Sistemas de Deteção de Intrusões usando Análise de Padrões do Tráfego de Rede

Versão Final

Por

Luís Filipe Bento Morais

Orientador: Doutor Pedro Miguel Mestre Alves da Silva

Co-orientador: Doutor Carlos Manuel José Alves Serôdio

Dissertação submetida à
UNIVERSIDADE DE TRÁS-OS-MONTES E ALTO DOURO
para obtenção do grau de
MESTRE
em Engenharia Eletrotécnica e de Computadores, de acordo com o disposto no
Regulamento Geral dos Ciclos de Estudo Conducentes ao Grau de Mestre na UTAD
DR, 2.ª série – N.º 133 – Regulamento n.º 658/2016 de 13 de julho de 2016

Sistemas de Deteção de Intrusões usando Análise de Padrões do Tráfego de Rede

Versão Final

Por

Luís Filipe Bento Morais

Orientador: Doutor Pedro Miguel Mestre Alves da Silva

Co-orientador: Doutor Carlos Manuel José Alves Serôdio

Dissertação submetida à
UNIVERSIDADE DE TRÁS-OS-MONTES E ALTO DOURO
para obtenção do grau de
MESTRE
em Engenharia Eletrotécnica e de Computadores, de acordo com o disposto no
Regulamento Geral dos Ciclos de Estudo Conducentes ao Grau de Mestre na UTAD
DR, 2.ª série – N.º 133 – Regulamento n.º 658/2016 de 13 de julho de 2016

Orientação Científica :

Doutor Pedro Miguel Mestre Alves da Silva

Professor Auxiliar do
Departamento de Engenharias da Escola de Ciências e Tecnologia da
Universidade de Trás-os-Montes e Alto Douro

Doutor Carlos Manuel José Alves Serôdio

Professor Associado c/Agregação do
Departamento de Engenharias da Escola de Ciências e Tecnologia da
Universidade de Trás-os-Montes e Alto Douro

"Activity is the only road to knowledge"

George Bernard Shaw (1856 – 1950)

A quem dedico, ***Aos meus pais***

Sistemas de Deteção de Intrusões usando Análise de Padrões do Tráfego de Rede

Luís Filipe Bento Morais

Submetido na Universidade de Trás-os-Montes e Alto Douro
para o preenchimento dos requisitos parciais para obtenção do grau de
Mestre em Engenharia Eletrotécnica e de Computadores

Resumo —Com a evolução da humanidade apareceram cada vez formas mais rápidas de comunicar até que foi criada a *Internet* na sua versão primordial datada dos anos 70. Começaram a ser criados protocolos para garantir uma comunicação mais rápida e fiável e o número de utilizadores foi sempre crescendo ao longo dos anos. Tendo em conta o crescimento da rede começaram também os problemas de acesso não autorizado aos equipamentos de terceiros. Iniciou-se então a procura pela segurança e evitar ao máximo que informação privada, pessoal ou empresarial seja exposta em público. Com as crescentes exigências do mundo empresarial as *firewalls* têm um papel preponderante na segurança e regras de acesso a portas estão a entrar em desuso passando a *firewalls* inteligentes que reconhecem as ameaças por exemplo pelo conteúdo dos pacotes, isto é, quando um ficheiro é descarregado pela primeira vez é também descarregado para uma máquina virtual e sujeito a vários testes para verificar se é maligno, caso seja maligno a base de dados é atualizada impedindo que outros sistemas sejam afetados por este ficheiro. Além de firewalls há outros tipos de software que controlam o número de pacotes na rede, e caso seja anormal para esses algoritmos devido a aprenderem quais os padrões de comunicação normais e quais os potencialmente perigosos, conseguindo bloquear e/ou alertar o responsável. Esta dissertação tem como objetivo estudar metodologias e algoritmos que permitam avaliar se uma ligação é segura ou não tomando em conta alguns parâmetros recolhidos sobre a ligação.

Palavras Chave: Ataques informáticos, Ameaças, Segurança Informática, *Internet*, Informação confidencial.

Intrusion Detection System using Patern Analysis of Network Trafic

Luís Filipe Bento Morais

Submitted to the University of Trás-os-Montes and Alto Douro
in partial fulfillment of the requirements for the degree of
Master of Science in Electrical Engineering and Computers

Abstract —With the evolution of humanity appeared faster forms of communicating until the creation of Internet, in the primordial version dated on 70's. Protocols creation started to ensure faster and reliable communication . The number of users starts to grow, and keeps growing over the years. With the growth of the network also starts the problems with unauthorized access to the third party systems, so started the search for security, avoiding that private information, personal or business leaks to public. With the growing requirements of the business world, firewalls have a leading role in security, rules of access to ports are going into disuse giving space to intelligent firewalls that recognize threats for the content, e.g, when a file is downloaded for the first time it is also downloaded to a virtual machine and tested to clear if it is trustable, or not trustable, then the database will be updated preventing other systems to be infected by this file. Besides firewalls there are another kinds of software that control the number of packets in the network, and if the number is unusual for this algorithms due to learn which patterns of communication are usual and which are the dangerous ones, with this they can block/alert the person in charge. This dissertation has the objective of study working methods and algorithms that allows measuring if the connection is safe or unsafe taking in consideration some parameters gathered about the connection.

Keywords: Cyber attacks, Threats, IT security, Internet, Private Information.

Agradecimentos

Em primeiro lugar quero agradecer aos meus orientadores, Professor Doutor Pedro Miguel Mestre Alves da Silva do Departamento de Engenharia da Escola de Ciência e Tecnologia da Universidade de Trás os Montes e Alto Douro e Professor Carlos Manuel José Serôdio do Departamento de Engenharia da Escola de Ciência e Tecnologia da Universidade de Trás os Montes e Alto Douro. A sua disponibilidade e orientação ao longo de todo o percurso académico e sobretudo durante o desenvolvimento desta dissertação, todo conhecimento foi transmitido e terá impacto também no futuro profissional.

Um especial agradecimento ao Professor Doutor Paulo Alexandre Cardoso Salgado que apesar de não ser meu orientador demonstrou sempre muita disponibilidade para ajudar nesta dissertação, ajudandando no algoritmo Modelo de Markov.

Quero agradecer aos meus pais, por todo o apoio durante o meu percurso académico pois sem eles nada disto seria possível, estando sempre ao meu lado apoiando-me e depositando em mim toda a confiança para que chegasse a onde cheguei.

Quero também agradecer a todos os meus amigos que sempre me ajudaram e ultrapassaram comigo todas as adversidades destes últimos anos, sem vocês nunca teria chegado a este ponto da minha vida, um obrigado por tudo.

UTAD Luís Morais

Vila Real, Outubro 2019

Índice geral

Resumo **ix**

Abstract **xi**

Agradecimentos **xiii**

Índice de tabelas **xvii**

Índice de figuras **xix**

Glossário, acrónimos e abreviaturas **xxi**

1 Introdução **1**
1.1 Contexto 1
1.2 Motivação e Objetivos 2
1.3 Estrutura do Documento 3

2 Enquadramento Tecnológico **5**
2.1 *Internet e Firewalls* 5
2.1.1 Importância das *Firewalls* no Crescimento da *Internet* 9
2.1.2 Firewalls 10
2.1.3 *Next Generation Firewalls* 12
2.2 Segurança de Informação e Tipos de Ataque 13
2.2.1 Classificação de Tipo de Ataques 14
2.2.2 *Packet Sniffer* 17

2.2.3 *3-Way HandShake* 20
2.3 Redes Neuronais 23
2.3.1 Funções de Activação 26
2.3.2 Long Short-Term Memory 27
2.3.3 Função de Custo 30
2.3.4 Tensorflow 32
2.3.5 Modelo de Markov 33
2.3.6 Trabalhos Relacionados 35

3 Conceção e Implementação **39**
3.1 Recolha de Dados 39
3.2 Desenvolvimento do Programa para Tratamento de Dados 42
3.2.1 Estudo dos Algortimos 44
3.3 Cadeia de Markov 47

4 Testes e Resultados **51**
4.1 Resultados dos Testes 51
4.1.1 Tensorflow 51
4.1.2 Modelo de Markov 58

5 Conclusões e Trabalho Futuro **65**
5.1 Conclusões 65
5.2 Trabalho Futuro 66

A Anexos **69**

Referências Bibliográficas **75**

Índice de tabelas

2.1 Exemplo de portas e protocolos de camada 7 9

4.1 Resultados dos 50 testes em condições ideais em%. 53
4.2 Resultados dos 50 testes em % com linhas de (0)-Solução B. 55

A.1 Resultados dos 50 testes em condições ideais referentes à 4.1 71
A.2 Resultados dos 50 testes da Solução B referentes à 4.2 73

Índice de figuras

2.1 Modelo OSI 6
2.2 Modelo TCP/IP 7
2.3 Número de hosts desde 1993[1] 10
2.4 Exemplo de inspeção proxy 12
2.5 Exemplo de pacote TCP/IP 18
2.6 *3-Way HandShake* 21
2.7 Exemplo de finalização de ligação 22
2.8 Máquina de estados *flags TCP* 22
2.9 Neurónio Simples [2] 24
2.10 Exemplo de rede neuronal 25
2.11 Exemplo de diagrama *LSTM* 27
2.12 Unidade *LSTM* 29
2.13 Hierarquia *Tensorflow* 32
2.14 Exemplo Cadeia de transição 34

3.1 Modo de recolha de pacotes. 40
3.2 Exemplo de Tráfego recolhido pelo *TCPDump*. 42
3.3 Exemplo de Tráfego filtrado. 43

3.4 Exemplo da procura de *flags* usando a Figura 3.3. 43
3.5 Exemplo de Dados de entrada para a rede neuronal. 45
3.6 Exemplo da Solução A. 46
3.7 Exemplo da Solução B. 47
3.8 Exemplo de grafo. 48

4.1 Output dos dados de tentativa de intrusão em situação ideal. 53
4.2 Output do dados de tentativa de intrusão-Solução A. 55
4.3 Evolução da função de custo ao longo do treino. 56
4.4 *Output* em dados de tentativa de intrusão-Solução B. 57
4.5 Evolução da Função de Custo. 57
4.7 Matriz transição original. 59
4.6 Grafo da matriz de transição original. 59
4.8 Exemplo de dados de ligação normal. 60
4.9 Exemplo de dados de tentativa de intrusão. 60
4.10 Matriz de transição após teste (ligação normal). 60
4.11 Matriz de transição após teste (tentativa de intrusão). 61
4.12 Exemplo de dados em teste. 61
4.13 Matriz de transição número 2. 62
4.14 Matriz de transição número 1. 62

Glossário, acrónimos e abreviaturas

Lista de acrónimos

Sigla	**Expansão**
ACK	*Acknowledge*
AP	*Access Point*
API	*Application Programming Interface*
DDoS	*Destributed Denial of Service*
DNS	*Domain Name System*
DOS	*Denial of Service*
FIN	*Finish*
ICMP	*Internet Control Message Protocol*
IDS	*Intrusion Detection System*
IP	*Internet Protocol*
IPv4	*Internet Protocol version4*
IPv6	*Internet Protocol version6*
ISP	*Internet service provider*

Sigla	**Expansão**
LAN	*Local Area Network*
LSTM	*Long Short-Term Memory*
MAC	*Medium Access Control*
MLP	*MUltilayer Perceptron*
OSI	*Open System Interconection*
OSPF	*Open Shortest Path First*
PC	*Personal Computer*
QoS	*Quality of Service*
ReLU	*Rectified Linear Unit*
RNN	*Recurrent Neural Network*
RST	*Reset*
SNMP	*Simple Network Management Protocol*
TCP	*Transmission Control Protocol*
TCP/IP	*Transmission Control Protocol/Internet Protocol*
UDP	*User Datagram Protocol*
URL	*Uniform Resource Locator*
UTAD	Universidade de Trás-os-Montes e Alto Douro
VLAN	*Virtual Local Area Network*

Lista de abreviaturas

Abreviatura	**Significado(s)**
e.g.	por exemplo
i.e.	isto é, por conseguinte

Introdução

1.1 Contexto

O aparecimento dos computadores veio a demonstrar ser uma evolução enorme para a humanidade, a capacidade de processamento destes foi crescendo e é impensável hoje em dia viver num mundo em que estes não estejam presentes. Mais tarde apareceu a *Internet* e com ela a necessidade de comunicar à distância com a maior rapidez possível, com o uso quase ubíquo do *Wi-Fi* ficou virtualmente presente em todo o lado.

Se praticamente todos os seres humanos estão ligados através de um ou vários dispositivos à mesma "teia", então, esta "teia" tem de estar o melhor protegida possível para evitar que sejam furtados dados pessoais ou, pior que isso, este serviço se degrade ou seja destruído. Para evitar que acedam aos sistemas informáticos remotamente foram criadas *firewalls* que impedem que utilizadores externos sem permissões acedam a dados privados e façam uso de recursos físicos, e.g., *botnet*, ou recursos pessoais do utilizador e.g., fotos, documentos. Atualmente, a tecnologia evolui a uma velocidade nunca antes vista, todos os dias há atualizações quer seja em *hardware* ou *software*, o que ontem era topo de gama amanhã pode já ter sido

destronado.

Para o *hardware* estar protegido o utilizador tem de evitar ao máximo o acesso aos *sites* não-fidedignos/suspeitos, abrir *e-mails* de assunto suspeito por exemplo, mas nem sempre é possível evitar estas práticas ou por desconhecimento do utilizador, por necessidade devido à profissão, ou por um *missclick* numa mensagem numa rede social que abre esse tipo de *site*. Então é aqui que entra a segurança informática, que evita que o utilizador sofra perda de dados pessoais ou até mesmo a perda total da informação *PC* no pior dos casos. No âmbito desta dissertação são analisadas e implementadas metodologias e algoritmos que permitam detetar a ocorrência de situações anómalas através das flags.

1.2 Motivação e Objetivos

Como consequência do crescimento da *Internet* os riscos de acesso a esta são permanentes, assim sendo a proteção é essencial. A primeira barreira de proteção de uma rede para outra é a Firewall, esta filtra o tráfego que entra e sai da rede. O método atual que existe nas *firewalls* tradicionais é através da inspeção de estado, porta e protocolo que está a ser utilizado [3], e.g., o protocolo *HTTPS* usa a porta 443 logo caso isto se confirme o tráfego será aceite. Esta regra será igual para outros protocolos e portas e o tráfego é aceite ou bloqueado.

Este tipo de *firewalls* tem um consumo de recursos bastante reduzido quando comparado com as *New Generation Firewalls*. Apesar de ser bastante simples, este tipo de *firewalls*, protege os utilizadores contra algumas tentativas de intrusão. Quando o tipo de ataque ainda não é conhecido o *hacker* tem acesso ao *PC* do utilizador até o anti-virus/*firewall* serem atualizados.

Visto que neste momento será impensável viver num mundo em que não tenhamos acesso à *Internet* é então essencial apostar na segurança da rede/dispositivo, é com este pressuposto que foi sugerido testar uma nova abordagem de reconhecer uma potencial tentativa de ataque a um servidor/*PC*. O objetivo desta dissertação é

analisar metedologias e algoritmos que permitam analisar padrões de comunicações, através da análise de *flags* dos pacotes *TCP*. Os algoritmos de análise de padrões deverão ser capazes de detetar tráfego considerados "normal"e "não normal". Os dados de treino de comunicações consideradas "normais"serão recolhidas e guardados numa base de dados, para treino dos algoritmos a utilizar. Nesta dissertação esses algoritmos serão baseados em Redes Neuronais.

Esta dissertação pretende ter uma contribuição para a possivel proteção de qualquer computador ligado à *Internet* pois analisa as ligações que são efectuadas analisando as *flags* utilizadas.

1.3 Estrutura do Documento

A presente dissertação desenvolvida está organizada em cinco capítulos. No Capítulo 1, Introdução, é abordado o contexto, a motivação e os objectivos a que esta se propõe.

No Capítulo 2, Enquadramento Tecnológico, é efectuada uma descrição das metodologias utilizadas assim como as características dos diferentes tipo de ataques e soluções possíveis até agora usados. São apresentados e comentados diferentes trabalhos relacionados com o tema desta dissertação, para se perceber as abordagens (técnicas e métodos) mais populares à resolução deste problema. É feita uma descrição dos tipos de ataques mais comuns, dos tipos de *firewalls* existentes, assim como os protocolos. É abordado, também, o *packet sniffer*, assim como as redes neuronais, e qual o objetivo da sua utilidade nesta dissertação.

No Capítulo 3, Conceção e Implementação, são descritas todas as premissas da implementação da referida aplicação e, as etapas que foram necessárias à sua execução.

No Capítulo 4, Testes e Resultados, são descritos com detalhe cenários de teste e respectivos resultados. São também descritos alguns problemas encontrados e as soluções encontradas para os resolver.

No Capítulo 5, Conclusões e Trabalho Futuro, são apresentadas as conclusões sobre os resultados obtidos durantes as diferentes fases desta dissertação, assim como alguns melhoramentos que permitirão otimizar ou melhorar o seu funcionamento.

Enquadramento Tecnológico

2.1 *Internet e Firewalls*

Criado em 1971, o modelo *OSI (Open System Interconection)* foi um dos pontos mais importantes para a comunicação que conhecemos hoje em dia, é divido em 7 camadas distintas umas das outras mas que servem umas às outras i.e., a camada 2 e.g., serve a camada 3 e é servida pela camada 1. O modelo *OSI* é apresentado na figura 2.1, sendo que estão definidos alguns protocolos e a camada em que eles são inseridos.

O modelo é divido em camadas para dar abstração à camada superior, sendo que essas camadas são as seguintes:

- Fisica: é a primeira camada do modelo *OSI*, esta é reponsável pela ligação entre dispositivos, aqui a informação é apenas em forma de *bits*, ao receber dados transforma em zeros e uns e envia para a camada de Ligação de Dados [4].
- Ligação de Dados: a principal função desta camada é assegurar que os dados são transferidos sem erros entre dois nós na mesma rede [4].

Figura 2.1 – Modelo OSI

- Rede: nesta camada é realizada a transmissão de dados entre dois *hosts* em redes diferentes, é assegurado o *routing* mais rápido dos trajetos disponíveis [4].

- Transporte: é responsável pela entrega da mensagem, há também controlo de fluxo de informação e controlo de erros [4].

- Sessão: é encarregue de estabelecer, manter, e terminar as sessões, além disto também é responsável pela autenticação e assegura a segurança [4].

- Apresentação: é a camada de tradução i.e., é responsável pela encriptação e desencriptação, e pela compatibilidade entre camadas de aplicação de sistemas diferentes [4].

- Aplicação: a última camada é a que fornece serviços às aplicações [4].

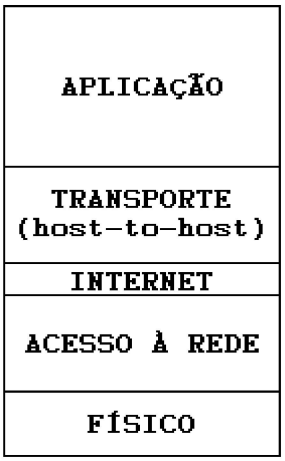

Figura 2.2 – Modelo TCP/IP

Outro grande modelo de comunicação é o *TCP/IP(Transmission Control Protocol/Internet Procotol)*, tal como o modelo *OSI* é dividido em camadas que são responsáveis por grupos de tarefas. O modelo é apresentado na figura 2.2, sendo que cada uma das camadas é responsável pelas seguintes funções:

- Fisica: camada com a as caracteristicas fisicas da comunicação como a natureza da ligação [5].
- Acesso à rede: responsável pela receção dos datagramas e encaminhá-los para a respectiva rede [6].
- Internet: camada que lida com a comunicação entre nós e encaminhamento entre redes diferentes [6].
- Transporte: camada responsável pela comunicação entre duas aplicações, assegura que os dados chegam em sequência e sem erros [6].
- Aplicação: os utilizadores usam as aplicações que podem aceder a serviços na *Internet*, cada aplicação escolhe o tipo de transporte necessário [6].

  A grande diferença entre o modelo *OSI* e o modelo *TCP/IP* é a deteção de erros. Enquanto o modelo *OSI* tem verificação de erros nas 4 primeiras

camadas, o modelo *TCP/IP* transfere esta verificação apenas para a camada de Transporte [6], além disto as funções das três ultimas camadas no modelo *OSI* é implementado apenas uma camada do modelo *TCP/IP*.

Protocolos de Transporte

O conjunto de protocolos TCP/IP fornece dois grandes protocolos de transporte, o *TCP* e o *UDP (User Datagram Protocol)*, tendo cada um diferentes características.

O *UDP* não garante a entrega de mensagens, não garante que elas chegam por ordem, podendo as mensagens ser perdidas, chegar duplicadas ou desordenadas. Portanto este protocolo de transporte fornece uma ligação não confiável de entrega de mensagens [6]. Quando usado tem de ser a aplicação a responsabilizar-se pela ordem dos pacotes, pacotes em falta ou até duplicados [6].

Ao contrário do protocolo apresentado anteriormente, o protocolo de transporte *TCP* assegura a entrega dos pacotes, assegurando que não há perda de dados, e caso sejam perdidos pacotes que sejam reenviados[6]. Portanto o protocolo *TCP* assegura uma ligação confiável para entrega de dados ao contrário do protocolo *UDP*.

Os protocolos de transporte TCP e UDP são usados em serviços da porta 0 a 65535, sendo que da porta 1 até à porta 1023 são *"Well Known Ports"*[7] i.e., são portas reservadas a protocolos já existentes e que não são usadas a não ser por esses protocolos. Os serviços usam o mesmo número de porta quer seja usado o protocolo de transporte *TCP* ou*UDP*, como se pode ver na tabela 2.1. Apesar de todos os serviços da tabela 2.1 poderem usar ambos os protocolos de transporte é mais usual ser usado apenas um deles, e.g., o *DNS* costuma usar o *UDP* em casos excepcionais usa o *TCP*.

protocolo	porta	Protocolo de transporte
FTP (data)	20	TCP
FTP(control)	21	TCP
DNS	51	UDP/TCP
HTTP	80	UDP/TCP
POP3	110	UDP/TCP
HTTPS	443	UDP/TCP

Tabela 2.1 – Exemplo de portas e protocolos de camada 7

Da porta 1024 até à 49151 são portas registadas [7] algumas delas com serviços associados, mas, que nada impede que sejam usadas para outros fins caso seja necessário, da porta 49152 à 65535 são portas privadas ou dinâmicas [8].

Se uma porta estiver aberta ela está à escuta pronta a ser comunicada, caso esteja fechada então irá ser recebido uma *flag RST*, assim fazendo uma verificação às portas verifica-se a vulnerabilidade do sistema.

2.1.1 Importância das *Firewalls* no Crescimento da *Internet*

A *Internet* é uma ferramenta essencial tanto ao trabalho como ao lazer e é usada por vários dispositivos todos os dias. A *Internet*, tem tido um grande crescimento nas últimas duas décadas estando neste momento a manter o número de hosts perto de um milhar de milhão, como se pode ver na Figura 2.3. Com este crescimento da *Internet* além dos aspectos positivos, vêm os perigos associados como e.g., possível furto de documentação privada, dados privados dos clientes.

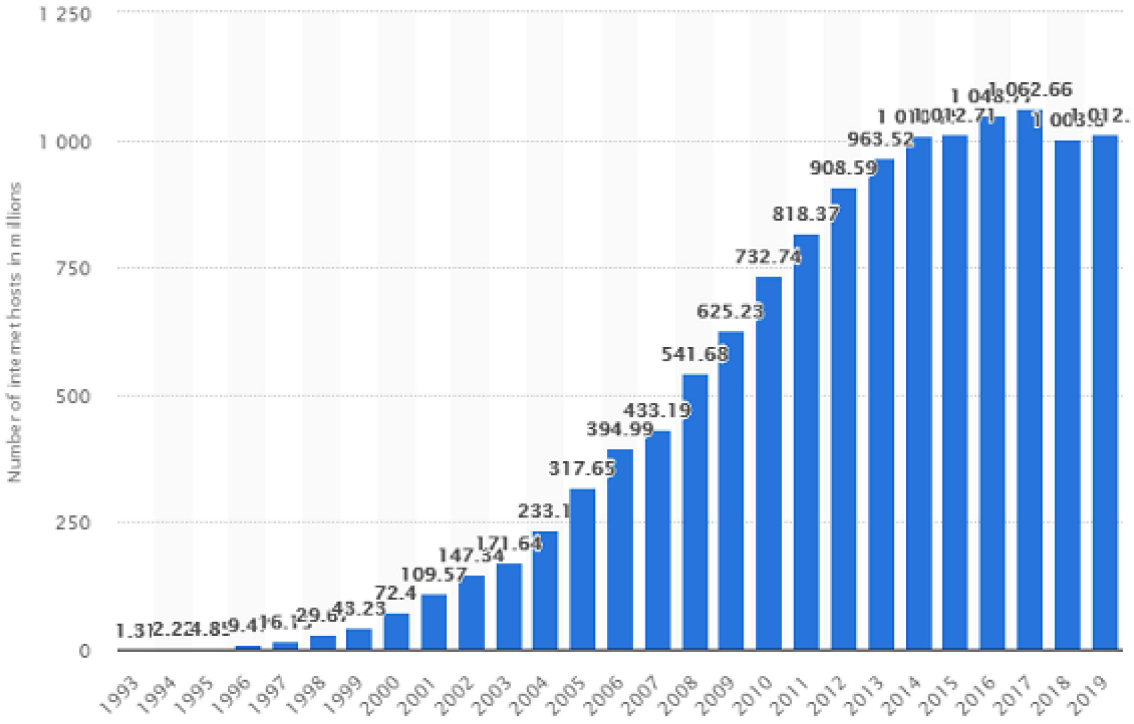

Figura 2.3 – Número de hosts desde 1993[1]

2.1.2 Firewalls

Uma *firewall* é um conjunto de componentes *hardware* e/ou *software* colocado entre duas redes, que filtram o tráfego entre elas de acordo com as politicas de segurança impostas [9].

Dependendo do tipo de *firewall* usada esta pode providenciar as seguintes funções [10]:

- Conservação de endereço de *IP* e tráfego para fora da rede interna, i.e., muitas *firewalls* funcionam como *routers* [10].
- Diferenciação de redes i.e., a *firewall* é a primeira fronteira entre duas redes, como cria uma distinção entre redes ajuda a gerir o tráfego [10].

- Proteção contra *DoS*, ataques de *scanning* e *sniffing* ,i.e., a *firewall* como pode funcionar como ponto único de entrada de trafego pode limitar ou bloquear tráfego [10].
- Filtros permitem autorizar/bloquear tráfego através de endereço *IP* ou através de número de porta [10].
- Servidores *proxy* são geralmente os únicos tipos de *firewalls* capazes de gerir tráfego através da inspeção de *URL* e conteúdo de pacotes [10].

Tipos de *Firewalls [10] [11]*

Filtering Simples

É a forma mais básica de proteção, apenas bloqueia o tráfego baseado no par porta/protocolo, i.e., todo o tráfego é bloqueado ou permitido dependendo do protocolo e porta. Não deteta ataques mas apesar disso é usado pela rapidez [10].

Stateful Inspection Packet Fitering

Neste tipo de *firewalls* já há alguma segurança através de tabelas de conexão, i.e., quando há uma receção de um *ACK*, consegue-se determinar se é legitimo ou não comparando o pacote com a correspondente entrada na tabela. Estas entradas são também desligadas após um certo tempo evitando uma longa extensão na tabela de sessões [11].

Application Proxies

Neste tipo de proteção é realizado uma inspeção profunda ao conteúdo. Estas *firewall* atuam como intermediário entre os dois *hosts*. As conexões são analisadas até à camada de aplicação para determinar se é permitida a entrada dos dados ou negada, desta forma pode-se por exemplo bloquear um protocolo. Este tipo de *firewall* por

outro lado tem um grande problema de performance já que é o ponto de entrada de todo o tráfego e necessita de ser analisado. Na figura 2.4 é apresentada uma *proxy firewall* e a camada onde é realizada a análise do tráfego, é analisada a última camada (Aplicação) [11].

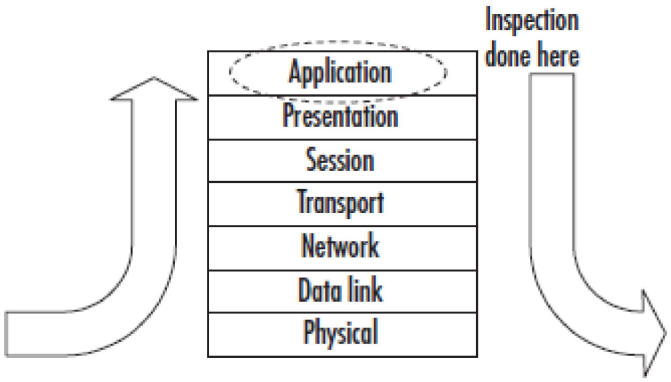

Figura 2.4 – Exemplo de inspeção proxy

2.1.3 *Next Generation Firewalls*

Next Generation Firewalls também conhecidas apenas pela sigla *NGFW*, ao contrário das *firewalls* tradicionais que filtram o tráfego por porta ou protocolo, estas filtram o tráfego identificando qual o protocolo que envia o tráfego para a rede. Além de reconhecer o protocolo desencripta os dados para saber qual a aplicação que está a enviar os dados através do protocolo.

Este tipo de *firewalls* consegue bloquear ou deixar passar tráfego conforme os endereços *IP* de destino do tráfego, analisa o tráfego assim que chega o primeiro pacote e não apenas quando todo o tráfego já está descarregado, este processo diminui a latência já que é analisado em tempo real assim que chega à rede. Reconhecendo o tráfego, consegue deixar passar tráfego dentro da mesma rede apenas para certos utilizadores, podendo bloquear esse tráfego para outros.

Além da proteção já mencionada as *NGFW* oferecem maior segurança que apenas as credenciais, estas *firewalls* conseguem usar autenticação multi-factor para impedir

o acesso aquando do furto das credenciais [12].

Para além de todas estas vantagens as *NGFW* justam no mesmo local a proteção contra ataques e a *VPN* por exemplo [13].

2.2 Segurança de Informação e Tipos de Ataque

Com o passar dos anos as empresas começaram a reparar que além do património material que possuíam e.g edifícios, veículos, fábricas, outro tipo de património muito importante era o património intelectual que possuíam[10]. Toda a informação documentada, toda a informação que era transmitida entre colaboradores em rede tinha tanto valor quanto o património material. Então é de todo o interesse que informação confidencial continue assim e não esteja sujeita a espionagem industrial. A Segurança informática é a eficiência da rede resistir a eventos acidentais ou eventos maliciosos que possam comprometer a informação armazenada ou a comunicação interna ou externa dessa organização. As principais áreas de preocupação na segurança de informação é conhecida pelo acrónimo em inglês CIA [14]:

- Confidencialidade : a informação apenas pode ser acedida por pessoas autorizadas ao serviço, mantendo assim o sigilo da informação [14].
- Integridade: assegurar que a informação não é modificada por alguém não autorizado, isto é feito por *CheckSums* [1] e *Hashes*[2] para confirmar a integridade dos dados transferidos [14].
- Disponibilidade: apesar de secreta a informaçao tem de estar sempre disponível caso um funcionário autorizado queira consultar [14].

Por outro lado para se alcançar a máxima segurança tem de se adicionar mais algumas áreas às acima descritas [10]:

[1] *CheckSums*-Assegura que toda a informação enviada foi recebida [15]

[2] *Hashes*-Assegura que o ficheiro não foi corrompido comparando com o valor anteriormente calculado do ficheiro hash [16]

- Autenticação – o mais comum neste caso será a *password* mas, com a evolução da tecnologia pode, também, ser feita através dados biométricos, e.g., impressão digital ou iris [10].

- Autorização/controlo de acesso – confirmar que após o utilizador ser autorizado a ter acesso ao sistema apenas tem acesso aos ficheiros/conteúdo/recursos autorizado o que pode ser feito através de *routers* ou *firewalls*.

- Não repudiação – isto é usado para caso se inicie uma transação se saiba quem a iniciou para que não consiga negar que foi ele/ela a iniciá-la, a *public key*[3] é usada para isto.

- Auditável – assegurar que as atividades podem ser monitorizadas e registadas *(logs)* para detetar usos sem autorização, isto pode ser feito e.g., *login* por sistema operativo, sistema de deteção de intrusão entre outros.

2.2.1 Classificação de Tipo de Ataques

Ataques de *"Social Engineering"*

Ao contrário dos outros não explora falhas de *software* ou *hardware*, nem requer muito conhecimento informático, explora sim as falhas humanas, como o desejo de cooperação a uma instituição com credibilidade. São ataques de pessoas que sabem falar e são persuasivos no seu discurso. Este tipo de ataque é baseado na confiança e quem faz o ataque tira vantagens do utilizador fazendo-o fornecer dados bancários, *passwords*, dados de acesso, etc. É também a forma mais fácil de ter acesso a uma rede de computadores na maior parte dos casos [10].

[3] *public key*- ligada à encriptação de dados conhecido como o par *public key e private key* serve para autentificar a identidade eletrónica ou para aceder a dados encriptados. A *public key* é publicada mas apenas desencripta a informação cifrada com a correspondente *private key* [17]

Spear Phishing

Spear Phishing é realizado através de *e-mail* com o objetivo de roubar dados confidenciais, ou tentando instalar *malware*[4] no computador atacado por este tipo de ataque. Chega à caixa de correio eletrónico um *e-mail* que aparentemente é de uma fonte fidedigna, mas que leva o destinatário a *sites* de *malware* [19].

Ataque *DoS*

DoS sigla para *Denial of Service*, é a negação de serviço, apesar de não haver furto ou destruição de informação, é uma escolha bastante popular para ataques a redes de computadores. Além de *DoS* tem-se também *DDoS* sigla para *Distributed Denial of Service*, que usam computadores intermediários os chamados agentes que têm pedaços de código malicioso adormecidos chamados *zombies*. Quando o *hacker* ativa os *zombies* começa o ataque de todos os computadores, este tipo de ataques são difíceis de localizar porque podem vir ataques de vários pontos do mundo, incluindo ou não do computador do *hacker*. Este tipo de ataque é realizado criando uma grande quantidade de tráfego para o *IP* sob ataque impedindo assim o normal funcionamento. Este tipo de ataques são protegidos pelos *ISP (Internet Service Provider)* mas apenas em dimensão empresarial, para o cliente comum o *ISP* ainda não aplica anti *DDoS* [10].

Scanning e Spoofing

Um *scanner*, no contexto de redes de segurança, refere-se a um *software* que obtém informação sobre o dispositivo. *Scanning* é um ataque que remotamente determina quais as portas que estão abertas, possibilitando assim o acesso. *Scanners* podem ser usados também para um fim benéfico, pois se for usado pelo administrador do sistema pode prevenir o ataque sabendo quais as portas abertas [10]. *Spoofing* é a

[4] *malware*- o termo é a contração das palavras inglesas *malicious software*, com o objetivo de danificar dispositivos, roubar dados [18]

ação de falsificar informação com o objetivo de aceder ilegitimamente a informação confidencial. Este roubo pode acontecer através de *e-mail* ou *website* por exemplo fazendo-se passar por uma fonte confiável [20].

Trojans

Um dos tipos de intrusão é através de *trojans*, este tipo de *software* parece não ter nenhum efeito negativo e aparenta até ser algo com uma função útil para o utilizador, normalmente escritos por *hackers* para contornar sistemas de segurança. Depois de instalado, o *trojan* consegue explorar falhas de segurança ou executar ações no computador que está sob ataque tais como apagar, modificar ou enviar ficheiros, ou até mesmo instalar outros programas maliciosos.

O *trojan* consegue executar ações fazendo-se passar pelo utilizador e torna-se ainda mais perigoso caso seja instalado no administrador do sistema. *Trojans* podem ser instalados por executáveis que até podem ser iniciados através de um *site* caso o utilizador tenha autorizado *scripts* a correrem automaticamente mal seja concluido o *download* [10].

Vírus

Outro tipo de intrusão é através de vírus que são instalados sem o conhecimento e executam ações não desejáveis. Replicam-se infetando os sistemas através do disco onde estão ou através da *Internet*. É possível distribuir vírus como anexo de e-mail ou através de um *Web site*. Novos vírus são criados diariamente e os anti vírus são actualizados para manter os utilizadores em segurança. Vírus podem fazer algo tão simples como abrir janelas que o utilizador não abriu, como sendo mais perigosos e apagar ficheiros do disco do utilizador, podendo até ser perigosos para a rede do utilizador [10].

Worms

Uma *worm* é um programa que consegue viajar pela rede, e se replica podendo infectar computadores ligados à mesma rede. Inicialmente as worms eram usados com fins legítimos em manutenção de redes, mas a sua eficiência em multiplicar-se tão rápido levou a que *hackers* começassem a usá-los para fins tanto abusivos quanto lucrativos [10].

2.2.2 *Packet Sniffer*

Um *Packet Sniffer* é um programa que está a ser executado em *background* numa rede e que recebe os dados que passam em toda a rede [21], não são usados apenas para fins lesivos, apesar de os intrusos os usarem para capturar dados não encriptados e *passwords* que lhes vai permitir o acesso aos sistemas.

Apesar destes dispositivos/*software* serem associados a fins lesivos ajudam também os administradores a saber o tipo de dados que passam na rede e a quantidade, podem ser configurados para capturar apenas certas interfaces, apenas algum tipo de tráfego, ou capturar tudo que passa na rede, pode também ser usado contra os próprios intrusos pois o tráfego que eles injetam na rede também é capturado pelo *sniffer*. Apesar de difíceis de detetar há também *software* que deteta *sniffers* não autorizados como o Sentinel ou Antisniff.

Um sniffer pode ser dividido nos seguintes componentes [22]:

- *Buffer* é o dispositivo onde são capturados os pacotes da rede, normalmente são usados 2 tipos, em um deles os dados são capturados ininterruptamente e no segundo os novos pacotes substituem os pacotes mais antigos.

- *Driver de captura*: a parte mais importante pois captura os dados da rede e guarda-os no *Buffer*.

- *Real-time Analysis* faz uma pequena análise das frames tal como são capturadas.

Quando um pacote é capturado vem apenas um conjunto de *bytes* sem significado que é separado como é mostrado na Figura 2.5.

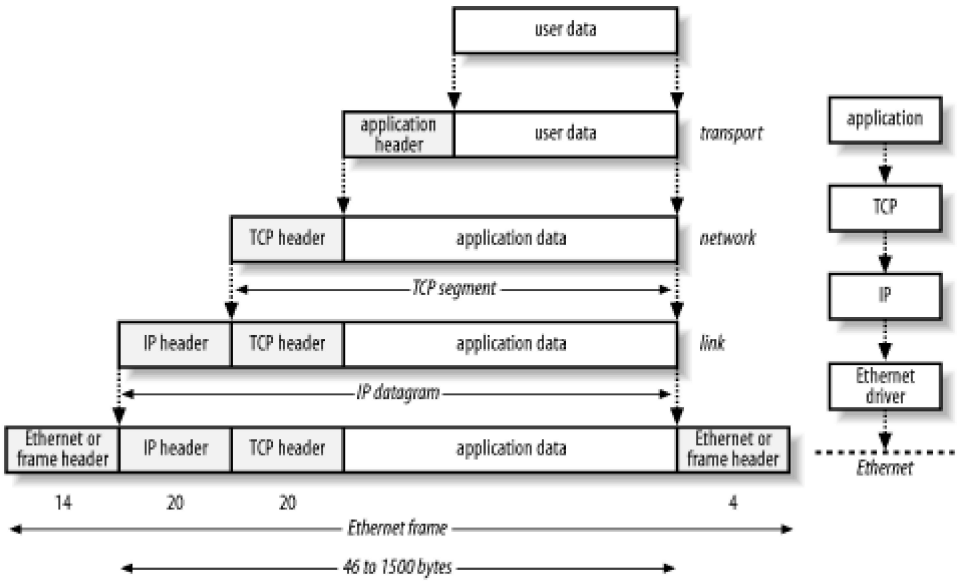

Figura 2.5 – Exemplo de pacote TCP/IP

No conjunto de bytes do *IP Header* vêm alguns dados como:

- Versão de *IP* – diz qual a versão *IP* que está a ser usada *IPv4* ou *IPv6*
- Tamanho do *IP Header*- especifica o datagrama em 32-bit
- Protocolo – indica à *layer* acima qual o protocolo que receberá este pacote (i.e, *TCP ou UDP*)
- Endereço de origem – especifica qual o endereço de *IP* que está e enviar os dados

- Endereço de destino – especifica o endereço de *IP* que irá receber os dados

O *TCP Header* contém outro tipo de dados também importantes para a transmissão de dados:

- Porta de destino e origem – identifica os extremos da camada de transporte.
- *Flags* – Este tipo de *bytes* são importantes para confirmar a receção e envio de dados e para fechar a ligação como irá ser explicado posteriormente.
- *Checksum* – é usado para garantir a integridade dos dados
- Dados - Contém os dados que serão tratados pela *layer* de aplicação.

Neste tipo de transmissão algo muito importante são as *flags*, são apenas um *bit*, que garantem que a ligação é bem executada e que ambos os *hosts* estão prontos a receber e enviar dados, além disto são também responsáveis por finalizar uma ligação, garantindo que não serão perdidos dados por uma das máquinas ter encerrado a ligação mais cedo que o previsto. Apesar de haver várias *flags*, as mais usadas são o *SYN*, *ACK* e *FIN*. Estas *flags* são usadas em todas as ligações que sejam fidedignas e em que a troca de dados tenha corrido da forma correta. As *flags* que se usam são as seguintes [23]:

- *SYN* – *flag* de sincronização, é a primeira *flag* usada quando se tenta abrir a ligação do tipo *3-Way Handshake*, que irá ser explicado mais adiante, e apenas deveria ser usada no primeiro pacote de quem envia e recebe.
- *ACK* – a *flag* de reconhecimento *"Acknowledgment"* é usada para informar que um pacote foi recebido com sucesso. Esta *flag* só é usada caso os dados sejam recebidos de maneira correta, caso contrário serão reenviados e nesse cenário será enviado o *ACK*. É também usada para estabelecer ligação no *3-Way Handshake* assim como, para finalizar uma ligação.

- *FIN* – esta *flag* é usada quando não há mais dados para transmitir, logo esta *flag* irá ser usada no último pacote para fechar a ligação entre servidor e cliente.
- *URG* – esta *flag*, tal como o acrónimo sugere, é para pacotes urgentes e faz com que o recetor processe primeiro estes pacotes passando-os à frente de todos os dados para ser processados.
- *PSH*- a *flag PUSH*, tal como na *flag URG*, é processada primeiro que os outros dados que estão em *Buffer*.
- *RST* – significa *RESET* e é usada pelo recetor quando são recebidos dados de que ele não estava à espera.

2.2.3 *3-Way HandShake*

Quando se dá inicio a uma transmissão é efetuado um processo que se dá o nome de *3-Way Handshake*, este processo acontece apenas no protocolo de transporte *TCP*. O *3-Way Handshake*, como mostra na Figura 2.6, consiste em estabelecer a ligação entre duas máquinas, para isso usam-se *flags*. Quando o *Client* quer comunicar envia um *SYN* ao *Server*, além da *Flag* são também enviadas as portas de comunicação. Quando o *Server* recebe o *SYN* envia para o *Client* um pacote com duas flags *SYN* e *ACK*. Após a recepção pelo *Client* destas *flags* é enviado um *ACK* e a partir deste momento está estabelecida a comunicação podendo começar a ser transmitidos dados [24]. Além de garantir que ambas as máquinas estão prontas para receber dados, nesta troca de *SYN* e *ACK* ambas as máquinas acordam quais os números de sequências iniciais. Quando o *Client* inicia a ligação envia um número de sequência x, a máquina *Server* envia um *ACK x+1* especificando que está à espera de receber esta sequência no próximo pacote recebido. Além disto envia também a sua própria sequência, neste caso designada por y, que servirá para designar qual a sequência enviada.

Quando a ligação é terminada, também há um procedimento que o protocolo *TCP/IP* tem como modelo para fechar a ligação, como se pode ver na Figura 2.7. A *flag FIN*

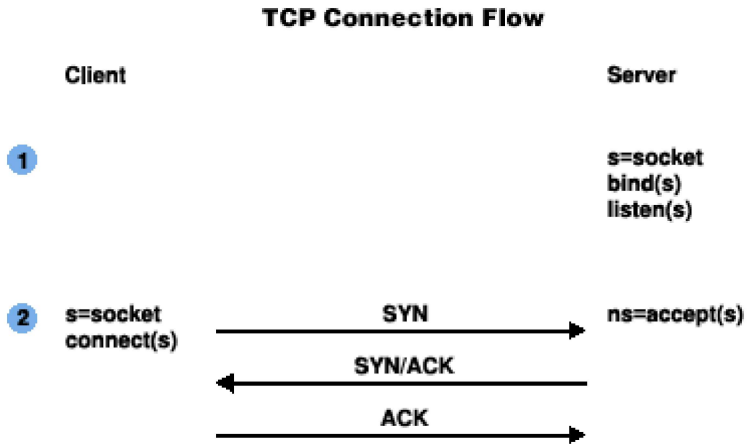

Figura 2.6 – *3-Way HandShake*

é usada para este fim. Neste caso o *Client* envia um *FIN* para o *Server* indicando que a partir deste momento não haverá mais pacotes a ser transmitidos [25]. Quando o *Server* recebe o *FIN* envia um *ACK* para o *Client*, aguarda então que a aplicação encerre a conexão e envia um *FIN* para o *Client*. Após o *Client* receber o *FIN* envia um *ACK* e a ligação é fechada [6]. O *ACK* ser enviado antes do *FIN* acontece porque a aplicação pode demorar bastante tempo a encerrar a conexão, impedindo assim o reenvio do *FIN* por parte do *Client*. Estes processos de envio de *flags* fazem com que a ligação inicie e termine sobre protocolos bem estabelecidos e que os dois *Hosts* estejam coordenados e prontos a receber ou enviar pacotes.

Na Figura 2.8 é apresentada uma máquina de estados das ligações *TCP*, esta máquina de estados representa todos os tipos de ligação possiveis quando uma ligação é fidedigna. Logo todos os padrões que não estiverem mencionados nesta máquina de estados serão suspeitos e motivo de análise.

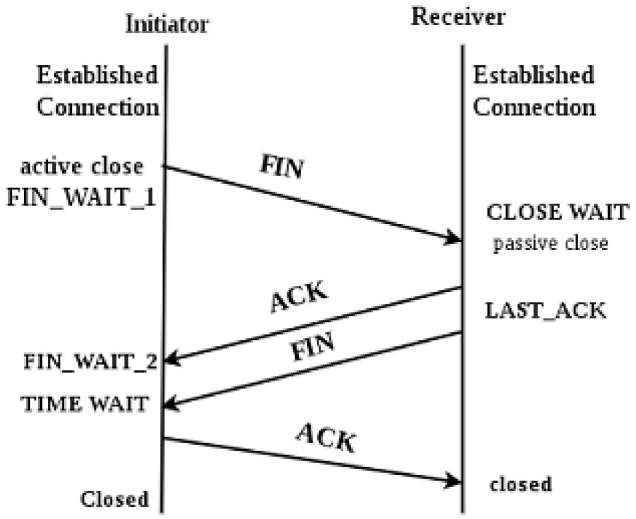

Figura 2.7 – Exemplo de finalização de ligação

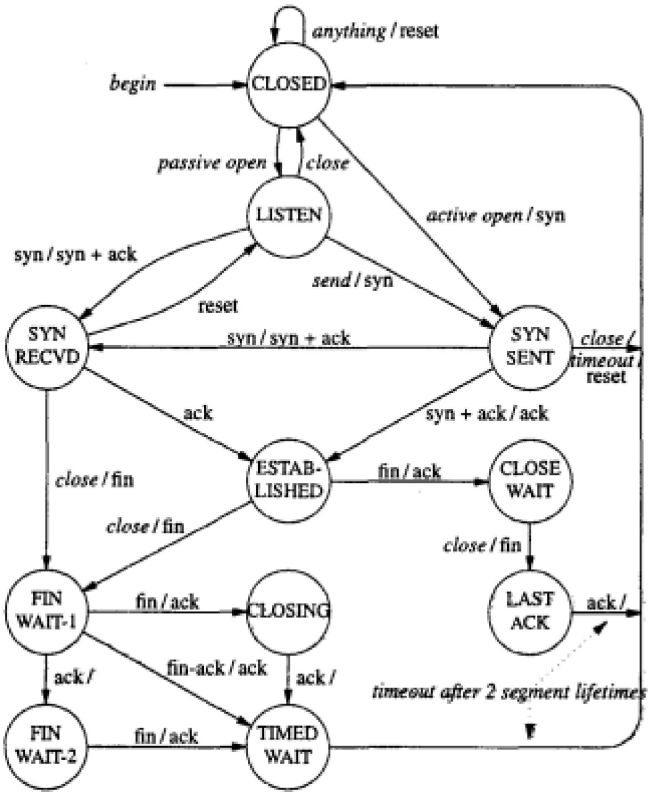

Figura 2.8 – Máquina de estados *flags TCP*

2.3 Redes Neuronais

As redes Neuronais comecou a ser falado em 1943 sendo a sua descoberta atribuida a dois investigadores da Universidade de Chicago, o neurofisiologista Warren McCullough e o matemático Walter Pitts [26].

Uma rede neuronal é um sistema computacional preparado para aprender através de funções, conseguindo assim entender e traduzir dados de *input*, produzindo um *output* com valor para análise humana [27]. As redes neuronais são baseadas nos humanos e na forma como os humanos detetam os *inputs* através dos sensores que todos têm, e.g tato, olfato etc, e os *Outputs* que o corpo humano tem depois desses dados passarem pelo cérebro. Estes dados que a redes neuronais avaliam, são introduzidos em algoritmos dependendo do que se quer descobrir, e.g padrões, imagens, etc, simulando assim o cérebro humano. As redes neuronais são bastante usadas em áreas como bolsa, analise numérica, deteção de fraude. Usam algoritmos como *k-means* por exemplo que tratam todos os dados para sinalizar movimentos suspeitos de terceiros.

A forma mais simples de uma rede neuronal é o neurónio, cada neurónio pode ser visto como um nó. A Figura 2.9 mostra o *perceptron* um dos neurónios mais usados em redes neuronais. O *perceptron*, tem um vetor de *Inputs Xn*, cujo produto vetorial com um vetor de pesos, *Wn*, retorna um *Output Z* como se pode verificar pela equação 2.1[2].

Após a soma dos pesos aplica-se uma função de activação representada na figura 2.9 pela letra σ. A função de activação é chamada de função degrau, pelas suas semelhanças. Caso o input seja maior ou igual a 0, o *Output* será 1, caso contrário o *Output* é 0 como se pode ver na equação 2.2. Este é o modelo matemático mais simples para um neurónio simples, a unidade fundamental de uma rede neuronal.

$$z = \sum_{i=1}^{n} W_i \times x_i + b \tag{2.1}$$

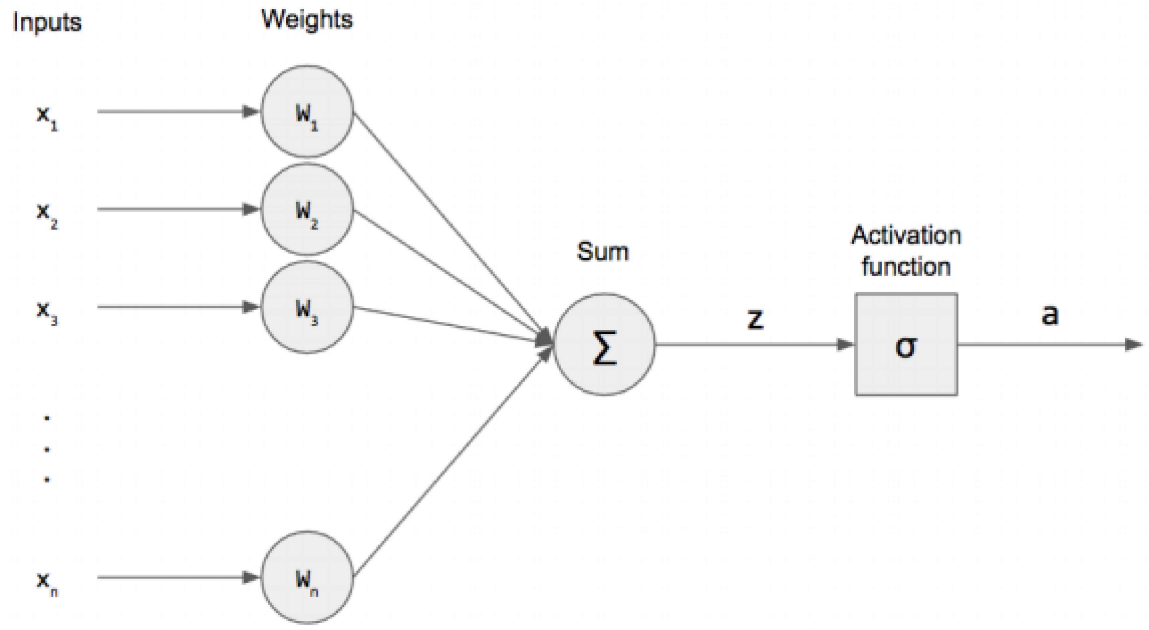

Figura 2.9 – Neurónio Simples [2]

$$\sigma(q) \begin{cases} 1 & q \geq 0 \\ 0 & q < 0 \end{cases} \tag{2.2}$$

O *perceptron* pode ser usado para formar uma rede neuronal, esta é composta por vários neurónios,um exemplo é a *MLP(Multilayer Perceptron)*, tal como apresentado na Figura 2.10. Há uma camada de *Inputs*, estes são multiplicados por pesos, aplica-se a função de activação, e retorna uma saída na *layer* de *Output*. Apesar de o exemplo apenas possuir uma camada escondida, é usual usar-se nas redes neuronais mais do que uma. Tal como nos *Outputs*, pode apenas possuir um, ou como no exemplo apresentado mais do que um.

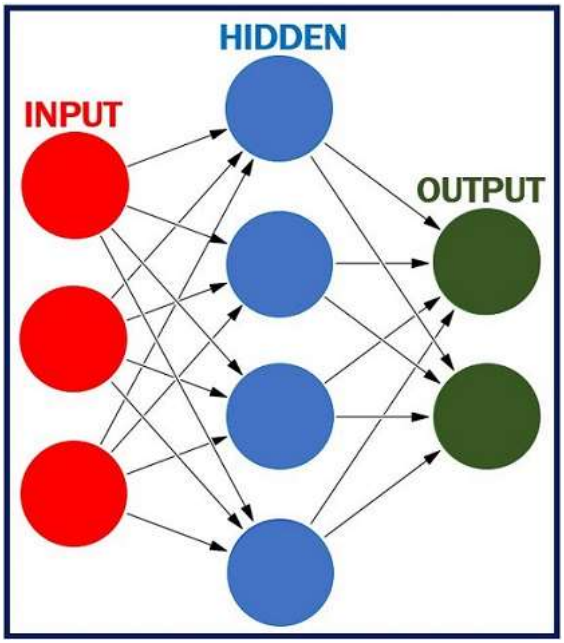

Figura 2.10 – Exemplo de rede neuronal

As redes neuronais podem ser submetidas a três tipos de aprendizagens [28]:

- **Aprendizagem supervisionada**: esta é a estratégia mais simples, pois a entrada e a saída estão tabeladas, e o algoritmo o que faz é tentar adaptar ao máximo os dados de maneira a eles se ajustarem a uma das soluções apresentadas, alguns exemplos são regressão linear, máquinas de vetores.

- **Aprendizagem não supervisonada**: Neste caso não há dados tabelados com entradas que correspondem a saídas, sendo assim a rede neuronal avalia os dados introduzidos e a função de custo encaminha a rede, ajustando-a ao algoritmo utilizado [26]. Este tipo de algoritmos são usados para agrupar dados, os algoritmos mais usados são *k-means* e *apriori*.

- **Aprendizagem reforçada**: Este tipo de aprendizagem além de ter dados para a rede ser treinada é forçada a obter resultados positivos, no caso de obter resultados negativos é punida, o que faz que com o passar das épocas a rede aprenda a adaptar os dados obtendo melhores resultados.

Para treino de uma rede neuronal há três diferentes passos:

- o *Input* passa pelas *Hidden Layers*, acabando na *Layer de Output*, onde a rede faz uma previsão (*forward pass*).
- esta previsão é comparada com o objetivo, usando a função custo.
- este erro é propagado para as *layers* anteriores, adaptando assim os pesos de cada neurónio, usando o respectivo gradiente da função de custo. O gradiente é um vector que aponta qual a direção que minimiza o erro da função de custo, esta função de custo permite perceber a adaptação aos dados, quando menor for melhor será a adaptação.

2.3.1 Funções de Activação

A função de activação é a ultima operação no fim do neurónio, esta determina se o neurónio fica excitado, e quão excitado fica. Já foi referido uma das funções de activação, no entanto há bastantes mais do que a referida e que serão agora explicadas.

A activação Sigmóide funciona com *Outputs* entre 0 e 1, e a função identidade não se aproxima da origem [29], é descrita pela equação 2.3:

$$f(x) = \sigma(x) = \left(\frac{1}{1 + e^{-x}}\right) \tag{2.3}$$

Outra função de activação bastante usada é a tangente hiperbólica(TanH), ao contrário da anterior o *Output* pode ser negativo, com valores entre -1 e 1, e a função identidade aproxima-se da origem [30], é descrita pela equação 2.4:

$$f(x) = \sigma(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})} \tag{2.4}$$

A função de activação *ReLU(Rectified Linear Unit)* [31] tem outputs de valores entre 0 e + ∞, e a função identidade não se aproxima da origem para valores negativos [31]. É descrita pela equação 2.5:

$$f(x) \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases} \tag{2.5}$$

ReLU mostrou ser bastante eficiente no treino de redes neuronais, comparando com as outras duas funções de activação apresentadas anteriormente [32].

2.3.2 Long Short-Term Memory

As redes *LSTM(Long Short-Term Memory)* foram propostas por Sepp Hochreiter e Jurgen Schmidhuber, e conseguem guardar informação em memória que é depois usada para para outras células . As unidades *LSTM* são compostas por uma célula, uma *Input Gate*, uma *Output Gate* e uma *Forget Gate*. As *Gates* regulam o fluxo de informação para dentro da célula, estas aprendem que porções de informação são relevantes e quais são irrelevantes. Na Figura 2.11 é apresentada uma unidade *LSTM*.

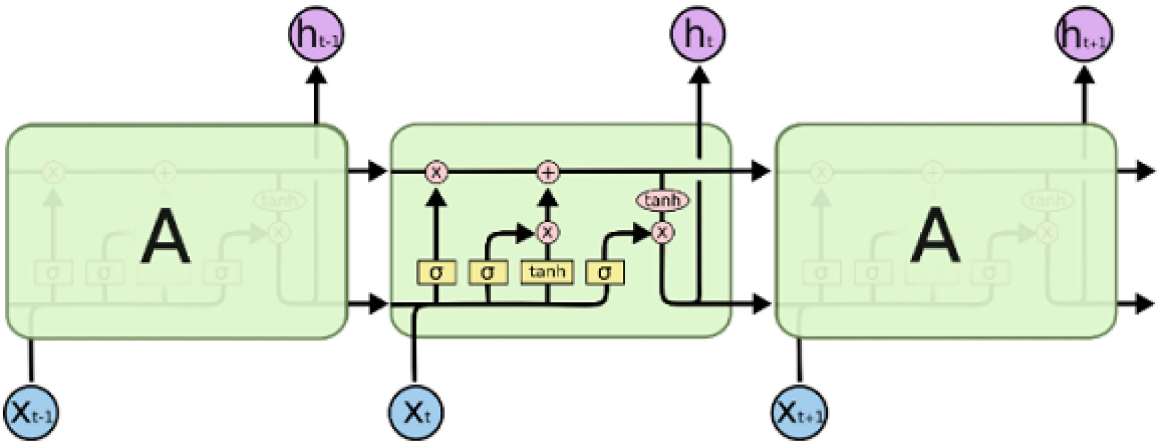

Figura 2.11 – Exemplo de diagrama *LSTM*

Com tantos parâmetros os resultados com este tipo de redes neuronais podem ser bastante bons, por outro lado a taxa de aprendizagem tem de ser escolhida com cuidado, pois sofre do problema de explosão de grandiente, segundo [33], o tamanho da rede neuronal e a taxa de aprendizagem são os parâmetros cruciais numa rede LSTM. As equações que definem as *Gates* representadas na Figura 2.12 de uma rede *LSTM* são as definidas em 2.6, 2.7, 2.8, 2.9, 2.10, 2.11 [34]:

$$f_t = \sigma \times (W_f \times [h_{t-1}, x_t] + b_f) \tag{2.6}$$

$$i_t = \sigma \times (W_i \times [h_{t-1}, x_t] + b_i) \tag{2.7}$$

$$\tilde{C_t} = tanh \times (W_C \times [ht - 1, x_t] + b_C) \tag{2.8}$$

$$C_t = f_t \times C_{t-1} + i_t \times \tilde{C_t} \tag{2.9}$$

$$o_t = \sigma \times (W_o \times [h_{t-1}, x_t] + b_o) \tag{2.10}$$

$$h_t = o_t \times tanh(C_t) \tag{2.11}$$

$x_t \in R$: vector de *Input LSTM*;

$f_t \in R$: activação do vector de esquecimento da *gate*;

$i_t \in R$: activação do vector de *Input* da *gate*;

$o_t \in R$: activação do vector de *Input* da *gate*;

$h_t \in R$: vetor *hidden* de estado;

$c_t \in R$: vetor de estado da célula;

$W \in R, U \in R, b \in R$: são os pesos das matrizes e o parâmetros do vector *bias*(offset) que será aprendido durante o treino.

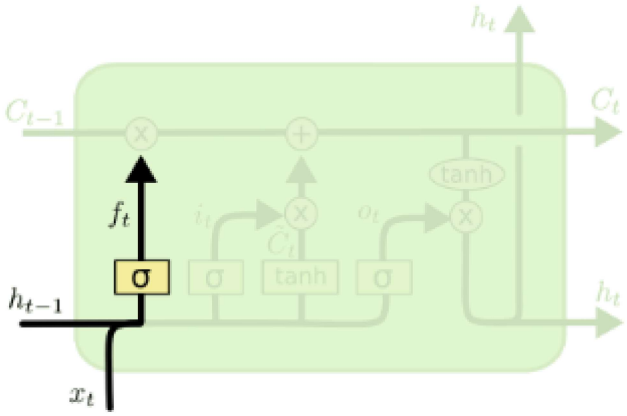

Figura 2.12 – Unidade *LSTM*

O primeiro passo é decidir qual a informação não será utilizada pela célula. Isto acontece analisando o $ht-1$ o x_t e o *Output* será entre 0 e 1 para cada número na célula de estado C_{t-1}, onde 1 será manter os dados e 0 descartar os dados.

O passo seguinte será decidir qual a informação a guardar na célula de estado. A *Sigmoid layer* decide que valores vão ser atualizados, a *Tanh layer* cria um vector de novos candidatos $\tilde{C}_t$ que podem ser adicionados ao estado. Com a combinação destes dois é criada a atualização do estado.

Faz-se a actualização da célula C_{t-1} para a nova célula C_t. Esta atualização é executada tendo em conta o quanto o algoritmo decidiu atualizar cada valor.

Após todos estes passos a rede está em condições de gerar um *Output*, que neste caso ainda será passado por uma *Sigmoid layer*. O *Output* será a multiplicação do estado da célula passada pela *layer* Tanh e da *Sigmoid*.

2.3.3 Função de Custo

Antes dos parâmetros da rede neuronal serem atualizados para que a rede continue o treino, o custo tem de ser calculado para esses parâmetros serem atualizados de forma correta. A esta função dá-se o nome de função de custo. Existem várias funções de custo, sendo a apresentada na equação 2.12 a função de custo *binary crossentropy*:

$$L(y, \tilde{y}) = -\frac{1}{N}\sum_{i=0}^{N}(y \times log(\tilde{y}_i) + (1-y) \times log(1-\tilde{y}_i)) \tag{2.12}$$

$\tilde{y}$: é o valor valor previsto

N : é o número de dados

y : é o valor de entrada de dados

O *Output* desta função de custo é um valor entre 0 e 1.

O gradiente é o método mais popular para otimizar as redes neuronais, o gradiente descendente é o modo de minimizar a função de custo da rede neuronal. Para isto acontecer há alguns métodos mais conhecidos como o *RMSprop* ou o *Adam*, sendo este último o que se irá dar mais enfâse. Na equação 2.13 mostra-se o atualizador de gradientes que o otimizador *Adam* usa para calcular o gradiente em cada um dos passos:

$$v_t = (1-\beta_2)\sum_{i=1}^{t}\beta_2^{t-i} \times (g_i)^2 \tag{2.13}$$

onde :

β_2: taxa de declinio do gradiente

g_i^2: gradiente em cada uma das *epochs* do treino da rede neuronal

OverFit e UnderFit

Overfit e Underfit estão ambos relacionados com o treino, uma das partes mais importantes na construção de qualquer rede é o treino. O *Overfit* é quando acontece um treino intensivo demais, a função de custo é mínima, apesar disso não se consegue adaptar aos dados em teste. O que acontece é que a rede se adaptou a todos os pontos, atingindo padrões secundários que não seriam o foco para que a rede foi produzida. O *Underfit* ocorre quando ainda há possibilidade de melhorar os dados de teste. Este caso pode acontecer porque o modelo não é o melhor para os dados para os dados, ou simplesmente porque não foi treinada o suficiente, isto significa que a rede não aprendeu os padrões relevantes dos dados de treino [35].

Técnicas de Regulação

As técnicas de regulação são usadas para evitar o *Overfitting*, tal como já referido é a tentativa de evitar que se adapte demasiado aos dados. Neste caso usam-se dois tipos de reguladores: L1 e L2.

Usando o *L2* a regulação tende a diminuir os parâmetros extremos, executando o regulador adiciona um peso à função de custo para assim a conseguir controlar [35], isto é descrito na equação 2.14:

$$Cost = CostFunction + \lambda_2 \times \sum_{i=0}^{N} W_i^2 \tag{2.14}$$

Usando o *L1* a regulação tende a diminuir os parâmetros menos importantes, tal como o anterior é adicionado peso à função custo [35], isto é descrito na equação 2.15:

$$Cost = CostFunction + \lambda_2 \times \sum_{i=0}^{N} |W_i| \tag{2.15}$$

Outra técnica utilizada para regulação é a técnica de *Dropout*, esta técnica altera aleatoriamente para um valor nulo o *Output* durante o treino. Cada neurónio na rede tem uma probabilidade de ser omitido alterando assim a saída [35].

2.3.4 Tensorflow

Tensorflow é uma uma plataforma *open-source* usada em *machine learning*. O sucesso desta plataforma *open source* é a flexibilidade que o Tensorflow oferece ao utilizador. Com Tensorflow o utilizador pode usar ferramentas de alto nivel assim como na construção do modelo pode alterar o treino a função de custo apenas chamando outra função já definida no sistema. Na Figura 2.13 está representada a hierarquia de TensorFlow, desde onde pode o código ser executado até às bibliotecas que esta ferramente oferece ao utilizador.

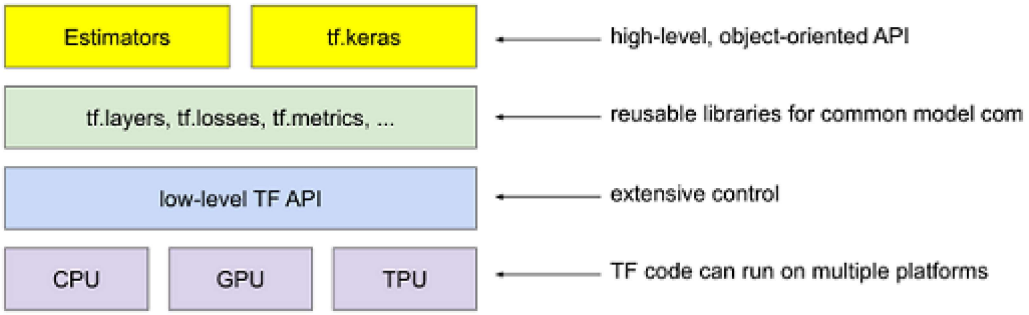

Figura 2.13 – Hierarquia *Tensorflow*

Keras é uma *API* que contém funções e com facilidade criar modelos. Esta ferramenta é capaz de funcionar em *Tensorflow ou Theano*, tal como *TensorFlow* tira partido tanto do processador como da *GPU* de maneira diminuir o tempo de compilação de programas.

Compilação de modelos

Modelos em TensorFlow são definidos usando bibliotecas já construidas facilitando ao utilizador a construção do modelo. Nesta construção são usados neurónios a

unidade básica de uma rede neuronal, o número de neurónios a função de custo, o número de iterações, toda esta informação é escolhida na construção do modelo em Tensorflow. Antes de treinar o modelo são preciso algumas configurações como por exemplo:

- ***Função loss***: mede quão preciso é o modelo durante o treino, quer-se minimizar a função de custo para guiar o modelo na direção certa [36].
- ***Optimizer*** é como o modelo se atualiza com base nos dados introduzidos e na função *loss* [36].
- ***Epochs***:é o número de vezes que o modelo é exposto ao conjunto de treino. Em cada iteração o *Optimizer* tenta ajustar os pesos com o objetivo de ajustar a função (minimizar ou maximizar) [37].
- ***batch size*** é o número de instâncias de treino observadas antes do *optimizer* executar a actualização de pesos [37].

Após esta configuração será necessário fornecer os dados para a rede neuronal encontrar padrões, pode-se observar a adaptação aos dados através da função de custo, quanto menor for este custo melhor será a adaptação, logo melhor será o *Output* da rede neuronal.

2.3.5 Modelo de Markov

A Cadeia de Markov é um modelo matemático que representa transições de estados usando probabilidades. A propriedade de Markov diz-nos que o estado futuro depende apenas do estado actual, não dependendo de nenhum estado passado sendo dada pela equação 2.16.

$$P(X_n = i_n | X_{n-1} = i_{n-1}) = P(X_n = i_n | X_0 = i_0, X_1 = i_1, ...X_{n-1} = i_{n-1}) \quad (2.16)$$

Para cada cadeia de Markov os estados são finitos e há uma matriz de transição que pode ser reprensentada visualmente por um grafo. Como se representa na figura 2.14, os estados são representados por círculos, e as probabilidades são representadas pelas linhas que apontam para o próximo estado desde que este seja não nulo. Esta matriz é normalmente representada pela letra **P**.

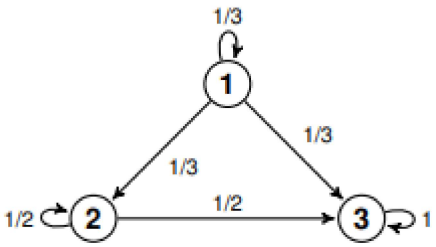

Figura 2.14 – Exemplo Cadeia de transição

Para ser considerado um evento de Markov o processo tem de ser considerado discreto, e o que acontence no instante t_{n-1} onde está no estado X_{n-1} em nada influencia o estado X_n no instante t_n. Tal como ja referido a matriz P é a matriz de transição de uma cadeia de Markov, já a matriz U é o vector de probabilidade que representa a matriz de distribuição inicial. Sendo assim a probabilidade da cadeia estar no estado s_i, após n fases é dado pela equação 2.17.

$$u^n = u \times P^n \tag{2.17}$$

Uma cadeia de Markov diz-se irredutível se for possivel atingir qualquer estado através de outro estado e não apenas através de um único caminho, i.e., no grafo todos os estados têm de estar ligados.

$$Pr(X_{n_{ij}} = j | X_0 = i) = P_{ij}^{n_{ij}} \tag{2.18}$$

onde i e j são os estados inicial e final respetivamente.

A cadeia de Markov pode ser definida como periódica ou aperiódica dependendo de por quantos estados têm de se passar para retornar ao mesmo estado de partida.

As cadeias de Markov podem ser classificadas como recorrentes, caso saindo de um estado haja a probabilidade de 1 se voltar a ele. Caso haja a possibilidade de por exemplo a cadeia entrar em *loop* e nunca mais ser possivel voltar ao estado a cadeia denomina-se de Transiente.

2.3.6 Trabalhos Relacionados

Nesta secção, serão abordados trabalhos e soluções que estão relacionados com o problema exposto nesta dissertação. Alguns dos trabalhos evitam outro tipo de ataques mas usando também *flags*, embora a abordagem ou objectivo possam diferir um pouco do apresentado.

Detecting TCP SYN Flood Attack Based on Anomaly Detection

Neste estudo [38], os autores procuram resolver um tipo de ataque, o *TCP SYN Flood*, em que o servidor é bombardeado com pedidos *SYN* de maneira a inutilizá-lo. Os autores focam-se em detetar actividade maliciosa detetando padrões de tráfego de rede. Há então três tipos de análise de comportamento:

- Usa-se o protocolo para detetar pacotes que são pequenos demais, que violam especificamente o protocolo das camadas de aplicação.
- Deteção por *rates* que deteta situações de *flooding* no tráfego usando modelo de tempo e volume de tráfego *DoS*.
- Deteta o comportamento ou relaciona as mudanças de como um *host* individual ou grupo de *hosts* interage com uma ou mais redes.

Neste trabalho é detetado a anomalia de comportamento analisando o *IP Header* e o *TCP Header* usando o *Payload*. Caso o *IP Header* seja igual a 20 *bytes* verifica

se o *Payload* é normal, caso não seja reporta ao administrador, no caso de ser normal analisa outro pacote. Além do filtro de pacotes, registaram também o uso do processador e da memória, tudo isto foi feito nos dois casos: no caso em que o tráfego é considerado normal e no caso em que a rede se encontra sob ataque.

No caso do filtro de tráfego, este foi configurado para pacotes *TCP*, e no caso das *flags* quando há bastantes *flags SYN e RST* muito provavelmente está a ocorrer um ataque.

Os autores realizaram o *download* em duas situações, no primeiro teste o *download* foi realizado com o tráfego normal de rede, com todas as oscilações de velocidade a que a rede esta sujeita, o *download* demorou um minuto. Na segunda situação a rede estava sob ataque *Syn Flood*, neste caso a transferência do mesmo documento demorou dez horas, isto prova que a rede fica praticamente inutilizada. Além disto provou-se que sob ataque o uso do processador aumenta significativamente. Detetaram também no teste que os pacotes maliciosos chegam todos da porta 1024 ou 3072, e tem sempre o mesmo tamanho de **Header** 448 Bytes.

Detecting distributed denial-of-service attacks by analyzing TCP SYN packets statistically

Neste caso [39], os autores fizeram um estudo sobre *DoS* ou *DDoS* dependendo se o ataque é realizado de vários computadores ou apenas de um computador. Os autores optaram por fazer a monitorização e classificação do tráfego, foi dividido em 5 categorias dependendo dos padrões de *flags*, sendo as categorias as seguintes:

- *Group N*: tráfego que inicia com *3-Way Handshake* e é encerrado pela *flag FIN ou RST.*
- *Group Rs*: tráfego que termina com a *flag RST* antes de haver um *SYN/ACK* e a ligação é terminada assim porque o *host* não está disponível para este serviço de *SYN request.*
- *Group Ra*: tráfego que termina com a *flag RST* antes de haver um *ACK* para

o *SYN/ACK*e a ligação é encerrada assim porque o pacote *SYN/ACK* foi enviado para um *host* falso.

- *Group Ts* tráfego contendo apenas pacotes com a *flag SYN*. Este tipo de acessos são terminados por *timeout* e não como normalmente com *flag RST e FIN*. Há 3 razões para acontecer: ou é um *SYN flood*; o servidor não está preparado para responder a este tipo de pedido; ou a rede está muito sobrecarregada e descarta a resposta *SYN/ACK*.
- *Group Ta*: tráfego contendo apenas pacotes com a *flag SYN e SYN/ACK*, possivelmente porque os pacotes *ACK* foram descartados terminando assim todas as ligações por *timeout*.

Neste estudo, após a recolha do tráfego definiu-se N como tráfego normal e tudo o resto como tráfego incompleto e chega-se à conclusão que há um uso muito maior da *flag SYN* quando a rede tem tráfego incompleto isto é, pode estar sob ataque. Além, disto nota-se uma variação muito maior de *SYN* quando o servidor está a receber tráfego incompleto apesar de 85.1% do tráfego recolhido ser considerado tráfego normal.

Deep Packet Inspection Using Parallel Bloom Filters

Neste estudo [40], os autores usaram a teoria de filtros de *Bloom*, estes são estruturas que permitem armazenar informação de maneira probabilística. Este tipo de filtros analisam todos os *byte strings* que entram distribuindo-os por filtros de *Bloom*, isto é, o primeiro *byte* entra no primeiro filtro, o primeiro e segundo *byte* entram no segundo filtro, até ao byte final em que entram todos os bytes e é analisada toda a *string*. Quando um filtro de *Bloom* deteta um possível problema este é passado por um analisador que deteta se de facto se trata de um problema ou apenas um falso positivo. Os autores chegam à conclusão que quanto mais forem os padrões programados para o filtro reconhecer, maior será a probabilidade de falsos positivos. De realçar que este teste foi executado em rede de fibra logo, o número de pacotes e tempo de análise tem de ser bastante rápido para não causar *delay* na rede.

Conceção e Implementação

Para testar as soluções apresentadas foram gerados diferentes padrões de comunicação, algumas consideradas como normais e outras consideradas como possível tentativa de ataque. Esses dados foram capturados utilizando um *packet sniffer*. Os dados utilizados nesta dissertação foram capturados no laboratório de Redes da Universidade de Trás-os-Montes e Alto Douro, onde foi possível implementar uma rede local de testes.

Neste capítulo será abordado o estudo que serviu de base à conceção deste trabalho, incidindo nos pontos fulcrais, recolha de dados e treino da rede neuronal, para melhor entender a teoria que se quis defender nesta dissertação nomeadamente se uma rede consegue através de padrões de acesso aprender o que é um acesso considerado normal, ou um possível padrão de ataque.

3.1 Recolha de Dados

Como já referido o objectivo é aferir se uma ligação é ou não um possivel ataque, para este efeito usam-se padrões de comunicação. Estes padrões foram gerados, considerando alguns como padrões de acesso normais, e outros considerados padrões

de possivel tentativa de ataque.

Para recolha de tráfego da rede é necessário usar um *Packet Sniffer*, que inicialmente foi desenvolvido em *C* usando as bibliotecas *libpcap*. Esta solução demonstrou-se infrutífera porque este *Packet Sniffer* não recolhia todo o tráfego da rede, sendo que para recolher tráfego tinham de se aplicar inúmeros filtros. Visto que há bastantes *Packet Sniffers* disponíveis, que recolhem todo o tráfego da rede, optou-se pela escolha do *TcpDump*. Para começar a desenvolver o projeto foi necessário estudar e avaliar se seria possível este tipo de estudo, para isto começou-se por executar o *Packet Sniffer*, para a recolha de pacotes de uma rede de testes. A Figura 3.1 ilustra o cenário utilizado para a recolha de dados e os componentes da rede:

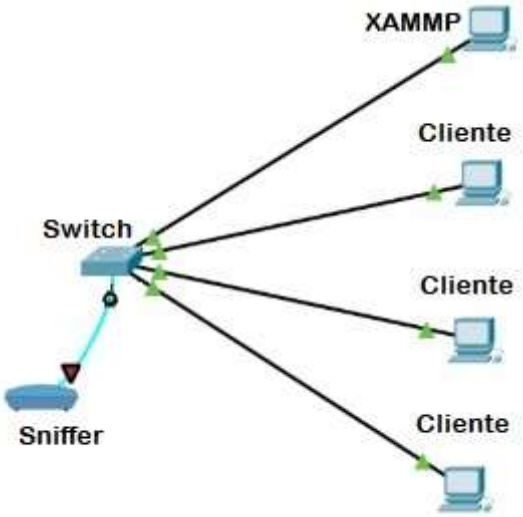

Figura 3.1 – Modo de recolha de pacotes.

- Um *Packet Sniffer* neste caso foi executado numa máquina virtual.
- Um *switch layer* 3 que para além de *switch* funcionará também como *router*, e executará o protocolo de *routing* definido o *OSPF(Open Shortest Path First)*.
- *End devices* que neste caso serão vários computadores a gerar tráfego de teste para depois serem tratados.
- Um Servidor *Web* usando *XAMPP*.

Estes tipos de testes foram executados na Universidade de Trás-os-Montes e Alto Douro em ambiente fechado e controlado de forma a melhor identificar que tipo de dados eram injetados na rede. Este processo de recolha de dados foi realizado em ambiente fechado por 2 razões: garantir que se sabia o tipo de ligação, e por não ser possível capturar dados privados de outros utilizadores por poderem conter dados pessoais, e.g., passwords, ficheiros.

Na concepção o objetivo será:

- Recolher os dados injetados na rede;
- Tratar os dados de modo a que, apenas, se obtenham os dados dos protocolos *ICMP e HTTP e HTTPS*;
- Após este tratamento avaliar a *flags* utilizadas no caso do protocolo *HTTP e HTTPS*;
- Verificar se as *flags* estão pela ordem correta e se não há alguma inconformidade;

Para a injeção de dados foi criado um *script* que faz acessos na rede a um *Website* ou faz *"pings"* a outras máquinas na rede. Para acesso ao *Website* foi usado o *software XAMPP*,[1] plataforma que executa o servidor Apache para serem simulados acessos *HTTP e HTTPS*. Para recolha de dados está em *background* o *Packet Sniffer* a gravar os dados que passam na rede.

Na Figura 3.1 está o cenário de recolha de dados, 3 computadores que vão aceder a um outro computador que tem *XAMPP* instalado. Os 3 computadores estão ligados em *Vlan* diferentes. Todas as máquinas estão ligadas ao mesmo *Switch* que sendo de *layer 3* funciona tanto como *Switch*, como *router* [41], a este está também ligado um outro computador que apenas tem como objetivo a captura dos pacotes da rede.

Todos os dados recolhidos são recolhidos em bruto,i.e., tudo o que passa na rede é capturado pelo *Packet Sniffer* e guardado num ficheiro do tipo *.pcap* que será onde

[1] *XAMPP- plataforma all-in-one que contem servidor Apache, PHP e MySQL*

estão guardados todos os dados. Para este estudo não terão interesse todos os dados mas estarão sempre acessíveis neste documento.

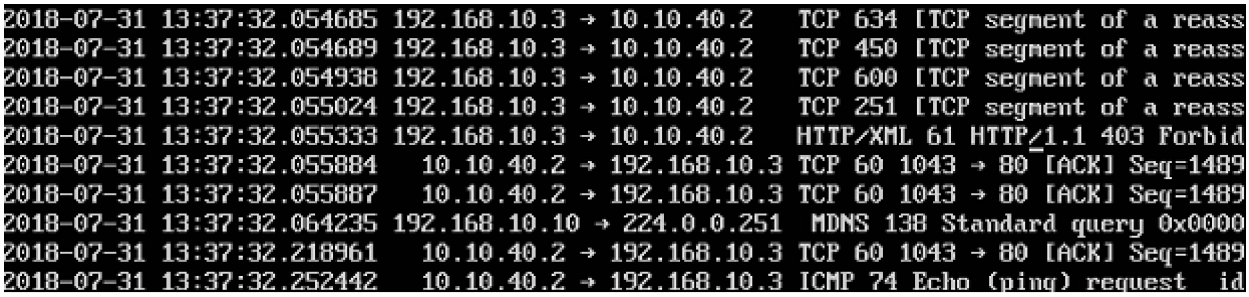

Figura 3.2 – Exemplo de Tráfego recolhido pelo *TCPDump*.

Na Figura 3.2 é apresentado um excerto do tráfego capturado pelo *TCPDump*, este recolhe todos os dados que passam na rede, ainda que para este trabalho nem todos são de interesse. Os dados recolhidos necessitam então de tratamento para depois serem injetados na rede neuronal para classificação do tipo de ligação.

3.2 Desenvolvimento do Programa para Tratamento de Dados

Após realizada a recolha de dados através do *sniffer* como foi referenciado no subcapítulo anterior procedeu-se à filtragem dos dados recolhidos. Como manusear um documento do tipo *.pcap* não é fácil optou-se por passar o documento *.pcap* para um documento do tipo *.txt*.

A filtragem é realizada da seguinte modo:

- Procura pela palavra *HTTP, HTPPS e ICMP*
- Selecionar tráfego interessante para o estudo: endereços *IP* usados na transação, hora da transação, portas usadas e *flag*

No caso do protocolo *ICMP* foi guardada também se era um *Request* ou um *Reply*. A Figura 3.3 apresenta o tráfego depois de ser filtrado com os parâmetros interessantes para estudo.

```
13:37:32.735402 10.10.40.2 → 192.168.10.3 1043 → 80 [ACK]
13:37:32.918551 10.10.40.2 → 192.168.10.3 1043 → 80 [ACK]
13:37:38.246644 192.168.10.3 → 10.10.40.2 80 → 1043 [FIN,
13:37:38.246647 10.10.40.2 → 192.168.10.3 1043 → 80 [ACK]
13:37:42.742688 10.10.40.2 → 192.168.10.3 1043 → 80 [RST,
13:37:25.238073 10.10.40.2 → 192.168.10.3 Echo (ping) request
13:37:25.238473 192.168.10.3 → 10.10.40.2 Echo (ping) reply
13:37:26.239003 10.10.40.2 → 192.168.10.3 Echo (ping) request
13:37:26.239007 192.168.10.3 → 10.10.40.2 Echo (ping) reply
13:37:27.240966 10.10.40.2 → 192.168.10.3 Echo (ping) request
13:37:27.241454 192.168.10.3 → 10.10.40.2 Echo (ping) reply
13:37:28.243880 10.10.40.2 → 192.168.10.3 Echo (ping) request
13:37:28.244325 192.168.10.3 → 10.10.40.2 Echo (ping) reply
```

Figura 3.3 – Exemplo de Tráfego filtrado.

Após esta seleção o documento é filtrado novamente à procura das flags usadas em cada pacote que o *Packet Sniffer* capturou da rede. Isto é processado linha a linha procurando por cada uma das *flags*, no caso de a *flag* ser encontrada a coluna correpondente passa a (1) sendo que todas as outras colunas permanecem a (0), no caso de nenhuma *flag* ser encontrada a linha é preenchida com (0). Este processo facilita a entrada das flags para a rede neuronal analisar, o exemplo deste processo está representado na Figura 3.4

```
0, 1, 0, 0, 0
0, 1, 0, 0, 0
0, 0, 1, 0, 0
0, 1, 0, 0, 0
0, 0, 0, 1, 0
```

Figura 3.4 – Exemplo da procura de *flags* usando a Figura 3.3.

3.2.1 Estudo dos Algortimos

Após feita a recolha dos dados e feita a filtragem os dados estão praticamente prontos a serem analisados pela rede neuronal. O *Output* da rede será a decisão da rede neuronal sobre a ligação que está a ser estudada, se a rede acha que a ligação segue os parâmetros normais, se acha que não é normal e poderá ser um possivel ataque.

Visto *Tensorflow* ser uma biblioteca *open-source* pareceu ser uma opção viável à resolução do problema. A primeira implementação usada foi [42], uma rede neuronal que, com base na palavra anterior, tentava pressupor a palavra seguinte. Esta tentativa não correspondeu ao resultado pretendido, pois o *Output* tinha pouco valor visto que ao passo de avaliar padrões tentava sempre presumir a próxima palavra do padrão apresentado. Além do *Output* ter pouco valor, o tempo de treino da rede era bastante demorado. A documentação do *site Tensorflow* foi alterado aquando da escrita da dissertação já não estando disponível o mesmo algoritmo *online* na documentação do *site*.

Foi tentada outra abordagem usando a biblioteca *TensorFlow*, mas usando a *API Keras*, com a ajuda desta *API* foi possivel fazer o pretendido desde o inicio, um *Output* com valor que avaliasse os padrões de acesso.

A rede neuronal é treinada com várias situações de acesso, algumas delas são consideradas ligações fidedignas e com o padrão correto de acesso, outras são consideradas como possíveis tentativas de ataque. No caso de ser um padrão normal é classificado como (1), no caso de ser um possível ataque (0).

Uma ligação com o padrão correto significa que as *flags* chegaram com a ordem correta, i.e., *3-Way HandShake* após isto são transferidos os dados e a ligação é terminada como demonstrado na figura 2.7. Uma ligação ser considerada um possível ataque significa e.g., foi iniciada a ligação com um *SYN*, houve um *SYN/ACK*, mas não foi respondido com um *ACK*, terminando a ligação por *timeout* com um *RST*. Obviamente uma ligação pode terminar assim e não ser um ataque, o cliente por ter perdido a ligação à *Internet*, o pacote ter sido perdido ou a ligação estar bastante

```
S, 0, 0, 0, 0
S, A, 0, 0, 0
0, A, 0, 0, 0
0, A, 0, 0, 0

0, A, 0, 0, 0
0, A, 0, 0, 0
0, 0, F, 0, 0
0, A, 0, 0, 0

0, 0, F, 0, 0
0, A, 0, 0, 0
```

Figura 3.5 – Exemplo de Dados de entrada para a rede neuronal.

congestionada.

A implementação funciona resumidamente da seguinte forma:

1. Recolha das *flags* da ligação em estudo.
2. Treino da rede neuronal para reconhecer os padrões de treino.
3. *Output* de um valor entre 0 e 1 (ligação passivel de ser um ataque o *Output* será um valor próximo de 0, ligação dentro dos padrões considerandos normais o *Output* será um valor próximo de 1).

Poderá acontecer também o *Output* não se aproximar de nenhum dos valores, neste caso os dados necessitam de ser analisados, para aferir se se trata de uma ligação não conhecida mas segura, ou se será um possível ataque.

Após serem procuradas as *flags* por incompatibilidade do *software* os (1), que significa a presença da *flag* foi alterado para a primeira letra desta. Então os dados de entrada para a rede neuronal ficaram como se apresenta na Figura 3.5

Apesar da solução encontrada para resolver o problema, esta tem uma limitação que consiste no tamanho dos vectores usados, os vectores de treino, de teste, e de

predict, todos têm de ter o mesmo tamanho. Para estes vetores terem tamanhos iguais foram testadas 2 soluções:

1. **Solução A**: Na primeira solução são adicionadas linhas de zeros mas no fim da recolha de dados apenas para uniformizar o número de dados, até ter o mesmo tamanho dos matrizes de treino. Com esta solução pretende-se ver qual a interferência das linhas de zeros no final da ligação.

2. **Solução B**: Na segunda solução é adicionado uma linha de zeros no final de cada três linha de *flags* de maneira a tentar causar menos interferência nos dados. Com esta solução o objectivo será perceber se deste modo a interferência no *Output* será menor que na Solução A.

```
S, 0, 0, 0,0,
S, A, 0 ,0,0,
0, A ,0 ,0,0,
0, A, 0, 0,0,

0 ,A, 0, 0,0,
0, A, 0, 0,0,
0 ,0, F, 0,0,
0, A, 0, 0,0,

0 ,0, F, 0,0,
0, A, 0, 0,0,
0 ,0, 0, 0,0,
0, 0, 0, 0,0,
```

Figura 3.6 – Exemplo da Solução A.

Será também apresentada a solução perfeita em que não são adicionadas linhas de zeros, isto significa que não haverá perturbação por adicionar linhas de zeros. Esta experiência servirá também para ver qual o efeito dos zeros no resultado. Neste teste será considerado que um valor inferior a 0.1 no *Output*, pois em teste de tamanho (12) em que apenas são dadas para análise letras por exemplo o *Output* é próximo

```
S, 0, 0, 0, 0,
S, A, 0 ,0, 0,
0, A ,0 ,0, 0,
0, 0 ,0 ,0, 0,

0, A, 0, 0, 0,
0, A, 0, 0, 0,
0, A, 0, 0, 0,
0, 0, 0, 0, 0,

0, 0, F, 0, 0,
0, A, 0, 0, 0,
0, 0, F, 0, 0,
0, A, 0, 0, 0
```

Figura 3.7 – Exemplo da Solução B.

de 0.1. Logo com a introdução de zeros será expectável que o *Output* seja próximo deste valor.

Nas Figuras 3.6 e 3.7 estão representados exemplos da Solução A e Solução B respectivamente. Ambas as soluções foram treinadas com os mesmos parâmetros, a rede foi treinada com 50 *epochs* de maneira a evitar o *Overfit*.

3.3 Cadeia de Markov

Devido aos problemas encontrados na implementação de uma solução utilizando Rede Neuronais, tal como a impossibilidade de treinar com padrões de vários tamanhos, foi decidido tentar uma implementação recorrendo à Cadeia de Markov para reconhecimento de padrões nas *flags* de uma comunicação usando *TCP*.

Para esta implementação usou-se o *software Matlab*, pela acessibilidade a grafos para a representar a matriz P que já foi referida no Capitulo 2. No *script* implementou-se rotinas para a matriz ser facilmente acedida através de um grafo, neste grafo estão presentes os estados que representam as *flags* que são usadas na transferência de pacotes, além dos estados estão representadas por linhas as probabilidades de

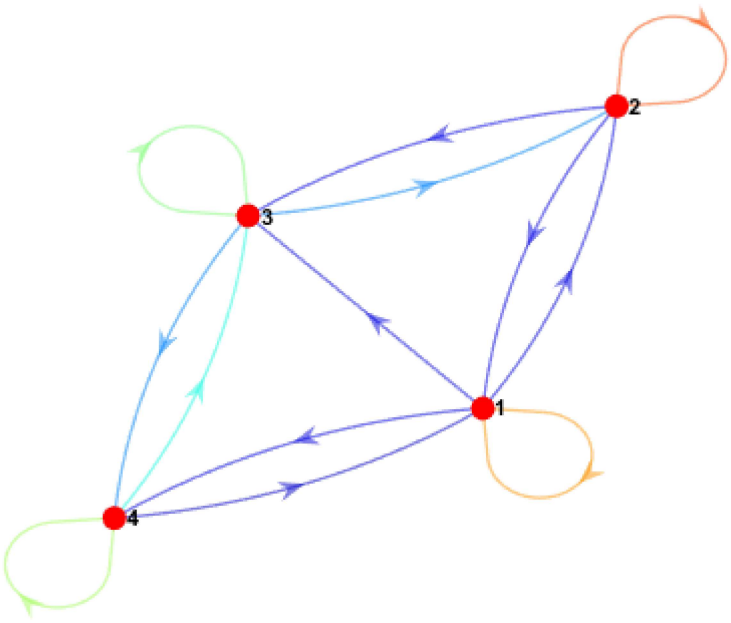

Figura 3.8 – Exemplo de grafo.

transição de estado. No caso da transição são representadas por cores, i.e., quanto mais azul for a seta maior a probabilidade de ser perto de zero, quanto maior a probabilidade de transição mais vermelha a seta se torna. Observando a Figura 3.8 o estado 2 está bastante mais próximo do vermelho logo há uma maior probabilidade de chegando ao estado 2 se manter. Por outro lado este mesmo estado tem outras duas ligações, uma ao estado 3 e outra ao estado 1, ambas a azul, logo estas ligações são de baixa probabilidade.

Como este *software* considera estados diferentes *SYN ACK* este estado foi alterado para apenas **s**.

A probabilidade de mudança de *flag* é representada pela matriz **P** e foi representada com base nos dados que foram recolhidos pelo *Sniffer*. Sendo assim a matriz foi adaptada a uma ligação fidedigna para se perceber a influência das transições na matriz depois dos testes, e avaliar a viabilidade da cadeia de Markov neste tipo de

dados.

O teste é executado usando as letras que correspondem às flags, na matriz de transição. Com estes dados é criado o grafo inicial usando o DTMC (**Discrite Time Markov Chain** da matriz. A Figura 3.8 é um exemplo da DTMC de uma matriz com quatro estados.

São testados os dois tipos de ligação de maneira a perceber qual a influência que têm na matriz de transição. Após isto as matrizes serão comparadas com a matriz original P, e perceber se houve alterações. As *strings* têm no caso da tentativa de intrusão 3 estados e no caso da ligação fidedigna 10 estados. No caso da tentativa de intrusão é uma tentativa de *DDoS*.

Testes e Resultados

Neste capítulo são apresentados os resultados com mais relevância dos testes descritos no capítulo 3, na explicação do funcionamento da aplicação, bem como na apresentação de problemas e possíveis soluções encontradas ao longo do seu desenvolvimento.

4.1 Resultados dos Testes

4.1.1 Tensorflow

Neste subcapítulo serão apresentados, em detalhe os resultados dos testes realizados e quais as soluções encontradas para a resolução do problema em questão. Como já referido no Capitulo 3 foram executadas três tipos de soluções baseadas na mesma biblioteca, *TensorFlow*, com a *API Keras* de maneira a normalizar as matrizes de entrada, enquanto que numa das soluções é adicionada uma linha de zeros para normalizar a entrada de dados no outro são adicionadas também linhas de zeros mas neste caso no fim das *flags* todas. É apresentada a solução onde não há interferência das linhas de zeros. Em ambos os casos foram usados dados do mesmo tamanho

(160) e com o mesmo modelo de compilação, no modelo de sem linhas de zeros o tamanho utilizado foi (80).

Para construção do modelo em *TensorFlow* foram definidos os seguintes parâmetros:

- *loss* - *binary crossentrophy*
- *optimizer* - *Adam*
- *metrics* - *categorical accuracy*

O modelo de compilação foi escolhido pelas características dos dados recolhidos, por exemplo a função de custo foi a *binary crossentropy* pois foi aproxima-se da resposta *Yes/No.*

O teste executado à rede neuronal foi efetuado várias vezes representando um ataque e uma ligação normal. Para este teste a rede foi treinada com dados recolhidos em ambiente fechado, sendo depois injetada com dados reais de ligações efetuadas. Não sendo, portanto, os dados de teste iguais aos de treino, para assim aferir se de facto a rede está a reconhecer os padrões.

Considerou-se que todos os *Outputs* abaixo de 0.1 são considerados ataques e todos os valores acima disto são considerados ligações dentro dos padrões normais.

O resultado ideal a apresentar é as comunicações possuirem todas o mesmo tamanho, e ser um tamanho *Standard* e não ser necessário acrescentar nada para a ligação ser avaliada pela rede neuronal. Esta foi a primeira experiência que foi realizada, todas as comunicações usavam vectores de tamanhos fixos (80). Na Figura 4.1 apresenta-se o *Output* dos dados que a rede deveria considerar como possível tentativa de intrusão.

Como se pode verificar pela Tabela 4.1 e pela Figura 4.1 os resultados da ligação que seria considerada uma possível tentativa de intrusão são praticamente idênticos, sendo que em 92% dos casos a rede considerou a ligação como uma tentativa

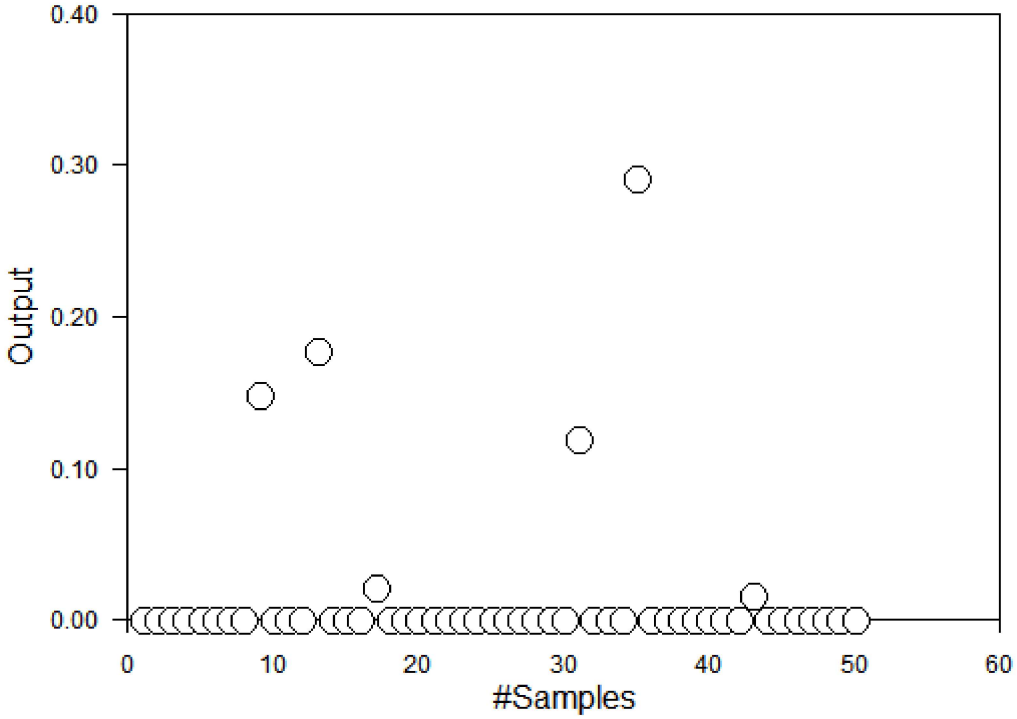

Figura 4.1 – Output dos dados de tentativa de intrusão em situação ideal.

de intrusão. Os valores desta tabela encontram-se em anexo A.1 sendo a tabela apresentada a % dos 50 testes.

Como se pode verificar também pela tabela 4.1 no caso em que a ligação obedece aos padrões de comunicação em 88% das vezes esse *Output* foi considerado como normal.

	Teste Normal	Teste ataque
Normais	88%	8%
Tentativa de Intrusão	12%	92%

Tabela 4.1 – Resultados dos 50 testes em condições ideais em%.

Visto a limitação no treino da rede neuronal e todas as ligações possuirem diferentes tamanhos foram adicionadas linhas de zeros para suprimir esta limitação.

A- Solução normalizada com zeros no fim

Como já referido à limitação de treino da rede neuronal ser de matrizes todas do mesmo tamanho foram adicionadas linhas de zeros no final da comunicação para uniformizar todos os dados injetados e de teste à rede neuronal.

Foram executados 50 testes à rede neuronal, sempre com os mesmos dados para análise para avaliar intensivamente a consistência do valor de *Output*. Como se pode ver na Figura 4.2 o valor de *Output* de dados de tentativa de intrusão situa-se sempre praticamente no mesmo valor cerca de 0.25. Isto acontece porque o valor da Função de custo como se pode ver na Figura 4.3 é quase sempre uniforme, nunca abaixo dos 0.5, valor muito alto para o treino de uma rede neuronal.

Podia-se estar no caso que a rede não está a obter treino suficiente, *Underfit*, mas foram alteradas as épocas para números bastante superiores e a Função de Custo continuou praticamente igual e os *outputs* foram sempre com números idênticos aos indicados nos gráficos. Como a Função de Custo é sempre bastante alta os valores de *Output* são de valor muito duvidoso pois a rede não está a conseguir detetar padrões, muito provavelmente porque em ambos os casos: tentativa de intrusão e ligação normal as ligações têm bastantes linhas no fim de zeros, passando os padrões que se querem analisar a ser secundários.

Portanto todos os *outputs* têm um valor de 0.25 em média para todos os casos em análise, isto quer dizer que esteja o servidor sob ataque ou esteja na presença de uma ligação normal o valor é o mesmo sendo assim impossível detetar o tipo de ligação que está em análise. Esta solução fica assim inviabilizada como se pode provar pelas Figuras 4.3 e 4.2. Independentemente dos dados de entrada todos os valores de saída são iguais ainda assim visto que *Keras* é bastante usado para detetar padrões testou-se uma outra solução de maneira a testar se os resultados seriam passíveis de análise e seriam detetados padrões.

B- Solução normalizada com zeros a cada 3 flags

Visto a solução acima exposta não apresentar resultados satisfatórios everedou-se por outra solução sabendo já as limitações que se tinha. Neste caso as linhas de zeros foram adicionadas no final de cada linha de 3 *flags*, e fez-se também como na solução

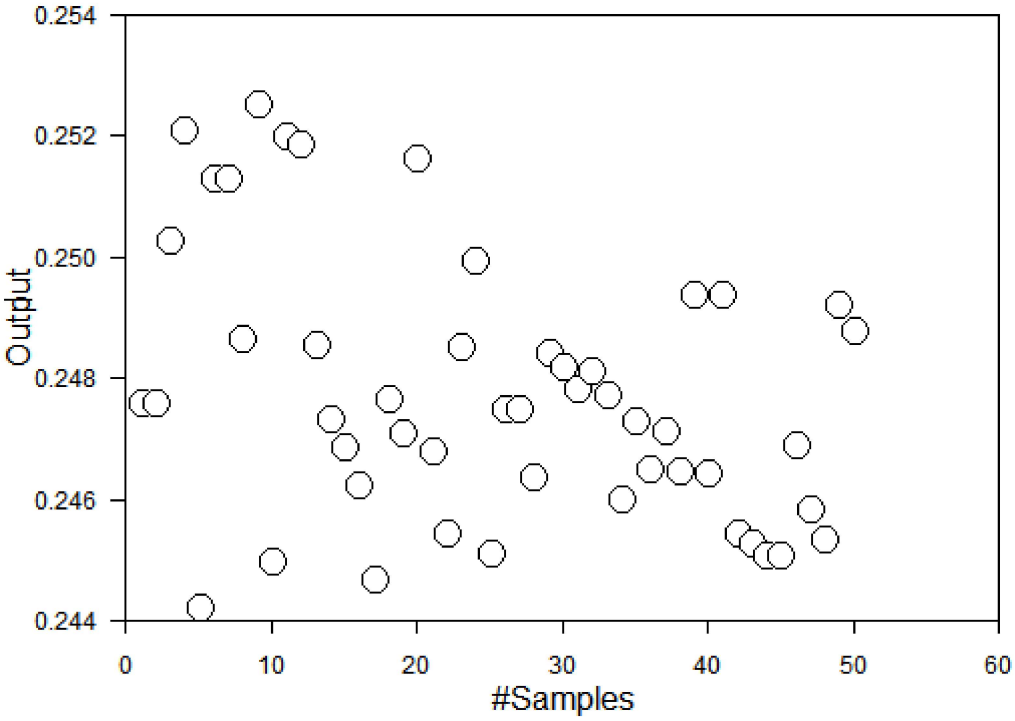

Figura 4.2 – Output do dados de tentativa de intrusão-Solução A.

	Teste Normal	Teste ataque
Normais	58%	14%
Tentativa de intrusão	42%	86%

Tabela 4.2 – Resultados dos 50 testes em % com linhas de (0)-Solução B.

A 50 testes nas mesmas condições com a mesma configuração, de maneira também a poderem ser comparados caso fosse o caso, tal como em cima foram criados dados de treino e de validação diferentes para a rede ter acesso a vários exemplos. Tal como se pode ver na Figura 4.4 onde estão representados os resultados dos dados que foram dados para analisar como tentativa de intrusão, têm praticamente todos valores bastantes baixos, sendo que os ataques foram treinados para ter valor o mais próximo de (0) possível, então está de acordo com o pretendido.

Nos 50 testes efectuados tem-se que apenas 14% dos dados que a rede neuronal devia avaliar como tentativa de intrusão foram classificados como ligações normais, este

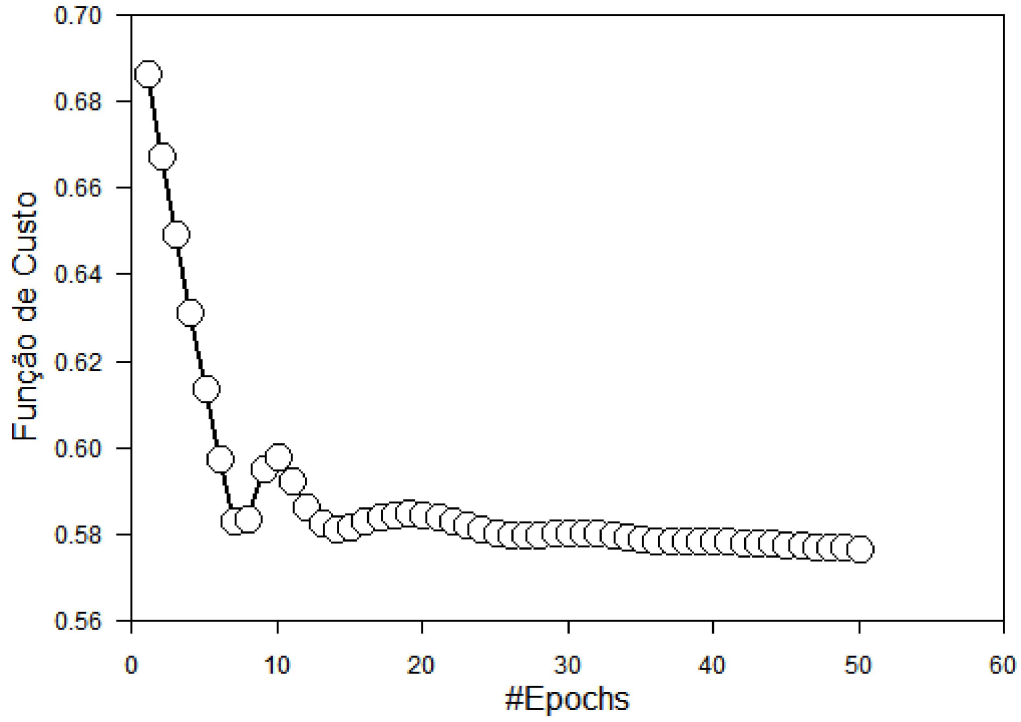

Figura 4.3 – Evolução da função de custo ao longo do treino.

valor representa que apenas 7 destas ligações foram considerados normais como se pode observar na Tabela 4.2. Os valores desta tabela encontram-se em anexo A.2 sendo estes dados apresentados de mais fácil compreensão.

Como se pode ver na Figura 4.4 há um valor a cor diferente, esse valor está a cor diferente pois apesar de ser um ataque a rede considerou-o com um *Output* de valor alto, indicativo de que o classificou como uma ligação normal. Neste valor foi acompanhada a evolução da Função de Custo para comprovar que com uma Função de Custo alta os valores de *Output* são bastante afetados. Como se pode verificar pela Figura 4.5 a Função de Custo vai sempre tendendo para valores mínimos, até que por volta da época 30 a Função de Custo sobe para valores elevados. A Função de Custo quando é elevada faz com que o *Output* tenha um valor pouco credível.

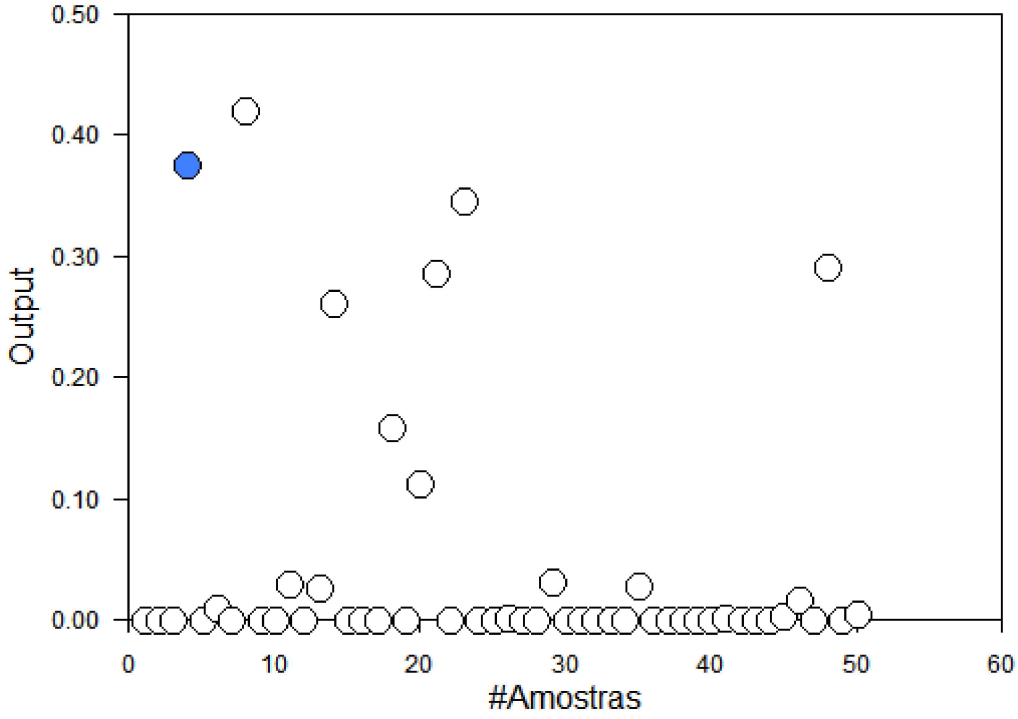

Figura 4.4 – *Output* em dados de tentativa de intrusão-Solução B.

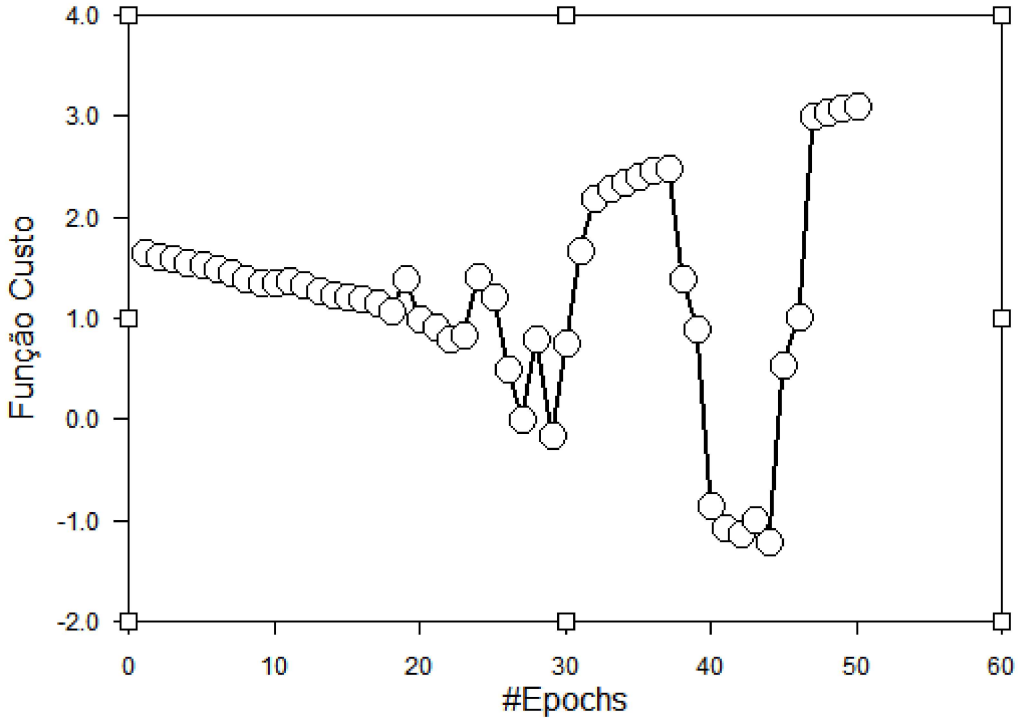

Figura 4.5 – Evolução da Função de Custo.

Analisando a Tabela 4.2 pode-se também observar que na ligação que a rede deveria considerar como uma ligação dentro dos padrões normais apenas em 58% dos casos foi assim considerada, nos outros 42% foi considerada como um possível ataque.

4.1.2 Modelo de Markov

Neste teste foram usadas 10 strings que foram retiradas do para testar a possibilidade de uso deste tipo de modelo. De relembrar que o modelo de Markov apenas tem em consideração o presente estado em que se encontra e apenas isso é considerado para o estado seguinte.

A matriz de transição foi calculada com os dados que se tinham recolhido pelo *sniffer* em ambiente fechado. Trata-se de uma matriz com 6 linhas e 6 colunas com as *flags SYN, ACK, FIN, RST, PUSH, SYN/ACK*, em todos os casos foi apenas usado a letra inicial. Cada ligação ficticia tem 10 transições de estado no caso da ligação normal e 3 estados no caso da tentativa de intrusão, pode acontecer o estado ser mantido no caso da *flag ACK*.

Na figura 4.7 é apresentada a matriz de transição usada nos testes, dessa matriz de transição foi criado o grafo da figura 4.6 . Como se pode observar mais facilmente através do grafo esta cadeia de Markov é aperiodica pois não se consegue atingir um mesmo estado através de outro com o mesmo periodo, transiente e redutivel.

Através do grafo, pode-se observar também que o estado 1 apenas pode ir para o estado 2, isto porque depois da *flag SYN* apenas pode haver a *flag SYN / ACK*, mesmo sendo um possível ataque. Após este estado o grafo tem acesso a todos os estados não sendo possível o acesso ao estado 1 de novo. Também através da observação do grafo observa-se que apenas o estado 3 voltar a ele próprio, visto que corresponde à *flag ACK* e aquando da receção de dados esta é usada bastantes vezes tanto pelo servidor como pelo cliente .

```
P=[0.0 1.0 0.0 0.0 0.0 0.0 ;...
   0.0 0.0 0.9 0.0 0.0 0.1;...
   0.0 0.0 0.7 0.2 0.05 0.05;...
   0.0 0.0 0.9 0.0 0.0 0.1;...
   0.0 0.0 1.0 0.0 0.0 0.0;...
   0.0 0.0 1.0 0.0 0.0 0.0];
```

Figura 4.7 – Matriz transição original.

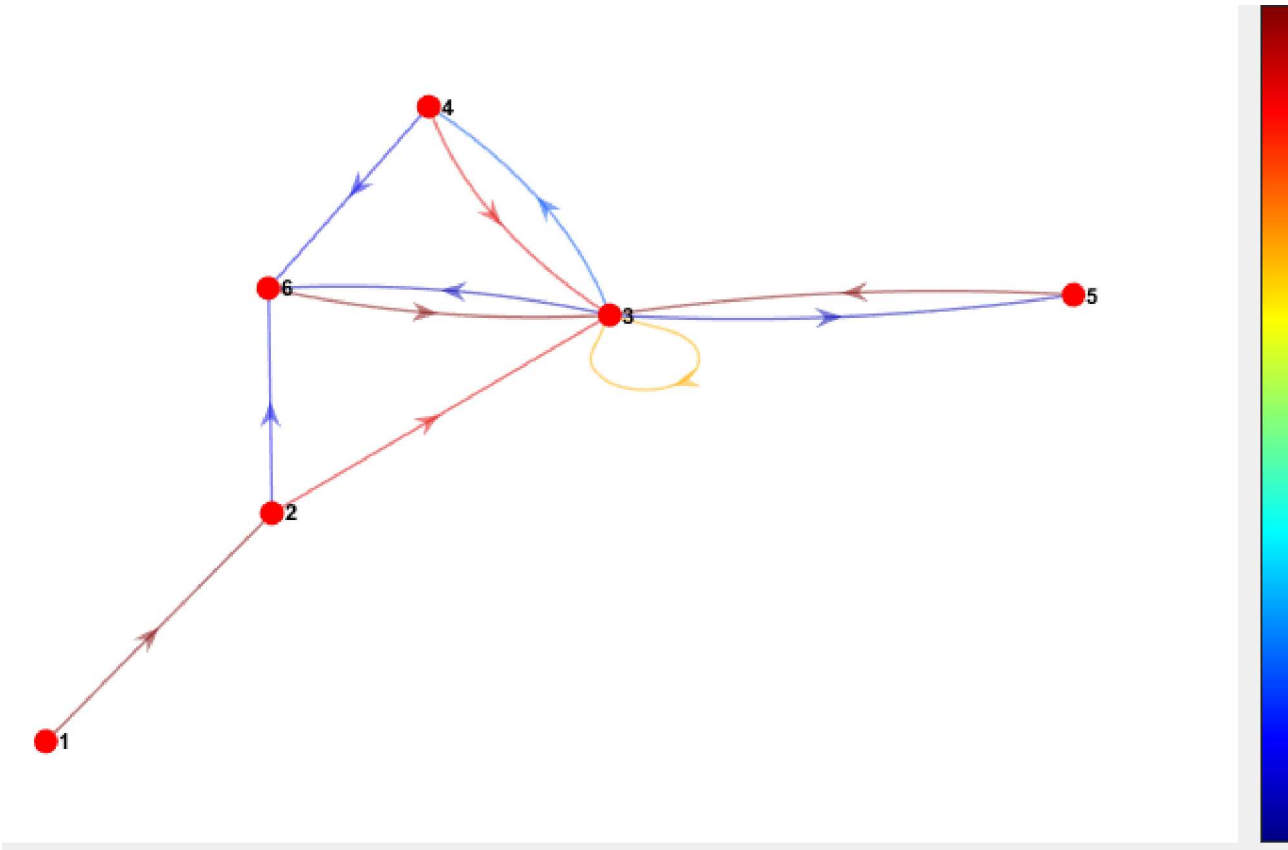

Figura 4.6 – Grafo da matriz de transição original.

Na Figura 4.8 e 4.9 estão representados os dados que serviram de teste ao Modelo de Markov, na Figura 4.8 estão as *flags* que serviram de teste como ligação fidedigna, sendo que na Figura 4.9 estão os dados da tentativa de intrusão.

Na Figura 4.10 tem-se a matriz de transição após o teste com os dados de ligação

```
Str=['SsAAPAFAFA'
     'SsAAPAFAFA'
     'SsAAAAFAFA'
     'SsAAAAFAFA'
     'SsAAPAFAFA'
     'SsAAAAFAFA'
     'SsAAPAFAFA'
     'SsAAAAFAFA'
     'SsAAAAFAFA'
     'SsAAAAFAFA']
```

Figura 4.8 – Exemplo de dados de ligação normal.

```
'SsR'
'SsR'
'SsR'
'SsR'
'SsR'
'SsR'
'SsR'
'SsR'
'SsR'
'SsR'
```

Figura 4.9 – Exemplo de dados de tentativa de intrusão.

```
Pe =

         0    1.0000         0         0         0         0
         0         0    1.0000         0         0         0
         0         0    0.4783    0.4348    0.0870         0
         0         0    1.0000         0         0         0
         0         0    1.0000         0         0         0
         0         0         0         0         0         0
```

Figura 4.10 – Matriz de transição após teste (ligação normal).

```
Pe =

     0     1     0     0     0     0
     0     0     0     0     0     1
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
```

Figura 4.11 – Matriz de transição após teste (tentativa de intrusão).

```
1- Str=['SsAAFAFAAAA']

2- Str=['SsAAAAAFAFA']
```

Figura 4.12 – Exemplo de dados em teste.

normal. Comparando com a matriz da Figura 4.7 nota-se algumas diferenças apesar de ter sido testada com uma ligação normal. Na Figura 4.11 observa-se a matriz transição após o teste com a ligação possivel tentativa de intrusão, nesta matriz as alterações são significativas alterando bastante a matriz quando comparada com a matriz da Figura 4.7.

Contudo para comprovar se o uso da Cadeia de Markov seria mesmo passível de ser usada para este tipo de tentativas foi realizado outro tipo de teste. Neste teste foram testadas 2 vetores que estão na imagem 4.12. No número 1 tem-se uma ligação que não acaba como uma ligação normal e no número 2 uma ligação normal onde começa com o *3-Way Handshake* tem a transferência de dados e após isto termina com a série de *flags FIN*. Como se pode ver pela imagem 4.12 as *flags* usadas são as mesmas apesar de estarem em posições temporais diferentes.

Como se pode observar pelas figuras 4.14 e 4.13, apesar dos dados serem bastante diferentes as matrizes são quase idênticas. Com isto fica provado que a Cadeia de Markov não pode ser usada para identificar ataques através de *flags*, porque não vê a ligação como um todo, mas apenas avalia o estado presente, não tendo em conta

```
Pe =

         0    1.0000         0         0         0         0
         0         0    1.0000         0         0         0
         0         0    0.6667    0.3333         0         0
         0         0    1.0000         0         0         0
         0         0         0         0         0         0
         0         0         0         0         0         0
```

Figura 4.13 – Matriz de transição número 2.

```
Pe =

         0    1.0000         0         0         0         0
         0         0    1.0000         0         0         0
         0         0    0.6000    0.4000         0         0
         0         0    1.0000         0         0         0
         0         0         0         0         0         0
         0         0         0         0         0         0
```

Figura 4.14 – Matriz de transição número 1.

o passado.

Problemas encontrados

Devido à recolha de dados demorar algum tempo foi preciso adicionar algumas condicionantes para que a aplicação não seja interrompida.

- Devido às impossibilidades de introduzir dados de tamanhos diferentes foram encontradas duas soluções e foram comparadas, isto acontece porque a rede é treinada com um tamanho fixo ficando assim impossibilitada de se adaptar a outros tamanhos de dados que poderiam ser injectados. Para a resolução deste problema foram adicionadas linhas aos dados de entrada fazendo assim com que os dados tenham tamanhos iguais aos que a rede foi treinada.
- Melhoramento da Função de Custo - A Função de Custo tornou-se o maior

problema neste trabalho já que tem de ser controlada para que os resultados sejam fidedignos, para isto usaram-se reguladores para tentar ao máximo conter o valor da Função de Custo fazendo assim que os resultados sejam melhores e a % de erro no *Output* seja cada menor.

- Incapacidade de funcionar em todos os dispositivos pois usa *TcpDump*, sem este software instalado não há recolha de dados e impossibilita a análise dos dados.

Conclusões e Trabalho Futuro

5.1 Conclusões

O crescimento da *Internet* e a necessidade desta leva a que a segurança seja cada vez maior, o desconhecimento dos perigos por parte dos utilizadores leva a que seja fulcral haver segurança dos sistemas e avisos de possibilidade de intrusão para que o uso dos utilizadores seja o mais seguro possível. Perante isto empresas e utilizadores têm de investir na segurança das informações pessoais e confidenciais e todos os dias surgem ameaças que têm de ser combatidas.

Visto isto têm aparecido novas metodologias de proteção com o objetivo de bloquear ameaças que já foram confirmados pelo *software* de proteção. O sistema implementado nesta dissertação tem o objetivo de realizar a proteção de uma máquina atendendo à maneira como é realizada a conexão. Através das *flags* será possível aferir se uma ligação é fidedigna ou não comparando com os dados que a rede neuronal já possui.

Sendo assim, após a análise há um *Output* sobre a classificação que a rede neuronal deu à ligação em estudo. Os testes foram realizados em ambiente fechado na Universidade de Trás-os-Montes e Alto Douro com vários tipos de ligações a serem testadas.

Do ponto de vista do utilizador raramente será notado que houve um acesso não fidedigno ao computador, pois na maioria das vezes o *hacker* não bloqueia o acesso do utilizador aos recursos.

No caso da solução de Cadeia de Markov é uma solução tem alterações significativas na matriz transição, mas que tem limitações próprios da cadeia, neste caso essa característica impede que o padrão seja visto como um todo, apenas o estado actual conta para a análise. Com isto ficou provado que a Cadeia de Markov não pode ser utilizada para detetar o padrão de *flags*.

Apesar de poder ser necessário melhorar alguns aspectos descritos no subcapítulo *Trabalho Futuro*, ficou, no entanto, demonstrado que com uma rede neuronal é possível classificar as ligações através de padrões de *flags*. Este método em nada substitui a inspeção dos pacotes já que com esse método tem-se uma maior certeza dos dados recebidos.

Com isto pode-se concluir que a solução encontrada satisfaz o que foi proposto nesta dissertação, ficando assim outras abordagens para um trabalho futuro que possam solucionar algumas limitações abaixo expostas.

5.2 Trabalho Futuro

A aplicação que foi desenvolvida para a solução do problema atinge o objectivo da dissertação em parte ficando ainda reservado para o futuro algum trabalho para a otimizar e melhorar a maneira como os dados são tratados. Sendo assim será necessário:

- Melhorar a forma como os dados são recolhidos otimizando a recolha conforme o tráfego, i.e., neste momento são recolhidos 1000 pacotes e só ao final deste número são armazenados. Esta situação em algumas redes é bastante tempo para recolher este número de pacotes, ao passo que em outras redes em menos de 1 segundo há a passagem deste número de pacotes na rede.

- Apesar desta solução funcionar, apenas aceita vetores com constante pré-definida. Um melhoramento a implementar no futuro seria arranjar uma solução que detete padrões em vetores de diferentes tamanhos e assim poderem ser analisados os dados em bruto como foram recolhidos como por exemplo a utilização de *Fully Convolutional Networks(FCN)* e de *Spatial Pyramid Pooling(SPP).*

- Recolha do endereço *IP* que ataca a rede e adicionar este endereço *IP* a uma *blacklist* de maneira a que mais nenhuma ligação do endereço *IP* em questão seja preventivamente aceite. Apesar da facilidade da mudança de endereço *IP* seria uma medida preventiva.

- Descartar resultados em que a *loss* seja grande, ou então através de reguladores para controlar melhor a *loss*, já que este factor influencia muito o resultado final dos dados para análise.

- Em complemento ao ponto acima mencionado, descartar testes em que a *loss* seja grande e testar de novo os dados, isto faria com que apesar de demorar mais tempo na análise, o *Output* seria credível.

Anexos

Número de Teste	Teste Normal	Teste ataque	Loss
1	0.67644227	1.6305578e-06	-2.2158
2	0.35790956	5.7493206e-05	-2.1253
3	0.51519406	0.00019896	-1.2548
4	0.5180062	8.91897e-06	-2.2186
5	0.40077513	2.2415485e-05	-2.3548
6	2.0733163e-05	2.1082866e-05	2.1453
7	0.33042577	3.7822613e-06	-2.1869
8	0.37939638	1.2451033e-05	-0.0068
9	0.47561452	0.14861247	0.3579
10	0.44953427	0.00014165	0.0625
11	7.380921e-05	7.357253e-05	3.3541
12	0.2988691	1.6615e-05	-0.01584
13	0.40781912	0.17704543	0.3482
14	0.40639272	9.397101e-06	-3.1535
15	0.3409643	6.866532e-06	-2.5783
16	0.4525297	7.721767e-06	-2.6547
17	0.5806708	0.02194764	-0.0258
18	0.37120074	9.4190555e-06	-3.2975
19	0.37101665	3.0297439e-05	-2.8563
20	0.45966405	7.2368794e-06	-3.1635
21	0.50225395	9.655023e-07	-3.3589
22	0.35311696	2.1657768e-05	-2.5763
23	0.41302562	2.3439843e-05	-2.2965
24	0.4751702	1.6463379e-06	-1.9536
25	0.24573845	1.606826e-05	-0.0092
26	0.54545474	6.672164e-06	-3.2546
27	0.45596853	1.4659821e-05	-2.8357
28	0.1297195	6.918176e-06	-0.0025
29	0.23997626	4.0797742e-05	-1.1563
30	0.37025833	1.3511601e-05	-1.9715
31	0.50832874	0.1190336	0.3462
32	0.36950913	1.2340857e-05	-1.4256
33	0.6101127	0.00045095	0.0143
34	0.4174536	1.0974473e-05	-1.7624

Número de Teste	Teste Normal	Teste ataque	Loss
35	0.4985735	0.2914927	0.3854
36	0.5056791	7.9403935e-06	-3.2548
37	2.2684226e-05	2.2817356e-05	2.5214
38	0.4692053	1.6872074e-05	-2.5382
39	0.41474003	1.4943083e-05	-2.6294
40	0.43710145	1.5183278e-05	-2.9635
41	0.4528278	7.649688e-06	-3.2453
42	1.3039867e-05	1.3252832e-05	3.0219
43	0.46687213	0.01581127	0.1267
44	2.3439554e-05	2.3439666e-05	2.9153
45	0.45133442	1.3984028e-05	-2.6524
46	2.9135158e-06	2.9181456e-06	2.5237
47	0.47269467	1.6260592e-06	-3.0253
48	0.32703194	4.7132773e-05	-2.1293
49	0.38610557	2.0793577e-05	-2.4582
50	0.18531676	9.3528564e-05	-1.4298

Tabela A.1 – Resultados dos 50 testes em condições ideais referentes à 4.1

Número de Teste	Teste Normal	Teste ataque	Loss
1	0.6643925	1.4736197e-08	-2.2537
2	0.4979429	0.00023335	0.1952
3	0.48200342	0.00020505	0.1589
4	0.4770255	0.37506005	3.1101
5	6.632526e-05	6.648128e-05	2.2549
6	0.01002023	0.00988341	1.1249
7	0.36920702	0.00026368	0.2896
8	0.48153713	0.4197652	2.9148
9	0.00063396	0.00061542	1.5249
10	0.46755973	0.0005421	0.2058
11	0.03002465	0.02956536	1.1549
12	0.49925202	0.00037852	-1.3655
13	0.02606467	0.02589274	1.3859
14	0.4229664	0.2615925	2.2546
15	0.0006153	0.00053179	1.0257
16	0.00020438	0.00020362	1.2371
17	0.4272106	0.00081015	-1.2497
18	0.46036607	0.15841562	-1.1283
19	0.5039155	1.2088244e-05	-2.4298
20	0.51218367	0.11252602	-1.1753
21	0.50959283	0.28566185	2.4196
22	6.249885e-07	6.267146e-07	-2.9527
23	0.46638823	0.3456704	2.2695
24	1.203633e-05	1.2134051e-05	-2.5196
25	0.51108044	1.9480703e-06	-3.1538
26	0.00123058	0.00119698	1.1673
27	1.166345e-06	1.1626059e-06	-2.4529
28	2.0427136e-05	2.0524027e-05	-3.2496
29	0.03289605	0.03176127	1.0843
30	0.43185896	1.6198159e-05	-3.1852
31	1.8680247e-06	1.8498923e-06	3.1738
32	0.5712814	1.2073967e-06	-3.0273
33	0.5473122	1.1623692e-05	-3.0246
34	1.7940216e-05	1.6868518e-05	-2.9438

Número de Teste	Teste Normal	Teste ataque	Loss
35	0.47942623	0.02787809	-1.0182
36	0.5942596	1.3356329e-06	-3.1673
37	0.51436853	5.7698894e-06	2.7651
38	1.6129744e-07	1.6099146e-07	-2.8491
39	0.42018867	6.034894e-06	-2.9482
40	0.5261534	2.196254e-06	-3.1492
41	0.00162963	0.00159444	1.0257
42	0.4097318	3.20257e-05	-2.5179
43	0.3523939	7.886946e-05	-2.6481
44	0.59224695	4.1817574e-07	-2.5834
45	0.0033901	0.00334266	1.0128
46	0.01709065	0.01666633	1.0529
47	2.0138862e-06	2.019972e-06	-3.1025
48	0.4827169	0.29069933	-1.5294
49	0.44505847	1.2189363e-05	-2.4934
50	0.005301	0.00521836	1.0369

Tabela A.2 – Resultados dos 50 testes da Solução B referentes à 4.2

Referências bibliográficas

[1] Clement J. https://www.statista.com/statistics/264473/number-of-internet-hosts-in-the-domain-name-system/, 2019. Last accessed 15 September 2019. xix, 10

[2] M. Deshpande. https://pythonmachinelearning.pro/perceptrons-the-first-neural-networks/. Last accessed 15 October 2019. xix, 23, 24

[3] https://www.cisco.com/c/en/us/products/security/firewalls/what-is-a-firewall.html, 2018. Last accessed 29 September 2019. 2

[4] CloudFlare. https://www.cloudflare.com/learning/ddos/glossary/open-systems-interconnection-model-osi/. Last accessed 15 October 2019. 5, 6

[5] Feup. Pequena História da Internet. https://paginas.fe.up.pt/~mrs01003/TCP_IP.htm. Last accessed 20 February 2020. 7

[6] C.E. Douglas. Internetworking with tcp/ip principles, protocols, and architecture. volume 4, pages 190–230. Prentice Hall, 200. 7, 8, 21

[7] S. Zander ; T. Nguyen ; G. Armitage. Automated traffic classification and application identification using machine learning. In *The IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05)*, 2005. 8, 9

[8] M. Kojo K. Ono M. Stiemerling L. Eggert A. Melnikov W. Eddy A. Zimmermann B. Trammell J. Touch; E. Lear, A. Mankin and J. Iyengar. `https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml?&page=135`, 2020. Last accessed 15 October 2019. 9

[9] Bellovin S.M. Cheswick W.R. Firewalls and internet security: Repelling the wily hacker. volume 2. Addison-Wesley Professional, 2003. 10

[10] Shinder T. Shimonski R.., Shinder D. Best damn firewall book period. volume 1, pages 1–60. Syngress Publishing, 2003. 10, 11, 13, 14, 15, 16, 17

[11] Cisco. What is a Firewall? `https://www.cisco.com/c/en/us/products/security/firewalls/what-is-a-firewall.html`. Last accessed 10 April 2020. 11, 12

[12] PaloAlto Networks. What is a Firewall? `https://www.paloaltonetworks.com/cyberpedia/what-is-a-firewall`. Last accessed 10 April 2020. 13

[13] Daniel Kligerman Doug Maxwell Cherie Amon Allen Keele Drew Simonis, Corey S. Pincock. Check point next generation security administration. volume 1, pages 1–40. Syngress Publishing, 2002. 13

[14] J. J. Vicente. `https://www.cncs.gov.pt/content/files/brochura_2.pdf`, 2017. Last accessed 15 September 2019. 13

[15] M. Rouse. `https://searchsecurity.techtarget.com/definition/checksum`. Last accessed 15 September 2019. 13

[16] Y. Victor M. Jones M. Wenzel, L. Latham. `https://docs.microsoft.com/en-us/dotnet/standard/security/`

ensuring-data-integrity-with-hash-codes. Last accessed 15 September 2019. 13

[17] IBM. https://www.ibm.com/support/knowledgecenter/en/SSB23S_1.1.0.13/gtps7/s7pkey.html. Last accessed 15 October 2019. 14

[18] J. Regan. https://www.avg.com/pt/signal/what-is-malware. Last accessed 15 October 2019. 15

[19] Kaspersky. https://www.kaspersky.com.br/resource-center/definitions/spear-phishing. Last accessed 15 October 2019. 15

[20] Avast. https://www.avast.com/pt-br/c-spoofing. Last accessed 15 October 2019. 16

[21] Zahid M. Siddiqui M.R. Mohammed A. Q., Iqbal A. Network Traffic Analysis and Intrusion Detection Using Packet Sniffer. In *2010 Second Internacional Conference on Communication Software and Networks*, 2010. 17

[22] Magers Daniel. Packet sniffing: An integral part of network defense. Sans Institute, 2002. 17

[23] https://www.keycdn.com/support/tcp-flags, 2018. Last accessed 29 September 2019. 19

[24] SujanRai. TCP 3-Way Handshake Process. https://www.geeksforgeeks.org/tcp-3-way-handshake-process. Last accessed 25 September 2019. 20

[25] Howard et al. AsynTask. https://patentimages.storage.googleapis.com/e9/8b/a4/eb30e97d5a07bc/US7284272B2.pdf, 2007. Last accessed 25 September 2019. 21

[26] Jonas Demuro. Cognitive science applied to computer learning theory. https://www.techradar.com/news/what-is-a-neural-network, 2018. Last accessed 29 September 2019. 23, 25

[27] What is a Neural Network. https://deepai.org/machine-learning-glossary-and-terms/neural-network. Last accessed 29 September 2019. 23

[28] C. Brownlee. https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms, 2016. Last accessed 15 September 2019. 25

[29] TensorFlow. https://www.tensorflow.org/api_docs/python/tf/keras/activations/sigmoid. Last accessed 15 October 2019. 26

[30] TensorFlow. https://www.tensorflow.org/api_docs/python/tf/keras/activations/tanh. Last accessed 15 October 2019. 26

[31] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, pages 807–815. Omnipress, 2010. 27

[32] A. Bordes X. Glorot and Y. Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–322, 2011. 27

[33] J. Koutník B. R. Steunebrink K. Greff, R. K. Srivastava and J. Schmidhuber. Lstm:a search space odyssey. In *IEEE Transactions on Neural Networks and Learning Systems*, pages 2221–2230, 2017. 28

[34] http://colah.github.io/posts/2015-08-Understanding-LSTMs/. Last accessed 15 October 2019. 28

[35] TensorFlow. https://www.tensorflow.org/tutorials/keras/overfit_and_underfit. Last accessed 29 September 2019. 31, 32

[36] https://marketbusinessnews.com/financial-glossary/neural-networks/k. Last accessed 29 September 2019. 33

[37] Pal S. Gulli A. Deep learning with keras. volume 1, pages 16–32. Packt Publishing Ltd, 2017. 33

[38] Ghani M.A.H.A Haris S.H.C., Ahmad R.B. Detecting TCP SYN Flood Attack Based on Anomaly Detection. `https://ieeexplore.ieee.org/abstract/document/5635797`, 2010. Last accessed 29 September 2019. 35

[39] Murata M. Ohsita Y., Ata S. Detecting distributed denial-of-service attacks by analyzing TCP SYN packets statistically. In *IEEE Global Telecommunications Conference, 2004. GLOBECOM '04.*, 2004. 36

[40] Sproull T.S. Lockwood J.W. Dharmapurikar S., Krishnamurthy P. Deep packet inspection using parallel bloom filters. *IEEE Computer Society*, 2004. 37

[41] Catalyst 4500 series switch cisco ios software configuration guide. `https://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst4500/12-2/25ew/configuration/guide/conf/l3_int.html`. Last accessed 15 October 2019. 41

[42] TCP Connection Termination | FIN Segment. `https://www.tensorflow.org/tutorials/text/text_classification_rnn`. Last accessed 25 September 2019. 44