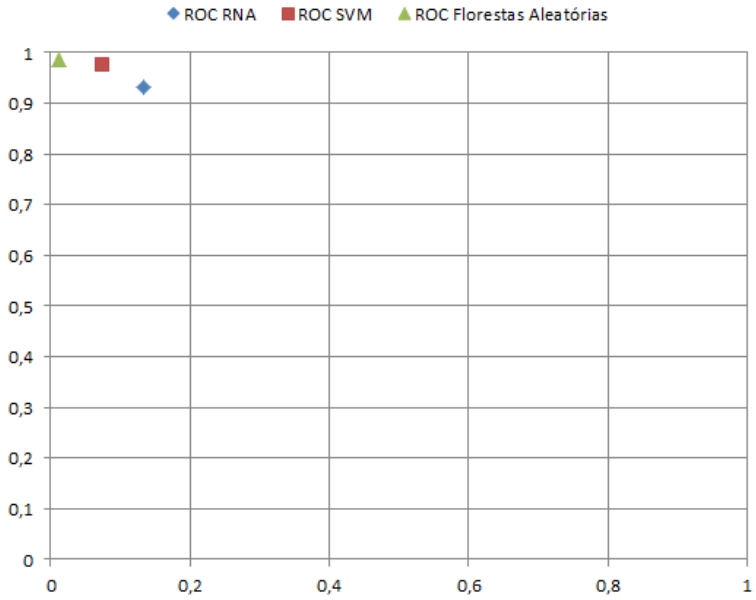
Fonte: Do Autor (2016)

A acurácia média da aplicação das técnicas de Inteligência Computacional é apresentada na Figura 4.10. Percebe-se um alto grau de desbalanceamento entre os resultados apresentados para a técnica de Redes Neurais Artificiais, porém afirma-se a alta capacidade de generalização do método, o qual obteve uma acurácia média próxima a 99%, para dados desconhecidos. Já a técnica de Máquinas de Vetores de Suporte apresenta melhor equilíbrio entre as duas bases de dados, apresentando, também, um grau de generalização. Por fim, na técnica de Florestas Aleatórias, a qual proporciona melhores resultados com os dados de teste da base de dados ISCX 2012, percebe-se um grau menor de generalização.

4.5 Considerações finais

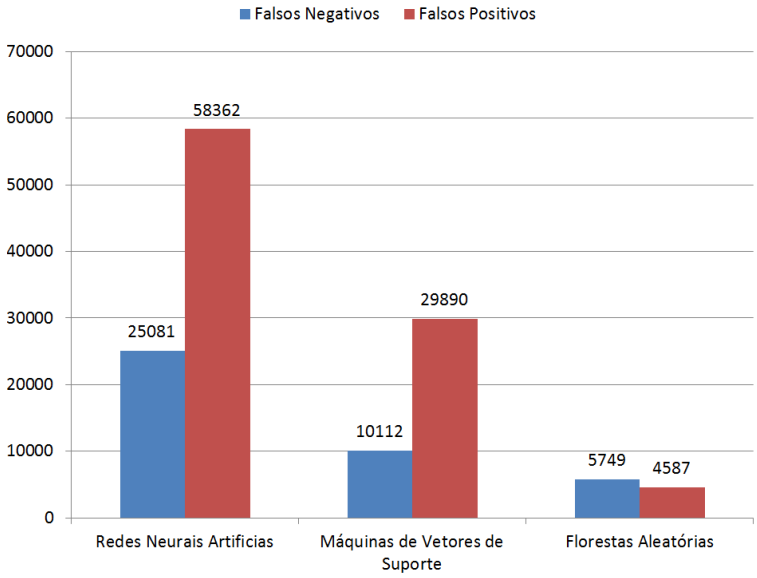
Visto os comparativos das técnicas de Inteligência Computacional, apresentados na seção anterior (4.4), percebe-se a alta eficácia das mesmas. Entretanto, alguns aspectos como o número de falsos positivos e falsos negativos (Figuras 4.5 e 4.7), e também, a capacidade de generalização do método, precisam ser levados em conta. Para a base de dados ISCX 2012, o

Figura 4.4 – Gráfico ROC com a Base de Dados ISCX 2012



Fonte: Do Autor (2016)

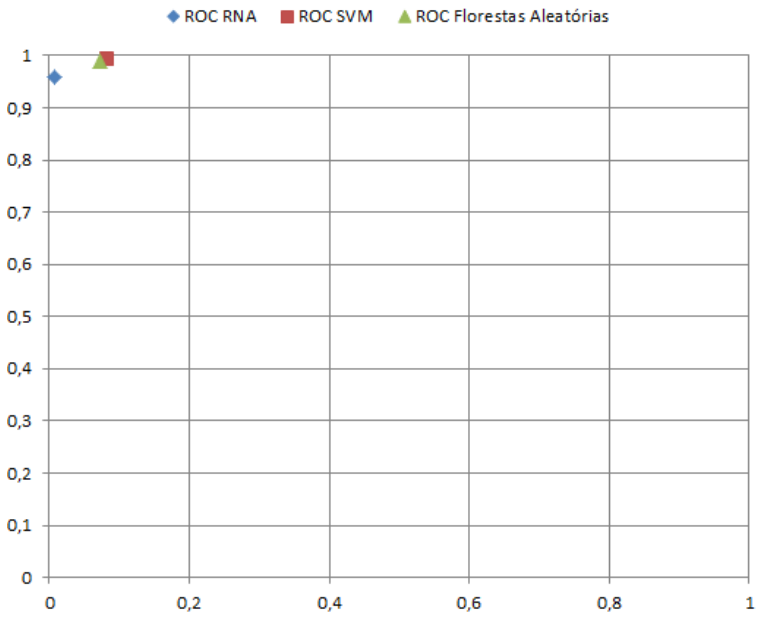
Figura 4.5 – Gráfico da Relação de Falsos Positivos e Falsos Negativos entre as técnicas, utilizando a Base de Dados ISCX 2012



Fonte: Do Autor (2016)

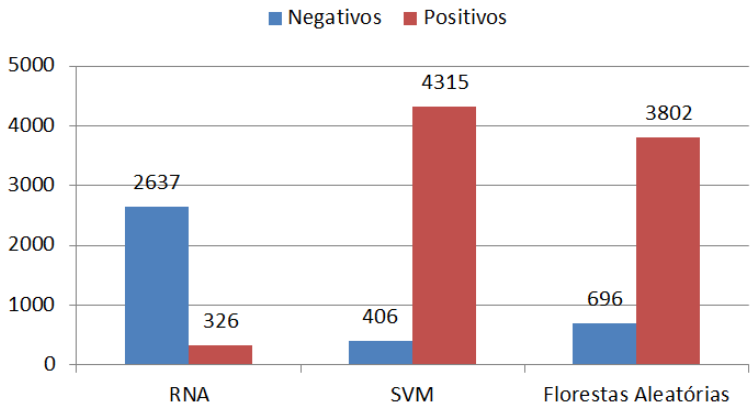
classificador de Florestas Aleatórias claramente demonstrou-se superior aos demais, fator que não acontece para dados desconhecidos (base de dados da API). Todavia, com os testes utilizando dados desconhecidos (base de dados da API), a técnica de Redes Neurais Artificias

Figura 4.6 – Gráfico ROC com a Base de Dados da API



Fonte: Do Autor (2016)

Figura 4.7 – Gráfico da Relação de Falsos Positivos e Falsos Negativos entre as técnicas, utilizando a Base de Dados da API



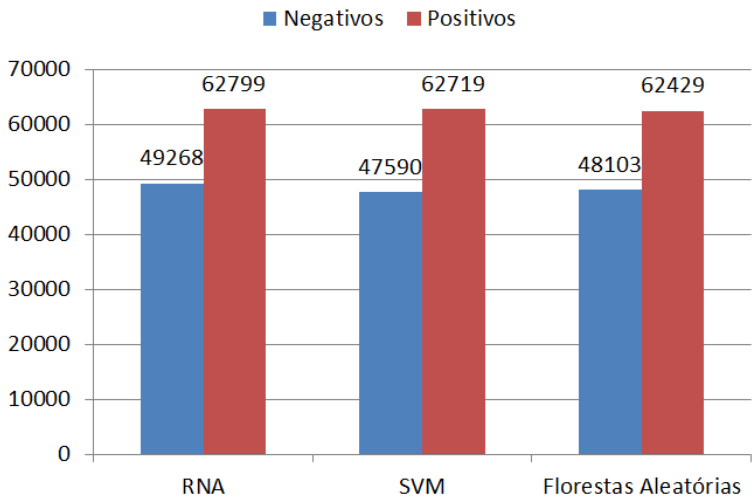
Fonte: Do Autor (2016)

apresentou a melhor acurácia média (Figura 4.10), mas também apresentou o maior número de Falsos Negativos (Figura 4.7), o qual representa o pior cenário [2].

Haja vista os resultados obtidos, é possível afirmar a alta eficácia de todos os métodos utilizados para problemas de reconhecimento de intrusão em redes de computadores. Sendo assim, fica a critério do utilizador do sistema a definição de uma ou mais características para a escolha do método. Para o caso da aplicação em um sistema embarcado, recomenda-se a técnica

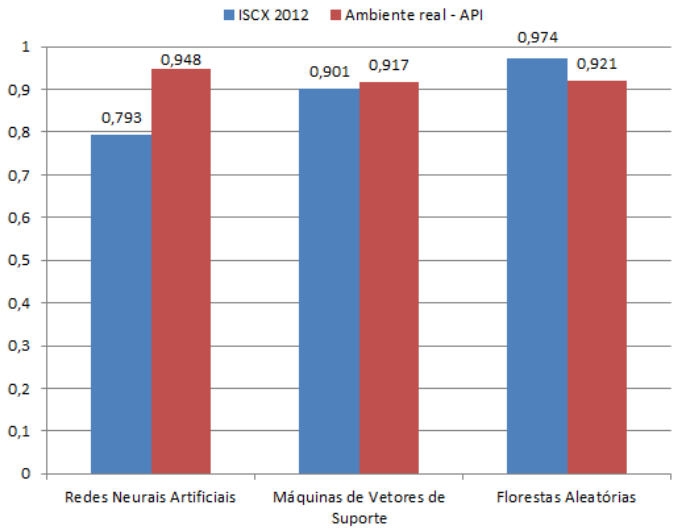
[2] A representação do pior cenário, em virtude do alto número de falsos negativos, significa que vários ataques passariam pelo NIDS, sem que sejam detectados.

Figura 4.8 – Gráfico da Relação de Verdadeiros Positivos e Verdadeiros Negativos entre as técnicas, utilizando a Base de Dados da API



Fonte: Do Autor (2016)

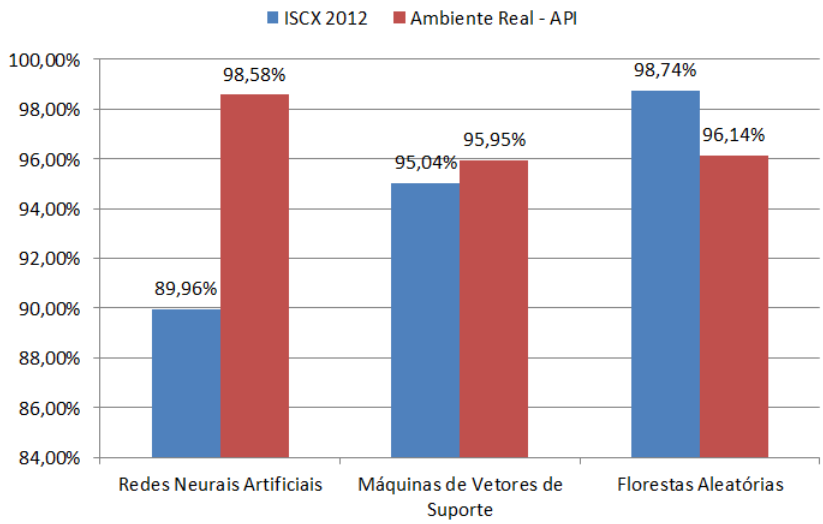
Figura 4.9 – Gráfico comparativo dos valores resultantes do Coeficiente Kappa



Fonte: Do Autor (2016)

com menor utilização de memória, sendo elas, por ordem crescente de utilização: Redes Neurais Artificias, Florestas Aleatórias e Máquinas de Vetores de Suporte. A próxima seção apresenta o funcionamento do NIDS em ambiente real.

Figura 4.10 – Gráfico da Acurácia Média utilizando todas as técnicas de Inteligência Computacional em ambas as Bases de Dados

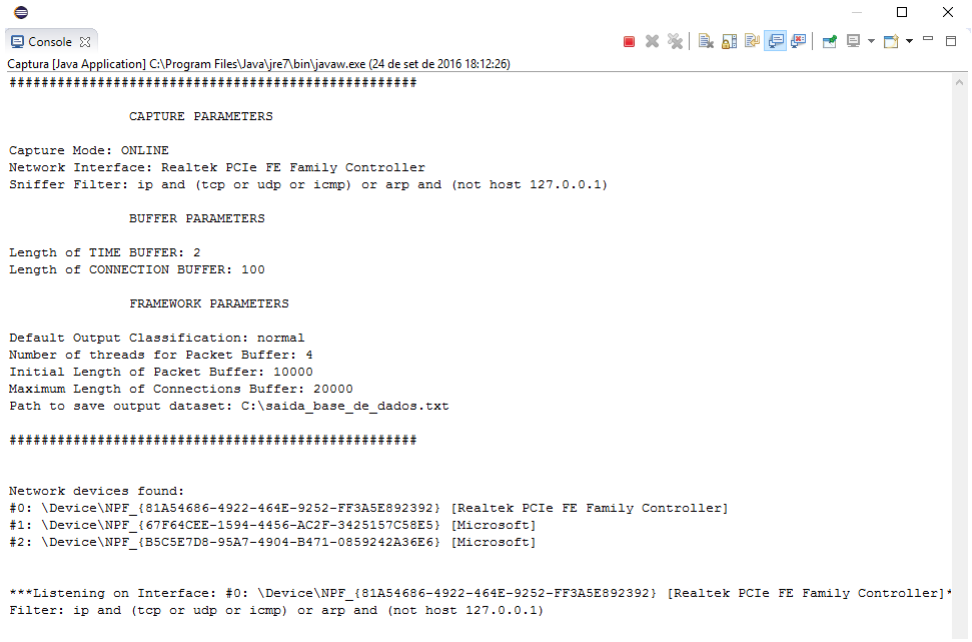


Fonte: Do Autor (2016)

4.6 Funcionamento do NIDS em ambiente real

Esta seção apresenta o funcionamento do NIDS em ambiente real. Para a demonstração utilizou-se um computador com *Microsoft Windows 10* e uma máquina virtual com *Linux Ubuntu 14.04*. A Figura 4.11 apresenta a inicialização do sistema, onde são definidas as interfaces, modos de utilização, filtros de rede e tamanho dos tipos de *buffer*.

Figura 4.11 – Inicialização da API em ambiente real

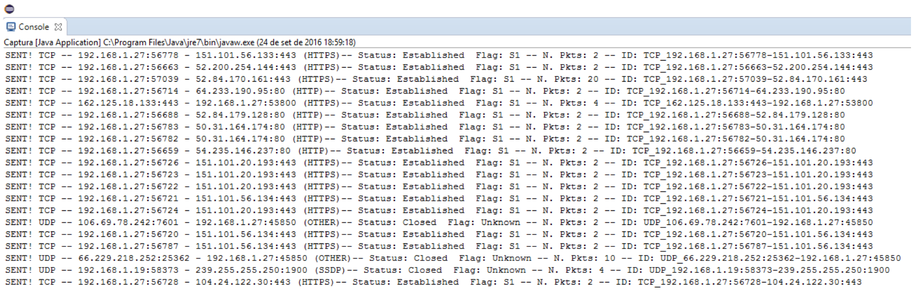


Fonte: Do Autor (2016)

A Figura 4.12 apresenta o funcionamento do sistema após alguns minutos de utilização. Ao ocorrer os *timeouts*, pré-definidos, da conexão, a mesma é despachada, via *socket*, para o motor de classificação (representado na Figura 4.13). O motor de classificação recebe a conexão e faz a predição, a qual pode ser encaminhada para um módulo de notificações ou apenas descartada. Como exemplo de utilização do motor de classificação, utilizou-se a técnica de Redes Neurais Artificiais, sendo possível implementá-la para as demais técnicas. A plataforma *Microsoft Azure*, também, permite que um *web service* [3] seja criado, permitindo, assim, a comunicação entre os módulos, no caso da utilização de Máquinas de Vetores de Suporte.

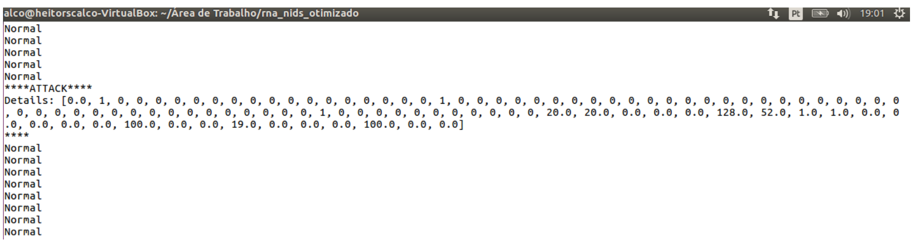
[3] É uma solução utilizada para a integração de sistemas e para comunicação entre aplicações diferentes.

Figura 4.12 – API enviando as conexões, via *socket*, com Microsoft Windows 10



Fonte: Do Autor (2016)

Figura 4.13 – API recebendo as conexões, via *socket*, com Linux Ubuntu 14.04



Fonte: Do Autor (2016)

5 CONCLUSÃO

Sistemas de Detecção de Intrusão são elementos indispensáveis para a manutenção da segurança em uma rede de computadores. Neste trabalho desenvolveu-se um NIDS, para ambiente real, utilizando três diferentes técnicas de Inteligência Computacional: Redes Neurais Artificias, Máquinas de Vetores de Suporte e Florestas Aleatórias. Essas técnicas foram aplicadas ao problema de detecção de intrusos em redes de computadores.

Após a aplicação e análise das técnicas de Inteligência Computacional, em diferentes infraestruturas e tipos de tráfego, notou-se um certo equilíbrio entre elas. Isso comprova a eficácia da utilização de Inteligência Computacional, para realizar o reconhecimento de intrusão, em redes de computadores, principalmente, de ataques de negação de serviço, sem que seja necessário adicionar fatores oriundos de *hosts* em específico.

As taxas médias de acurácia das técnicas de Redes Neurais Artificiais, Máquinas de Vetores de Suporte e Florestas Aleatórias foram de até 98,74%, para a base de dados ISCX 2012 e de até 98,58%, para a base de dados construída manualmente, com auxílio da API desenvolvida. Obtiveram-se índices de falsos positivos e falsos negativos aceitáveis, em relação às outras propostas apresentadas no decorrer deste trabalho.

Quanto à API desenvolvida, observou-se a estabilidade satisfatória da aplicação, mesmo com cargas altas de utilização de banda. Na comunicação entre os módulos, não se notou qualquer tipo de gargalo, nem mesmo perda de pacotes (por utilizar o protocolo TCP).

Além da contribuição científica deste trabalho, é apresentado um grande potencial comercial, para redes de pequeno porte, utilizando sistemas embarcados de baixo custo. Por fim, a próxima seção apresenta algumas propostas de continuidade, para trabalhos futuros.

5.1 Propostas de continuidade

Como propostas de continuidade estão incluídos alguns aprimoramentos do sistema como:

1. Realizar testes com outros classificadores.
2. Construir um NIDS com um comitê de técnicas de Inteligência Computacional, desta forma a classificação majoritária é levada em consideração.
3. Reescrever a API em linguagem de baixo nível para aplicar em um sistema embarcado.

4. Testar o método em uma rede de grande porte.

5. Inserir no vetor de características um parâmetro que contabilize a quantidade de ruído nos dados e a quantidade de ativos na rede.

REFERÊNCIAS

AL-JANABI, S. T. F.; SAEED, H. A. A neural network based anomaly intrusion detection system. In: DEVELOPMENTS IN E-SYSTEMS ENGINEERING, 2011, New York. **Proceedings...** New York: IEEE, 2011. p. 221–226.

AMBWANI, T. Multi class support vector machine implementation to intrusion detection. In: INTERNATIONAL JOINT CONFERENCE ON, 3., 2003, New York. **Proceedings...** New York: IEEE, 2003. p. 2300–2305.

ARAÚJO, N. V. d. S. et al. Kappa-artmap fuzzy: uma metodologia para detecção de intrusos com seleção de atributos em redes de computadores. In: WORKSHOP DE GERÊNCIA E OPERAÇÃO DE REDES E SERVIÇOS, 18., 2013, Brasília. **Anais...** Brasília: SBRC, 2013.

ATTOH-OKINE, N. O. Analysis of learning rate and momentum term in backpropagation neural network algorithm trained to predict pavement performance. **Advances in Engineering Software**, Amsterdam, v. 30, n. 4, p. 291–302, Abr. 1999.

AZEVEDO, R. P. d. **Detecção de ataques de negação de serviço em redes de computadores através da transformada Wavelet 2d.** 2012. 98 p. Dissertação (Mestrado em Informática) – Universidade Federal de Santa Maria, Santa Maria, 2012.

BARFORD, P. et al. A signal analysis of network traffic anomalies. In: ACM SIGCOMM INTERNET MEASUREMENT WORKSHOP, 2., 2002, London. **Proceedings...** London: ACM, 2002. p. 71–82.

BARRETO, J. M. **Introdução às redes neurais artificiais**. Florianópolis: Editora da UFSC, 2002. 62 p.

BATISTA, G. **Pré-processamento de dados em aprendizado de máquina supervisionado**. 2003. 232 p. Tese (Doutorado em Ciências da Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2003.

BEN-HUR, A.; WESTON, J. A user's guide to support vector machines. In: CARUGO, O.; EISENHARB, F. **Data mining techniques for the life sciences**, Amsterdam: Springer, p. 223–239, 2010.

BING, X.; XIAOSU, C.; NING, C. An efficient tcp flow state management algorithm in high-speed network. In: INFORMATION ENGINEERING AND ELECTRONIC COMMERCE, 2009, New York. **Proceedings...** New York: IEEE, 2009.

BISHOP, C. M. **Neural networks for pattern recognition**. Oxford: Oxford University Press, 1995. 504 p.

BOLDT, A. S. et al. **Coleções nucleares e associação do teor de óleo de cártamo com variáveis ecogeográficas por inteligência computacional**. 2014. 67 p. Tese (Doutorado em Genética) – Universidade Federal de Viçosa, Viçosa, 2014.

BRAGA, A. d. P.; FERREIRA, A. C. P. d. L.; LUDERMIR, T. B. **Redes neurais artificiais**: teoria e aplicações. São Paulo: LTC, 2007. 238 p.

BREIMAN, L.; CUTLER, A. **Random Forests**: classification description. [S.l.: s.n.], 2016. Disponível em: <http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm>. Acesso em: 15 mar. 2016.

BURGES, C. J. A tutorial on support vector machines for pattern recognition. **Data mining and knowledge discovery**, New York, v. 2, n. 2, p. 121–167, 1998.

CHANG, C.-C.; LIN, C.-J. LIBSVM: a library for support vector machines. **ACM Transactions on Intelligent Systems and Technology**, New York, v. 2, p. 1–27, 2011.

CHEN, W.-H.; HSU, S.-H.; SHEN, H.-P. Application of SVM and ANN for intrusion detection. **Computers & Operations Research**, New York, v. 32, n. 10, p. 2617–2634, Oct. 2005.

COHEN, J. et al. A coefficient of agreement for nominal scales. **Educational and Psychological Measurement**, Durham, v. 20, n. 1, p. 37–46, 1960.

CUNHA NETO, R. P. d. **Sistema de Detecção de Intrusos em Ataques Oriundos de Botnets Utilizando Método de Detecção Híbrido**. 2011. 101 p. Dissertação (Mestrado de Engenharia de Eletricidade) – Universidade Federal do Maranhão, São Luís, 2011.

CYBENKO, G. **Continuous valued neural networks with two hidden layers are sufficient**. Illinois: University of Illinois at Urbana-Champaign, 1988. 431 p.

DENG, H.; ZENG, Q.-A.; AGRAWAL, D. P. Svm-based intrusion detection system for wireless ad hoc networks. In: VEHICULAR TECHNOLOGY CONFERENCE, 58., 2003, New York. **Proceedings...** New York: IEEE, 2003. p. 2147–2151.

DUDA, R. O.; HART, P. E.; STORK, D. G. **Pattern classification**. New York: John Wiley and Sons, 2012. 680 p.

FAWCETT, T. An introduction to roc analysis. **Pattern recognition letters**, Amsterdam, v. 27, n. 8, p. 861–874, 2006.

FILIPPETTI, M. A. **CCNA5.0 - Guia Completo de Estudo**. Florianópolis: Visual Books, 2011. 544 p.

FONSECA, R.; SILVA, P.; SILVA, R. Acordo inter-juízes: o caso do coeficiente kappa. **Laboratório de Psicologia**, Portugal, v. 5, n. 1, p. 81–90, 2013.

HAN, C. et al. An intrusion detection system based on neural network. In: IEEE. MECHATRONIC SCIENCE, ELECTRIC ENGINEERING AND COMPUTER INTERNATIONAL CONFERENCE ON, 2011, New York. **Proceedings...** New York: IEEE, 2011. p. 2018–2021.

HEINEN, M. R.; OSÓRIO, F. S. **Autenticação de Assinaturas utilizando Análise de Componentes Principais e Redes Neurais Artificiais**. Ribeirão Preto: UNISINOS, 2006. 6 p.

HPING3. **Hping3-alpha-1 released**. [S.l.: s.n], 2016. Disponível em: <http://www.hping.org/hping3.html>. Acesso em: 15 mar. 2016.

HSU, C.-W.; CHANG, C.-C.; LIN, C.-J. **A practical guide to support vector classification**. Taipei: National Taiwan University, 2003. 16 p.

IATAC, T. M. W. **Information Assurance Technology Analysis Center (IATAC)**: Information assurance tools report - intrusion detection systems. 6. ed. Indiana: Information Assurance Technology Tools, 2009. 53 p.

JALIL, K. A.; KAMARUDIN, M. H.; MASREK, M. N. Comparison of machine learning algorithms performance in detecting network intrusion. In: IEEE. NETWORKING AND INFORMATION TECHNOLOGY INTERNATIONAL, 2010, New York. **Proceedings...** New York: IEEE, 2010. p. 221–226.

JING-XIN, W.; ZHI-YING, W.; KUI, D. A network intrusion detection system based on the artificial neural networks. In: INTERNATIONAL CONFERENCE ON INFORMATION SECURITY, 3., 2004, New York. **Proceedings...** New York: IEEE, 2004. p. 166–170.

KUKIELKA, P.; KOTULSKI, Z. Analysis of different architectures of neural networks for application in intrusion detection systems. In: COMPUTER SCIENCE AND INFORMATION INTERNATIONAL MULTICONFERENCE ON, 2008, New York. **Proceedings...** New York: IEEE, 2008. p. 807–811.

LABS, I. **Intrusion detection systems**: buyer's guide. [S.l.: s.n.], 1999. 53 p.

LAUREANO, M. A. P.; MAZIERO, C. A.; JAMHOUR, E. Detecção de intrusão em máquinas virtuais. In: SIMPÓSIO DE SEGURANÇA EM INFORMÁTICA, 5., 2003, São José dos Campos. **Anais...** São José dos Campos: São José dos Campos, 2003. p. 7.

LI, H. et al. Network intrusion detection based on support vector machine. **Journal of Computer Research and Development**, Amsterdam, v. 6, p. 799–807, 2003.

LI, Y. et al. An efficient intrusion detection system based on support vector machines and gradually feature removal method. **Expert Systems with Applications**, Amsterdam, v. 39, n. 1, p. 424–430, 2012.

LIMA, R. A. G. **Utilização de Sistemas Inteligentes para Classificação de Tráfego Malicioso**. 2014. 80 p. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal Viçosa, Viçosa, 2014.

LORENA, A. C.; CARVALHO, A. C. de. Uma introdução às support vector machines. **Revista de Informática Teórica e Aplicada**, Porto Alegre, v. 14, n. 2, p. 43–67, 2007.

MAFRA, P. M. et al. **POLVO-IIDS**: Um sistema de detecçao de intrusão inteligente baseado em anomalias. [S.l.: s.n.], 2008. 13 p.

MALTAINFOSEC. **The concept of Intrusion Detection Systems**. [S,l.], 2011. Disponível em: <http://maltainfosec.org/archives/26-The-concept-of-Intrusion-Detection-Systems.html>. Acesso em: 15 mar. 2016.

MICROSOFT. **Microsoft Azure Machine Learning Studio**. [S.l.: s.n.], 2016. Disponível em: <https://studio.azureml.net>. Acesso em: 15 mar. 2016.

MOUSTAFA, N.; SLAY, J. The evaluation of network anomaly detection systems: Statistical analysis of the unsw-nb15 data set and the comparison with the kdd99 data set. **Information Security Journal: A Global Perspective**, Washington, v. 25, n. 1/3, p. 18–31, 2016.

MUKKAMALA, S.; JANOSKI, G.; SUNG, A. Intrusion detection using neural networks and support vector machines. In: IEEE. Neural Networks IJCNN'02. OF THE 2002 INTERNATIONAL JOINT CONFERENCE ON, 2., 2002, New York. **Proceedings...** New York: IEEE, 2002. p. 1702–1707.

NISSEN, S.; NEMERSON, E. **Fast artificial neural network library**. [S.l.: s.n.], 2000. Disponível em: <dk/fann/html/files/fann-h.html>. Acesso em: 18 mar. 2016.

NMAP. **NMAP Security Scanner**. [S.l.: s.n], 2016. Disponível em: <https://nmap.org/>. Acesso em: 16 mar. 2016.

OSHIRO, T. M. **Uma abordagem para a construção de uma única árvore a partir de uma Random Forest para classificação de bases de expressão gênica**. 2013. Dissertação (Mestrado em Bioinformática) — Universidade de São Paulo, 2013.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in python. **Journal of Machine Learning Research**, Oxford, v. 12, p. 2825–2830, Out. 2011.

POSTEL, J. et al. Rfc 792: Internet control message protocol. **Journal Post**, Essex, 1981. Disponível em: <https://tools.ietf.org/html/rfc792>. Acesso em: 16 mar. 2016.

PRATI, R.; BATISTA, G.; MONARD, M. Curvas ROC para avaliação de classificadores. **Revista IEEE América Latina**, Caribe, v. 6, n. 2, p. 215–222, 2008.

PRATI, R. C. et al. Uma experiência no balanceamento artificial de conjuntos de dados para aprendizado com classes desbalanceadas utilizando análise ROC. In: Workshop de Inteligência Artificial, 4., 2003, Chile. **Anais...** Chile: [s.n.], 2003. p. 28–33.

RAJAGOPAL, D.; THILAKAVALLI, K.; FATHIMA, K. S. A. New approach to monitoring internet access along with usage of bandwidth using intrusion detection system. **International Journal of Security and Its Applications**, Oxford, v. 9, n. 6, p. 183–194, 2015.

RAO, X.; DONG, C.-X.; YANG, S.-Q. An intrusion detection system based on support vector machine. **Journal of Software**, Sussex, v. 4, n. 14, p. 1–6, 2003.

RAUBER, T. W. **Redes neurais artificiais**. Espírito Santo: Universidade Federal do Espírito Santo, 2005.

RODRIGUES, R. C. B. et al. **Máquinas de vetores de suporte aplicadas à classificação de defeitos em couro bovino**. Logan: Universidade Católica Dom Bosco Campo Grande, 2007. 6 p.

RÜCKSTIESS, T.; FELDER, M.; SCHMIDHUBER, J. State-dependent exploration for policy gradient methods. In: WALTER, D.; KATHARINA, M. (Ed.). **Machine learning and knowledge discovery in databases**, Amsterdam: Springer, p. 234–249, 2008.

SA, L. C. d. Árvores, florestas e sua função como preditores: Uma aplicação na avaliação do grau de maturidade de empresas. **Af-Revista PMKT**, São Paulo, p. 1–6, 2011.

SALEM, M.; BUEHLER, U. Reinforcing network security by converting massive data flow to continuous connections for ids. In: INTERNET TECHNOLOGY AND SECURED TRANSACTIONS INTERNATIONAL CONFERENCE FOR, 8., 2013, New York. **Proceedings...** New York: IEEE, 2013. p. 570–575.

SALEM, M.; REISSMANN, S.; BUEHLER, U. Persistent dataset generation using real-time operative framework. In: COMPUTING, NETWORKING AND COMMUNICATIONS INTERNATIONAL CONFERENCE ON, 2014, New York. **Proceedings...** New York: IEEE, 2014. p. 1023–1027.

SANTOS, E. M. d. **Teoria e Aplicação de Support Vector Machines à Aprendizagem e Reconhecimento de Objetos Baseado na Aparência**. 2002. 121 p. Dissertação (Mestrado em Informática) – Universidade Federal da Paraíba, Campina Grande, 2002.

SCALCO, H. N. Reconhecimento de intrusão em redes de computadores utilizando pybrain. In: CONGRESSO BRASILEIRO DE INTELIGÊNCIA COMPUTACIONAL, 12., 2015, Paraná. **Anais...** Paraná: UTFPR, 2015. p. 1–6.

SCARFONE, P. M. K. **Guide to Intrusion Detection and Prevention Systems (IDPS)**. Gaithersburg: National Institute of Standards and Technology Gaithersburg, 2007. 127 p.

SCHAUL, T. et al. Pybrain. **The Journal of Machine Learning Research**, Oxford, v. 11, p. 743–746, 2010.

SCHAUL, T.; SCHMIDHUBER, J. Scalable neural networks for board games. in: ALIPPI, C. et al. **Artificial neural networks–ICANN 2009**, Amsterdam: Springer, p. 1005–1014, 2009.

SCIKIT. Documentation sklearn.ensemble.random Forest Classifier. **Scikitlearn.org**, 2016. Disponível em: <http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>. Acesso em: 10 mar. 2016.

SEHNKE, F. et al. Policy gradients with parameter-based exploration for control. In: KURKOVA-POHLOVA, V.; KOUTNIK, J. **Artificial neural networks-ICANN 2008**, Amsterdam: Springer, p. 387–396, 2008.

SEN, N.; SEN, R.; CHATTOPADHYAY, M. An effective back propagation neural network architecture for the development of an efficient anomaly based intrusion detection system. In: COMPUTATIONAL INTELLIGENCE AND COMMUNICATION NETWORKS INTERNATIONAL CONFERENCE ON, 2014, New York. **Proceedings...** New York: IEEE, 2014. p. 1052–1056.

SHALEV-SHWARTZ, S. et al. Pegasos: primal estimated sub-gradient solver for svm. **Mathematical Programming**, Amsterdam, v. 127, n. 1, p. 3–30, Mar. 2011.

SHIRAVI, A. et al. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. **Computers & Security**, Amsterdam, v. 31, n. 3, p. 357–374, 2012.

SILVA, J. R. C. D. **Sistemas de Detecção de Intrusão com técnicas de inteligencia artificial**. 2011. 157 p. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Viçosa, Viçosa, 2011.

SIMON, H. **Redes neurais**: princípios e prática. Porto Alegre: Bookman, 2001. 900 p.

SNORT IDS. **Snort.org**. [s.l.], 2015. Disponível em: <https://www.snort.org/#documents>. Acesso em: 10 mar. 2016.

SOUZA, E. P. d.; MONTEIRO, J. A. S. Estudo sobre sistema de detecção de intrusão por anomalias uma abordagem utilizando redes neurais. In: WORKSHOP DE GERÊNCIA E OPERAÇÃO DE REDES E SERVIÇOS, 14., 2008, Vitória. **Anais...** Vitória: [s.n.], 2008.

STOLFO, J. et al. **Cost-based modeling and evaluation for data mining with application to fraud and intrusion detection**. New York: [s.n.], 2000. 39 p.

T50. T50 very fast network stress tool. **Source Forge**, [s.n.], 2016. Disponível em: <https://sourceforge.net/projects/t50/>. Acesso em: 10 mar. 2016.

TANENBAUM, A. S. **Computer networks**. 5. ed. New York: Pearson, 2011. 962 p.

TAVALLAEE, M. et al. A detailed analysis of the kdd cup 99 data set. In: SYMPOSIUM ON COMPUTATIONAL INTELLIGENCE, 2008, New York. **Proceedings...** New York: IEEE, 2009. p. 1–6.

TCPDUMP. **Name**. [S.l.: s.n.], 2015. Disponível em: <http://www.tcpdump.org/tcpdump_man.html>. Acesso em: 10 mar. 2016.

UCHOA, J. Q. **Algoritmos imunoinspirados aplicados em segurança computacional: utilizaçao de algoritmos inspirados no sistema imune para detecçao de intrusos em redes de computadores**. 2009. 279 p. Tese (Doutorado em Bioinformática) – Universidade Federal de Minas Gerais, Belo Horizonte, 2009.

VAPNIK, V. N. Constructing learning algorithms. In: VAPNIK, V. **The nature of statistical learning theory**, New York: Springer, p. 119–166, 1995.

WANG, J. **Computer network security**: theory and practice. Massachusetts: Springer, 2009. 400 p.

WU, S. X.; BANZHAF, W. The use of computational intelligence in intrusion detection systems: A review. **Applied Soft Computing**, Amsterdam, v. 10, n. 1, p. 1–35, Jan. 2010.

YASSIN, W. et al. Anomaly-based intrusion detection through k-means clustering and naives bayes classification. In: INTERNATIONAL CONFERENCE ON COMPUTING AND INFORMATICS, 4., 2013, Sarawak. **Proceedings...** Sarawak: [s.n.], 2013. p. 298–303.

ZHANG, J.; ZULKERNINE, M. **Network intrusion detection using random forests.** Kingston: Queen's University, 2005. 9 p.

APÊNDICE A – Relação dos Ataques Realizados

Página em Branco.

Tabela 1 – Ataques Realizados

Protocolo	**Ataque**	**Ferramenta**	**Comando**
TCP	*SYN Flood*	*t50*	t50 [*IP*] - -flood -S - -turbo t50 [*IP*] - -flood -S - -turbo - -dport 20 t50 [*IP*] - -flood -S - -turbo - -dport 21 t50 [*IP*] - -flood -S - -turbo - -dport 22 t50 [*IP*] - -flood -S - -turbo - -dport 80
TCP	*ACK Flood*	*t50*	t50 [*IP*] - -flood -A - -turbo t50 [*IP*] - -flood -A - -turbo - -dport 20 t50 [*IP*] - -flood -A - -turbo - -dport 21 t50 [*IP*] - -flood -A - -turbo - -dport 22 t50 [*IP*] - -flood -A - -turbo - -dport 80
TCP	*FIN Flood*	*t50*	t50 [*IP*] - -flood -F - -turbo t50 [*IP*] - -flood -F - -turbo - -dport 20 t50 [*IP*] - -flood -F - -turbo - -dport 21 t50 [*IP*] - -flood -F - -turbo - -dport 22 t50 [*IP*] - -flood -F - -turbo - -dport 80
TCP	*RST Flood*	*t50*	t50 [*IP*] - -flood -R - -turbo t50 [*IP*] - -flood -R - -turbo - -dport 20 t50 [*IP*] - -flood -R - -turbo - -dport 21 t50 [*IP*] - -flood -R - -turbo - -dport 22 t50 [*IP*] - -flood -R - -turbo - -dport 80
TCP	*URG Flood*	*t50*	t50 [*IP*] - -flood -U - -turbo t50 [*IP*] - -flood -U - -turbo - -dport 20 t50 [*IP*] - -flood -U - -turbo - -dport 21 t50 [*IP*] - -flood -U - -turbo - -dport 22 t50 [*IP*] - -flood -U - -turbo - -dport 80
TCP	*PSH Flood*	*t50*	t50 [*IP*] - -flood -P - -turbo t50 [*IP*] - -flood -P - -turbo - -dport 20 t50 [*IP*] - -flood -P - -turbo - -dport 21 t50 [*IP*] - -flood -P - -turbo - -dport 22 t50 [*IP*] - -flood -P - -turbo - -dport 80
UDP	*UDP Flood*	*hping3*	hping3 -2 -c 10000 –flood –rand-source -w 64 -d 120 -p 80 [IP] hping3 -2 -c 10000 –flood –rand-source -w 64 -d 120 [IP]
TCP UDP ICMP	*Port Scanning*	*NMAP*	nmap -sS [IP] nmap -sS -sV [IP] nmap -sS -sV -Pn [IP] nmap -sS -sV -Pn -O [IP] nmap -sT [IP] nmap -PM [IP] nmap -PE [IP] nmap -sV [IP]

APÊNDICE B – Código-Fonte do Módulo de Captura da API

Disponível em: https://github.com/heitorscalco/NIDSProject

```
1  package capturator;
2  import java.io.BufferedReader;
3  import java.io.FileReader;
4  import java.util.ArrayList;
5  import java.util.Date;
6  import java.util.List;
7
8  import org.jnetpcap.Pcap;
9  import org.jnetpcap.PcapBpfProgram;
10 import org.jnetpcap.PcapIf;
11 import org.jnetpcap.packet.PcapPacket;
12 import org.jnetpcap.packet.PcapPacketHandler;
13 import org.jnetpcap.protocol.network.Arp;
14 import org.jnetpcap.protocol.network.Icmp;
15 import org.jnetpcap.protocol.network.Ip4;
16 import org.jnetpcap.protocol.tcpip.Tcp;
17 import org.jnetpcap.protocol.tcpip.Udp;
18
19 import com.sun.corba.se.impl.orbutil.concurrent.Mutex;
20
21 import capturator.Packet;
22 import manager.ConnectionsManager;
23
24 import org.jnetpcap.packet.format.FormatUtils;
25
26 import preprocessor.Config;
27
28
29 public class Capture {
30
31   public static ArrayList<Packet> packet_buffer = new ArrayList<
       Packet>();
```

```
32  public static Mutex mutex = new Mutex();
33  public static Mutex mutex_packet_buffer = new Mutex();
34  public static long num_pkts = 0;
35
36  //FUNCAO PRINTFRAMEWORKPARAMETERS OCULTA PARA ECONOMIA DE ESPACO
37
38  private static boolean listLabelsOfConnections(){
39    System.out.println("\n\nLOADING THE FILE THAT CONTAINS THE LABELS
        .");
40    System.out.println("This may take some time, Please Wait. \n\n");
41    try{
42      FileReader arq = new FileReader(Config.PATH_LABELS_FILE);
43      BufferedReader lerArq = new BufferedReader(arq);
44      String linha = lerArq.readLine();
45      String unique_id;
46      while (linha != null) {
47        String[] words = linha.split(";");
48        String datahora = words[5].split(":")[0];
49        //{TIMESTAMP -- PROTOCOLO -- IP_ORIGEM:PORTA_ORIGEM --
            IP_DEST:PORTA_DEST}
50        if(words[1].equals("tcp_ip")){
51          unique_id = datahora + "--" + Config.TAG_TCP + "--" + words
              [0] + ":" + words[2] + "--" + words[3] + ":" + words[4];
52        } else if (words[1].equals("udp_ip")) {
53          unique_id = datahora + "--" + Config.TAG_UDP + "--" + words
              [0] + ":" + words[2] + "--" + words[3] + ":" + words[4];
54        } else if (words[1].equals("icmp_ip") || words[1].equals("
            ipv6icmp")) {
55          unique_id = datahora + "--" + Config.TAG_ICMP + "--" +
              words[0] + ":" + words[2] + "--" + words[3] + ":" +
              words[4];
56        } else if (words[1].equals("ip") || words[1].equals("arp") ||
            words[1].equals("igmp")) {
```

```
57            unique_id = datahora + "--" + Config.TAG_ARP + "--" + words
                  [0] + ":" + words[2] + "--" + words[3] + ":" + words[4];
58          } else {
59            continue;
60          }
61          Config.CONNECTIONS_LABELED.put(unique_id, words[7]);
62
63          linha = lerArq.readLine();
64        }
65        arq.close();
66        System.out.println("File loaded successfully!!!");
67        return(true);
68      } catch(Exception ex){
69        System.out.println("An error has ocurred, exiting now...");
70        return(false);
71      }
72    }
73
74    public static void main(String[] args){
75      try{
76        System.loadLibrary("jnetpcap");
77      }catch(Exception ex){
78        ex.printStackTrace();
79        return;
80      }
81
82      //printFrameworkParameters(); OCULTA
83
84      if(Config.INCLUDE_LABELS_FILE){
85        if(listLabelsOfConnections() == false){
86          return;
87        }
88      }
89
```

```
90      List<PcapIf> alldevs = new ArrayList<PcapIf>(); // Will be filled
            with NICs
91          StringBuilder errbuf = new StringBuilder();    // For any
               error msgs
92          Pcap pcap;
93          if(Config.CAPTURE_ONLINE == true){
94            Pcap.findAllDevs(alldevs, errbuf);
95
96            if (alldevs.isEmpty()) {
97                System.err.printf("Can't read list of devices, error is
                      %s", errbuf.toString());
98                return;
99            }
100
101           System.out.println("Network devices found:");
102           int i = 0, if_aux = -1;
103           String if_aux_name = null, if_aux_description = null;
104           for (PcapIf device : alldevs) {
105               String description = (device.getDescription() != null)
                      ? device.getDescription() : "No description
                      available";
106               if((device.getName().matches(Config.INTERFACE_TO_SNIFF)
                       == true) || (device.getDescription().matches(Config
                      .INTERFACE_TO_SNIFF) == true)){
107                 if_aux = i;
108                 if_aux_name = device.getName();
109                 if_aux_description = device.getDescription();
110               }
111               System.out.printf("#%d: %s [%s]\n", i++, device.getName
                      (), description);
112           }
113
114           if(if_aux == -1){
```

```
115             System.err.printf("Network device %s not found!\n",
                    Config.INTERFACE_TO_SNIFF);
116             return;
117           }
118
119
120           PcapIf device = alldevs.get(if_aux);
121
122           int snaplen = 64 * 1024;           // Capture all packets,
                 no trucation
123           int flags = Pcap.MODE_PROMISCUOUS; // capture all packets
124           int timeout = 10 * 1000;           // 10 seconds in millis
125           pcap = Pcap.openLive(device.getName(), snaplen, flags,
                 timeout, errbuf);
126
127           if (pcap == null) {
128               System.err.printf("Error while opening device for
                      capture: [Permission Error?] " + errbuf.toString());
129               return;
130           }
131
132           System.out.printf("\n\n***Listening on Interface: #%d: %s
                 [%s]***\n", if_aux, if_aux_name, if_aux_description);
133
134         } else {
135       //OFFLINE
136       pcap = Pcap.openOffline(Config.PATH_TO_OFFLINE_PCAP, errbuf);
137       if (pcap == null) {
138         System.err.println(errbuf);
139         return;
140       }
141         }
142
143         PcapBpfProgram program = new PcapBpfProgram();
```

```
144         int optimize = 1;          // 0 = false
145         int netmask = 0xFFFF0000; // 255.255.0.0
146
147         System.out.println("Filter: " + Config.FILTER+"\n");
148
149         if (pcap.compile(program, Config.FILTER, optimize, netmask)
                != Pcap.OK) {
150           System.err.println(pcap.getErr());
151           return;
152         }
153
154         if (pcap.setFilter(program) != Pcap.OK) {
155           System.err.println(pcap.getErr());
156           return;
157         }
158
159         ConnectionsManager gerenciador = new ConnectionsManager();
160     gerenciador.startMonitoringThread(); //Start the monitor of
            buffer
161     gerenciador.updateMovingAverage();
162     for (int i = 0; i < Config.NUM_THREADS_OF_BUFFER; i++) {
163       gerenciador.startManagerThread(); //Start the consumer of
              packets buffer
164     }
165
166         PcapPacketHandler<String> jpacketHandler = new
               PcapPacketHandler<String>() {
167           Tcp tcp = new Tcp();
168             Udp udp = new Udp();
169             Icmp icmp = new Icmp();
170             Arp arp = new Arp();
171             Ip4 ip = new Ip4();
172
173           public void nextPacket(PcapPacket packet, String user) {
```

```
174
175             Packet pkt = new Packet();
176
177             if (packet.hasHeader(ip)){
178               pkt.setIPsrc(FormatUtils.ip(ip.source()));
179               pkt.setIPdst(FormatUtils.ip(ip.destination()));
180               pkt.setTimestamp(new Date(packet.getCaptureHeader().
                     timestampInMillis()));
181               pkt.setIpID(ip.id());
182               pkt.setTTL(ip.ttl());
183
184               if(ip.checksum() == ip.calculateChecksum()){
185                 pkt.setWrong(0);
186               } else {
187                 pkt.setWrong(1);
188               }
189
190               if (packet.hasHeader(tcp)){
191             pkt.setProtocol(Config.TAG_TCP);
192                 pkt.setSport(tcp.source());
193                 pkt.setDport(tcp.destination());
194                 pkt.setLength(tcp.getLength());
195
196                 //Flags
197                 pkt.setSYN(tcp.flags_SYN());
198                 pkt.setACK(tcp.flags_ACK());
199                 if(tcp.flags_SYN() == true && tcp.flags_ACK() == true
                        ){
200                   pkt.setSYNACK(true);
201                 }
202                 pkt.setRST(tcp.flags_RST());
203                 pkt.setFIN(tcp.flags_FIN());
204                 pkt.setPSH(tcp.flags_PSH());
205
```

```
206                 if (tcp.flags_URG() == false) {
207                   pkt.setUrgptr(0);
208                 } else {
209                   pkt.setUrgptr(1);
210                 }
211               } else if(packet.hasHeader(udp)) {
212                 pkt.setProtocol(Config.TAG_UDP);
213                 pkt.setSport(udp.source());
214                 pkt.setDport(udp.destination());
215                 pkt.setLength(udp.getLength());
216               } else if(packet.hasHeader(icmp)){
217                 pkt.setProtocol(Config.TAG_ICMP);
218                 pkt.setLength(icmp.getLength());
219                 pkt.setIcmp_ID(Integer.parseInt((String.format("%8s",
                      Integer.toBinaryString(packet.getUByte(38)))).
                    replace(' ', '0') + String.format("%8s", Integer.
                    toBinaryString(packet.getUByte(39))).replace(' ',
                    '0')), 2));
220                 pkt.setIcmp_Message(Integer.parseInt((String.format("
                    %8s", Integer.toBinaryString(packet.getUByte(34)))
                    .replace(' ', '0')), 2));
221               }
222             } else if (packet.hasHeader(arp)){
223               pkt.setProtocol(Config.TAG_ARP);
224               pkt.setLength(arp.getLength());
225               pkt.setIPdst(Integer.parseInt((String.format("%8s",
                    Integer.toBinaryString(packet.getUByte(38))).replace
                    (' ', '0')),2) + "." +Integer.parseInt((String.
                    format("%8s", Integer.toBinaryString(packet.getUByte
                    (39))).replace(' ', '0')),2) + "." + Integer.
                    parseInt((String.format("%8s", Integer.
                    toBinaryString(packet.getUByte(40))).replace(' ', '0
                    ')),2) + "." + Integer.parseInt((String.format("%8s"
```

```
                , Integer.toBinaryString(packet.getUByte(41))).
                replace(' ', '0')),2));
226             pkt.setIPsrc(Integer.parseInt((String.format("%8s",
                Integer.toBinaryString(packet.getUByte(28))).replace
                (' ', '0')),2) + "." + Integer.parseInt((String.
                format("%8s", Integer.toBinaryString(packet.getUByte
                (29))).replace(' ', '0')),2) + "." + Integer.
                parseInt((String.format("%8s", Integer.
                toBinaryString(packet.getUByte(30))).replace(' ', '0
                ')),2) + "." + Integer.parseInt((String.format("%8s"
                , Integer.toBinaryString(packet.getUByte(31))).
                replace(' ', '0')),2));
227             pkt.setTimestamp(new Date(packet.getCaptureHeader().
                timestampInMillis()));
228           } else {
229             System.out.printf("Invalid IP Header.\n\n");
230             return;
231           }
232
233         packet_buffer.add(pkt);
234         packet = null;
235         pkt = null;
236
237         try { Thread.sleep(1); } catch (InterruptedException e) { e.
              printStackTrace(); }
238         num_pkts++;
239         }
240     };
241
242     pcap.loop(Pcap.LOOP_INFINITE, jpacketHandler, "");
243         pcap.close();
244
245         try {
246       Thread.sleep(200000);
```

```
247     } catch (InterruptedException e1) {
248       // TODO Auto-generated catch block
249       e1.printStackTrace();
250     }
251         ConnectionsManager.kill();
252     System.out.println("\n\n\n\n\n\n\n\n\n\n NUMBER OF PKTS: "+num_pkts
          +" \n\n\n\n\n\n\n\n\n\n");
253
254         System.exit(0);
255   }
256 }
```

APÊNDICE C – Código-Fonte do Módulo de Gerenciamento e Monitoramento de *Timeouts* da API

```
1  package manager;
2  import preprocessor.Output;
3  import preprocessor.Config;
4  import java.util.Iterator;
5  import java.util.LinkedHashMap;
6  import java.util.Map;
7  import java.util.Set;
8  import java.io.IOException;
9  import java.text.SimpleDateFormat;
10 import java.util.ArrayList;
11 import java.util.Date;
12 import java.util.GregorianCalendar;
13 import java.util.concurrent.TimeUnit;
14 import com.sun.corba.se.impl.orbutil.concurrent.Mutex;
15 import capturator.Capture;
16 import capturator.Connection;
17 import capturator.Packet;
18
19 public class ConnectionsManager extends Thread {
20   public static LinkedHashMap<String, Connection> conexoes = new
       LinkedHashMap<String, Connection>();
21   private ArrayList<Integer> MovingAverage = new ArrayList<Integer>()
       ;
22   public static boolean isrunning = true;
23   static Output saida = new Output();
24   public static Date last_timestamp = null;
25   private Mutex mutex_last_timestamp_var = new Mutex();
26   private Mutex mutex_connections = new Mutex();
27
28
29   public void updateMovingAverage(){
30     new Thread() {
```

```
31        public void run(){
32          this.setPriority(NORM_PRIORITY);
33          int length_packets, size_buffer;
34          while(isrunning){
35            length_packets = 0;
36            size_buffer = 0;
37            try {
38              Capture.mutex_packet_buffer.acquire();
39              length_packets=Capture.packet_buffer.size();
40              Capture.mutex_packet_buffer.release();
41              if(length_packets > 0){
42                if(MovingAverage.size() > Config.WINDOW_MOVING_AVERAGE)
43                  MovingAverage.remove(0);
44                MovingAverage.add(length_packets);
45
46                if(MovingAverage.size() >= Config.WINDOW_MOVING_AVERAGE
                      ){
47                  for (int i = 0; i < MovingAverage.size(); i++) {
48                    size_buffer += MovingAverage.get(i);
49                  }
50                  Config.TAM_PACKET_BUFFER = (int) Math.round((
                        size_buffer/MovingAverage.size()));
51                }
52              }
53              Thread.sleep(Config.INTERVAL_TO_ANALYSE_MOVING_AVERAGE);
54            } catch (InterruptedException e) {
55              System.out.println("Problem in UpdateMovingAverage thread
                    .");
56              System.out.println(e);
57            }
58          }
59        }
60      }.start();
61    }
```

```
62
63  public void startMonitoringThread(){
64    new Thread() {
65      public void run(){
66        this.setPriority(NORM_PRIORITY);
67        while(isrunning){
68          try {
69            verifyTimeouts();
70            System.gc();
71            Thread.sleep(20000);
72          } catch (InterruptedException e) {
73            System.out.println(e);
74          }
75        }
76      }
77    }.start();
78  }
79
80  public void startManagerThread(){
81    new Thread() {
82      public void run(){
83        Packet pkt = null;
84        this.setPriority(MAX_PRIORITY);
85        while(isrunning){
86          pkt = null;
87          try {
88            Capture.mutex_packet_buffer.acquire();
89            if(Capture.packet_buffer.isEmpty() == false){
90              pkt = Capture.packet_buffer.get(0);
91              Capture.packet_buffer.remove(0);
92            }
93            Capture.mutex_packet_buffer.release();
94          } catch (InterruptedException e1) {
```

```
95              System.out.println("Problema no acquire 1 funcao
                    managerthread");
96            }
97            if(pkt != null){
98              synchronized (pkt) {
99                try {
100                     mutex_last_timestamp_var.acquire();
101                     last_timestamp = pkt.getTimestamp();
102                     mutex_last_timestamp_var.release();
103                 manageConnection(pkt);
104               } catch (InterruptedException e) {
105                 e.printStackTrace();
106               }
107             }
108           }
109           try { Thread.sleep(1); } catch (InterruptedException e) {}
110         }
111       }
112     }.start();
113   }
114
115   public static void startOutputThread(final Connection conexao){
116     new Thread() {
117       public void run(){
118         this.setPriority(NORM_PRIORITY);
119         try {
120           try {
121             saida.outputTreatment(conexao);
122           } catch (InterruptedException e) {
123             e.printStackTrace();
124           }
125         } catch (IOException e) {
126           System.err.println("Erro ao despachar conexao %s" + conexao
                .getUnique_id());
```

```
127             e.printStackTrace();
128           }
129         }
130       }.start();
131     }
132
133     public static void kill(){
134       isrunning = false;
135     }
136
137     @SuppressWarnings({"rawtypes"})
138     public static void setTimeoutToALL() throws InterruptedException{
139         Iterator<?> it;
140         synchronized (conexoes) {
141           Set<?> set = conexoes.entrySet();
142           it = set.iterator();
143         }
144         Connection conexao_aux = null;
145         try{
146         while(it.hasNext()) {
147           try {
148             Map.Entry me = (Map.Entry)it.next();
149             synchronized (conexoes) {
150               conexao_aux = conexoes.get(me.getKey());
151             }
152             startOutputThread(conexao_aux);
153           } catch (Exception e) {
154             System.out.println("ERROR - while, setTimeoutToALL");
155           }
156
157         }
158         } catch (Exception e) {
159           System.out.println("Error in Function SetTimeoutToALL");
160         System.out.println(e);
```

```
161     }
162   }
163
164   @SuppressWarnings({ "unchecked", "rawtypes" })
165   private void verifyTimeouts() throws InterruptedException{
166       GregorianCalendar data = new GregorianCalendar();
167       mutex_last_timestamp_var.acquire();
168       if((Config.CURRENT_TIMESTAMP_TO_COMPARE == false) && (
              last_timestamp != null)){
169         data.setTime(last_timestamp);
170       }
171       mutex_last_timestamp_var.release();
172
173       long diferenca = 0;
174
175       LinkedHashMap<String, Connection> conexao_aux = new
              LinkedHashMap<String,Connection>();
176       Iterator<?> it;
177       synchronized (conexoes) {
178         conexao_aux = (LinkedHashMap<String, Connection>) conexoes.
              clone();
179         Set<?> set = conexao_aux.entrySet();
180         it = set.iterator();
181     }
182
183       try{
184       while(it.hasNext()) {
185         Map.Entry me = (Map.Entry)it.next();
186           if(conexao_aux.get(me.getKey()).getTimeouted() == false){
187             diferenca = TimeUnit.MILLISECONDS.toSeconds((data.
                  getTimeInMillis() - conexao_aux.get(me.getKey()).
                  getUltimoTS().getTime()));
188             if(diferenca > conexao_aux.get(me.getKey()).getTimeout())
                  {
```

```
189             conexao_aux.get(me.getKey()).setTimeouted(true);
190             /**
191              * Despacha a Conexao para ser tratada e enviada para as
                    tecnicas de I.A
192              */
193             Connection conexao = conexao_aux.get(me.getKey());
194             startOutputThread(conexao);
195             synchronized (conexoes) {
196               conexoes.remove(me.getKey());
197             }
198           }
199           }
200       }
201       } catch (Exception e) {
202         System.out.println("Error in Function VerifyTimeout");
203       System.out.println(e);
204     }
205   }
206
207   private void manageConnection(Packet pkt) throws
        InterruptedException{
208     String unique_id, reverse_unique_id;
209     boolean status_unique = false, status_reverse_unique = false;
210
211     try{
212       if(pkt.getProtocol().equals(Config.TAG_ICMP)){
213         unique_id = pkt.getProtocol() + "_" + pkt.getIPsrc()+"-"+pkt.
              getIPdst()+"_"+pkt.getIcmp_ID();
214         reverse_unique_id = pkt.getProtocol() + "_" + pkt.getIPdst()+
              "-"+pkt.getIPsrc()+"_"+pkt.getIcmp_ID();
215       } else if (pkt.getProtocol().equals(Config.TAG_ARP)) {
216         unique_id = pkt.getProtocol() + "_" + pkt.getIPsrc()+"-"+pkt.
              getIPdst();
```

```
217           reverse_unique_id = pkt.getProtocol() + "_" + pkt.getIPdst()+
                  "-"+pkt.getIPsrc();
218         } else {
219           unique_id = pkt.getProtocol() + "_" + pkt.getIPsrc()+":"+pkt.
                  getSport()+"-"+pkt.getIPdst()+":"+pkt.getDport();
220           reverse_unique_id = pkt.getProtocol() + "_" + pkt.getIPdst()+
                  ":"+pkt.getDport()+"-"+pkt.getIPsrc()+":"+pkt.getSport();
221         }
222
223         mutex_connections.acquire();
224         synchronized (conexoes) {
225           status_unique = conexoes.containsKey(unique_id);
226           status_reverse_unique = conexoes.containsKey(
                  reverse_unique_id);
227         }
228         mutex_connections.release();
229
230
231         if(status_unique){
232           addPktInConnection(pkt, unique_id);
233         } else if (status_reverse_unique){
234           addPktInConnection(pkt, reverse_unique_id);
235         } else {
236           mutex_connections.acquire();
237           createConnection(pkt, unique_id);
238           mutex_connections.release();
239         }
240         pkt = null;
241       } catch (Exception e) {
242         System.err.println("Some problem with the packet.");
243         System.out.println(e);
244       }
245
246     }
```

```
247
248   private Connection updateConnectionStatus(Connection conexao,
          Packet pkt){
249     /**
250      * Calcula o status e flags da conexao.
251      * O status serve para definir o timeout corretamente.
252      * As Flags servem para a classificacao nas tecnicas de I.C
253      *
254      * Based on:
255      * An Efficient TCP Flow State Management Algorithm in High-speed
             Network - Xiong Bing, Chen Xiaosu, Chen Ning
256      *
257      * Possible flags:
258      * S0 = Connection attempt seen, no reply.
259      * S1 = Connection established, not terminated.
260      * S2 = Connection established and close attempt by originator
           seen, but no reply from responder.
261      * S3 = Connection established and close attempt by responder
           seen, but no reply from originator
262      * SF = The connection was normally established and terminated.
263      * REJ = Connection attempt rejected.
264      * RSTO = Connection established, originator aborted by sending a
             RST.
265      * RSTR = Connection established, responder aborted by sending a
           RST.
266      * RSTOS0 = Originator sent a SYN followed by RST, we never saw a
             SYN ACK from the responder.
267      * RSTRH = Responder sent a SYN ACK flowed by a RST, we never saw
             a SYN from the originator.
268      * SH = Originator sent a SYN followed by a FIN, we never saw a
           SYN ACK from the responder.
269      * SHR = Responder sent a SYN ACK followed by a FIN, we never saw
             a SYN from the originator.
270      * OTH = Midstream traffic, we never saw a SYN.
```

```
271       */
272
273     if(conexao.getConnectionStatus().equals("Unknown")){
274       if(pkt.getSYN() == true || pkt.getSYNACK() == true){
275         conexao.setConnectionStatus("Handshake");
276         conexao.setFlagConexao("S0");
277       } else if (pkt.getACK() ==  true){
278         conexao.setConnectionStatus("Established");
279         conexao.setFlagConexao("S1");
280       } else if (pkt.getFIN() == true){
281         conexao.setConnectionStatus("Termination");
282         if(pkt.getDirection().equals(Config.TAG_SourceToDest)){
283           conexao.setFlagConexao("SH");
284         } else {
285           conexao.setFlagConexao("SHR");
286         }
287       } else {
288         conexao.setConnectionStatus("Other");
289         conexao.setFlagConexao("OTH");
290       }
291
292     } else if (conexao.getConnectionStatus().equals("Handshake")){
293       if(pkt.getRST()){
294         conexao.setConnectionStatus("Closed");
295         if(pkt.getDirection().equals(Config.TAG_SourceToDest)){
296           conexao.setFlagConexao("RSTOSO");
297         } else {
298           conexao.setFlagConexao("RSTRH");
299         }
300
301       } else if(pkt.getSYNACK()){
302         return(conexao);
303
304       } else if (pkt.getFIN()){
```

```
305           conexao.setConnectionStatus("Termination");
306           if(pkt.getDirection().equals(Config.TAG_SourceToDest)){
307             conexao.setFlagConexao("SH");
308           } else {
309             conexao.setFlagConexao("SHR");
310           }
311
312         } else if (pkt.getACK()){
313           conexao.setConnectionStatus("Established");
314           conexao.setFlagConexao("S1");
315
316         } else {
317           conexao.setConnectionStatus("Other");
318           conexao.setFlagConexao("OTH");
319         }
320
321       } else if (conexao.getConnectionStatus().equals("Established")){
322         if(pkt.getACK()){
323           return(conexao);
324         } else if(pkt.getFIN()){
325           conexao.setConnectionStatus("Termination");
326           if (pkt.getDirection().equals(Config.TAG_SourceToDest)){
327             conexao.setFlagConexao("S2");
328           } else {
329             conexao.setFlagConexao("S3");
330           }
331         } else if (pkt.getRST()){
332           conexao.setConnectionStatus("Closed");
333           if (pkt.getDirection().equals(Config.TAG_SourceToDest)){
334             conexao.setFlagConexao("RSTO");
335           } else {
336             conexao.setFlagConexao("RSTR");
337           }
338         }
```

```
339
340     } else if (conexao.getConnectionStatus().equals("Termination")){
341       if(pkt.getACK()){
342         conexao.setConnectionStatus("Closed");
343         conexao.setFlagConexao("SF");
344       } else {
345         return(conexao);
346       }
347
348     } else if (conexao.getConnectionStatus().equals("Closed")){
349       return(conexao);
350     }
351     return(conexao);
352   }
353
354
355   private synchronized void addPktInConnection(Packet pkt, String
          unique_id){
356     Connection conexao = null;
357     synchronized (conexoes) {
358       try {
359         conexao = conexoes.get(unique_id);
360       } catch (Exception e) {
361         return;
362       }
363     }
364     conexao.setUltimoTS(pkt.getTimestamp());
365     conexao.setVirtualTS(pkt.getTimestamp());
366     conexao.setDuration(TimeUnit.MILLISECONDS.toSeconds((conexao.
          getUltimoTS().getTime() - conexao.getPrimeiroTS().getTime())))
          ;
367
368     if(conexao.getProtocol().equals(Config.TAG_ICMP) == false){
369       conexao.addWrong(pkt.getWrong());
```

```
370         conexao.addUrgptr(pkt.getUrgptr());
371       }
372
373       /**
374        * Define a direcao do pacote.
375        */
376       if(pkt.getIPsrc().equals(conexao.getIPsource())){
377         pkt.setDirection(Config.TAG_SourceToDest);
378         if(pkt.getProtocol().equals(Config.TAG_ARP) == false){
379           conexao.setSTTL(pkt.getTTL());
380         }
381         conexao.addSourceToDestPkts();
382         conexao.addSizeFromSourceToDest(pkt.getLength());
383       } else {
384         pkt.setDirection(Config.TAG_DestToSource);
385         if(pkt.getProtocol().equals(Config.TAG_ARP) == false){
386           conexao.setDTTL(pkt.getTTL());
387         }
388         conexao.addDestToSourcePkts();
389         conexao.addSizeFromDestToSource(pkt.getLength());
390       }
391
392       conexao.addPacketCounter();
393
394       if(conexao.getProtocol().equals(Config.TAG_TCP)){
395         conexao = updateConnectionStatus(conexao, pkt);
396         if(conexao.getConnectionStatus().equals("Handshake")){
397           conexao.setTimeout(Config.TIMEOUT_TCP_HANDSHAKE);
398         } else if(conexao.getConnectionStatus().equals("Established")){
399           conexao.setTimeout(Config.TIMEOUT_TCP_ESTABLISHED);
400         } else if(conexao.getConnectionStatus().equals("Termination")){
401           conexao.setTimeout(Config.TIMEOUT_TCP_TERMINATION);
402         } else if(conexao.getConnectionStatus().equals("Closed")){
403           conexao.setTimeout(Config.TIMEOUT_TCP_CLOSED);
```

```
404         } else {
405           conexao.setTimeout(Config.TIMEOUT_OTHER);
406         }
407       } else if (conexao.getProtocol().equals(Config.TAG_UDP)){
408         conexao.setTimeout(Config.TIMEOUT_UDP);
409       } else if(conexao.getProtocol().equals(Config.TAG_ARP)){
410         conexao.setTimeout(Config.TIMEOUT_ARP);
411       } else {
412         conexao.setTimeout(Config.TIMEOUT_ICMP);
413       }
414
415       if(conexao.getDont_Analyse() == true){
416         conexao.setTimeout(Config.TIMEOUT_DONT_ANALYSE);
417       }
418
419       synchronized (conexoes) {
420         conexoes.remove(unique_id);
421         conexoes.put(unique_id, conexao);
422       }
423     }
424
425
426     @SuppressWarnings("unchecked")
427     private void createConnection(Packet pkt, String unique_id){
428       Connection conexao = new Connection();
429       conexao.setProtocol(pkt.getProtocol());
430       conexao.setUnique_id(unique_id);
431
432       if(conexao.getProtocol().equals(Config.TAG_TCP)){
433         conexao.setUrgptr(pkt.getUrgptr());
434         if(pkt.getSYNACK() == true){
435           conexao.setIPsource(pkt.getIPdst());
436           conexao.setSport(pkt.getDport());
437           conexao.setIPdestination(pkt.getIPsrc());
```

```
438             conexao.setDport(pkt.getSport());
439             conexao.setSizeFromDestToSource(pkt.getLength());
440             conexao.addDestToSourcePkts();
441             conexao.setSizeFromSourceToDest(0);
442             conexao.setDTTL(pkt.getTTL());
443             pkt.setDirection(Config.TAG_DestToSource);
444           } else {
445             conexao.setIPsource(pkt.getIPsrc());
446             conexao.setSport(pkt.getSport());
447             conexao.setIPdestination(pkt.getIPdst());
448             conexao.setDport(pkt.getDport());
449             conexao.setSizeFromDestToSource(0);
450             conexao.setSizeFromSourceToDest(pkt.getLength());
451             conexao.addSourceToDestPkts();
452             conexao.setSTTL(pkt.getTTL());
453             pkt.setDirection(Config.TAG_SourceToDest);
454           }
455         } else if (conexao.getProtocol().equals(Config.TAG_UDP) ||
              conexao.getProtocol().equals(Config.TAG_ICMP)){
456           conexao.setIPsource(pkt.getIPsrc());
457           conexao.setSport(pkt.getSport());
458           conexao.setIPdestination(pkt.getIPdst());
459           conexao.setDport(pkt.getDport());
460           conexao.setSizeFromDestToSource(0);
461           conexao.setSizeFromSourceToDest(pkt.getLength());
462           conexao.addSourceToDestPkts();
463           conexao.setSTTL(pkt.getTTL());
464           pkt.setDirection(Config.TAG_SourceToDest);
465         } else if (conexao.getProtocol().equals(Config.TAG_ARP)){
466           conexao.setIPsource(pkt.getIPsrc());
467           conexao.setIPdestination(pkt.getIPdst());
468           conexao.setSizeFromDestToSource(0);
469           conexao.setSizeFromSourceToDest(pkt.getLength());
470           conexao.addSourceToDestPkts();
```

```
471         pkt.setDirection(Config.TAG_SourceToDest);
472       }
473
474       if (pkt.getIPsrc() == pkt.getIPdst() && pkt.getSport() == pkt.
              getDport()){
475         conexao.setLand(1);
476         } else {
477           conexao.setLand(0);
478         }
479
480       conexao.setPrimeiroTS(pkt.getTimestamp());
481       conexao.setUltimoTS(pkt.getTimestamp());
482       conexao.setVirtualTS(pkt.getTimestamp());
483
484
485       conexao.setTimeouted(false);
486       conexao.addPacketCounter();
487       if(pkt.getProtocol().equals(Config.TAG_TCP)){
488         /**
489          * Foi definido como 20 segundos porque quando cria a conexao
                presume-se que esteja em handshake
490          */
491         conexao.setTimeout(Config.TIMEOUT_TCP_HANDSHAKE);
492         conexao = updateConnectionStatus(conexao, pkt);
493         conexao.setService(defineService(conexao.getDport(), conexao.
              getSport()));
494         conexao.setWrong(pkt.getWrong());
495       } else if (pkt.getProtocol().equals(Config.TAG_UDP)){
496         conexao.setFlagConexao(Config.FLAG_UDP);
497         conexao.setConnectionStatus("Closed");
498         conexao.setTimeout(Config.TIMEOUT_UDP);
499         conexao.setWrong(pkt.getWrong());
500         conexao.setService(defineService(conexao.getDport(), conexao.
              getSport()));
```

```
501     } else if (pkt.getProtocol().equals(Config.TAG_ARP)){
502       conexao.setTimeout(Config.TIMEOUT_ARP);
503       conexao.setService(Config.TAG_ARP);
504       conexao.setFlagConexao(Config.FLAG_ARP);
505     } else { //ICMP
506       conexao.setTimeout(Config.TIMEOUT_ICMP);
507       conexao.setService(Config.TAG_ICMP);
508       conexao.setFlagConexao(Config.FLAG_ICMP);
509     }
510
511     if(conexoes.size() < Config.TAM_BUFFER){
512       conexao.setDont_Analyse(true);
513       conexao.setTimeout(Config.TIMEOUT_DONT_ANALYSE);
514     }
515
516     //{TIMESTAMP -- PROTOCOL -- IP_SOURCE:SOURCE_PORT -- IP_DEST:
          DEST_PORT}
517     SimpleDateFormat format = new SimpleDateFormat("dd/MM/yyyy HH");
518     pkt.setVirtualTimestamp(format.format(pkt.getTimestamp()));
519     conexao.setVirtualUniqueID(pkt.getVirtualTimestamp()+"--"+conexao
          .getProtocol()+"--"+conexao.getIPsource()+":"+conexao.getSport
          ()+"--"+conexao.getIPdestination()+":"+conexao.getDport());
520     conexao.setReverseVirtualUniqueID(pkt.getVirtualTimestamp()+"--"+
          conexao.getProtocol()+"--"+conexao.getIPdestination()+":"+
          conexao.getDport()+"--"+conexao.getIPsource()+":"+conexao.
          getSport());
521
522     LinkedHashMap<String, Connection> conexoes_temp = null;
523     synchronized (conexoes) {
524       conexoes_temp = (LinkedHashMap<String, Connection>) conexoes.
            clone();
525     }
526     conexao.parm_time = Output.getTimeBuffer(conexao, conexoes_temp);
```

```
527     conexao.parm_connections = Output.getConnectionsBuffer(conexao,
            conexoes_temp);
528     conexoes_temp = null;
529     synchronized (conexoes) {
530       conexoes.put(unique_id , conexao);
531     }
532   }
533
534   private String defineService(int Dport, int Sport){
535     String service = "";
536     if(Config.services.containsKey(Dport) == true){
537       service = Config.services.get(Dport);
538     } else if(Config.services.containsKey(Sport) == true){
539       service = Config.services.get(Sport);
540     } else {
541       service = "OTHER";
542     }
543     return(service);
544   }
545 }
```

APÊNDICE D – Código-Fonte do Pré-processamento dos Vetores recebidos do Módulo de Monitoramento de *Timeouts*

```
1  package preprocessor;
2
3  import java.util.LinkedHashMap;
4  import java.util.Set;
5  import java.util.concurrent.TimeUnit;
6
7  import capturator.Connection;
8  import manager.ConnectionsManager;
9  import preprocessor.Config;
10 import java.io.FileWriter;
11 import java.io.IOException;
12 import java.io.PrintWriter;
13 import preprocessor.Communicator;
14
15
16 public class Output {
17
18   public synchronized static void writeInFile(String string) throws
         IOException {
19     FileWriter arquivo = new FileWriter(Config.PATH_TO_OUTPUT, true);
20     PrintWriter gravarArq = new PrintWriter(arquivo);
21     gravarArq.printf("%s\n",string);
22     arquivo.close();
23   }
24
25   public synchronized void outputTreatment(Connection conexao) throws
         IOException, InterruptedException{
26     if(conexao.getDont_Analyse() == false){
27       String parametros = "";
28 //       System.out.printf("SENT! %s -- %s:%d - %s:%d (%s)-- Status: %
     s  Flag: %s -- N. Pkts: %s -- ID: %s \n", conexao.getProtocol(),
     conexao.getIPsource(), conexao.getSport(), conexao.
```

```
    getIPdestination(), conexao.getDport(), conexao.getService(),
    conexao.getConnectionStatus(), conexao.getFlagConexao(), conexao.
    getNumOfPackets(), conexao.getUnique_id());
29
30
31      parametros =  Float.toString(conexao.getDuration())+Config.
           SEPARATOR+
32              conexao.getProtocol()+Config.SEPARATOR+
33              conexao.getService()+Config.SEPARATOR+
34              conexao.getFlagConexao()+Config.SEPARATOR+
35              Integer.toString(conexao.getSizeFromSourceToDest())+
                   Config.SEPARATOR+
36              Integer.toString(conexao.getSizeFromDestToSource())+
                   Config.SEPARATOR+
37              Integer.toString(conexao.getLand())+Config.SEPARATOR+
38              Integer.toString(conexao.getWrong())+Config.SEPARATOR+
39              Integer.toString(conexao.getUrgptr())+Config.SEPARATOR+
40              Integer.toString(conexao.getSTTL())+Config.SEPARATOR+
41              Integer.toString(conexao.getDTTL())+Config.SEPARATOR+
42              Integer.toString(conexao.getSourceToDestPkts())+Config.
                   SEPARATOR+
43              Integer.toString(conexao.getDestToSourcePkts())+Config.
                   SEPARATOR;
44
45      parametros = parametros + conexao.parm_time + conexao.
           parm_connections;
46
47      if(Config.INCLUDE_LABELS){
48        if(Config.INCLUDE_LABELS_FILE){
49          synchronized (Config.CONNECTIONS_LABELED) {
50            if(Config.CONNECTIONS_LABELED.containsKey(conexao.
                   getVirtualUniqueID()) == true){
51              if(Config.CONNECTIONS_LABELED.get(conexao.
                   getVirtualUniqueID()).equals("Normal")){
```

```
52                  parametros = parametros + Config.TAG_NORMAL+Config.
                        TERMINATOR;
53                } else {
54                  parametros = parametros + Config.TAG_ATTACK+Config.
                        TERMINATOR;
55                }
56                writeInFile(parametros);
57              } else if (Config.CONNECTIONS_LABELED.containsKey(conexao
                    .getReverseVirtualUniqueID()) == true){
58                 if(Config.CONNECTIONS_LABELED.get(conexao.
                       getReverseVirtualUniqueID()).equals("Normal")){
59                   parametros = parametros + Config.TAG_NORMAL+Config.
                        TERMINATOR;
60                 } else {
61                   parametros = parametros + Config.TAG_ATTACK+Config.
                        TERMINATOR;
62                 }
63                writeInFile(parametros);
64              } else {
65                System.out.println("Key NOT FOUND: "+ conexao.
                      getVirtualUniqueID());
66                parametros = parametros + Config.TAG_DEFAULT+Config.
                      TERMINATOR+"NOT_FOUND";
67              }
68            }
69          } else {
70            parametros = parametros + Config.TAG_DEFAULT+Config.
                  TERMINATOR;
71            writeInFile(parametros);
72          }
73
74        } else if(Config.ENABLE_SOCKET == true && Config.CAPTURE_ONLINE
              == true){
75          send_through_socket(parametros);
```

```
76          System.out.printf("SENT! %s -- %s:%d - %s:%d (%s)-- Status: %
                s  Flag: %s -- N. Pkts: %s -- ID: %s \n", conexao.
                getProtocol(), conexao.getIPsource(), conexao.getSport(),
                conexao.getIPdestination(), conexao.getDport(), conexao.
                getService(),conexao.getConnectionStatus(), conexao.
                getFlagConexao(), conexao.getNumOfPackets(), conexao.
                getUnique_id());
77        } else {
78          parametros = parametros + Config.TERMINATOR;
79        }
80      } else {
81  //        System.out.printf("DELETED! %s -- %s:%d - %s:%d (%s)-- Status
      : %s  Flag: %s -- N. Pkts: %s -- ID: %s \n", conexao.getProtocol()
      , conexao.getIPsource(), conexao.getSport(), conexao.
      getIPdestination(), conexao.getDport(), conexao.getService(),
      conexao.getConnectionStatus(), conexao.getFlagConexao(), conexao.
      getNumOfPackets(), conexao.getUnique_id());
82      }
83
84      synchronized (ConnectionsManager.conexoes) {
85        ConnectionsManager.conexoes.remove(conexao.getUnique_id());
86      }
87    }
88
89    private synchronized void send_through_socket(String parametros){
90      try {
91        Communicator.sendMessage(parametros);
92      } catch (Exception e) {
93        System.out.println("Socket Error " + e);
94      }
95    }
96
97    public synchronized static String getTimeBuffer(Connection conexao,
          LinkedHashMap<String, Connection> buffer){
```

```
98      String parameters = null;
99      long primeiro_ts = conexao.getPrimeiroTS().getTime();
100     int count_same_host = 0,
101       count_same_service = 0,
102       count_diff_service = 0,
103       count_same_host_syn_error = 0,
104       count_same_service_syn_error = 0;
105
106
107     if(buffer.isEmpty() == false){
108         Set<String> sorted_connection_keys = buffer.keySet();
109         long diferenca = primeiro_ts - TimeUnit.SECONDS.toMillis(
                Config.TIME);
110
111         for(int i=1; i<=buffer.size(); i++){
112           if (buffer.get(sorted_connection_keys.toArray()[buffer.size
                ()-i]).getPrimeiroTS().getTime() > diferenca){
113             /**
114              * Same host connections
115              */
116             if((buffer.get(sorted_connection_keys.toArray()[buffer.
                   size()-i]).getIPdestination().equals(conexao.
                   getIPdestination()) == true) ||
117                (buffer.get(sorted_connection_keys.toArray()[buffer.
                      size()-i]).getIPsource().equals(conexao.
                      getIPdestination()) == true)) {
118               count_same_host++;
119               /**
120                * serror_rate = % of connections that have ``SYN''
                     errors
121                */
122               if((buffer.get(sorted_connection_keys.toArray()[buffer.
                     size()-i]).getConnectionStatus() == "Handshake") ||
```

```
123                 (buffer.get(sorted_connection_keys.toArray()[buffer.
                        size()-i]).getSYN() > 2)){
124                 count_same_host_syn_error++;
125               }
126             }
127
128             /**
129              * Same Service Connections
130              */
131             if(buffer.get(sorted_connection_keys.toArray()[buffer.
                    size()-i]).getService().equals(conexao.getService())
                    == true ){
132               /**
133                * srv_count = number of connections to the same
                       service as the current connection in the past two
                       seconds
134                */
135               count_same_service++;
136
137               /**
138                * srv_serror_rate = % of connections that have ''SYN''
                        errors
139                */
140               if((buffer.get(sorted_connection_keys.toArray()[buffer.
                      size()-i]).getConnectionStatus() == "Handshake") ||
141                  (buffer.get(sorted_connection_keys.toArray()[buffer.
                        size()-i]).getSYN() > 2)){
142
143                 count_same_service_syn_error++;
144               }
145             } else {
146               count_diff_service++;
147             }
148           } else {
```

```
149             /**
150              * Iterator will never find more connections
151              */
152             break;
153           }
154         }
155       }
156
157
158     /**
159       count: continuous.
160       srv_count: continuous.
161       serror_rate: continuous.
162       srv_serror_rate: continuous.
163       same_srv_rate: continuous.
164       diff_srv_rate: continuous.
165       srv_diff_host_rate: continuous.
166      */
167
168     parameters =  Integer.toString(count_same_host)+Config.SEPARATOR+
169             Integer.toString(count_same_service)+Config.SEPARATOR+
170
171             //SYN ERROR
172             percentageCalculation(count_same_host,
                    count_same_host_syn_error)+Config.SEPARATOR+
173             percentageCalculation(count_same_service,
                    count_same_service_syn_error)+Config.SEPARATOR+
174
175             //Same Service Rate
176             percentageCalculation((count_same_service+
                    count_diff_service), count_same_service)+Config.
                    SEPARATOR+
```

```
177             percentageCalculation((count_same_service+
                    count_diff_service), count_diff_service)+Config.
                    SEPARATOR+
178
179             //Same Service Different Host Rate
180             percentageCalculation(count_same_service,
                    count_diff_service)+Config.SEPARATOR;
181     return(parameters);
182   }
183
184   private synchronized static String percentageCalculation(int count,
          int parm){
185     if(count>0){
186       return(Float.toString((100*parm)/count));
187     } else {
188       return("0");
189     }
190   }
191
192 //  public synchronized String getConnectionsBuffer(Conexao conexao,
      String parameters){
193   public synchronized static String getConnectionsBuffer(Connection
        conexao, LinkedHashMap<String, Connection> buffer){
194     /**
195      *  Some probing attacks scan the hosts (or ports) using a much
            larger time interval than two seconds,
196      *  for example once per minute.  Therefore, connection records
            were also sorted by destination host,
197      *  and features were constructed using a window of 100
            connections to the same host instead of a time window.
198      *    dst_host_count: continuous.
199         dst_host_srv_count: continuous.
200         dst_host_same_srv_rate: continuous.
201         dst_host_diff_srv_rate: continuous.
```

```
202         dst_host_same_src_port_rate: continuous.
203         dst_host_srv_diff_host_rate: continuous.
204         dst_host_serror_rate: continuous.
205         dst_host_srv_serror_rate: continuous.
206         dst_host_rerror_rate: continuous. -> Removed
207         dst_host_srv_rerror_rate: continuous. -> Removed
208      */
209
210
211     String parameters = null;
212     int count = 0,
213       serror_rate = 0,
214       same_srv_rate = 0,
215       diff_srv_rate = 0,
216
217       srv_count = 0,
218       srv_serror_rate = 0,
219       srv_diff_host_rate = 0,
220       same_src_port_rate = 0;
221
222     //Contador do tamanho do Buffer
223       int counter = (buffer.size() < Config.TAM_BUFFER) ? buffer.size
              () : Config.TAM_BUFFER;
224     if(buffer.isEmpty() == false){
225         for(int i=1; i<=counter; i++){
226
227           /**
228            * Same Destination
229            */
230           if(buffer.get(buffer.keySet().toArray()[buffer.size()-i]).
                 getIPdestination().equals(conexao.getIPdestination()) ==
                  true ||
```

```
231             buffer.get(buffer.keySet().toArray()[buffer.size()-i]).
                    getIPsource().equals(conexao.getIPdestination()) ==
                    true ){
232             count++;
233
234             /**
235              * serror_rate = % of connections that have ``SYN''
                    errors
236              */
237             if((buffer.get(buffer.keySet().toArray()[buffer.size()-i
                    ]).getConnectionStatus() == "Handshake") ||
238                (buffer.get(buffer.keySet().toArray()[buffer.size()-i
                       ]).getSYN() > 2)){
239             serror_rate++;
240             }
241
242             /**
243                * Same Service
244                */
245               if(buffer.get(buffer.keySet().toArray()[buffer.size()-i
                       ]).getService().equals(conexao.getService()) == true
                       ){
246                 same_srv_rate++;
247               } else {
248                 diff_srv_rate++;
249               }
250         }
251
252
253         /**
254          * Same Service
255          */
256         if(buffer.get(buffer.keySet().toArray()[buffer.size()-i]).
                getService().equals(conexao.getService()) == true ){
```

```
257             srv_count++;
258
259             /**
260              * serror_rate = % of connections that have ``SYN''
                    errors
261              */
262             if((buffer.get(buffer.keySet().toArray()[buffer.size()-i
                   ]).getConnectionStatus() == "Handshake") ||
263                (buffer.get(buffer.keySet().toArray()[buffer.size()-i
                      ]).getSYN() > 2)){
264             srv_serror_rate++;
265             }
266
267             /**
268              * Different hosts
269              */
270             if( (buffer.get(buffer.keySet().toArray()[buffer.size()-i
                   ]).getIPdestination().equals(conexao.getIPdestination
                   ()) == false) &&
271               (buffer.get(buffer.keySet().toArray()[buffer.size()-i])
                     .getIPsource().equals(conexao.getIPdestination()) ==
                      false)) {
272                srv_diff_host_rate++;
273               }
274           }
275
276           if(buffer.get(buffer.keySet().toArray()[buffer.size()-i]).
                   getSport() == conexao.getSport()){
277             same_src_port_rate++;
278           }
279         }
280       }
281
282     parameters =  //Counters
```

```
283             Integer.toString(count)+Config.SEPARATOR+
284             Integer.toString(srv_count)+Config.SEPARATOR+
285
286             //Same/Different Services Rate
287             percentageCalculation(count, same_srv_rate)+Config.
                    SEPARATOR+
288             percentageCalculation(count, diff_srv_rate)+Config.
                    SEPARATOR+
289
290             //Same source port
291             percentageCalculation(counter, same_src_port_rate)+Config
                    .SEPARATOR+
292
293             //Same Service Different Host
294             percentageCalculation(srv_count, srv_diff_host_rate)+
                    Config.SEPARATOR+
295
296             //Syn Errors
297             percentageCalculation(count, serror_rate)+Config.
                    SEPARATOR+
298             percentageCalculation(srv_count, srv_serror_rate)+Config.
                    SEPARATOR;
299
300     return(parameters);
301   }
302
303 }
```

APÊNDICE E – *Script* de Treinamento com o *Pybrain*

```
1  # encoding:utf-8
2
3  #Nao esquecer de importar as bibliotecas modificadas!
4  #local do pybrain
5  #/usr/local/lib/python2.7/dist-packages/PyBrain-0.3.3-py2.7.egg/
       pybrain/
6
7  import os, pickle
8  from pybrain.datasets import SupervisedDataSet
9  from pybrain.tools.shortcuts import buildNetwork
10 from pybrain.supervised import BackpropTrainer
11 from pybrain.tools.validation import ModuleValidator, Validator #Para
       o MSE
12 from pybrain.structure.modules import SigmoidLayer
13 import funcoes
14 import time
15 import datetime
16
17
18 ######################
19 # Programa principal
20 learningrate = 0.01;
21 momentum = 0.9;
22 verbose = True;
23 escreve_arquivo = True;
24
25 ts_inicial = datetime.datetime.now()
26
27 conjunto_de_dados = funcoes.preprocessamento();
28 conjunto_de_dados, dados_para_teste = conjunto_de_dados.
       splitWithProportion(proportion = 0.8);
29 print "Conjunto de dados: "+str(len(conjunto_de_dados))
30 print "Teste: "+str(len(dados_para_teste))
```

```
31
32
33 network = buildNetwork( conjunto_de_dados.indim,
34             20,
35             20,
36             conjunto_de_dados.outdim,
37             bias=True,
38             hiddenclass=SigmoidLayer,
39             outclass=SigmoidLayer);
40
41
42 trainer = funcoes.treina_rede(  network=network,
43         conjunto_de_dados=conjunto_de_dados,
44         learningrate =learningrate,
45         momentum = momentum,
46         verbose = verbose,
47         maxEpochs = 25,
48         continueEpochs= 10,
49         escreve_arquivo = escreve_arquivo,
50         dados_teste = dados_para_teste);
51
52 ts_final = datetime.datetime.now()
53
54 print "Duracao: " + str(ts_final-ts_inicial)
55 print("Total de Epocas: %d" % trainer.totalepochs);
```

APÊNDICE F – Conjunto de Funções utilizadas no treinamento com o *Pybrain*

```
1
2  # encoding:utf-8
3  from pybrain.datasets import SupervisedDataSet, UnsupervisedDataSet
4  from pybrain.tools.shortcuts import buildNetwork
5  from pybrain.supervised import BackpropTrainer
6  from pybrain.tools.shortcuts import buildNetwork
7  from pybrain.datasets import ClassificationDataSet
8  import pickle
9  import sys, os, os.path, tty, termios
10 from pybrain.tools.validation import *
11 from pybrain.structure.modules import *
12 from pylab import *
13 from matplotlib import pyplot
14 from pybrain.tools.neuralnets import NNregression
15 from pybrain.tools.customxml.networkreader import NetworkReader
16 from pybrain.tools.customxml.networkwriter import NetworkWriter
17 import list_parameters as funcoes_extras
18
19
20 base_de_entrada = "bases/basecompleta.txt";
21 nome_do_arquivo = "treinamento_rede_neural";
22 nome_arquivo_grafico = "grafico/dados_grafico";
23 num_colunas_arquivo = 29;
24 nome_arquivo_teste = "baseteste";
25
26 def preprocessamento(arquivo = base_de_entrada):
27   conjunto_de_dados = SupervisedDataSet(96,1)
28
29   f = open(arquivo,'r');
30   for line in f.xreadlines():
31     dados_novos = None;
32     dados_novos = [];
```

```
33      dados =  line.split(",", num_colunas_arquivo ); #Separa as linhas
            por virgula.
34      cont = 0;
35      for dado in dados:
36        if (cont == 1): #Ajusta protocolo da camada de transporte.
37          dados_novos = checkvetor(dados_novos, dado, funcoes_extras.
                protocolos())
38        elif (cont == 2):  #Ajusta servicos
39          dados_novos = checkvetor(dados_novos, dado, funcoes_extras.
                servicos())
40        elif (cont == 3):  #Ajusta flags.
41          dados_novos = checkvetor(dados_novos, dado, funcoes_extras.
                flags())
42        elif ((cont == num_colunas_arquivo-1)):
43          if (dado == "normal;\n"):
44            result = 0.1
45          else:
46            result = 0.9
47          conjunto_de_dados.addSample(dados_novos, [result])
48          break
49        else:
50          dados_novos.append(dado)
51        cont = cont + 1
52    f.close()
53    return conjunto_de_dados
54
55  def checkvetor(dados, dado_atual, vetor):
56    for dado in vetor:
57      if(dado.upper() == dado_atual.upper()):
58        dados.append(1)
59      else:
60        dados.append(0)
61    return dados
62
```

```
63 def treina_rede(network, conjunto_de_dados, learningrate=0.01,
       momentum=0.99, verbose = True, maxEpochs = 100, continueEpochs =
       10, escreve_arquivo = True, dados_teste = False):
64   if(verbose == True):
65     print "****************************************************";
66     print "          Iniciando o treinamento..."
67     print "****************************************************";
68     print "\n";
69
70   trainer = BackpropTrainer(network, conjunto_de_dados, learningrate=
         learningrate, momentum=momentum, verbose=verbose, weightdecay
         =0.001);
71
72   treinamento = trainer.trainUntilConvergence(dataset=
         conjunto_de_dados,
73                         maxEpochs=maxEpochs,
74                         continueEpochs = continueEpochs,
75                         verbose=True,
76                         validationProportion=0.1, # Early-stopping
77                         return_all_errors = True);
78
79   if(dados_teste != False):
80     #Grava os dados de teste para testar depois.
81     os.path.isfile(nome_arquivo_teste);
82     try:
83        with open(nome_arquivo_teste, 'r') as f:
84            os.remove(nome_arquivo_teste);
85     except IOError:
86       pass
87     fileObject = open(nome_arquivo_teste, 'w');
88     pickle.dump(dados_teste, fileObject);
89     fileObject.close();
90
91   if (escreve_arquivo == True):
```

```
92      os.path.isfile(nome_do_arquivo);
93      try:
94          with open(nome_do_arquivo, 'r') as f:
95              os.remove(nome_do_arquivo);
96      except IOError:
97        pass
98
99      #Salva a rede treinada.
100     NetworkWriter.writeToFile(trainer.module, nome_do_arquivo);
101
102     os.path.isfile(nome_arquivo_grafico);
103     try:
104         with open(nome_arquivo_grafico, 'r') as f:
105             os.remove(nome_arquivo_grafico);
106     except IOError:
107       pass
108     fileObject = open(nome_arquivo_grafico, 'w');
109     pickle.dump(treinamento, fileObject);
110     fileObject.close();
111
112   return trainer;
```

APÊNDICE G – Conjunto de Protocolos e Serviços para o Pré-processamento

```
1  # encoding:utf-8
2
3  def protocolos():
4    protocolo = []
5    protocolo.append("tcp")
6    protocolo.append("udp")
7    protocolo.append("icmp")
8    protocolo.append("unknown")
9    return protocolo
10
11
12 def servicos():
13   servico = []
14   servico.append("AUTH")
15   servico.append("BGP")
16   servico.append("CSNET_NS")
17   servico.append("CTF")
18   servico.append("DAYTIME")
19   servico.append("DISCARD")
20   servico.append("DNS")
21   servico.append("ECHO")
22   servico.append("EFS")
23   servico.append("EXEC")
24   servico.append("FINGER")
25   servico.append("FTP")
26   servico.append("FTP-DATA")
27   servico.append("GOPHER")
28   servico.append("HTTP")
29   servico.append("HTTPS")
30   servico.append("ICMP")
31   servico.append("IMAP4")
32   servico.append("IRC")
33   servico.append("ISO_TSAP")
```

```
34    servico.append("KLOGIN")
35    servico.append("LDAP")
36    servico.append("LINK")
37    servico.append("LLMNR")
38    servico.append("LOGIN")
39    servico.append("NETBIOS_DGM")
40    servico.append("NETBIOS_NS")
41    servico.append("NETBIOS_SSN")
42    servico.append("NETSTAT")
43    servico.append("NNTP")
44    servico.append("NTP_U")
45    servico.append("OTHER")
46    servico.append("POP2")
47    servico.append("POP3")
48    servico.append("PRINTER")
49    servico.append("PRIVATE")
50    servico.append("REMOTE_JOB")
51    servico.append("SMTP")
52    servico.append("SPOTIFY")
53    servico.append("SQL_NET")
54    servico.append("SSDP")
55    servico.append("SSH")
56    servico.append("SUNRPC")
57    servico.append("TELNET")
58    servico.append("TFTP_U")
59    servico.append("TIME")
60    servico.append("URP_I")
61    servico.append("UUCP")
62    servico.append("UUCP_PATH")
63    servico.append("VNC")
64    servico.append("VOIP")
65    servico.append("WHOIS")
66    servico.append("X11")
67    servico.append("Z39_50")
```

```
68    return servico
69
70
71  def flags():
72    flag = []
73    flag.append("SF")
74    flag.append("S1")
75    flag.append("REJ")
76    flag.append("S2")
77    flag.append("S0")
78    flag.append("S3")
79    flag.append("RSTO")
80    flag.append("RSTR")
81    flag.append("RSTOS0")
82    flag.append("OTH")
83    flag.append("SH")
84    flag.append("Unknown")
85    flag.append("ICMP")
86    return flag
```

APÊNDICE H – Módulo de Classificação Utilizando Redes Neurais Artificiais

```
# encoding:utf-8

#Socket server with trained Neural Network.

import socket
import thread
import funcoes_server as functions
from pybrain.tools.customxml.networkreader import NetworkReader

def connected(con, cliente, network):
    while True:
    msg = con.recv(2048)
    if not msg:
      break
    elif len(msg) >= 3:
      dado = functions.preprocessamento(dados = msg)
      result = network.activate(dado)
      if result > 0.5:
        print "****ATTACK****"
        print "Details: " + str(dado)
        print "****"
        #TODO: Adicionar notificacao por email, sms [...]
      else:
        print "Normal"
    con.close()
    thread.exit()


def main():
  network = NetworkReader.readFrom("treinamento_rede_neural")
  HOST = ''
  PORT = 59001
  tcp = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
34    orig = (HOST, PORT)
35    tcp.bind(orig)
36    tcp.listen(1)
37    print "Listening on port %s" % PORT
38    while True:
39        con, cliente = tcp.accept()
40        thread.start_new_thread(connected, tuple([con, cliente, network
              ]))
41
42
43 main()
```

APÊNDICE I – *Script* de Treinamento com o Scikit-Learn

```
1  from sklearn import datasets
2  from sklearn.datasets import make_blobs
3  from sklearn.ensemble import RandomForestClassifier
4  from numpy import random
5  import numpy as np
6  import cPickle
7  import sys
8  import datetime
9  import matplotlib.pyplot as plt
10
11
12 def splitWithProportion(dataset, answer_dataset, proportion = 0.8):
13   """Produce two new datasets, the first one containing the fraction
       given
14   by `proportion` of the samples."""
15   indicies = random.permutation(len(dataset))
16   separator = int(len(dataset) * proportion)
17
18   leftIndicies = indicies[:separator]
19   rightIndicies = indicies[separator:]
20
21   train_ds = []
22   train_ds_as = []
23   test_ds = []
24   test_ds_as = []
25
26   for x in leftIndicies:
27     train_ds.append(dataset[x])
28     train_ds_as.append(answer_dataset[x])
29   for x in rightIndicies:
30     test_ds.append(dataset[x])
31     test_ds_as.append(answer_dataset[x])
32
```

```
33    return train_ds, train_ds_as, test_ds, test_ds_as
34
35
36
37  def addSampleToTrain(arquivo):
38    # Apenas se o preprocessamento ja foi realizado
39    tam_vetor = 96
40    dados_treinamento = []
41    dados_resposta = []
42    f = open(arquivo,'r');
43    for line in f.xreadlines():
44      dados_novos = None;
45      dados_novos = [];
46      dados =  line.split(",", tam_vetor); #Separa as linhas por
          virgula.
47      cont = 0;
48      for dado in dados:
49        if ((cont == tam_vetor)):
50          if(dado == "0.1;\n"):
51            dado = 0
52          else:
53            dado = 1
54          dados_treinamento.append(dados_novos)
55          dados_resposta.append(dado)
56          break
57        else:
58          dados_novos.append(dado)
59        cont = cont + 1
60    f.close()
61    return dados_treinamento, dados_resposta
62
63
64  def plotImportanceGraphic(rfc):
65    importances = rfc.feature_importances_
```

```
66    std = np.std([tree.feature_importances_ for tree in rfc.estimators_
          ],
67                 axis=0)
68    indices = np.argsort(importances)[::-1]
69    for f in xrange(0,96):
70      print("%d. feature %d (%f)" % (f + 1, indices[f], importances[
            indices[f]]))
71
72
73    # Plot the feature importances of the forest
74    # plt.figure()
75    # plt.title("Feature importances")
76    # plt.bar(range(0,96), importances[indices], color="r", yerr=std[
        indices], align="center")
77    # plt.show()
78
79  def main():
80    flag = 0
81    if(len(sys.argv) == 3):
82      try:
83        X_train = cPickle.load(open(sys.argv[1], "rb"))
84        y_train = cPickle.load(open(sys.argv[2], "rb"))
85      except Exception, e:
86        flag = 1
87    elif(len(sys.argv) == 2):
88      X_train, y_train = addSampleToTrain(sys.argv[1])
89    else:
90      print "************************"
91      print("ERROR! Usage:\npython train_rf.py [File_to_train]\nor \
            npython train_rf.py [preprocessed_file_to_train_x] [
            preprocessed_file_to_train_y]\n")
92      print "************************"
93    flag = 1
94
```

```
95  if(flag == 0):
96    X_train, y_train, X_test, y_test = splitWithProportion(X_train,
          y_train, proportion = 0.8)
97    print "Saving training and testing files..."
98
99    print "Training..."
100
101   # Train uncalibrated random forest classifier on whole train and
          validation
102   # data and evaluate on test data
103   rfc = RandomForestClassifier(n_estimators=1000,
104                                max_depth = 3,
105                                min_samples_leaf=5,
106        min_samples_split = 2,
107                                max_features = 25,
108                                verbose = 1,
109                                n_jobs = -1)
110
111   rfc.fit(X_train, y_train)
112
113   print "Saving Training..."
114
115   with open('treinamento_rf', 'wb') as f:
116     cPickle.dump(rfc, f)
117
118   print "Testing..."
119
120   print "RFC: " + str(rfc)
121   predicted = rfc.predict(X_test)
122
123   i = 0
124   acertos = 0
125   erros = 0
126   vn = 0
```

```
127     vp = 0
128     fp = 0
129     fn = 0
130     for result in y_test:
131       if(predicted[i] == result):
132         acertos += 1
133         if(result == 0):
134           vn += 1
135         else:
136           vp += 1
137       else:
138         erros += 1
139         if(predicted[i] == 0):
140           fn += 1
141         else:
142           fp += 1
143       i += 1
144
145     print "\n\n********************"
146     print "VERDADEIRO POSITIVO: " + str(vp)
147     print "VERDADEIRO NEGATIVO: " + str(vn)
148     print "FALSO POSITIVO: " + str(fp)
149     print "FALSO NEGATIVO: " + str(fn)
150     print "********************"
151
152
153     print "Acertos: " + str(acertos) + ", Erros: " + str(erros)
154     percentage = (float(acertos) * 100)/(float(acertos)+float(erros))
155     print "Eficacia: % .6f" % percentage,"%"
156
157     plotImportanceGraphic(rfc)
158
159
160 main()
```

APÊNDICE J – Módulo de Classificação Utilizando Florestas Aleatórias

```
from sklearn import datasets
from sklearn.datasets import make_blobs
from sklearn.ensemble import RandomForestClassifier
import cPickle
import sys
import datetime

def addSample(arquivo):
  # Apenas se o preprocessamento ja foi realizado
  tam_vetor = 96
  dados_treinamento = []
  dados_resposta = []
  f = open(arquivo,'r');
  for line in f.xreadlines():
    dados_novos = None;
    dados_novos = [];
    dados =  line.split(",", tam_vetor); #Separa as linhas por
       virgula.
    cont = 0;
    for dado in dados:
      if ((cont == tam_vetor)):
        if(dado == "0.1;\n"):
          dado = 0
        else:
          dado = 1
        dados_treinamento.append(dados_novos)
        dados_resposta.append(dado)
        break
      else:
        dados_novos.append(dado)
      cont = cont + 1
  f.close()
  return dados_treinamento, dados_resposta
```

```
33
34 def main():
35   flag = 0
36   if(len(sys.argv) == 3):
37     try:
38       X_test = cPickle.load(open(sys.argv[1], "rb"))
39       y_test = cPickle.load(open(sys.argv[2], "rb"))
40     except Exception, e:
41       flag = 1
42   elif(len(sys.argv) == 2):
43     try:
44       X_test, y_test = addSample(sys.argv[1])
45     except Exception, e:
46       flag = 1
47   else:
48     print "*****************************************"
49     print("ERROR! Usage:\npython test.py [File_to_xtest] [
         File_to_ytest]\n")
50     print("OR\n")
51     print("Usage:\npython test.py [File_to_test] \n")
52     print "*****************************************"
53   flag = 1
54
55   if(flag == 0):
56   rfc = cPickle.load(open('treinamento_rf', "rb"))
57
58   print "RFC: " + str(rfc)
59   predicted = rfc.predict(X_test)
60
61   i = 0
62   acertos = 0
63   erros = 0
64   vn = 0
65   vp = 0
```

```
66    fp = 0
67    fn = 0
68    for result in y_test:
69      print "Result: " + str(predicted[i]) + " ----> " + str(result)
70      if(predicted[i] == result):
71        acertos += 1
72        if(result == 0):
73          vn += 1
74        else:
75          vp += 1
76      else:
77        erros += 1
78        if(predicted[i] == 0):
79          fn += 1
80        else:
81          fp += 1
82      i += 1
83
84    print "\n\n*********************************"
85    print "VERDADEIRO POSITIVO: " + str(vp)
86    print "VERDADEIRO NEGATIVO: " + str(vn)
87    print "FALSO POSITIVO: " + str(fp)
88    print "FALSO NEGATIVO: " + str(fn)
89    print "*********************************"
90
91    print "Acertos: " + str(acertos) + ", Erros: " + str(erros)
92    percentage = (float(acertos) * 100)/(float(acertos)+float(erros))
93    print "Eficacia: % .6f" % percentage,"%"
94
95  main()
```