UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

RODRIGO APARECIDO DA SILVA BRAGA

RECONHECIMENTO DE TRÁFEGO PEER-TO-PEER UTILIZANDO REDES NEURAIS

Dissertação submetida à Universidade Federal de Itajubá
para a obtenção do Grau de Mestre em Engenharia Elétrica.

Orientador:
Otávio Augusto Salgado Carpinteiro

Itajubá
2007

RODRIGO APARECIDO DA SILVA BRAGA

RECONHECIMENTO DE TRÁFEGO PEER-TO-PEER UTILIZANDO REDES NEURAIS

Esta Dissertação foi julgada adequada para obtenção do Título de "Mestre em Ciências em Engenharia Elétrica", área de concentração Automação em Sistemas Elétricos Industriais e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica.

Itajubá, 01 de junho de 2007.

_____

Prof. Leonardo de Mello Honório, Dr.
Coordenador do Curso

Banca Examinadora:

_____

Prof. Otávio Augusto Salgado Carpinteiro, Dr.
Universidade Federal de Itajubá – UNIFEI
Orientador

_____

Prof. Benedito Isaías Lima Lopes, Dr.
Universidade Federal de Itajubá – UNIFEI
Examinador Interno

_____

Prof. José Demisio Simões da Silva, Dr.
Instituto Nacional de Pesquisas Espaciais – INPE
Examinador Externo

Aos meus pais Pedro e Janete
pelo apoio, exemplo de vida
e amor incondicional.

AGRADECIMENTOS

Primeiramente agradeço a Deus pela saúde e força de vontade que tem me proporcionado durante toda a vida.

Ao Prof. Dr. Otávio Augusto Salgado Carpinteiro pelo incentivo, paciência e sabedoria na condução deste trabalho. Pessoa pela qual criei muito respeito e admiração.

Aos pesquisadores do GRES – Grupo de Redes e Engenharia de Software da Universidade Federal de Itajubá pelo companheirismo e experiências compartilhadas, em especial ao Prof. Dr. Isaías Lima pelo auxílio nas implementações no Matlab e Prof. Dr. Edmilson Marmo Moreira assistência na revisão do trabalho.

Aos amigos Roberto Netto, Caio Flausino, Cezar Sant'Anna, Thiago Mikail, Jonas Borges e Rodrigo Rodrigues que contribuíram definitivamente na concepção deste trabalho. Obrigado pelas discussões nas quais ainda trocamos muitas experiências.

Aos meus companheiros de trabalho da Incubadora de Empresas de Base Tecnológica de Itajubá – INCIT, pela ceção da rede local para a coleta de dados.

A minha companheira Ana Flávia pela compreensão, paciência, encorajamento, tolerância e otimismo dispensados todos os dias.

Aos meus companheiros de república pelo apoio, incentivo e bons momentos que vivi durante minha graduação. Amigos para sempre.

Ao programa de pós-graduação da Universidade Federal de Itajubá, pela oportunidade e a CAPES pelo auxilio financeiro durante o primeiro ano de pesquisa.

Finalmente, gostaria de agradecer a todos aqueles que não contribuindo com este trabalho, não me perguntaram nada a respeito, proporcionando paz para a confecção deste.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E REDUÇÕES

ADSL – Asymmetric Digital Subscriber Line

API – Application Programming Interface

CLI – Command Line Interface

DNS – Domain Name Server

FTP – File Transfer Protocol

HTTP – Hyper Text Transfer Protocol

HTTPS – Hyper Text Transfer Protocol Secure

IDS – Intrusion Detection System

IOS – Interconnection Operation System

IP – Internet Protocol

ISP – Internet Service Provider

MLP – Multi Layer Perceptron

NAT – Network Address Translation

OSI – Open System Interconection

P2P – Peer-to-peer

POP – Post Office Protocol

PoP – Point of Presence

QoS – Quality of Service

RNP – Rede Nacional de Pesquisa

SMTP – Simple Mail Transfer Protocol

SSH – Secure Shell

TCP – Transmission Control Protocol

ToS – Type of Service

UDP – User Datagram Protocol

WAN – Wide Area Network

RESUMO

A ampla utilização das aplicações *peer-to-peer* – P2P – para o compartilhamento de arquivos tem causado problemas como, por exemplo, à possibilidade de acesso à materiais protegidos por direitos autorais, transmissão de vírus, *adwares* e *spywares* e, também, ocasionando

ABSTRACT

The ample use peer-to-peer applications - P2P - for the sharing archives has caused many problems with regard to the availability of access to materials protected for copyrights, transmission of virus, adwares and spywares and, also, have caused problems for administrators and manager of large computer networks, since the applications P2P are great consumers of bandwidth. The tools for traffic identification P2P have implemented one or a combination of the following models: filters of packages, identification of signatures, distribution of the connections and topology of P2P network, and crawler. However, with the intention of share material with protected content, applications P2P have incorporated aspects that do not allow the filtering of the traffic. The use of regulated ports for transference of data, the use of distributed architectures hinder the use of filters of packages, the cryptography hinders the recognition of signatures and the models of distribution of connections and crawler still possess limitations and are still study object. This work considers the use of a neural model MLP for the analysis and identification of traffic P2P. For the training of the model, they had been used given of normal traffic and P2P, that they had circulated into a edge router, in a real network. The analysis of the communication is carried through with information of the flows consolidated of all the connections and presents the advantages of not being intrusive, easy to use and, mainly, immune to techniques of camouflage used by applications P2P. The results of the MLP had been very satisfactory, all the experiments had gotten taxes of correct classifications superior to 85%. The percentages of found correct classifications in the experiments are equalized to the percentages gotten for the other models. Such results suggest, thus, the study of other neural models and the creation of tools that implement these models.

Key-words: netflow protocol, multilayer neural networks and peer-to-peer protocols.

CAPÍTULO 1 - INTRODUÇÃO

1.1 - Redes *peer-to-peer*

A Internet sempre teve como proposta promover a liberdade e democratização das informações, isto pode ser visto como acesso irrestrito a qualquer tipo de recurso disponível, em qualquer local e a qualquer momento.

No princípio da Internet, nos anos 60 e 70, a comunicação era baseada em um modelo em que um computador disponibilizava um serviço, e outros computadores ligados à rede tinham acesso e uso [1,2]. Contudo, não havia restrições na disponibilidade de serviços, todos os recursos eram distribuídos pela rede e todos os computadores exerciam um mesmo papel, ou seja, uma rede ponto-a-ponto, *peer-to-peer*.

Com o passar do tempo, a Internet foi perdendo esta característica, já que a necessidade de especialização, robustez, disponibilidade, segurança e as limitações de recursos de comunicação impulsionaram a mudança de arquitetura. Surgia, então, o modelo cliente/servidor, no qual alguns computadores superdimensionados, servidores, atendiam a uma infinidade de outros computadores, os clientes [1].

No final dos anos 70 e a partir dos anos 80, novas opções foram estudadas para se criar uma alternativa a este modelo, já que por definição o padrão cliente/servidor ignora os recursos computacionais presentes na maior parcela de computadores presentes na rede, os clientes. Foi proposta a utilização de sistemas distribuídos para aplicações específicas em que as idéias de autonomia, transparência, execução concorrente, tolerância a falhas e heterogeneidade de plataformas foram implementadas e amplamente utilizadas na década de 90[1].

No final dos anos 90 surgiu o *Napster*, um aplicativo de compartilhamento de arquivos que implementava uma rede *peer-to-peer*, utilizando o conceito original da Internet. O sucesso do *Napster* popularizou o modelo P2P, que é amplamente utilizado atualmente, com várias redes de colaboração e uma infinidade de aplicações [2].

O ressurgimento das redes *peer-to-peer* (P2P) proporcionou avanços tanto na forma de compartilhar as informações quanto na criação de redes colaborativas, tornando-se uma alternativa viável ao modelo cliente/servidor [2]. Essas redes possibilitaram a colaboração direta entre usuários, sem a dependência de servidores centrais, e permitiram, também, que

recursos computacionais fossem compartilhados através da Internet, mesmo com a presença de *firewalls* e/ou a utilização de endereços privados via NAT (*network address translation*).

Por suas características, as redes *peer-to-peer* proporcionam o livre compartilhamento de arquivos, e esse aspecto vem se tornando cada vez mais popular entre os usuários da Internet [7]. Pesquisas realizadas nos principais *backbones* (concentradores da comunicação de dados da Internet) pelo mundo, nos anos de 2003 e 2004, mostravam uma mudança no perfil da comunicação, o tráfego *peer-to-peer* chega a ocupar aproximadamente 60% de todo o tráfego da Internet, que anteriormente era ocupado por tráfego de *web* [3]. Atualmente, a comunicação P2P corresponde a uma grande fatia da comunicação, porém perdeu o primeiro posto para aplicações de serviços de *streaming* de som e vídeo (*Youtube*, rádios *online*, etc).

1.2 - Problemas causados pela comunicação *peer-to-peer*

As inovações proporcionadas pelos aplicativos P2P que permitem a livre troca de arquivos entre usuários, criam também problemas para a comunicação de dados. A abertura na transferência de informações disponibiliza acesso livre a materiais protegidos por direitos autorais como, por exemplo, filmes, músicas, livros, softwares, etc [4-7]. Esse aspecto também é encontrado em redes do tipo cliente/servidor, contudo nas redes P2P ele é ampliado.

O segundo aspecto negativo da comunicação P2P está ligado a uma característica técnica da comunicação. Aplicativos P2P são grandes consumidores de largura de banda. Por exemplo, um filme compartilhado, no formato dvd, pode gerar em torno de 4Gbytes de *download* e uma quantidade até maior de *upload* [3]. Assim, aplicativos P2P causam um grande impacto nas taxas de *download* e *upload*, pois, por sua natureza, exigem que a transferência de dados seja bidirecional em todos os *hosts*, ou seja, para receber arquivos (*download*), o usuário tem que também disponibilizar os seus arquivos (*upload*) [3, 5, 7].

Em sistemas de conexão assimétrica, como, por exemplo, o ADSL, esse fato é mais crítico, já que a conexão apresenta um canal de *upstream*, taxa de *upload* de dados, menor do que o de *downstream*, taxa de download de dados. Portanto, quando um aplicativo P2P é iniciado em um *link* assimétrico, a comunicação direta é prejudicada, pois o canal de *upload* fica carregado rapidamente, mesmo que o canal de *download* não esteja totalmente ocupado. Isto prejudica a transferência de dados como um todo, pois impede o processo de *handshake* das conexões TCP [30].

Outra característica nociva da comunicação de dados ampliada pelas redes P2P são os vírus, *spywares* (softwares que utilizam a conexão de Internet para

Outra linha de estudo tem examinado as propriedades da topologia das redes P2P, baseando-se no monitoramento das rotas e destinos dos pacotes e, também, na concentração e na relação entre os usuários que pertencem à rede [4]. Extensões desse modelo incorporaram outras propriedades como métricas para caracterização, usando volume de tráfego, conectividade do *host*, variação de volume de tráfego no tempo, duração da conexão e largura de banda utilizada.

O mapeamento mais direto da topologia é proposto com o uso do modelo *crawler*. Os *crawlers* são aplicações construídas para se conectar às redes P2P e obter informações em um dado instante. As informações recolhidas são endereços IPs de servidores, clientes e alguns dados de transferências de arquivos [10]. Com este tipo de informação é possível mapear a rede e implementar filtros por endereços e portas.

1.4 - Necessidade de uma nova solução

Por todas as razões descritas no item 1.2, as grandes instituições e corporações, como universidades, empresas, entidades do setor público, etc, sofrem com a comunicação P2P. Grandes instituições possuem, geralmente, *link*s de Internet de alta velocidade (da ordem de *megabits*) para *download* e *upload*, que proporcionam altas taxas de transferência de arquivos nas redes P2P. Essa razão faz com que as redes dessas entidades se tornem hospedeiros preferenciais de arquivos, funcionando como os principais fornecedores para outros usuários. No ano de 2002, apenas uma única universidade americana recebeu mais de 200 notificações de violação de direitos autorais, pelo compartilhamento de material protegido, através de sua rede [11].

Dadas as características atuais da comunicação *peer-to-peer* e as soluções existentes expostas até o momento, conclui-se que não há, hoje em dia, ferramentas comercialmente adequadas, que forneçam informações apropriadas para rápida análise e filtragem do tráfego P2P. As ferramentas atuais para controle e gestão de tráfego P2P estão defasadas em relação às técnicas utilizadas para a transferência de dados nestas redes.

Os modelos de identificação de tráfego *peer-to-peer* baseado em assinaturas, baseado na topologia da rede e baseado no mapeamento de endereços (*crawler*) são uma boa alternativa na solução do problema, contudo implementações para o ambiente real ainda não existem.

Este estudo propõe uma nova abordagem, utilizando redes neurais artificiais, que supre as deficiências dos filtros de pacotes e do modelo baseado em assinaturas, com desempenho comparável ao modelo de topologia sendo a metodologia e os resultados apresentados nos capítulos e seções do documento.

1.5 - Estrutura do trabalho de dissertação

Os capítulos que se seguem fundamentam e descrevem o modelo proposto e as técnicas utilizadas. O segundo capítulo fornece uma revisão das soluções para os problemas gerados pelos aplicativos *peer-to-peer*, encontradas na literatura, suas vantagens e deficiências. No segundo capítulo é feita a apresentado o modelo de referência OSI, muito utilizado para descrição do processo que ocorre em meios de comunicação de dados. O terceiro capítulo apresenta as ferramentas, sistemas e softwares utilizados para a implementação do modelo, são vistos também o protocolo Netflow, a ferramenta *flow-tools* e o ambiente do software Matlab. No terceiro capítulo também é mostrado a teoria de redes neurais e como elas são indicadas para o objetivo específico proposto. No quarto capítulo é mostrada a indução que conduz ao uso de redes neurais artificiais para classificação do tráfego *peer-to-peer*. O quinto capítulo descreve o ambiente de coleta, a forma de coleta de dados, o pré-processamento das informações, a forma de apresentação dos dados para o modelo neural proposto e os resultados obtidos. No sexto capítulo é ilustrada a forma de avaliação dos resultados da classificação, apresentando o desempenho obtido em cada experimento, e discorre sobre as vantagens e as limitações do modelo proposto. O sétimo capítulo faz-se uma análise geral apresentando quatro caminhos para a continuidade do desenvolvimento.

CAPÍTULO 2 - FUNDAMENTAÇÃO TEÓRICA

2.1 - Modelo de referência OSI

Para compreensão do estudo desenvolvido é necessário a apresentação do modelo de referência OSI, pois a terminologia determinada por este modelo será amplamente utilizada nos capítulos posteriores, em que serão descritos comportamentos de conexões, mecanismos de filtragem de pacotes, etc. O modelo OSI é mostrado na figura 2.1, na qual estão listados um *host* de origem e outro de destino se comunicando. O modelo OSI oferece um conjunto de padrões que, se seguidos, garantem maior compatibilidade e interoperabilidade entre diversas tecnologias de rede proprietárias ou não [30].

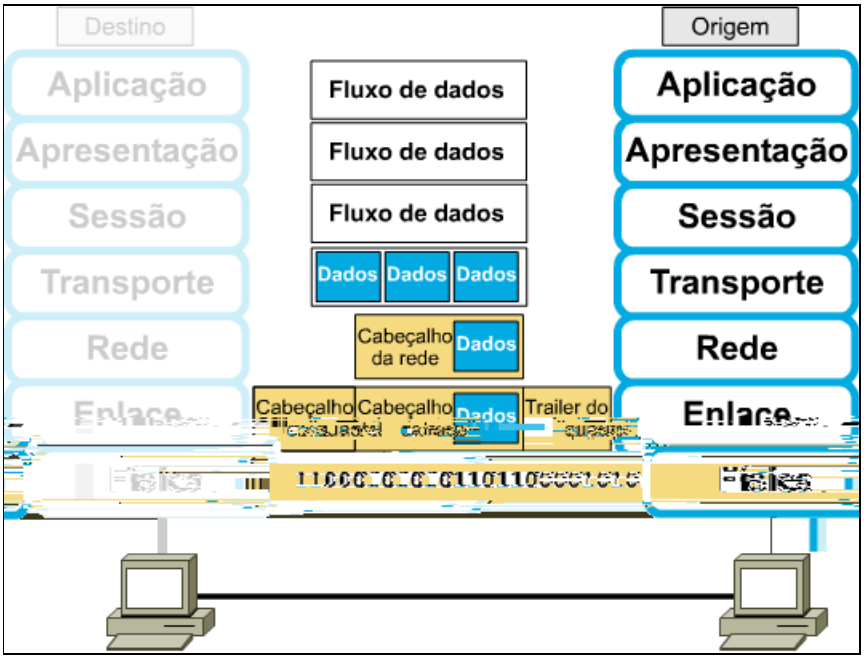


Figura 2. 1 – Modelo de referência OSI

O modelo de referência OSI é dividido em sete camadas e cada uma delas possui uma função particular no processo de comunicação, contendo suas especificações e os seus protocolos de comunicação. O protocolo é um conjunto de regras ou um acordo, que determina o formato e a transmissão de dados em uma comunicação de dados. A seguir serão

descritas as camadas do modelo OSI e seu contexto no objetivo específico deste estudo. As sete camadas do modelo OSI são [13,14]:

- **Camada 7 - camada de aplicação**

É a camada mais próxima do usuário. Ela difere das outras camadas, pois não fornece serviço para outra camada do modelo OSI e sim para os aplicativos nos sistemas finais. Alguns exemplos desses aplicativos são: navegadores de internet, processadores de texto, planilhas eletrônicas e programas de correio eletrônico. É nesta camada que funcionam os aplicativos *peer-to-peer* como, por exemplo, Azureus, Bearshare, BitTorret, Earthstation, eDonkey, eMule, KazaA, KazaA Lite, WinMx, etc. Para o estudo proposto serão tratados dois protocolos P2P que são implementados na camada de aplicação, o protocolo eDonkey2000 e o protocolo BitTorrent.

- **Camada 6 - camada de apresentação**

Esta camada assegura que a informação enviada pela camada de aplicação de um sistema de origem seja legível no sistema de destino. Uma das funções dessa camada é a estruturação dos dados que são usados pelos aplicativos, em que se pode incluir a compactação e a criptografia destes dados, recursos disponibilizados por alguns aplicativos *peer-to-peer*.

- **Camada 5 - camada de sessão**

Ela estabelece, gerencia e termina sessões entre aplicativos de dois *hosts*. A camada de sessão é responsável por prover serviços à camada de apresentação. Ela sincroniza e controla o diálogo entre as camadas de apresentação dos dois *hosts* e também gerencia a troca de dados entre eles.

- **Camada 4 - camada de transporte**

Esta camada é responsável pelo transporte de informações da origem para o destino, de forma confiável e precisa. Para fornecer o serviço de transporte de dados a camada de transporte estabelece, mantém e termina circuitos virtuais entre processos. Ela também utiliza os recursos de controle do fluxo de informações e detecção/recuperação de erros no transporte

dos dados. Nesta camada são implementados os protocolos TCP e UDP. As aplicações de filtragem de pacotes baseados em portas utilizam informações desta camada para a classificação dos pacotes.

- **Camada 3 - camada de rede**

Esta camada fornece conectividade e seleção de caminhos entre dois sistemas de comunicação. O esquema de endereçamento lógico [14] é usado pelos dispositivos que trabalham nesta camada para determinar o destino dos dados no momento em que eles se movem pelas redes. Nesta camada não existe nenhum mecanismo de verificação de erros, ela apenas tenta entregar os dados da maneira mais eficiente possível. O protocolo IP, Internet Protocol, é o protocolo mais utilizado na camada 3 e será o abordado neste estudo. Nessa camada são implementados os filtros de pacotes que utilizam o endereçamento IP. É nessa camada que operam os roteadores, equipamentos de duas ou mais interfaces de comunicação que são responsáveis por encaminhar os pacotes por redes lógicas diferentes.

- **Camada 2 - camada de enlace de dados**

Esta camada fornece acesso aos meios de rede e à transmissão física dos dados. Além disso, esta camada cuida da notificação de erros, da topologia física da rede, da entrega ordenada de quadros e do controle de acesso ao meio. Neste estudo as informações dos pacotes de camada 2 não serão utilizadas.

- **Camada 1 - camada física**

Esta camada define as características elétricas, mecânicas, funcionais e os procedimentos para iniciar, manter e terminar as conexões físicas destinadas à transmissão de bits entre os sistemas de origem e destino. Por regulamentar apenas informações dos meios físicos, esta camada não apresenta informações adicionais ao modelo proposto.

O modelo OSI, por si só, não define a arquitetura de uma rede. Isso acontece porque ele não especifica com exatidão os serviços e protocolos de cada camada. O modelo OSI simplesmente “diz o que cada camada deve fazer ou implementar” servindo apenas como referência e não como uma norma [14, 32].

A divisão da comunicação de dados em camadas oferece vantagens, entre elas se pode citar:

- Divisão da comunicação de dados em partes menores para tornar mais simples o entendimento e a diversidade de serviços prestados.
- Padronização dos componentes de rede, para permitir o desenvolvimento e o suporte, por parte dos vários fabricantes existentes.
- Viabilização da comunicação entre tipos diferentes de *hardware* e de *software* de rede, independente do fabricante, ou seja, interoperabilidade entre diferentes tecnologias.
- Evitar que as modificações em uma camada afetem as outras camadas, permitindo, assim, uma maior rapidez no seu desenvolvimento.

2.2 - Redes *peer-to-peer*

Redes *peer-to-peer* são redes que funcionam na Internet com o objetivo de compartilhar recursos entre os participantes, sendo que por princípio não há diferenciação entre os usuários. O *Peer-to-peer Working Group*, um consórcio que inclui IBM, Hewlett-Packard e outros lideres da indústria definiram o P2P como a forma de “compartilhamento de recursos computacionais por troca direta” [16].

Os sistemas *peer-to-peer* abandonaram a infra-estrutura centralizada e cada indivíduo ou *peer* integrante contribui com seus recursos computacionais em um ambiente de cooperação. A organização dos *peers* pode seguir vários modelos, contudo, em cada um deles a característica principal é a mesma: os *peers* interagem diretamente uns com os outros sem a necessidade de uma autoridade central. Outra característica das redes *peer-to-peer* é a capacidade de auto-organização. A topologia da rede, implementada na camada de aplicação do modelo OSI, se adapta dinamicamente à medida que os *peers* se conectam e desconectam da rede, de forma a manter a conectividade, disponibilidade e desempenho, sendo conhecida como uma rede *overlay*, pois é implementada sobre a rede TCP/IP [4].

Os aplicativos P2P foram construídos com a intenção de compartilhamento em larga escala, em que os usuários contribuem, principalmente, com seus recursos computacionais (*grid computing*) e arquivos [15]. Diferentemente das redes cliente/servidor, cujos recursos

estão localizados apenas no servidor central, as redes P2P possuem seus recursos distribuídos e replicados pelo ambiente computacional de acordo com a demanda, formando um sistema altamente escalável, devido ao grande número de *hosts*, equipamentos, largura de banda disponível e espaço em disco disponível [15].

O aspecto que tem atraído muitos usuários às redes *peer-to-peer* é o compartilhamento de arquivos, pois disponibiliza material de interesse comum a qualquer usuário do sistema. No compartilhamento de arquivos, os tamanhos das tr

rede Gnutella, porém tem sido abandonada devido à ineficiência na troca de mensagens, que devem atravessar muitos *servent*s até encontrar os *peers* adequados, aumentando o tempo de resposta.

- **Parcialmente centralizadas:** nestes modelos, alguns nós assumem mais responsabilidades que outros, tornando-se supernós, agindo como servidores locais para arquivos compartilhados por *peers* locais e disponibilizando conectividade com outros supernós. Neste ponto, a arquitetura da rede P2P forma uma hierarquia de dois níveis, fornecendo melhor desempenho e escalabilidade que o modelo puramente descentralizado [18]. Este novo modelo é usado em sistemas como o Kazaa e na nova versão do Gnutella. Observa-se que os supernós são eleitos dinamicamente, portanto evitam a criação de pontos únicos de falha.

- **Híbridas descentralizadas:** nesta arquitetura existe um servidor ou conjunto de servidores que facilitam a cooperação entre os *peers* e podem até disponibilizar serviços como a procura de arquivos. Esta arquitetura foi usada no Napster, portanto, provou ser suscetível a falhas, já que o Napster foi desativado. No entanto, algumas redes *peer-to-peer* modernas como, por exemplo, eDonkey2000, utilizam essa arquitetura com supernós espalhados pelo mundo inteiro. A diferença deste modelo para o parcialmente centralizado é que os supernós são fixos, e diferentemente do modelo cliente/servidor, a troca de arquivo é realizada entre os vários *peers* diretamente sendo que o supernó é um ponto de organização que auxilia alguns mecanismos da rede de compartilhamento. Por estas razões o desempenho deste modelo é superior, pois os supernós dedicados auxiliam na localização e compartilhamento de arquivos [18].

A figura 2.2 apresenta uma interpretação visual para as topologias utilizadas pelas redes de compartilhamento segundo a classificação determinada pelo seu grau de centralização.

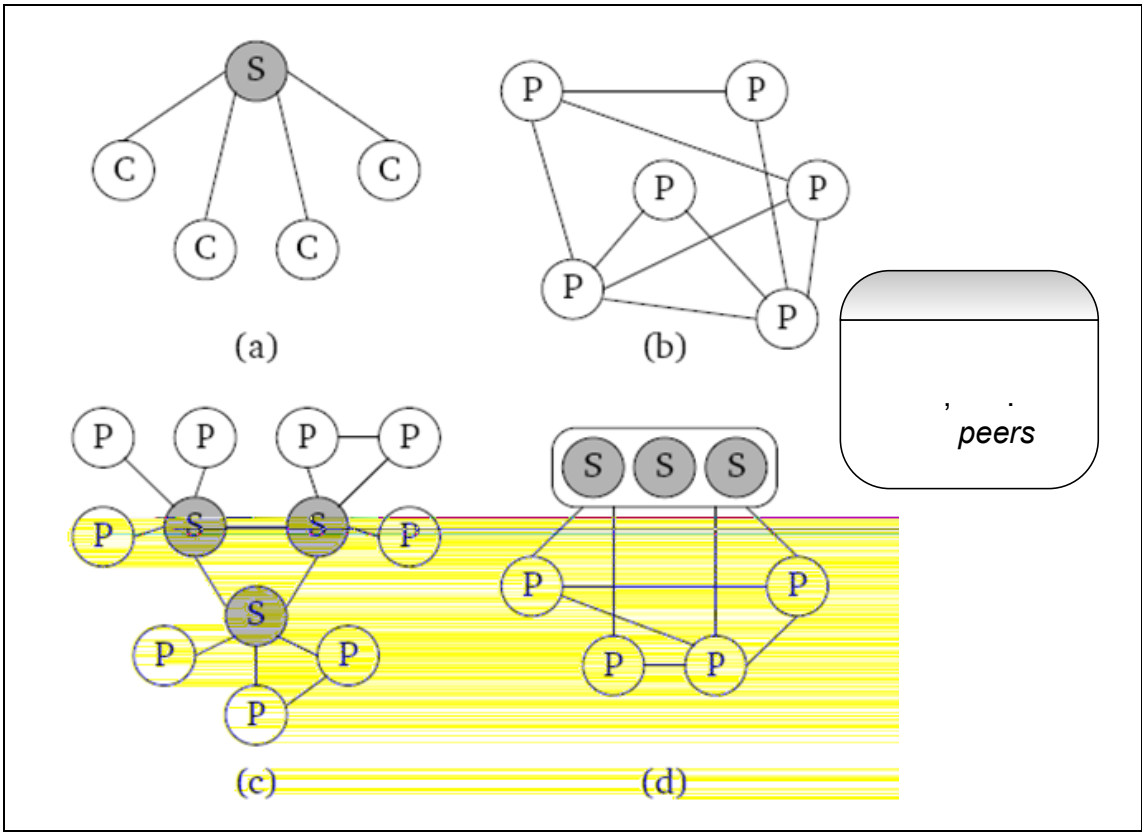


Figura 2. 2 – Diferentes graus de centralização
(a) Modelo cliente/servidor (b) Puramente descentralizada (c) Parcialmente centralizada (d) Híbrida descentralizada

As arquiteturas P2P também podem ser classificadas de acordo com a estrutura da rede. Os sistemas P2P constroem uma rede própria, *overlay*, sobre a rede de comunicação utilizada (na maioria dos casos sobre uma rede TCP/IP), em que cada *peer* mantém a mesma conexão ponto a ponto com os outros *peers*.

A topologia desta rede é altamente dinâmica, já que os nós entram e saem do ambiente freqüentemente, e sem qualquer relação com meios físicos ou protocolos de camadas inferiores. De acordo com a estrutura, as redes *peer-to-peer* podem ser [18]:

- **Desestruturadas:** nestas redes a localização dos dados não é relacionada com a topologia. Não existe uma forma de conhecer imediatamente onde está um dado recurso e os mecanismos de pesquisa são aleatoriamente conduzidos. Vários *servent* são consultados até que a localização do arquivo solicitado seja

encontrada ou então se descubra uma informação que possa levar ao destino do arquivo procurado. Geralmente, essas redes possuem baixo desempenho na localização de arquivos e problemas de escalabilidade, no entanto é largamente utilizada, pois seu esquema acomoda facilmente a população de usuários que está constantemente em mutação. Para amenizar os efeitos da não estruturação, estas redes são parcialmente centralizadas ou híbridas sendo que as pesquisas por arquivos continuam aleatórias, porém apenas no nível dos supernós. Os usuários consultam pelos arquivos apenas em seus supernós locais e os supernós trocam informações entre si para localizar o arquivo procurado, tornando assim este modelo de rede viável.

- **Estruturadas:** nestes sistemas a topologia é relacionada aos *hosts* e os recursos são relacionados às topologias. Os arquivos são armazenados em locais especiais no sistema P2P e existe um mecanismo para mapear os arquivos e suas localizações. São mantidas tabelas de rotas e as *queries* de procura podem ser encaminhadas para o *host* adequado de forma mais eficiente que no modelo não estruturado. A grande dificuldade deste modelo é manter uma tabela de rotas uma vez que a entrada e saída de *peers* na rede são constantes.

- **Fracamente estruturadas:** é uma solução que mescla redes estruturadas e desestruturadas. Em tais sistemas, existe um mapeamento entre a localização do arquivo e a topologia, porém a rota exata não é completamente especificada, ela aponta para a localização aproximada do arquivo procurado, desta forma as buscas podem resultar em falhas.

2.2.2 - Protocolo eDonkey2000

2.2.2.1 - Breve histórico

Este estudo utiliza o aplicativo eMule para acesso a rede eDonkey2000. O eDonkey2000, *ed2k,* foi a rede *peer-to-peer* criada pela empresa alemã *MetaMachine* em 2000 para a transferência de grandes arquivos, que ultrapassavam os limites dos gigabytes. O

aplicativo disponibilizado para acesso a esta rede de compartilhamento foi o *eDonkey*. Juntamente com o *eDonkey* a empresa criou a rede P2P *Overnet*, a 2ª geração das redes *peer-to-peer*, altamente descentralizada e mais rápida, com a intenção de lucrar com o uso do aplicativo desta rede P2P, que era pago [19].

Contudo, alguns desenvolvedores de sistemas insatisfeitos com o aplicativo da rede *ed2k* fornecido pela *MetaMachine* criaram, em 2002, o eMule, que popularizou a rede *ed2k*. No mesmo ano, foi criada uma rede de compartilhamento com características idênticas à rede *Overnet*, chamada *Kademlia*, suportada nas versões posteriores do aplicativo eMule. Atualmente, o eMule é capaz de se conectar na rede *eDonkey* e *Kademlia*. O fato do código do aplicativo eMule ser aberto proporciona uma série de clientes derivados, sendo: xMule, ml-Donkey, Shareaza, etc [19].

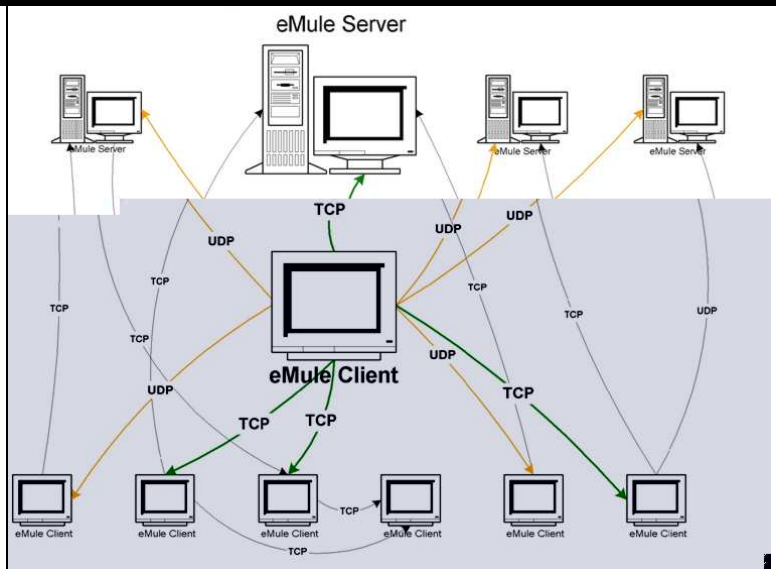
Nos sites de *download* de softwares, o eMule sempre figura entre os dez mais baixados. As proporções estimadas atingidas pela rede ed2k são [19]:

- Número de usuários simultâneos: 11,5 - 12,5 milhões;
- Número de arquivos compartilhados simultaneamente: 700 - 900 milhões.

Para este estudo é utilizado o aplicativo eMule compartilhando arquivos na rede eDonkey2000. O ambiente eDonkey2000 pode ser classificado como híbrido descentralizado, pois se baseia em servidores centrais fixos e espalhados, com a função de fornecer ao usuário pesquisas e indexação de arquivos.

2.2.2.2 - Visão simplificada dos mecanismos de conexão

O primeiro passo da conexão realizado pelo eMule, inicialmente é, a conexão particular entre o cliente e um servidor supernó. Em seguida, o aplicativo irá se conectar a outros servidores e a outros diversos clientes expandindo sua ligação na rede por conexões TCP e UDP [20, 34], conforme mostrado na figura 2.3.


eMule Server
eMule Server
eMule Server
eMule Server
eMule Server
TCP
UDP
UDP
UDP
TCP
TCP
TCP
TCP
TCP
UDP
UDP
eMule Client
UDP
TCP
TCP
TCP
TCP
eMule Client
eMule Client
eMule Client
eMule Client
eMule Client
eMule Client
TCP
TCP

que determina a prioridade entre os usuários que estão tentando acessar um mesmo recurso [20, 34].

2.2.3 - Protocolo BitTorrent

O BitTorrent foi criado por Bram Cohen em 2003 e a sua popularidade tem crescido rapidamente. O BitTorrent realiza o *download* de um arquivo de múltiplas fontes e paralelamente o *upload* para outras fontes das partes dos arquivos que ele já possui. Isto faz com que a disponibilidade de fontes seja muito grande no início da transferência e a quantidade de arquivos completos vá crescendo à medida que mais *servents* concluem os *downloads* [21].

Sua arquitetura é puramente descentralizada e os *peers* trocam arquivos diretamente, o problema da localização de arquivos existente neste tipo de arquitetura é solucionado pela forma estruturada na qual a rede é baseada. A localização dos arquivos não é mapeada automaticamente, é necessário que exista um *tracker*, ou seja, um servidor que armazena apenas informações da localização de arquivos à medida que estes são replicados pelo ambiente de compartilhamento [21].

2.2.3.1 - Visão simplificada dos mecanismos de conexão

Sendo um protocolo aberto, existe uma variedade de aplicativos disponíveis para ambientes Windows e Linux. Um pequeno arquivo com a extensão ".torrent" contém as informações dos arquivos e o processo de conectividade procede conforme mostrado na figura 2.4.

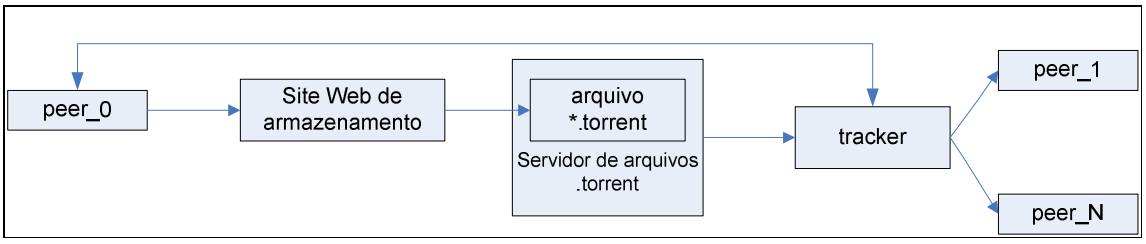


Figura 2. 4 – Processo de conexão do BitTorrent

Todo o processo mostrado na figura 2.4 pode ser descrito através dos seguintes passos [21]:

- **Passo 1:** Um usuário que deseja acessar a rede BitTorrent instala um aplicativo como, por exemplo, BitComet, ABCTorrent, GTorrent, etc. No caso deste estudo o aplicativo utilizado foi o Azureus, e o computador deste usuário será conhecido como peer_0;
- **Passo 2:** Um *website* usual contém os arquivos com a extensão ".torrent", e é utilizado como a interface de pesquisa. Neste site os usuários cadastram informações sobre os arquivos como, por exemplo, nome, tamanho do *download*, quantidade de partes e outros identificadores. Este dado é armazenado em um arquivo com a extensão ".torrent", nele estão todos os parâmetros necessários para o início do *download*, sendo: endereço IP e número da porta do *tracker*, número de fontes completas, quantidade de fontes parciais, etc;
- **Passo 3:** Com as informações do arquivo ".torrent" o aplicativo inicia a transferência do arquivo desejado. O aplicativo se conecta com o *tracker* e atualiza a lista de *peers* que possuem o arquivo desejado. Em seguida o *peer_0* se conecta a estes *peers* (*peer_1, ... , peer_N*) e transfere o arquivo.
- **Passo 4:** Os aplicativos se comunicam periodicamente com o *tracker* para obter informações sobre novas fontes para *download* do arquivo ou de outros *peers* para o *upload* dos arquivos. Todas as transferências de arquivos são realizadas através de conexões diretas entre os *peers*. Essas conexões diretas utilizam o protocolo BitTorrent para visualizar e manter a topologia da rede obtida através do *tracker*.

O usuário que possui na íntegra o arquivo a ser transferido é conhecido como *seed* e é necessária sua presença na topologia para que outros *peers* possam completar o *download* e também se tornarem *seeds*. Portanto, o número de *seeds* aumenta à medida que os dados são espalhados pela rede [21].

A maioria dos aplicativos que utiliza a rede BitTorrent possui a porta 6881 como porta padrão. Esse é o comportamento padrão dos aplicativos BitTorrent. No entanto, assim como no eMule, este comportamento pode ser alterado pelo usuário, para transferir arquivos por outras portas quaisquer [21].

2.2.4 - Necessidade de detecção

O grande crescimento e a intensidade de consumo de largura de banda, natural em aplicativos *peer-to-peer*, sugere que o tráfego P2P possa ter um impacto significante nas redes de comunicação de dados e pesquisas de caracterização de tráfego em *backbones* de comunicação têm comprovado esse fato [2,3,6]. Portanto, torna-se importante para os provedores de acesso à Internet (*Internet Service Provider*, ISP) detectar o tráfego P2P e desenvolver modelos de distribuição de carga, técnicas de engenharia de tráfego e novos padrões de planejamento para suportar essa realidade.

Conseqüentemente, sistemas P2P, em especial os de compartilhamento de arquivos, utilizados em conexões de banda larga em um provedor de acesso à Internet, pode diminuir de forma geral o desempenho da rede (latência, confiabilidade, disponibilidade, etc). Os usuários do serviço, que não utilizam P2P, ao ficarem insatisfeitos com o serviço prestado podem trocar de provedor. Nesse contexto, as aplicações que não podem sofrer atrasos no envio dos dados e necessitam baixa taxa de variação da banda no link são prejudicadas como, por exemplo, aplicações de voz e vídeo em tempo real.

Para garantir o correto desempenho dos tráfegos considerados regulares e/ou aplicações em tempo real, um ISP deve implementar formas de engenharia de tráfego (*shapping*, política de prioridades, classes de serviço, etc). Por sua vez, para implantar perfeitamente essas ações o ISP deve possuir um sistema que seja capaz de classificar o tráfego *peer-to-peer*. Este fato traz a necessidade de detectar corretamente o tráfego P2P, mesmo que este utilize técnicas de camuflagem.

Além dos aspectos técnicos, essa necessidade é importante na proteção dos conteúdos cobertos por direitos autorais, que podem ser encontrados nas redes *peer-to-peer*. Recentemente, algumas ISP têm sido responsabilizadas por infringir leis de direito autoral, o que as tem encorajado a encontrar uma solução definitiva para o problema causado pelo P2P [9], contudo essa solução fica cada vez mais distante, diante dos mecanismos de camuflagem utilizados pelos aplicativos. As dificuldades encontradas na detecção são discutidas no item subseqüente.

2.2.5 - Dificuldade de detecção

Atualmente, os aplicativos P2P mais modernos estão intencionalmente camuflando o tráfego que elas geram com a finalidade de circundar os potenciais *firewalls* e filtros de pacotes encontrados pelas redes [2-10].

Na primeira geração dos aplicativos P2P era relativamente fácil classificar o tráfego gerado devido ao uso de portas com números bem definidos para o acesso a cada rede *peer-to-peer*. No entanto, com o crescimento do número de aplicativos P2P, técnicas foram adicionadas para se evitar os filtros de pacotes que bloqueavam a transferência dos dados em determinadas portas. Nos dias de hoje, os aplicativos P2P são capazes de transferir os dados utilizando portas aleatórias, incluindo, principalmente, as portas regulamentadas da faixa de 1 até 1024 que muitas vezes garantem a passagem do tráfego P2P pelos filtros de pacotes. Estudos mostram que quase 15% do tráfego *peer-to-peer* utiliza a porta 80, tradicionalmente usada com tráfego *web* [2]. Existem também aplicativos P2P com suporte à criptografia, o que impede a utilização de sistemas baseados na verificação de assinaturas em pacotes *peer-to-peer*.

O problema da camuflagem é tão grave que as administrações dos grandes *backbones* têm reportado uma diminuição no tráfego *peer-to-peer*, que agora ocupa em torno de 16% do total trafegado. Porém, com o uso da camuflagem, está crescendo o volume de tráfego circulante sem identificação, que já representa 54% da quantidade total, que se supõe pertencer aos aplicativos da classe P2P [2,3].

Portanto, os meios atuais para controle e gestão de tráfego P2P estão defasados em relação às técnicas utilizadas para a transferência de dados nestas redes e não são capazes de fornecer mecanismos válidos para as administrações dos *backbones* e ISPs. Nos itens 2.3, 2.4, 2.5 e 2.6 são mostradas, de forma sucinta, as principais soluções empregadas na tentativa de minimizar o problema.

2.3 - Solução por utilização de filtros de pacotes

2.3.1 - Princípio de funcionamento

Os filtros de pacotes estão entre as técnicas mais complexas de análise de tráfego de informação, utilizando informações da camada de transporte e da camada de rede.

Em linhas gerais, um filtro de pacotes é um equipamento que se situa entre a rede local e a rede pública, sendo responsável por permitir ou negar a troca de dados entre as duas redes. Basicamente, ele opera permitindo apenas a passagem de informações das aplicações que são realmente necessárias, negando todo o resto da comunicação não explicitamente definida como permitida [22].

Os mecanismos que os filtros de pacotes, ou *firewalls* [36], utilizam para tomar suas decisões são baseados em regras [22]. Essas regras são sempre criadas pelo administrador da rede, que conhece as necessidades dos seus usuários. Os filtros trabalham nas camadas 3 e 4 do modelo OSI. Portanto, as decisões de filtragem se baseiam nas informações contidas nos cabeçalhos da camada de rede e de transporte, onde estão os endereços IP e as portas de serviço, respectivamente. Dois tipos de filtros de pacotes são conhecidos: os *stateless* e os *statefull*.

2.3.2 - Filtro de pacotes *stateless*

Os filtros *stateless* não mantêm os estados das conexões e são apenas um conjunto de regras que são aplicadas cada vez que um novo pacote é recebido pelo filtro. Quando um pacote chega, suas informações são comparadas com o conjunto de regras até que uma possua instruções sobre qual ação realizar sobre o pacote [31]. A vantagem desse modelo de filtro é que ele não ocupa memória para manter os estados das conexões, em contrapartida ele apresenta a desvantagem de que cada novo pacote deve ser checado com o conjunto de regras. Quando se possui uma base de regras muito extensa, pode-se gerar um atraso na comunicação, já que muitos pacotes podem sobrecarregar o processamento do equipamento devido à grande quantidade de regras e a checagem de pacotes.

2.3.3 - Filtro de pacotes *statefull*

Filtros baseados em *statefull packet filter* tomam as decisões de filtragem tendo como referência dois parâmetros:

- As informações dos cabeçalhos das camadas 3 e 4; e
- Uma tabela de estados, que guarda os estados de todas as conexões.

Os *firewalls* baseados em estado são capazes de manter o rastreamento de sessões de rede. Quando um pacote é recebido ele checa as informações com uma base de regras, da mesma forma que o filtro *stateless* [31]. Determinada a legitimidade da comunicação, o filtro manterá as informações de estado desta conexão, como os endereços IP de origem e destino, as portas de origem e destino, o número do pacote e etc. Todos os pacotes seguintes, que sejam continuação da primeira conexão classificada como legítima, serão permitidos.

Este tipo de filtro economiza tempo de processamento, porém requer grande quantidade de memória para armazenamento das informações das conexões. Em redes com grande variedade de tipos de tráfego ele é mais eficiente, pois não necessita consultar uma base de regras extensa a todo o momento [31].

2.3.4 - Deficiências

Apesar de apresentar boas funcionalidades na filtragem de pacotes, ambos os modelos *statefull* e *stateless* são deficientes, pois não conseguem filtrar o tráfego P2P que utiliza as portas padrões de outros protocolos, já que ele se baseia em listas de controle de acesso que sempre permitem pacotes nas portas padrões dos protocolos regulares utilizados pelos usuários da rede [2,5-8].

A aceitação e a recusa são baseadas em especificações, por exemplo, endereço IP de origem ou de destino, portas de origem ou de destino, e são ineficientes em bloquear o tráfego P2P em portas como, por exemplo, http/80, dns/53, ssh/22, https/443, ftp/20,21, pop/110, etc. Nestes casos, os filtros de pacotes são como portões sempre abertos para os tráfegos P2P.

2.4 - Modelo baseado em assinaturas

O modelo baseado em assinaturas realiza uma varredura no *payload*, área onde estão os dados, do pacote da comunicação, geralmente na camada 4, procurando por uma string binária característica do aplicativo P2P que gerou o tráfego [2,5-7]. Por este modelo é necessária uma avaliação separada de cada protocolo identificando o formato de cada pacote utilizado por cada aplicativo.

Como a documentação dos aplicativos P2P é deficiente, a maior parte dos estudos realizados [2,5-7] monitoram, utilizando *tcpdump,* o tráfego TCP/UDP de cada aplicativo na procura de uma string binária particular de cada cliente. A tabela 1 ilustra as características dos pacotes dos protocolos eDonkey2000 e BitTorrent.

Tabela 1 – Características dos pacotes dos protocolos P2P

Protocolo P2P	**String**	**Protocolo de transporte**	**Porta padrão**
eDonkey2000	0xe319010000 0xc53f010000	TCP/UDP	4661/4665
BitTorrent	“0x13Bit”	TCP	6881

A lista contendo as características dos nove protocolos P2P mais populares (eDonkey2000, Kazzaa, BitTorrent, OpenNap, WinMx, Gnutella, MP2P, Ares e Direct Connect) pode ser encontrada em [7].

Portanto, o emprego prático deste modelo utiliza um software que captura os pacotes, avalia o *payload* procurando por uma string particular e a compara com um conjunto de regras, atualizadas periodicamente, para identificar e marcar o tráfego P2P em relação ao tráfego normal, para posterior filtragem ou processamento em um roteador de borda [7].

Alguns sistemas utilizam a estrutura de sistemas de detecção de intrusão, IDS, criando regras apropriadamente de forma a identificar os padrões de string. A implementação utilizando o SNORT, o IDS de código aberto mais utilizado pelo mercado, apresentou sucesso. Foram construídas regras que capturavam os pacotes, procurando por mensagens específicas dos processos de conexão e transferências de arquivos dos aplicativos P2P, que geravam mensagens de alertas ao administrador do sistema, no caso de classificações *peer-to-peer* positivas [5,24].

Como base para comparação dos resultados obtidos pela utilização do modelo proposto é mostrada, na tabela 2, a precisão da detecção alcançada pelo modelo baseado em assinaturas retirado de [6].

Tabela 2 – Detecção P2P pelo modelo baseado em assinaturas

Protocolo	**Precisão da Detecção (%)**	
	Bytes	**Pacotes**
eDonkey2000	98,82%	93,63%
BitTorrent	99,78%	98,43%

2.4.1 - Limitações e deficiências

O modelo de verificação de assinaturas resolve o problema da aleatoriedade das portas de serviços, já que sua detecção é baseada no conteúdo estático de cada pacote *peer-to-peer*. Apesar de obter altas taxas de classificação correta de pacotes, existem algumas limitações e deficiências impedindo este modelo de se tornar a solução definitiva para o problema do P2P, sendo as principais:

- **Quantidade de pacotes:** para encontrar a string *peer-to-peer* característica do aplicativo o sistema pode ter de vasculhar toda a extensão do *payload* do arquivo. Para uma quantidade muito grande de pacotes a verificação de assinaturas pode ocasionar atrasos nas comunicações;
- **Mudanças nas *strings*:** na tentativa de camuflar o tráfego, os aplicativos podem alterar suas *strings* características, o que implicaria em uma manutenção constante do monitoramento dos aplicativos e freqüente atualização das regras nos sistemas implementados nos ambientes reais; e
- **Criptografia:** para esconder a string *peer-to-peer*, alguns aplicativos têm implementado algoritmos de criptografia na transferência de pacotes. Desta forma, a identificação das *strings* P2P com o uso de qualquer método para inspeção do *payload* é ineficiente.

2.5 - Modelo baseado na análise de topologia

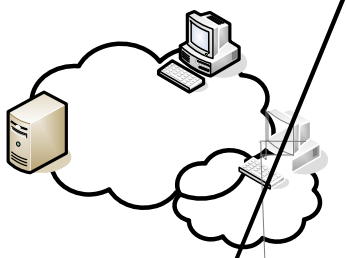
Alguns modelos, na tentativa de suprir as deficiências encontradas nos filtros de pacotes e no modelo baseado em assinaturas, estão utilizando aproximações que consideram a topologia da rede P2P.

Estes métodos examinam apenas os cabeçalhos dos pacotes para tentar identificar o tráfego *peer-to-peer* que é realizado por portas arbitrárias [7]. A visão heurística deste método é baseada na observação dos padrões de conexão dos endereços IPs de origem e destino dos pacotes na tentativa de obter a topologia da rede P2P na qual um determinado *host* pode estar conectado. O modelo é baseado em dois pontos chaves:

- **Endereços IP origem/destino:** os pares de endereços IPS que utilizam tanto o protocolo TCP quanto UDP para transferência de arquivos; e
- **Topologia dos *peers*:** estudando as características dos pares de endereços IP origem/destino se consegue identificar potenciais hosts conectados em redes P2P.

Baseado na primeira informação é possível encontrar o *host* que forma pares de comunicação com vários outros *hosts*, utilizando ambos os protocolos TCP/UDP, já que diferentemente dos aplicativos *peer-to-peer*, a maior parte das aplicações regulares utilizam apenas um único protocolo na camada de transporte.

Utilizando a segunda informação é possível detectar o P2P pela característica dos *peers* serem altamente distribuídos. A heurística utilizada pode ser ilustrada pela figura 2.5. Inicialmente, o *host* com o endereço *peer_0* se conecta ao supernó informando seu endereço IP e a porta em que ele aceitará as conexões. No segundo momento o supernó informa aos outros *peers* que o *host peer_0* se conectou à rede. Na terceira etapa os *peers* 1, 2 e 3 se conectam ao *peer_0* formando vários pares entre IP/porta de origem e IP/porta de destino, conforme mostrado na figura 2.5.



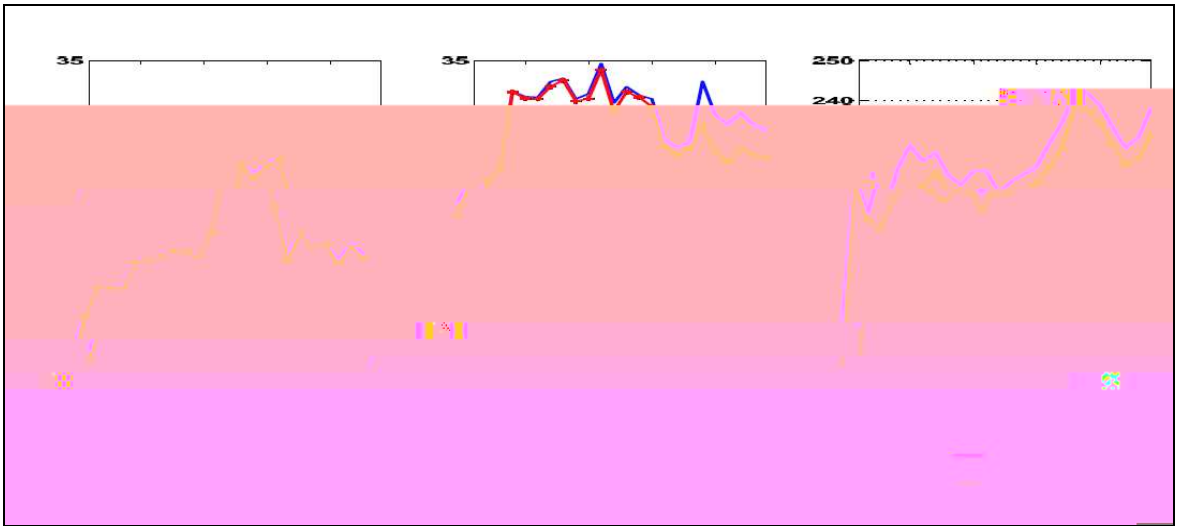


Figura 2. 6 – Comparação entre modelo de assinaturas e modelo de topologia

2.5.1 - Limitações

O modelo de topologia possui a deficiência de poder apresentar uma alta taxa de falsos positivos. Considerando que se aplicado em um link de *backbone*, com um vasto número de endereços IPs e fluxos circulantes, o tráfego normal pode ser confundido com tráfego P2P, caso esses possuam, mesmo que momentaneamente, o padrão parecido com o de um aplicativo *peer-to-peer* [7].

2.6 - Modelo *crawler*

Outra abordagem para identificação do tráfego P2P é o modelo *crawler* [23]. Este mecanismo é um aplicativo que possui a função de se conectar a uma rede de compartilhamento para obter informações dos *peers* (endereços e portas), dessa forma montando a topologia da rede P2P.

O *crawler* se conecta à rede e dispara requisições de procura para coletar os endereços IP que respondem aos seus chamados. Dessa maneira é possível reconstruir a topologia de *overlay* da rede *peer-to-peer*. A vantagem desse método é que ele identifica os endereços IP dos *hosts* e a partir desse ponto é possível realizar ações de controle sobre os fluxos pertencentes a esses endereços.

Por exemplo, para a rede eDonkey2000, é possível que o mecanismo descubra os *peers* principais e os clientes conectados a eles. A partir daí, qualquer estratégia pode ser adotada

como, por exemplo, bloquear o tráfego para o *peer* principal ou limitar a banda disponível para os clientes.

A vantagem desse modelo é que ele não necessita de nenhum conhecimento prévio sobre o protocolo, sendo uma boa característica quando se trabalha com protocolos P2P proprietários. Outros aspectos positivos são a imunidade à criptografia, monitoramento não intrusivo e não inserção de atraso nas comunicações.

A proposta do modelo *crawler* é simples, tornando-o uma estratégia viável, sendo que novos modelos híbridos entre o modelo *crawler*, modelo baseado em assinaturas e modelo baseado em topologia estão sendo propostos [24].

2.7 - Considerações

Este capítulo trouxe o modelo de referência OSI que auxilia na compreensão dos mecanismos descritos para filtragem de pacotes, modelo baseado em assinaturas, modelo baseado em topologia e o modelo crawler de mapeamento. A visão geral destes modelos para a classificação do tráfego *peer-to-peer*, seus mecanismos de funcionamento e deficiências auxiliam na compreensão do objetivo que se busca na construção do modelo proposto.

Portanto, no modelo proposto, utilizando-se inteligência artificial, procura-se a capacidade de classificar os fluxos exportados de um roteador utilizando o protocolo Netflow como pertencentes ou não a aplicativos *peer-to-peer*. Pretende-se com essa abordagem, criar um paradigma que seja imune às técnicas de camuflagem, já que até o instante, não se tem conhecimento de modificações intencionais no comportamento dos fluxos gerados pelos aplicativos P2P nos roteadores.

O modelo neural utilizado, o protocolo de coleta de dados, a ferramenta de captura do Netflow e o ambiente científico Matlab fornecem todo o ferramental necessário para tratar os dados e a construção do modelo proposto, e estes são descritos no capítulo 3.

CAPÍTULO 3 - O FERRAMENTAL

3.1 - Netflow

3.1.1 - O que é o Netflow?

Netflow é um componente presente nas versões mais novas do Software IOS de alguns equipamentos da empresa Cisco. Cisco é um fabricante de equipamentos para redes que atualmente fornece sistemas de hardware e software para aproximadamente 80% da estrutura da Internet [25].

O Netflow é utilizado para caracterização da operação das comunicações e fornece visibilidade ao administrador da rede, proporcionando ações de controle, avaliação de pontos críticos e planejamento de expansões. As informações que ele fornece são úteis para análise de [33]:

- Uso da rede e de aplicações;
- Utilização de recursos e desempenho da rede;
- Impactos em mudanças estruturais;
- Anomalias de comportamento, vulnerabilidade, etc

Com a análise dos dados sendo realizada por um administrador eficiente é possível entender quem, o que, quando, onde e como o tráfego está fluindo. Esta habilidade de caracterizar o tráfego IP e compreender como fluxos dos dados ocorrem torna possível determinar parâmetros da rede como, por exemplo, o desempenho e localização e solucionar problemas relacionados à comunicação de dados.

O Netflow é um protocolo que transporta informações dos fluxos (*flows*) que atravessam um roteador Cisco. Cada pacote que é encaminhado por um roteador é caracterizado por um conjunto de atributos. Estes atributos estão nos cabeçalhos IP, TCP ou UDP e são como uma identidade ou impressão digital do pacote, que determina se ele é único ou similar a outros pacotes. Tradicionalmente um fluxo (*flow*) pode ser identificado por 5, 6 ou 7 atributos, sendo [12, 33]:

- Endereço IP de origem (informação de camada 3);
- Endereço IP de destino (informação de camada 3);
- Porta de origem (informação de camada 4);

- Porta de destino (informação de camada 4);
- Tipo de protocolo da camada 4 (informação de camada 3);
- Classe de serviço (informação de camada 3);
- Interface do roteador (informação de camada 2).

No modelo proposto neste estudo apenas os parâmetros endereço IP de origem, endereço IP de destino, porta de origem, porta de destino e o protocolo da camada 4 são utilizados como identificadores dos fluxos. Todos os pacotes com os mesmos identificadores são agrupados como um único fluxo, e todas as informações armazenadas em uma base de dados do roteador, chamada Netflow *cache*. Estas informações dos gravada dos fluxos são úteis para entender o comportamento da rede, sendo elas [25, 35]:

- Endereço de origem: permite conhecer quem está originando o tráfego;
- Endereço de destino: informa quem está recebendo o tráfego;
- Portas: caracterizam o aplicativo que está realizando o tráfego;
- Classe de serviço: examina a prioridade do tráfego;
- Interface do roteador: informa como a rede está usando o roteador;
- Contagem de pacotes e bytes: indica a quantidade de tráfego;
- *Timestamps* de início e fim do fluxo: úteis para determinar a duração do fluxo;
- Endereço IP do próximo salto: fornece informações sobre o roteamento;
- Máscaras de subredes para endereços de origem e destino; e
- Flags TCP: permite examinar os *handshakes* TCP.

Especificamente, o Netflow é utilizado para análise de novas aplicações e seus impactos na rede, solução de problemas, entendimento dos gargalos da rede, validação de parâmetros de qualidade de serviço das redes (QoS), identificação de tráfego WAN não autorizado e detecção de anomalias.

3.1.2 - Por que utilizar o Netflow?

Para atingir o objetivo específico deste estudo, é necessário determinar o comportamento dos fluxos de comunicação *peer-to-peer* e dos protocolos regulares. No

capítulo 4 é mostrado como o Netflow fornece estas informações (contagem de pacotes e *bytes* e *timestamps* de início e fim do fluxo).

Existe também a vantagem da coleta do Netflow não ser intrusiva, sem interferência na rede ou na comunicação dos *peers*. O formato simples dos dados e da transmissão também proporciona um maior tempo de monitoramento, que fornece um histórico adequado para outros tipos de análise [9, 35].

A praticidade do Netflow também é outro ponto positivo já que um roteador de borda de uma ISP, sendo do fabricante Cisco e com o IOS adequado, pode ser configurado para exportar Netflow em poucos minutos, causando apenas um pequeno impacto na rede [9, 35].

3.1.3 - Forma de acesso às informações do Netflow

O acesso aos dados dos fluxos coletados pelo Netflow pode ser realizado através da interface de linha de comando (CLI) do roteador ou utilizando um aplicativo externo para criação de relatórios. A forma mais rápida de acesso é pela CLI do roteador, contudo uma ferramenta externa disponibiliza formas de processamento dos fluxos, que tornam as informações mais amigáveis ao administrador de rede [33]. Essas ferramentas externas são conhecidas como coletoras de Netflow. No presente caso é utilizado um coletor de Netflow, *freeware*, construído em linux, o *flow-tools*.

Para que o coletor receba as informações é necessário que o roteador esteja configurado para tal. De forma geral, os seguintes passos devem ser seguidos [25]:

- Roteador configurado para capturar informações dos fluxos e armazenar no netflow *cache*;
- Roteador deve ser configurado para exportar o netflow *cache* para um endereço IP e uma porta de destino em um servidor coletor específico e em intervalos de tempos determinados pelo administrador;
- O *cache* é preenchido com informações de todos os fluxos terminados e transmitido em um pacote UDP para o coletor; e
- Software coletor de Netflow, instalado no servidor, cria um histórico de todos os fluxos para gerar relatórios dos dados trafegados.

Um fluxo está pronto para ser exportado quando ele se torna inativo durante um determinado período de tempo (novos pacotes não são computados para o fluxo), ou quando as *flags* TCP indicam que o *flow* está terminado (por exemplo, flags FIN, RST).

Existem vários formatos para os pacotes exportados pelo Netflow, sendo que cada um desses formatos é conhecido como versão [12]. Existem as versões 1,3,5,6,7,8 e 9. As variações do formato do pacote e dados armazenados determinam as diferenças entre elas. As versões mais documentadas são as 5, 7 e 9. Para este estudo é utilizado a versão 5 que é a mais comumente encontrada em redes de comunicação de dados.

3.1.4 - Mecanismos de coleta – *Flow-tools*

Flow-tools é uma biblioteca e uma coleção de programas usados para coletar, enviar, processar e gerar relatórios dos dados do Netflow [26]. As ferramentas podem ser utilizadas juntas ou separadas em um ou vários servidores distribuídos por uma rede. As bibliotecas *flow-tools* fornecem uma API para o desenvolvimento de aplicações personalizadas para as versões 1,5,6 e 8. A coleção de programas é construída utilizando as linguagens de programação *Perl* e *Python*.

Os *flows* são exportados de um roteador e o coletor é capaz de armazenar e processar apenas uma versão do Netflow por vez, já que as ferramentas do *flow-tools* trabalham apenas com uma única versão por arquivo [26].

Os seguintes programas estão incluídos na versão do *flow-tools* utilizada [26]:

- ***flow-capture*** – coleta, comprime, armazena e gerencia o espaço em disco dos *flows* exportados pelo roteador;
- ***flow-cat*** – concatena os arquivos de *flows*, que tipicamente contêm informações de pequenas janelas de tempo exportadas, usualmente 5 a 15 minutos;
- ***flow-fanout*** – replica os datagramas NetFlow para um ou vários destinos. Facilita o captura de múltiplos coletores associados a um único roteador.
- ***flow-report*** – gerencia relatórios de conjuntos de dados do Netflow;
- ***flow-tag*** – marca os fluxos baseando-se no endereço IP ou número do sistema o qual o fluxo pertence. Utilizado para agrupar os *flows* segundo critérios do administrador da rede;
- ***flow-filter*** – filtra os fluxos baseando-se em qualquer um dos campos exportados, e é utilizado na linha de comando em conjunto com outros programas do pacote;

- ***flow-import*** – importa dados de um arquivo de fluxos no formato ASCII;
- ***flow-export*** – exporta dados para um arquivo de fluxos para o formato ASCII;
- ***flow-send*** – envia dados sobre o uso da rede, utilizando o protocolo NetFlow;
- ***flow-receive*** – apenas recebe os dados exportados usando o Netflow sem armazená-los no disco;
- ***flow-gen*** – gera dados de teste;
- ***flow-dscan*** – ferramenta simples para detecção da alguns tipos de escâner de rede e ataques de negação de serviço.
- ***flow-merge*** – mescla arquivos de *flow* em ordem cronológica;
- ***flow-expire*** – trabalha em conjunto com o *flow-capture* para apagar os fluxos que expiraram segundo critérios estabelecidos pelo administrador;
- ***flow-header*** – mostra informações do cabeçalho do protocolo Netflow;
- ***flow-split*** – divide os arquivos de *flows* em arquivos menores baseado em tamanho, tempo ou *tags*.

Apenas algumas das ferramentas são utilizadas neste estudo, sendo elas *flow-capture*, *flow-cat*, *flow-nfilter* e *flow-export*. No capítulo 4 é apresentado detalhadamente como elas serão empregadas e sua função no processo de captura e apresentação dos dados coletados.

3.2 - Redes Neurais

Desde a invenção do computador, o homem vem tentando criar sistemas que sejam capazes de se comportar e agir como seres humanos. Contudo, na essência, os seres humanos possuem habilidades diferentes das encontradas nos computadores. Os seres humanos possuem aptidões como reconhecer objetos pela visão, diferenciar pessoas pela audição, solucionar problemas complexos com diversas variáveis através de um conhecimento, ou experiência adquirida com a vivência. Os computadores são, por sua vez, eficientes em realizar cálculos com vários dígitos e com grande velocidade, localizar nomes e telefones de pessoas em cadastros e outros aspectos que não são tão eficientes nos seres humanos [39].

Entretanto, na busca de reproduzir nosso comportamento e forma de solucionar problemas através das máquinas surgiu um campo de conhecimento, a Inteligência Artificial.

A Inteligência Artificial tem conseguido avanços no emprego de diferentes técnicas que simulam as ações do ser humano na busca de solução de problemas.

Uma das técnicas criadas em Inteligência Artificial busca simular o comportamento do cérebro humano. A falta de conhecimento a respeito do cérebro humano ainda é muito grande, a ponto de várias perguntas simples como “O que é a mente?”, “Como eu penso?” ainda permanecerem sem uma resposta exata. Alguns poucos aspectos são conhecidos sobre o seu comportamento e construção.

Sabe-se que o cérebro humano é formado por inúmeros componentes, conhecidos como neurônios, que formam uma estrutura complexa e paralelizada. Conhece-se também que a arquitetura do cérebro é dividida em áreas dedicadas, que permitem a realização de várias tarefas simultâneas como ver, ouvir, falar e solucionar problemas, combinando diversas variáveis complexas, encontrando um resultado correto. Desta forma, o cérebro possui funções que permitem ao ser humano realizar tarefas intrinsecamente complexas, que o computador não consegue realizar.

Na tentativa de adicionar aos computadores aspectos inerentes aos seres humanos surgiu uma linha de pesquisa em Inteligência Artificial conhecida como redes neurais artificiais, que investiga a reprodução do funcionamento e da arquitetura do cérebro em sistemas computacionais [39].

A forma de modelar o cérebro humano utiliza-se de pequenas unidades interconectadas que são chamadas, também, de neurônios. A intenção é criar uma estrutura paralela com reações parecidas às encontradas no encéfalo humano.

3.2.1 - Neurônio Biológico

O estudo das redes neurais artificiais segue em busca de uma forma de representar a forma de conhecimento humano. Nesta busca, vários modelos matemático-computacionais foram desenvolvidos com vários aspectos presentes no neurônio biológico e no comportamento do sistema nervoso, particularmente, os existentes no cérebro humano [39].

O neurônio é a parte básica do sistema nervoso, existindo em dois tipos: os que realizam o processamento das informações e os que interconectam partes do cérebro entre si, ou conectam partes do corpo ao cérebro. No seu processo básico, o neurônio possui várias entradas (dendritos) por onde ele recebe estímulos, que são acumulados até um determinado

nível, onde o neurônio passa a enviar estímulos para os outros neurônios que estão ligados à suas terminações (axônio).

A figura 3.1 ilustra os componentes de um neurônio biológico, elemento básico do sistema nervoso. Nela estão ilustrados o corpo, os dendritos e o axônio.

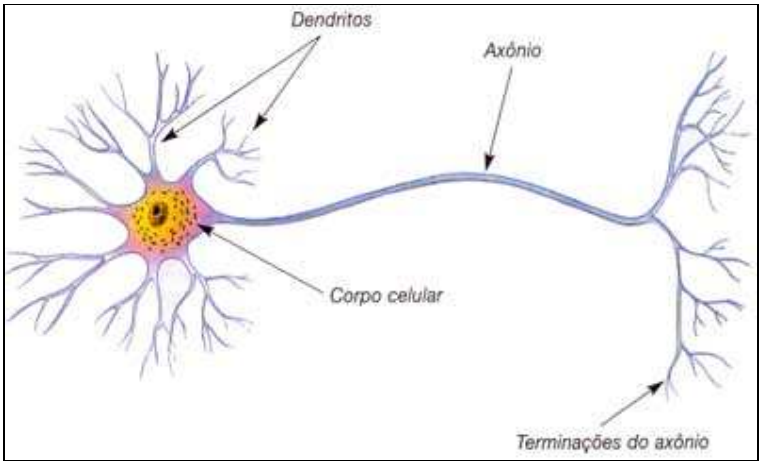


Figura 3. 1 – Neurônio biológico

Individualmente os neurônios realizam operações restritas, porém com as várias ligações e em grande número, eles proporcionam ao ser humano a capacidade de realizar tarefas complexas.

De forma geral, os axônios são responsáveis pela transmissão dos impulsos nervosos aos outros neurônios, relacionando-se com os dendritos que capturam os estímulos através das sinapses, enquanto no corpo celular encontram-se o núcleo e organelas responsáveis pelas demais atividades da célula.

Singularmente, o neurônio desempenha operações simples, entretanto, sabe-se que cada um realiza um somatório dos estímulos recebidos pelos seus dendritos. O axônio é responsável pela transmissão dos pulsos de tensão quando o potencial dentro do neurônio atinge um valor limite. O axônio termina em uma espécie de acionador, chamado sinapse, que realiza os contatos químicos entre os neurônios. Muitas sinapses podem atuar no mesmo neurônio. O conjunto das sinapses causa os estímulos no corpo do neurônio e o somatório de todos os estímulos determinam ou não a ativação do mesmo e a transmissão ou não do pulso pelo axônio [27].

O aprendizado no neurônio biológico ocorre pela modificação da eficiência das sinapses que interligam os neurônios. O conhecimento é determinado pela força das ligações químicas entre as células neurais. O ajuste do acoplamento entre as células é o aspecto mais importante no modelamento do neurônio biológico, e é conhecido como fator peso da conexão entre as unidades neurais artificiais [39].

3.2.2 - Modelamento de um neurônio

No modelamento do neurônio artificial, vários pequenos aspectos são abstraídos e apenas aspectos considerados importantes são extraídos, produzindo-se uma versão simplificada de forma a manter apenas o mesmo comportamento.

A função básica de um neurônio biólogo é acumular todos os estímulos em sua entrada e produzir ou não uma saída, caso o valor acumulado atinja certo valor limite. As excitações que chegam através dos dendritos possuem um determinado peso, que depende das sinapses entre o neurônio que envia sinal e o que o recebe. A intensidade da ligação química determina a quantidade de sinal que será enviada ao núcleo por aquela entrada. O resultado do acumulado, ou seja, resultado do somatório das entradas, ponderadas pelas sinapses, comparado com o valor limite decidirá se a saída pelo axônio será ativada ou não. Desta forma, para modelar um neurônio humano as seguintes características devem ser seguidas [27]:

- Saída com estados ligada ou desligada, que modela o comportamento do axônio;
- Cada entrada determinará um ganho diferente na ativação, modelando o comportamento das sinapses;
- O neurônio realiza o somatório das entradas recebidas; e
- O neurônio é ativado a partir de certa quantidade de entradas ativas, de forma que a saída depende do estímulo na entrada.

McCulloch e Pitts [27] propuseram um modelo para o neurônio biológico chamado de perceptron McCulloch-Pitts. O nome "perceptron" foi dado para os modelos de neurônios conectados de forma simples. Portanto, este modelo, mostrado na figura 3.2, tenta simular as funções biológicas que ocorrem dentro de uma célula do sistema nervoso implementando as características elucidadas anteriormente.

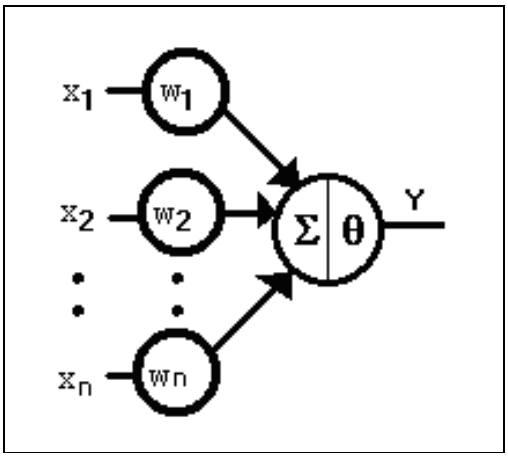


Figura 3. 2 – Modelo do neurônio artificial

As informações fornecidas por outros neurônios correspondem aos estímulos de entrada X_1, X_2,...,X_n, cada entrada possui ligações sinápticas W_1, W_2,...,W_n, conhecidas como pesos sinápticos. Os pesos sinápticos são fixos e a saída Y é obtida pela aplicação de uma função de limiar. O perceptron realiza o somatório das entradas X, ponderadas pelos pesos W, e compara com o nível limiar θ, somente ativando ou não a saída Y do neurônio se este nível for ultrapassado [27].

Este modelo implementa as funções necessárias, e o fato das entradas atuarem no neurônio para produzir a saída tornou o modelo conhecido na literatura como *feedforward*.

Pode-se formular matematicamente o modelo. Adimitindo-se n entradas e n pesos associados a cada linha de entrada, pode-se dizer que o sinal total na entrada, *net*, é o somatório do valor de cada entrada multiplicado pelo peso da mesma [27]:

$$net = \sum_{i=1}^{n} x_i w_i \tag{3.1}$$

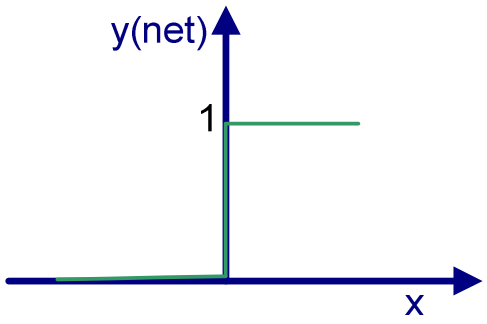
onde w_i é o peso ou ponderação na entrada i e x_i é o sinal na entrada i. A saída y terá valor 1 se a soma ponderada for maior que o limiar θ, e 0, se for menor. Assim,

$$y = f_h\left[\sum_{i=1}^{n} x_i w_i - \theta\right] \tag{3.2}$$

onde f_h é a função de *Heaviside*, figura 3.3, logo:

$$f_h(x) = \begin{cases} 1 ou "ligado" se\ x > 0 \\ 0 ou "desligado" se\ x \le 0 \end{cases}$$

(3.3)



Generalizando, pode-se concluir que, para os estados *0* e *1*, os pesos são diminuídos quando se busca a saída e *0* e são aumentados quando se pretende obter *1* na saída.

Todo o processo de treinamento está baseado no conhecimento das entradas que devem ser obtidas. Como se é guiado sabendo o resultado a ser obtido, a técnica é conhecida como aprendizado supervisionado.

Portanto, o algoritmo de aprendizagem de uma unidade perceptron, de forma a ser implementado computacionalmente, pode ser descrito pelos passos seguir [27]:

- Passo 1: Iniciar os pesos $w_i(t) = \{w_0; w_2;...; w_n\}$, como peso da entrada i no tempo t e o valor limite de "disparo" ou ativação da unidade perceptron, θ. Todos os pesos devem ser pequenos e aleatórios durante a inicialização;
- Passo 2: Apresentar os valores de entrada e de saída desejados. São mostrados os atributos $x_1; x_2;...; x_n$ do elemento e a saída esperada $y(t)$;
- Passo 3: Calcular a saída atual pelo somatório do produto entre as entradas e os pesos iniciais O resultado do somatório determina ou não a ativação do neurônio pela comparação com o valor de limiar. Como segue:

$$y = f_h\left[\sum_{i=1}^{n} x_i(t) w_i(t)\right] \tag{3.4}$$

- Passo 4: Caso o valor saída não apresente o valor esperado, é realizada a adaptação dos pesos. Através do descrito a seguir:
  Se correto, $w_i(t+1) = w_i(t)$
  Se saída 0, e deveria ser 1, então $w_i(t+1) = w_i(t) + x_i(t)$
  Se saída 1, e deveria ser 0, então $w_i(t+1) = w_i(t) - x_i(t)$

Basicamente, este é o algoritmo de aprendizado. Os passos de 2 até 4 são repetidos até que a saída apresente resultados dentro de uma tolerância esperada, ou seja, o elemento deve ser classificado corretamente. Contudo várias modificações foram sugeridas para o algoritmo.

Uma modificação proposta para melhorar o processo de aprendizagem foi a aplicação de um ganho à adaptação dos pesos, este ganho recebeu o nome de taxa de aprendizado. Variando seus valores entre *0* e *1*, a taxa de aprendizado multiplica o termo de adaptação dos pesos, dessa forma controla a velocidade de mudança e aprendizado, melhorando a adaptação dos neurônios da rede, e assim fornecendo mais estabilidade ao processo de treinamento [27].

Um outro algoritmo propõe que os pesos devem ser mudados mais rapidamente se saída estiver muito longe do valor desejado, e à medida que o valor esperado para a saída se a aproxima, os pesos devem ser apenas ligeiramente alterados. Esta proposta cria uma variável *erro*, que será obtida pela diferença entre a saída obtida e a esperada. Os pesos da rede serão ajustados proporcionalmente ao valor do *erro*, isto é, durante o processo de treinamento o passo dado não é em função da saída a ser esperada, mas em função do *erro* obtido para a saída esperada.

3.2.4 - Visão vetorial da unidade perceptron

Vetorialmente, a unidade perceptron é ativada ou não pela comparação do valor de limiar com o resultado do produto escalar entre o vetor de entrada X e o vetor de pesos W.

o45.62942(e)-6.85594(n)-4.5326C931 5

Conforme o processo de aprendizado, o vetor *W* é inicialmente aleatório, e com a apresentação dos elementos de cada classe ao sistema neural o processo de aprendizagem ajusta o vetor *W*, reduzindo o erro até que, no final, todos os elementos sejam corretamente classificados em suas classes.

Para as classes *1* e *2*, mostradas na figura 3.4, pode-se ilustrar o comportamento do processo de aprendizado do perceptron examinando o comportamento do vetor W conforme ilustrado na figura 3.5. A solução para a classificação das classes é produzir um vetor que crie uma linha que particiona o espaço em duas classes de padrões. A intenção é que o perceptron encontre esta linha sozinho, através do processo de aprendizado [27].

A linha é localizada ajustando os valores dos elementos do vetor de pesos, inicialmente aleatório e que aponta para qualquer lugar do espaço. As entradas são apresentadas, e com as saídas incorretas, o treinamento ajusta os pesos para diminuir o erro, mudando a posição do vetor *W*, uma quantidade finita de vezes, até encontrar a solução ideal ou aproximada. Dessa forma, o perceptron aprende a distinguir entre as classes 1 e 2.

Supondo que quatro iterações sejam suficientes para encontrar as partições corretas, pode-se observar a localização dos vetores W_1, W_2, W_3 e W_f , quando a divisão das classes é feita corretamente [27].

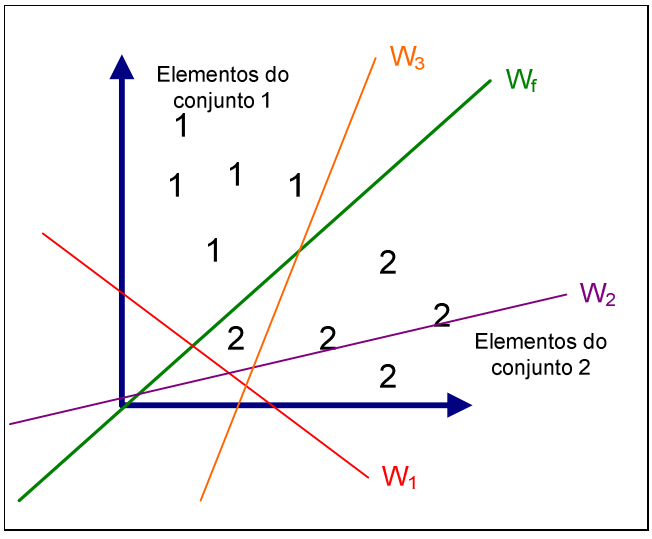


Figura 3. 5 – Divisão das classes pelo vetor W

A figura 3.6 mostra o comportamento dos vetores W_1, W_2, W_3 e W_f à medida que os pesos são ajustados em cada iteração.

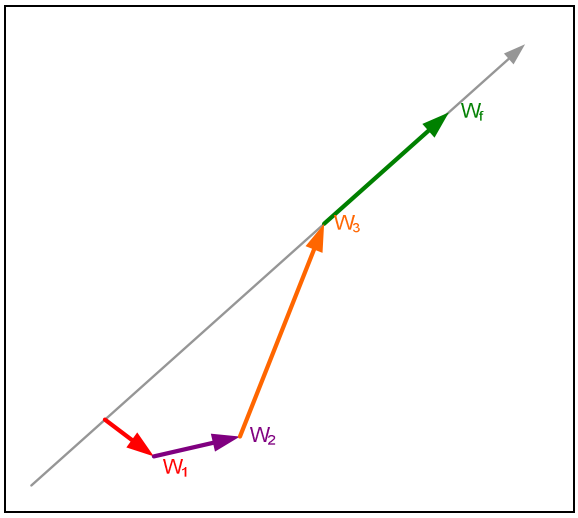


Figura 3. 6 – Comportamento do vetor W

Há, no entanto, situações em que não existe uma divisão linear entre as classes. Neste caso, a divisão das classes é mais complexa e o perceptron não consegue encontrar uma solução, conforme será descrito no item 3.2.5.

3.2.5 - Limitações da unidade perceptron

Para entender a limitação do modelo perceptron, deve-se lembrar que ele encontra uma linha que particiona o espaço em classes. Contudo existem situações em que a divisão entre as classes não pode ser realizada utilizando apenas uma única linha, pois a distribuição das classes no espaço é mais complexa [27].

O exemplo clássico que ilustra essa limitação é o caso da representação da função lógica OU - exclusivo, ou XOR. A função lógica XOR possui duas entradas e uma saída e produz como tabela verdade os resultados da tabela 3.

Tabela 3 – Tabela verdade do OU - exclusivo

Entrada X	Entrada Y	Saída Z
0	0	0
0	1	1
1	0	1
1	1	0

A figura 3.7 mostra a distribuição espacial das classes de saída *0* e *1* do OU - exclusivo. Pode-se observar de forma relativamente simples que não se consegue dividir as classes 0 e 1 com uma única linha. Este tipo de problema é conhecido como “linearmente inseparável”, no qual o perceptron não é capaz de encontrar a linha divisória.

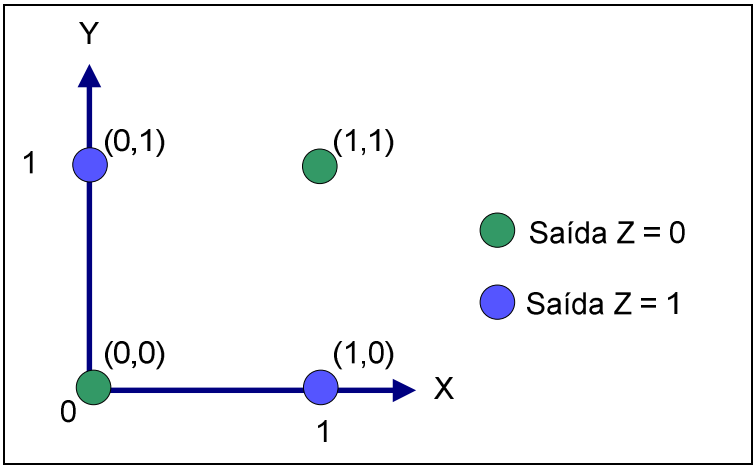


Figura 3. 7 – Ilustração espacial do problema OU - Exclusivo

Conseqüentemente, a limitação consiste no fato do modelo ser somente aplicável em problemas linearmente separáveis, o que praticamente encerrou os estudos na área de redes neurais da devida época. Somente com o surgimento do modelo Perceptron Multicamadas (ou Perceptron de Múltiplas Camadas, ou Multi Layer Perceptron ou ainda MLP) que houve avanços significativos para a continuação da pesquisa. Detalhes deste novo modelo são apresentados no item que se segue.

3.2.6 - Perceptron Multicamadas

Para superar as limitações encontradas no modelo do perceptron, a primeira abordagem foi a utilização de vários perceptrons para identificar pequenas áreas linearmente separáveis e combinar suas saídas em outro perceptron para produzir a separação final das classes. Por exemplo, para a solução do problema do OU - exclusivo, utiliza-se a arquitetura mostrada na figura 3.8 [27].

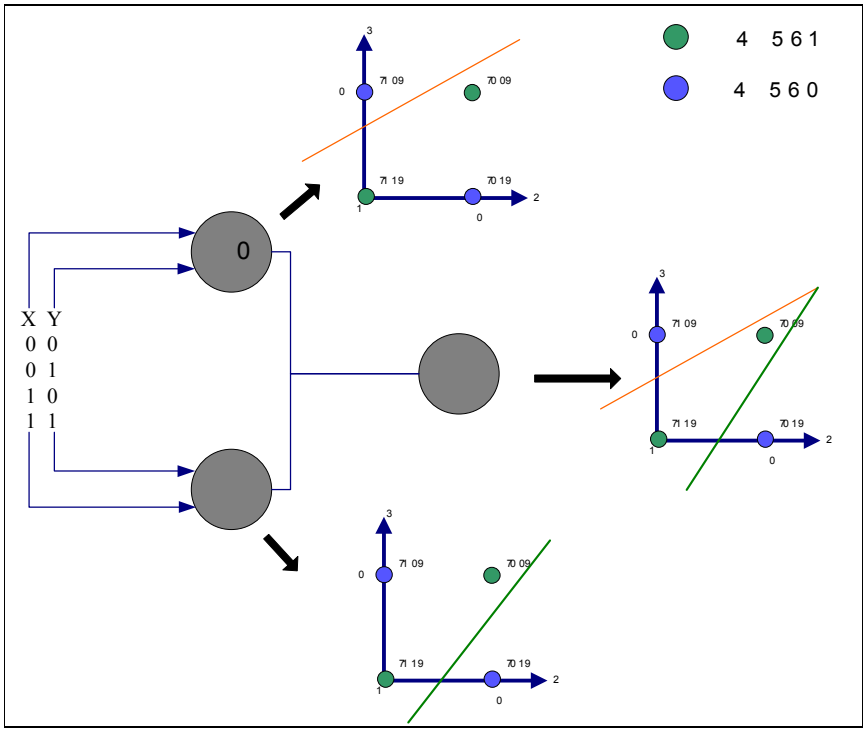


Figura 3. 8 – Solução do OU - exclusivo combinando três perceptrons

Fornecendo-se as saídas dos perceptrons 1 e 2 à entrada do perceptron 3 “combinam-se as duas linhas” para construir as partições que separam as classes de saída do OU - exclusivo.

Na arquitetura mostrada na figura 3.8, as unidades perceptron estão ligadas, a saída dos neurônios 1 e 2 servem como entrada para o neurônio 3, da camadas seguinte. As unidades perceptron continuam realizando os mesmos cálculos internamente, ou seja, o somatório das entradas ponderadas pelos pesos.

Para sistemas mais complexos ainda permanece um problema, já que pela função de *Heaviside*, as informações da entrada da rede neural são perdidas, uma vez que as camadas posteriores recebem apenas a informação 0 ou 1, de acordo com a ativação ou não dos neurônios da primeira camada. Portanto, o ajuste das camadas posteriores nem sempre produzem resultados satisfatórios, porque as nuâncias dos valores das entradas são ignoradas.

A solução encontrada para o problema foi alterar a função de ativação de *Heaviside* por outros tipos de funções ligeiramente diferentes. As outras funções de ativação escolhidas foram a sigmoidal e linear, mostradas na figura 3.9 [27].

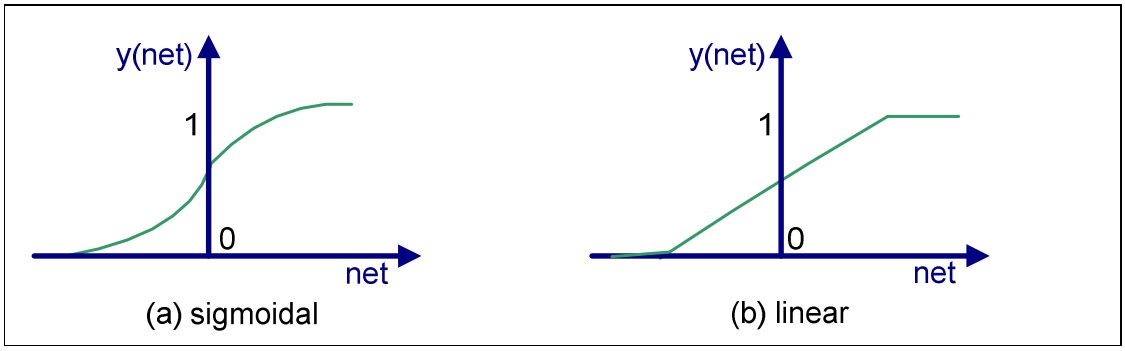


Figura 3. 9 – Funções de ativação

No caso das funções linear e sigmoidal, o valor da saída será praticamente 1,se a soma ponderada extrapolar o valor limite, da mesma forma, praticamente zero se o resultado for inferior ao limite. Para valores do somatório próximos ao valor limite, a saída oscila entre 0 e 1. Desta forma é possível que as nuâncias dos valores das camadas anteriores sejam passadas de forma mais significativa às camadas posteriores. Em termos práticos a função sigmoidal é geralmente escolhida, já que camadas de perceptrons com funções lineares podem ser reduzidas a uma única camada [27]. Desta forma foi criado o modelo de múltiplas camadas, conforme a figura 3.10.

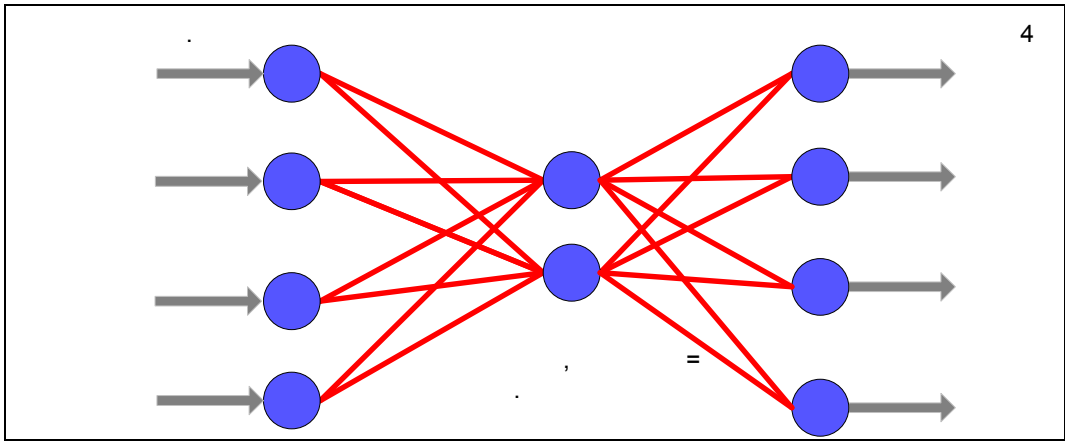


Figura 3. 10 – Modelo de múltiplas camadas com uma camada escondida

O modelo é basicamente composto por três níveis de camadas: a entrada, as intermediárias (ou ocultas) e de saída, podendo haver MLP com várias camadas escondidas.

No modelo MLP a camada de entrada tem apenas o propósito de distribuir os valores recebidos para a camada seguinte. Ela não realiza soma ponderada das entradas e também não compara as entradas com o valor de limiar.

A mudança na função de ativação e a adição de camadas escondidas implicam mudanças no sistema de aprendizado, já que a rede neural artificial deverá ser capaz de solucionar problemas mais complexos.

3.2.7 - Novo algoritmo de aprendizado

Para o modelo perceptron de uma camada era feita a comparação do valor da saída esperada com o valor obtido para os ajustes dos pesos, de forma que, na próxima iteração, a

A. Com os novos pesos de entrada de *A* é possível determinar a função de erro dos neurônios da camada anterior, por exemplo, para *B* e *C* é possível calcular o ajuste nos seus pesos de entrada a partir da diferença das suas saídas anteriores e as saídas atuais determinadas pelo ajuste de *A*. Desta forma o processo prossegue até que todos os pesos estejam reajustados.

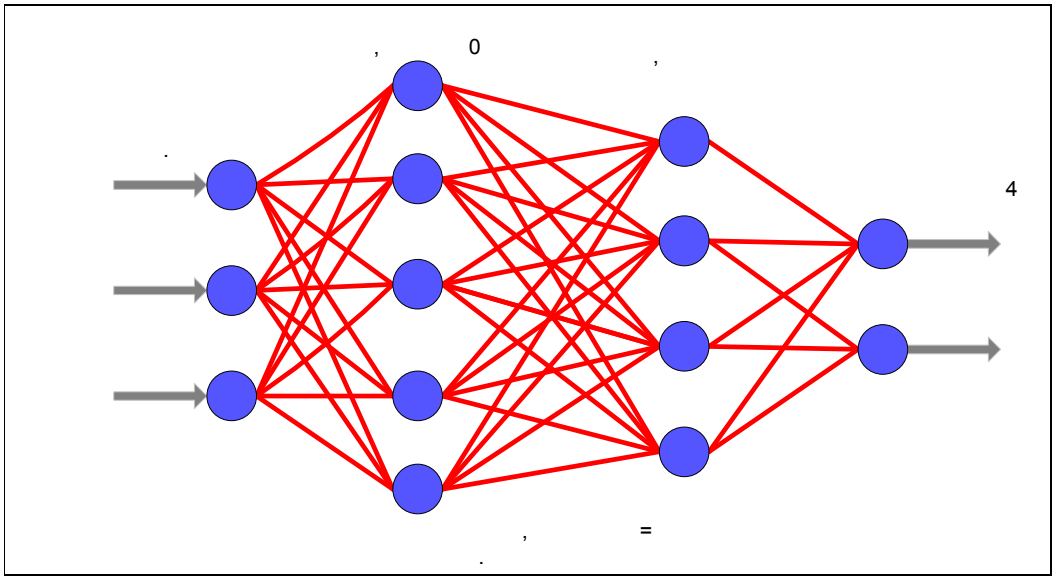


Figura 3. 11 – Modelo de múltiplas camadas com duas camadas escondidas

3.2.8 - Algoritmo MLP

O algoritmo para treinamento de uma rede neural artificial multicamadas, ou *Multi Layer Perceptron* (MLP), exige que as unidades possuam funções limitadoras não-lineares e continuamente diferenciáveis, assume-se, por conveniência, esta função como sendo a sigmoidal [27], uma vez que é a mais utilizada como função de ativação dos neurônios artificiais. Mostrada a seguir na equação 3.6.

$$f(t) = \frac{1}{1+e^{-kt}} \tag{3.6}$$

que apresenta derivada simples.

Assim sendo, o algoritmo de aprendizagem de uma MLP pode ser descrito pelos passos seguintes:

- Passo 1: Iniciar os pesos das sinapses $w_i(t) = \{w_0; w_2;...; w_n\}$ *de cada entrada i* no tempo *t* e o valor limite de "disparo", ou ativação, da unidade perceptron, θ. Todos os pesos devem ser pequenos e aleatórios durante a inicialização. Exceto por w_0 *que deve ser* $-\theta$ e x_0 sempre 1;
- Passo 2: Apresentar os valores de entrada e de saída desejados. São mostrados os atributos $X_p = x_0; x_2;...; x_{n-1}$ do elemento a serem apresentados à camada de entrada e os valores de saídas esperados $T_p = t_0;t_2;...;t_{m-1}$, em que *n* é o número de atributos de entrada e *m* o número de saídas; Para a classificação dos padrões T_p,todos os elementos da saída são configurados para 0, exceto aquele que representa a classe do elemento de entrada X_p. O erro para *P* é dado pela equação 3.7.

$$E_p = \frac{1}{2}\sum (t_{pj} - o_{pj})^2 \tag{3.7}$$

- Passo 3: Calcular a saída atual. Para cada camada é calculada a saída pela equação 3.8,

$$y_{pj} = f\left[\sum_{i=0}^{n} w_i(t)x_i(t)\right] \tag{3.8}$$

que será a entrada da camada posterior até que a última camada apresente os valores de saída obtidos, o_{pj}; e

- Passo 4: Adaptação dos pesos. Inicia-se na camada de saída e retorna para a camada imediatamente anterior, pela seguinte fórmula:

$$w_{ij}(t+1) = w_{ij}(t) + \eta\delta_{pj}o_{pj} \tag{3.9}$$

onde w_{ij} representa os pesos da unidade *i* para a unidade *j*, η é o ganho e δ_{pj} é um termo de erro para a entrada *p* na unidade *j*.

$$o_{pj} = f_i\left(\sum_i w_{ij}o_{pi}\right) \tag{3.10}$$

Para as unidades de saída:

$$\delta_{pj} = ko_{pj}(1-o_{pj})(t_{pj}-o_{pj}) \tag{3.11}$$

Para as unidades intermediárias:

$$\delta_{pj} = ko_{pj}(1-o_{pj})\sum_{k}\delta_{pk}w_{pk} \tag{3.12}$$

onde a soma é para as *k* unidades na camada posterior a camada da unidade *j*.

Basicamente, este é o algoritmo de aprendizado. Os passos de 2 até 4 são repetidos até que a saída apresente resultados dentro do esperado, ou seja, o elemento deve ser classificado corretamente.

3.2.9 - Classificação utilizando MLP

À medida que camadas e unidades são adicionadas à rede neural, mais partições são possíveis de serem encontradas. Cada uma dessas partições forma uma região convexa, ou seja, uma área no espaço, limitada, de forma que um par qualquer de pontos dentro desta região possa ser interligado sem atravessar a borda ou regiões diferentes [27, 39]. A adição de unidades na primeira camada escondida determina a quantidade de linhas que são utilizadas para construção das regiões. Por exemplo, para o problema do OU - exclusivo o sistema neural divide o espaço em duas regiões de classificação, conforme mostrado na figura 3.12, utilizando duas linhas, cada uma construída por uma única unidade da camada escondida.

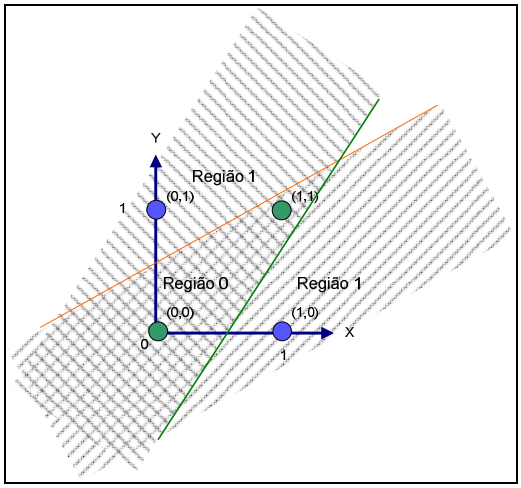


Figura 3. 12 – Regiões de decisão para o OU - exclusivo

A combinação de muitos neurônios na primeira camada escondida possibilita formar qualquer tipo de região poligonal convexa.

Outro aspecto a ser mencionado é a adição de uma segunda camada escondida à MLP. Ao se acrescentar outra camada de perceptrons após a primeira, esta segunda camada passará a receber em suas entradas informações de regiões convexas vindas das saídas da primeira camada. A combinação destas áreas convexas pode gerar outras áreas convexas ou não, porém o importante é que com duas camadas escondidas é possível criar interseções, sobreposições e produzir qualquer área arbitrária que possa separar as diferentes classes [27].

Assim sendo, um perceptron com uma camada de entrada, duas camadas escondidas e uma de saída é capaz de separar qualquer tipo de classe. A complexidade das figuras formadas depende do número de neurônios da rede neural, pois este número determina a quantidade de linhas que poderão ser utilizadas. Isso significa que quatro camadas é o suficiente para classificar qualquer tipo de problema.

3.2.10 - Generalização

Uma das maiores vantagens das redes neurais artificiais é sua capacidade de generalização que, em linhas gerais, é a capacidade de classificar padrões desconhecidos, ou seja, ainda não apresentados ao sistema neural [39].

A propriedade da generalização presente em perceptrons multicamadas permite que padrões sem erros definidos possam ser classificados devido à similaridade com os padrões puros. Isso significa também que padrões com ruídos, ou mal amostrados possam ser

classificados corretamente em virtude de sua similaridade com padrões legítimos. As redes neurais são capazes de interpolar e extrapolar com grande precisão. Essas características determinaram o sucesso das redes neurais artificiais em relação a outras técnicas inteligentes contemporâneas [27].

Pelas características de classificação, generalização e solução de problemas não lineares, as redes neurais foram escolhidas para classificação de tráfego *peer-to-peer* pela análise de comportamento, já que é difícil mapear através de uma função matemática o comportamento gerado pelos fluxos (*flows*).

3.3 - Ferramenta *NNTool* do Matlab

O Matlab é um ambiente científico, disponibilizado pela MathWorks, que possui uma vasta gama de funções matemáticas, processamento de sinais, ferramentas de controle, etc [28]. Trata-se de um aplicativo científico que é utilizado na implementação do modelo neural MLP proposto para o estudo. O conjunto de implementação de vários modelos neurais presentes no MatLab formam a ferramenta conhecida como *NNTool*.

Neste tópico é descrito, sucintamente, a utilização do *NNTool* na construção do modelo neural utilizado.

3.3.1 - Modelos de rede neural e suas implementações

Uma rede neural no Matlab pode ser criada de várias formas, podendo ser utilizado a função *newff*, que cria uma rede *feed-forward* com a regra *back-propagation* para o processo de aprendizado. Através deste comando é possível configurar vários parâmetros da rede neural, sendo [28]:

- Valores máximos e mínimos dos atributos dos elementos de entrada;
- Tamanho de cada uma das camadas escondidas;
- Tipo de função de ativação dos neurônios de cada camada, i. ex. *logsig* (sigmoidal logística), *purelin* (linear), etc;
- Função de treinamento da rede neural, como *trainlm*, *trainbfg*, etc que determinam otimizações na velocidade e uso de memória durante o cálculo;
- Função de aprendizado para os valores de pesos/conexões da rede como *learngd*, e *learngdm*; e

- Medida de desempenho utilizada, como *sse* (somatórios dos quadrados erros), *mse* (erro médio), etc.

No modelo construído para estudo foram utilizados os parâmetros configurados da seguinte forma:

- Valores máximos e mínimos dos atributos dos elementos de entrada entre 0 e 1;
- Modelos com 1 ou 2 camadas escondidas e variadas combinações de neurônios em cada uma delas;
- Função de ativação dos neurônios: *logsig* (sigmoidal logística) ilustrada na figura 3.13;

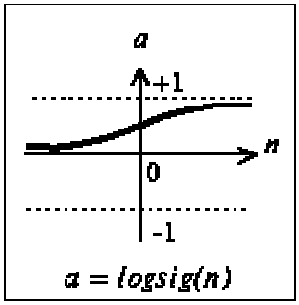


Figura 3. 13 – Função de transferência log-sigmoidal

- Função de treinamento de rede neural: *trainlm*, a mais veloz com maior uso de memória;
- Função de aprendizado dos valores de pesos/conexões da rede: *learngd*, que determina a de aprendizado em função da variação do vetor gradiente [28]; e
- Medida de desempenho utilizada: *sse*, somatórios dos quadrados dos erros.

Os parâmetros de treinamento da rede neural são configurados pela função *trainlm*, que atualiza os valores dos pesos/conexões de acordo com a otimização de Levenberg-Marquardt [28]. Essa função utiliza um conjunto de entradas para treinamento, outro conjunto de entradas para validação e um terceiro conjunto para testes da rede.

O desempenho da classificação dos elementos do conjunto de validação é utilizada para encerrar o processo de treinamento. Caso o desempenho da classificação do vetor de validação diminua ou permaneça inalterado por uma quantidade determinada de épocas o

treinamento é encerrado. Este mecanismo permite maior generalização dos pesos da rede neural, já que impede que o treinamento exaustivo torne a rede especialista apenas em detectar os elementos do conjunto de treinamento, e é conhecido como validação cruzada [28].

O conjunto de teste especificado na função *trainlm* é usado para checar o desempenho de classificação dos padrões não conhecidos pela rede neural.

Os principais parâmetros de entradas que devem ser configurados para o uso da função *trainlm* são os seguintes:

- Rede neural criada através da *newff*;
- Vetores de treinamento, validação e teste;
- Número de épocas;
- Taxa de aprendizado inicial;
- Limite de erro; e
- Tempo de treinamento;

E a função retorna os seguintes parâmetros:

- Número final de épocas;
- Desempenho da classificação dos conjuntos de treinamento, validação e teste; e
- Módulo do vetor gradiente que determina a taxa de aprendizado.

O treinamento pode ser encerrado caso ocorra as seguintes condições:

- Atingir um número máximo de épocas de treinamento;
- Quantidade máxima de tempo de treinamento é atingida;
- O desempenho alcança o valor desejado;
- Desempenho da classificação do conjunto de validação diminui ou não se altera por um determinado número de épocas.

Outro importante aspecto é que a *NNTool* possibilita a inicialização dos pesos e dos valores limites de forma aleatória e de maneira muito simples. A função *initnw* implementa o algoritmo de Nguyen-Widrow que gera valores aleatórios para as camadas ativas da rede neural, ou seja, camadas escondidas e de saída [28].

Com a facilidade proporcionada pelo Matlab foram implementadas duas MLPs. A primeira MLP possui apenas uma camada escondida e a segunda MLP duas camadas ocultas.

Após a descrição dos procedimentos realizados, no capítulo 5 são apresentados os resultados e os desempenhos das redes neurais com uma e duas camadas. No apêndice C consta a implementação construída no Matlab para as redes MLP.

CAPÍTULO 4 – MODELO PROPOSTO

Em busca de uma solução que se adapte às novas características de troca de arquivos em redes *peer-to-peer* uma abordagem mais profunda deve ser utilizada para identificar a comunicação de dados. Neste estudo é descrito um modelo de avaliação capaz de distinguir, através do comportamento dos fluxos da comunicação, os tráfegos *peer-to-peer* em relação aos protocolos "convencionais" (http, ftp, dns, etc). Nos experimentos realizados, dois protocolos convencionais são utilizados para comparação com o P2P, o protocolo http e ftp.

Alguns trabalhos sobre o comportamento das conexões *peer-to-peer* [3,4,9,10] ilustram as diferenças entre as condutas dos protocolos P2P e os protocolos regulares. Aspectos como tempo de conexão, volume de dados e número de pacotes são mostrados em gráficos que, ao serem analisados por administradores de redes experientes, permitem a distinção entre os dois tipos de tráfego.

4.1 - A identificação do tráfego P2P pelo operador humano

Estudos anteriores mostram que é possível a identificação do tráfego P2P pela análise dos fluxos que atravessam um roteador [3]. Dados obtidos através de arquivos coletados no formato binário, gerado pelo Netflow, no ponto de presença (PoP) da RNP em São Paulo (SP), são avaliados por operadores humanos que identificam os tráfegos P2P pelo comportamento básico (volume, vazão e duração) dos fluxos.

A figura 4.1 mostra o perfil de tráfego relacionado ao volume passante no PoP-SP. Pela avaliação dos gráficos, conclui-se que a contribuição de tráfego P2P não é relevante, visto que o volume gerado é substancialmente menor do que o tráfego gerado por aplicações *web* [3].

Em termos de quantidade de fluxos, pode ser observado na Figura 4.2 seu perfil para os fluxos *web* e P2P. Em contraste com o perfil do volume de tráfego, o número de fluxos P2P é substancialmente superior ao dos fluxos *web* [3].

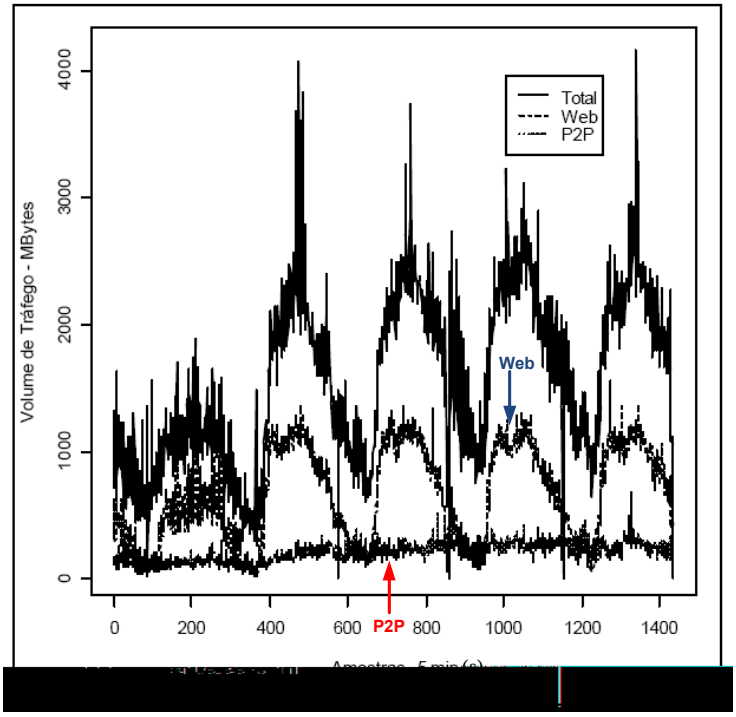


Figura 4. 1 – Perfil de tráfego em volume transferido (Mbytes) no PoP-SP

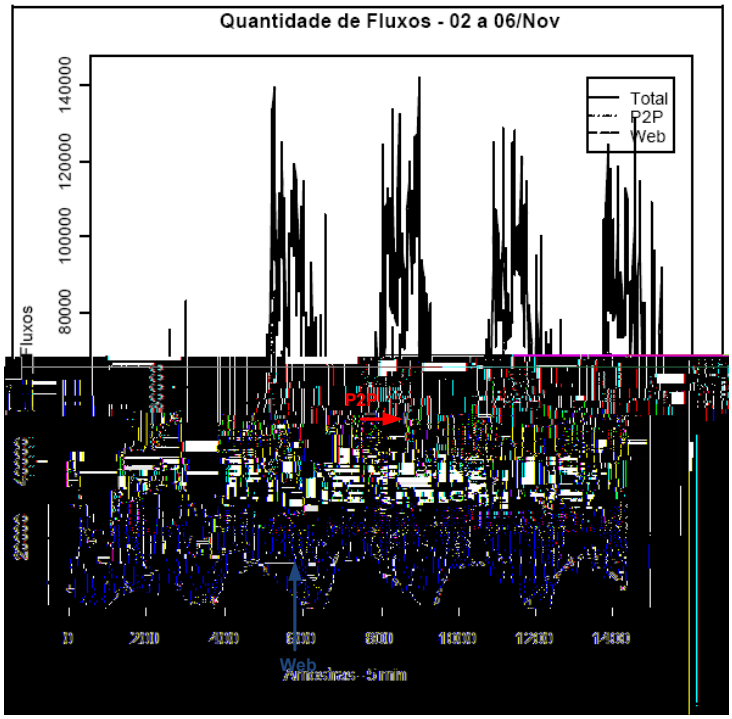


Figura 4. 2 – Perfil de tráfego em quantidade de fluxos (PoP-SP)

Através de técnicas de análise de conteúdo dos campos de dados dos pacotes, foi recentemente observado as sessões de downloads dos aplicativos P2P que estavam utilizando portas padrões (e.g., http/80, dns/53, etc) para a transferência de dados[1-7]. Com isso, quaisquer anomalias na distribuição do volume de tráfego por porta de serviço padrões conhecidas, podem intuitivamente fazer com que o administrador de redes conclua que o aplicativo P2P está trafegando dados pela portas padrão.

Na Figura 4.3 é apresentada a análise da contribuição do volume de tráfego gerado por fluxos de aplicações na porta 80 (tipicamente tráfego *web*) de acordo com sua classificação nas faixas de 0 a 10KB, de 10KB a 100KB, de 100KB a 1MB, de 1MB a 10MB, de 10MB a 100MB e de 100MB a 1GB. O perfil esperado do tráfego nessa porta é de maior contribuição no volume de tráfego associado às duas primeiras faixas de classificação [3]. Porém, o maior volume de tráfego gerado está associado às faixas de 100KB a 100MB. Como esse comportamento é inesperado, isso é um forte indicativo de que os aplicativos P2P estejam utilizando a porta 80 para transferência de dados.

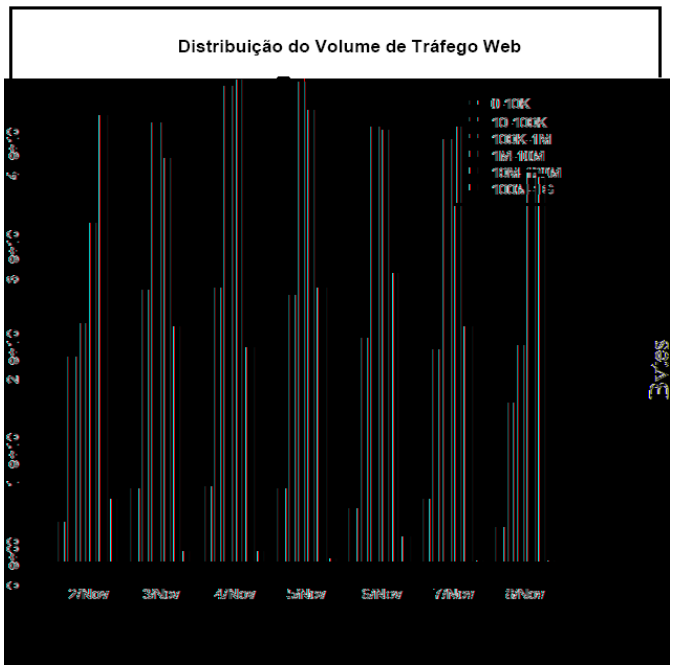


Figura 4. 3 – Distribuição do volume de tráfego na porta 80

Além da porta 80, essa análise foi estendida para a porta de serviço dns/53. Os resultados da Figura 4.4 apresentam perfis diferentes dos usuais, pois em alguns períodos aparece um grande volume de tráfego com fluxos na faixa de 10MB a 100MB, o que não é compatível com os tamanhos típicos de pacotes para esses serviços [3].

Figura 4. 4 – Distribuição do volume de tráfego na porta 53

Outra métrica utilizada para caracterizar o comportamento do tráfego P2P é a inferência sobre os tempos de conexões destes aplicativos. Na Figura 4.5 (gráfico log-log) esta distribuição aparenta ter as propriedades de uma distribuição *zipf* (*power law*) [3]. A propriedade básica de uma distribuição *zipf* é de um grande número de ocorrências de valores muito pequenos e a existência de cauda pesada para ocorrência de valores muito grandes que pode caracterizar uma comunicação *peer-to-peer*.

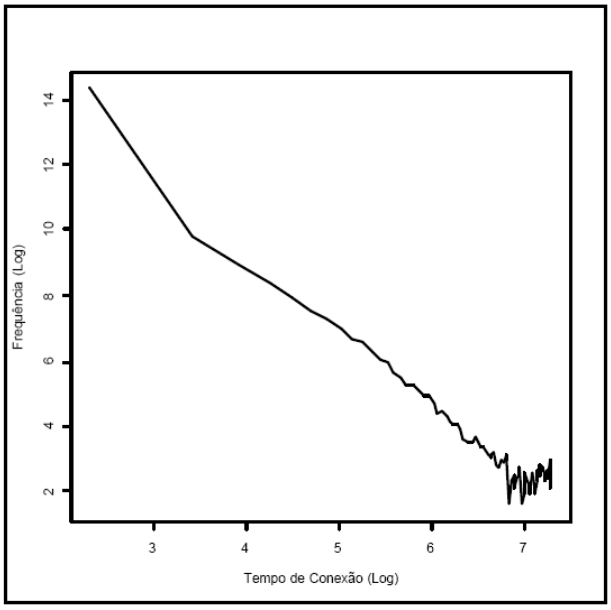


Figura 4. 5 – Distribuição do tempo de conexão dos aplicativos (log-log)

Pelas análises, os resultados mostram indícios de um grande impacto do tráfego dos aplicativos P2P no tráfego global da rede. Adicionalmente foram obtidos fortes indicativos de que a maior quantidade de tráfego P2P no *backbone* da RNP não é transferida através das portas tradicionalmente utilizadas pelos aplicativos P2P.

A grande contribuição do estudo desenvolvido em [3] é que a análise realizada pelos autores sobre os dados dos fluxos demonstra que operadores humanos podem identificar o tráfego P2P em relação ao tráfego gerado pelos protocolos convencionais como, por exemplo, http e dns, pela observação dos fluxos na comunicação de dados e no conhecimento prévio (experiência) sobre o comportamento de cada tipo de protocolo.

4.2 - A aplicação de redes neurais artificiais na detecção do tráfego P2P

Partindo da forma de análise mostrada acima, o sistema baseado em comportamento do tráfego utilizaria o reconhecimento de padrões dos fluxos gerados pela comunicação P2P para investigar a possibilidade de classificação do tráfego pela maneira como ele ocorre durante a conexão. Esse sistema deve ser capaz de analisar os dados da mesma forma que um ser humano especialista os analisaria. Logo, uma rede neural artificial que fosse treinada para reconhecer os fluxos P2P em relação aos protocolos regulares poderia executar a tarefa de classificação dos pacotes.

Para esse estudo é empregado o modelo neural perceptron multicamadas (MLP) para inferir sobre os tempos de conexões dessas aplicações, protocolo da camada de transporte, número de pacotes trafegados e número de *bytes* trafegados, identificando as conexões P2P. Esses dados são obtidos pela coleta de um protocolo proprietário da empresa Cisco, o Netflow [12], que fornece informações dos fluxos das conexões que atravessam os seus equipamentos.

Pelas características de classificação, generalização e solução de problemas não lineares, as redes neurais foram escolhidas para classificação de tráfego *peer-to-peer* a partir da análise de comportamento, já que é impossível mapear através de uma função matemática o comportamento gerado pelos fluxos (*flows*) [27].

Esta abordagem fornece um mecanismo para identificação e marcação de pacotes *peer-to-peer*, proporcionando que as decisões sobre o tratamento dos fluxos de comunicação sejam realizadas em outro equipamento como, por exemplo, um roteador de borda. Um modelo de solução para o ambiente real é sugerido no capítulo 7.

CAPÍTULO 5 – Metodologia

5.1 - Apresentação da metodologia

O capítulo 5 faz uma apresentação detalhada de todo o procedimento realizado em busca do objetivo proposto para este estudo. O diagrama de blocos mostrado na figura 5.1 fornece uma visão geral de todas as etapas do processo.

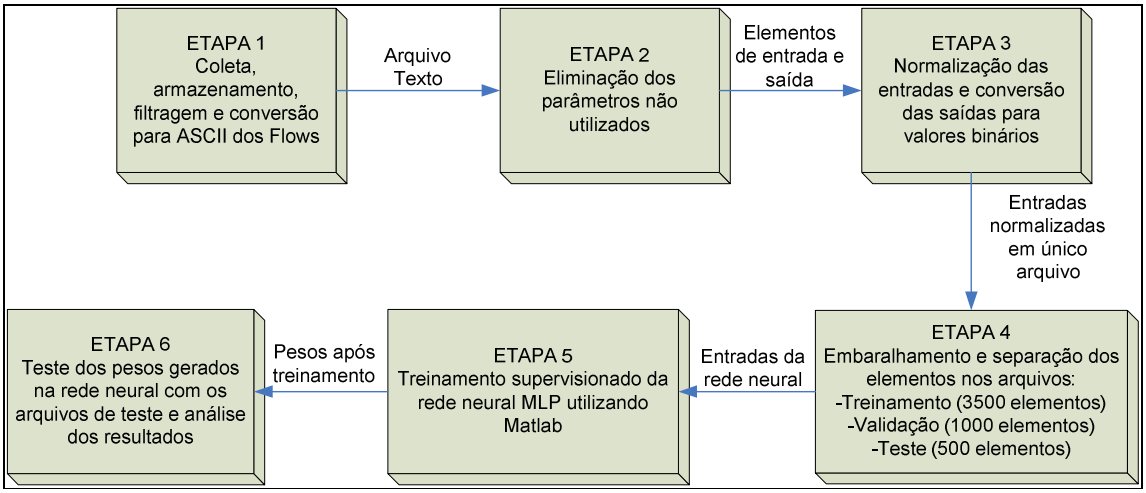


Figura 5. 1 – Diagrama de blocos das etapas do processo

Todas as seis etapas são detalhadas nos itens a seguir, bem como outras condições de ambiente que devem ser satisfeitas para que o experimento possa ser realizado com sucesso.

5.2 - Seleção do ambiente de coleta dos dados

Para a realização dos experimentos práticos é necessário coletar o protocolo Netflow contendo informações dos fluxos de comunicação gerados por uma rede privada com acesso à Internet, utilizando um roteador de borda Cisco com o IOS apropriado.

Por conseguinte, a escolha do ambiente de coleta foi influenciada pela facilidade de acesso aos equipamentos, pela possibilidade da alteração das configurações do roteador e pelas características de que a rede local possuía. Foi escolhida a rede da Incubadora de

Empresas de Base Tecnológica de Itajubá – INCIT, e a partir deste momento será referenciada como ambiente de coleta.

O ambiente de coleta possui algumas características desejadas para a execução deste estudo:

- **Ambiente heterogêneo:** existência de computadores de vários fabricantes, com diversas configurações e diferentes sistemas operacionais contendo aproximadamente 60 (sessenta) computadores.
- **Variedade de usuários:** o ambiente é compartilhado por aproximadamente 80 (oitenta) usuários que possuem diferentes perfis. Existem usuários que utilizam a rede apenas para tarefas como *e-mail* (pop e smtp) e *web* (http), outros publicam páginas na Internet (*ftp*) e fazem uso de protocolos específicos para aplicações de voz. Existem também usuários que durante o período noturno utilizam aplicações *peer-to-peer* para *download* de software, filmes, músicas, etc.
- ***Link* de Internet:** os usuários acessam a Internet através de uma conexão ADSL com 1000kbps de taxa de transferência de *download* e 300kbps de *upload*, disponibilizada pela operadora telefônica local.
- **Roteador Cisco Série 2600:** o *link* ADSL é conectado diretamente a um roteador com suporte ao protocolo Netflow. Esse roteador de borda é responsável pelo encaminhamento dos pacotes da rede interna para a rede do ISP.

5.3 - Descrição do ambiente

O ambiente de coleta é heterogêneo tanto no aspecto do perfil dos usuários quanto nos equipamentos que o compõem, porém uma descrição detalhada do ambiente não é o escopo deste estudo, logo, apenas os itens mais importantes são abordados nas definições que se seguem.

A rede interna é conectada à rede do provedor de serviços por um modem específico, que realiza o encapsulamento dos pacotes conduzidos pelo roteador, na tecnologia Ethernet, pelo *link* ADSL. Este *link* de Internet é distribuído pelos computadores da rede por meio de dois *switches* não gerenciáveis.

O dispositivo do ambiente de coleta mais importante para este estudo é o roteador de borda, o principal componente da rede, sendo um equipamento Cisco 2600. Este equipamento é o responsável pelo encaminhamento dos pacotes da rede interna para a rede externa. O

roteador é o gateway dos computadores da rede, possui o endereço IP privado 192.168.0.1 na interface *Ethernet0*, conectada à rede interna (figura 5.2). A interface *Ethernet1* é conectada ao modem ADSL, e a esta interface é atribuído um endereço IP público fornecido dinamicamente, através do protocolo dhcp, pelo provedor de serviço. A figura 5.2 ilustra a topologia do ambiente de coleta de dados.

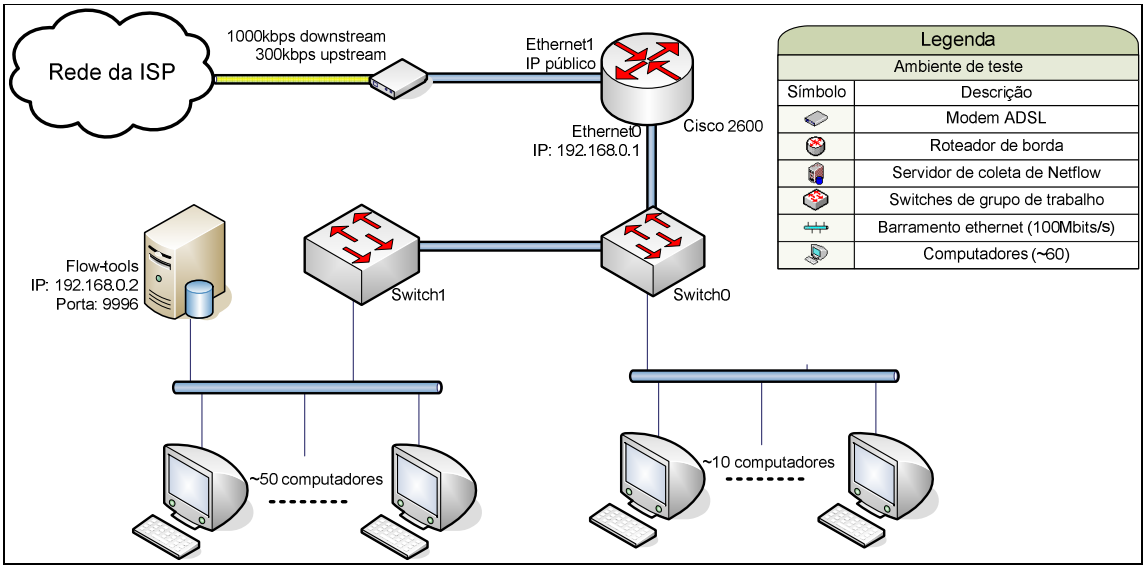


Figura 5. 2 – Ambiente de coleta de dados

O roteador Cisco 2600, com a versão 12.2 do software IOS, possui suporte ao protocolo Netflow [37]. Isso permite ao equipamento armazenar em *cache* as informações relacionadas aos fluxos de dados por suas interfaces. No ambiente de coleta, o roteador de borda foi configurado para exportar os fluxos de dados que atravessam a interface conectada a rede interna *Ethernet0*. Ambas as interfaces do roteador (*Ethernet0* e *Ethernet1*) poderiam ser escolhidas para exportar o protocolo Netflow, contudo a *Ethernet0* foi escolhida por já estar no mesmo segmento de rede do equipamento coletor do Netflow [37].

Conforme mencionado anteriormente no item 3.1, o protocolo Netflow possui muitas versões, cada qual com sua particularidade. Foi escolhida a versão cinco (NetflowV5), pois, além de ser a mais utilizada comercialmente, ela fornece todas as informações de comportamento do fluxo, necessárias para a análise proposta neste estudo.

Para a coleta dos fluxos, o roteador de borda Cisco 2600 foi configurado para exportar seu *cache* de Netflow versão 5 para o servidor de coleta no endereço 192.168.0.2 na porta

UDP 9996 (figura 5.2) a cada 300 segundos (5 minutos). A configuração completa do roteador é mostrada no apêndice A.

Neste ponto, é importante ressaltar que os dados exportados pelo roteador são consolidados, ou seja, eles representam fluxos de informações que já ocorreram, pertencem ao passado das conexões entre os *hosts* e a Internet. Em termos de implementação de um sistema comercial e prático, esta poderia ser uma limitação do sistema, já que ele trabalharia com informações que representam uma situação anterior, contudo, posteriormente, é proposta uma solução para este problema em um sistema prático.

Outro aspecto que merece comentários é a perda de pacotes transmitidos pelo Netflow durante a transferência das informações dos fluxos do roteador para o servidor coletor. As especificações [12,25] determinam que o protocolo da camada de transporte do pacote Netflow seja o UDP, que não possui nenhuma checagem de erros de transmissão. Desse modo, os pacotes enviados pelo roteador que se perdem pelo caminho e não chegam ao servidor de coleta não são retransmitidos. No modelo proposto, esse aspecto é abstraído e não se faz nenhuma adequação que leve em consideração essa perda de informações. No caso estudado é admitido que todos os pacotes transmitidos pelo roteador são recebidos pelo servidor de coleta.

5.4 - Forma de coleta, armazenamento e filtragem

Após a seleção do ambiente, configuração dos equipamentos e estabelecimento das condições iniciais se pode iniciar a etapa 1, mostrada no diagrama da figura 5.1. O servidor de coleta de dados do protocolo Netflow utiliza o conjunto de ferramentas *flow-tools*, descritas no capítulo 3, para captura, armazenamento e pré-processamento.

As ferramentas do conjunto *flow-tools* que são utilizadas para coleta, armazenamento, filtragem e exportação dos dados têm suas funções descritas a seguir:

- ***flow-capture*:** responsável pela coleta e armazenamento dos fluxos no disco rígido do servidor. Segundo as configurações fornecidas pelo administrador, o flow-capture grava os dados recebidos pela porta UDP 9996 em intervalos de tempo constantes, nesse caso, a cada 5 (cinco) minutos, e gera um novo arquivo de fluxos.
- ***flow-cat*:** como são gerados vários arquivos, a ferramenta *flow-cat* é utilizada para concatenar todas as informações dos fluxos em um único arquivo. A grande

vantagem de possuir apenas um arquivo é a possibilidade de processar toda a informação do Netflow coletada em apenas um lote.

- ***flow-nfilter***: é a ferramenta mais importante deste momento, pois é através dela que se separaram os *flows* correspondentes para cada protocolo que se deseja analisar em arquivos específicos. Por exemplo, para gerar os fluxos de análise do tráfego http foi criado um filtro que separa todos os fluxos que têm porta de destino igual a 80 (oitenta) e cria um novo arquivo contendo apenas estas informações.
- ***flow-export*:** ferramenta que exporta os dados no formato ASCII para posterior processamento e análise.

Neste ponto não são descritos em detalhes os comandos de cada uma das ferramentas nem a sintaxe de utilização pois não se tratar do foco do estudo. Estas informações podem ser encontradas no apêndice B.

Conforme dito anteriormente, foram coletados os fluxos dos protocolos regulares http e ftp e dos protocolos *peer-to-peer* eDonkey2000 e BitTorrent. A tabela 4 mostra a porta padrão utilizada pela ferramenta *flow-nfilter* para separar os fluxos de cada protocolo.

Tabela 4 – Portas da camada de transporte e respectivo protocolo da camada de aplicação

Protocolo	Porta
HTTP	80
FTP	20,21
eDonkey2000	TCP:4661/UDP:4665
BitTorrent	6881

Nesse ponto se admite que todos os tráfegos filtrados nas portas mostradas na tabela 4 correspondem aos protocolos determinados. Esse argumento está respaldado no controle que o autor possuía sobre os computadores que estavam gerando os tráfegos *peer-to-peer* e os tráfegos regulares.

Uma consideração deve ser feita a respeito do filtro do protocolo ftp: o protocolo da aplicação ftp pode se comportar de duas formas diferentes, a forma padrão e a forma passiva. Na forma padrão ocorre a negociação da conexão entre o cliente e o servidor na porta 20, em seguida, o servidor do arquivo abre uma porta superior à porta 1024 e a utiliza para se

comunicar com o cliente que utilizará a porta 21. Na forma passiva, após a etapa de negociação na porta 20, tanto o cliente quanto o servidor abrem uma conexão para transferir os arquivos em portas acima de 1024 [29]. Portanto, conhecendo o endereço IP do servidor ftp que opera em modo passivo, acrescentou-se um filtro de endereço IP ao filtro do protocolo ftp.

5.5 - Apresentação dos dados coletados

Após o processo descrito no item anterior, são gerados quatro arquivos de *flows* contendo informações dos protocolos http, ftp, eDonkey2000 e BitTorrent. Cada um desses arquivos contém 5000 (cinco mil) elementos de saída e cada elemento corresponde a um fluxo de comunicação gerado pelo respectivo protocolo.

O arquivo em sua forma original, gerado pelo *flow-export*, fornece 24 campos de informação para cada fluxo. Os campos estão elencados e descritos na tabela 5 abaixo. Assim tem início a segunda etapa do diagrama mostrado na figura 5.1.

Tabela 5 – Campos do Netflow coletados pelo flow-export

Nº	**Campo**	**Descrição**
1	unix_secs	Contador dos segundos desde 0000 UTC 1970
2	unix_nsecs	Nanosegundos residuais desde 0000 UTC 1970
3	sysuptime	Tempo em milisegundos desde que foi iniciado
4	exaddr	Endereço IP de quem exporta o fluxo
5	dpkts	Número de pacotes no fluxo
6	doctets	Número de bytes do fluxo
7	first	SysUpTime no início do fluxo
8	last	SysUpTime quando o último pacote do fluxo foi recebido
9	engine_type	Type of flow-switching engine
10	engine_id	Slot number of the flow-switching engine
11	srcaddr	Endereço IP de origem do pacote
12	dstaddr	Endereço IP de destino do pacote
13	nexthop	Endereço IP do roteador do próximo pulo
14	input	Interface do roteador por onde entra o pacote
15	output	Interface do roteador por onde sai o pacote
16	srcport	Número da porta de origem na camada 4
17	dstport	Número da porta de destino na camada 4
18	proto	Protocolo da camada 4 (TCP=6 UDP=17)
19	tos	Byte de tipo de serviço
20	tcp_flags	OR cumulativo da Flags TCP
21	src_mask	Bits da máscara do ip de origem

22	dst_mask	Bits da máscara do ip de destino
23	src_as	Número do sistema autônomo da origem
24	dst_as	Número do sistema autônomo do destino

Apesar do grande número de campos exportados, apenas cinco campos são utilizados. Existem diversas razões para o descarte dos outros dezenove campos fornecidos no arquivo. Os campos *SysUptime, unix_secs, unix_nsecs, engine_type, engine_id* e *exaddr* são campos que pertencem ao cabeçalho do pacote Netflow que possuía aquele fluxo, portanto não possuem nenhuma informação representativa do comportamento da conexão.

Os campos *srcaddr, dstaddr, nexthop, input, output, srcport, dstport, src_mask, dst_mask, src_as* e *dst_as* indicam informações estáticas sobre o fluxo como, por exemplo, os dados de endereçamento do pacote e do sistema a qual o pacote pertence. Outros campos fornecem parâmetros do roteador, como, por exemplo, as interfaces de entrada e saída do pacote e o endereço do próximo roteador que receberá o pacote daquele fluxo.

Dois campos merecem uma atenção especial: *tos* e *tcp_flags*. O campo *tos* indica o tipo de serviço que o pacote IP possuiria em uma rede de comunicação com suporte à QoS – Qualidade de Serviço. O campo *tcp_flags* indica as *flags* dos pacotes TCP da camada de transporte no pacote IP da camada de rede e são utilizados para identificar fluxos que finalizam, reiniciam, sincronizam ou configuram as conexões. Ambos os campos poderiam ser utilizados como complemento na determinação do protocolo da camada de aplicação utilizando o modelo neural proposto. Ambos os campos adicionariam informações das necessidades dos protocolos da camada de aplicação e dos fluxos de comunicação que atravessam o roteador. No entanto, quando considera-se os aplicativos *peer-to-peer* para a transferência de arquivos, a informação dos campos *tos* e *tcp_flags* passa a não ser confiável, pois os desenvolvedores de aplicativos *peer-to-peer*, com a intenção de burlar o sistema de detecção, podem modificar intencionalmente o cabeçalho dos pacotes IP, alterando os campos de tipo de serviço (*tos*) e as *flags* do pacote TCP (*tcp_flags*).

Feitas todas as considerações a respeito de quais campos não poderiam ser utilizados, restam apenas os campos *dpkts, doctets, first, last* e *proto*. Os campos *dpkts* (número de pacotes por fluxo) e *doctets* (número de bytes por fluxos) indicam diretamente atributos de comportamento do fluxo da comunicação, assim sendo a utilização destes como entradas do sistema de detecção de tráfego *peer-to-peer* por comportamento do fluxo é clara.

O campo *proto* indica qual o protocolo da camada de transporte aquele fluxo de comunicação utiliza. No caso deste estudo, apenas dois protocolos da camada de transporte

são utilizados: TCP e UDP. A necessidade da utilização do campo *proto* está fundamentada em dois aspectos: o primeiro deles é que existem aplicativos *peer-to-peer* que utilizam ambos os protocolos, o segundo ponto é que o comportamento das conexões é diferente para cada um dos protocolos. Enquanto no TCP existem mecanismos de controle que podem ocasionar a retransmissão de pacotes e a negociação de parâmetros de transferência de dados durante a conexão, o UDP não apresenta nenhuma forma de correção de erros ou negociação de taxas de transferência durante a conexão. Estas particularidades determinam diferenças no comportamento das conexões de cada protocolo, que associadas com as outras informações da entrada do sistema determinam padrões que podem ser aprendidos pelo sistema neural utilizado.

Considerações a respeito dos campos *first* e *last* também são necessárias: os campos *first* e *last* indicam respectivamente o *SysUpTime* no início e o *SysUpTime* no encerramento daquele fluxo de comunicação. Para fornecer um dado significativo para a entrada do sistema de detecção é necessário conhecer a duração do fluxo no roteador, portanto o módulo da diferença entre os campos *first* e *last* fornece o tempo que o fluxo levou para ser concluído. Este valor será denominado *diff*.

Conforme informado anteriormente, o campo *dstport* não é utilizado como entrada no sistema de detecção neural, pois a intenção principal é identificar um fluxo regular em relação ao P2P sem utilizar informações sobre as portas de comunicação. Contudo, para o sistema proposto, o *dstport* indica qual o protocolo da camada de aplicação que o fluxo do conjunto de fluxos coletados representa e é útil para fornecer o campo “saída esperada” para os processos de treinamento, validação cruzada e teste da rede neural utilizada.

Portanto, na tabela 6 são mostrados os campos (atributos) que serão utilizados como entrada do sistema de detecção.

Tabela 6 – Campos utilizados como entradas da rede neural

Campo	**Função**	**Descrição**
dpkts	Entrada	Número de pacotes no fluxo
doctets	Entrada	Número de bytes do fluxo
diff	Entrada	Diferença do SysUpTime no início e no término do fluxo
proto	Entrada	Protocolo da camada 4
dstport	Saída esperada	Indica o protocolo da camada de aplicação (http, ftp, eDonkey2000, BitTorrent)

5.6 - Tratamento e representação da informação dos fluxos

Conforme discutido no item 3.2, o desempenho de uma rede neural está intimamente ligado com a forma com que os dados são apresentados na entrada do sistema de reconhecimento de padrões. Este item é dedicado a uma descrição detalhada do tratamento dado aos dados brutos do protocolo Netflow e a forma com que eles são utilizados no sistema neural.

5.6.1 - Normalização dos dados

A etapa 3 (figura 5.1) consiste na normalização dos dados. O processo de normalização é necessário para que todos os valores numéricos que são apresentados à entrada da rede neural estejam na mesma faixa de valores e, portanto, tenham a mesma importância durante o processo de treinamento.

Por exemplo, nos dados coletados o campo *proto* possui 17 como valor máximo, já o campo *doctets* possui valores na casa de centenas de milhares. Caso os valores fossem apresentados ao sistema neural desta forma, o campo *doctets* possuiria maior importância que o campo *proto*, porém esse comportamento é indesejado já que qualitativamente, no modelo proposto, ambos os campos têm a mesma importância.

Para se normalizar os dados coletados, foram extraídos os valores máximos e mínimos de cada atributo, considerando todos os protocolos escolhidos para o estudo. Em seguida, todos os atributos do conjunto de fluxos de todos os protocolos foram divididos pelos valores máximos encontrados para cada um dos quatro atributos de entrada: *dpkts*, *doctets*, *diff* e *prot*. Para fixação é mostrada a equação 5.1, utilizada para a normalização das entradas.

$$V_{\substack{ATRIBUTO \\ NORMALIZADO}} = \frac{V_{ATRIBUTO}}{V_{\substack{MÁXIMO \\ ATRIBUTO}} - V_{\substack{MÍNIMO \\ ATRIBUTO}}} \tag{5.1}$$

Como os valores mínimos para todos os parâmetros é zero, a equação utilizada na prática é mostrada em 5.2.

$$V_{\substack{ATRIBUTO \\ NORMALIZADO}} = \frac{V_{ATRIBUTO}}{V_{\substack{MÁXIMO \\ ATRIBUTO}}} \tag{5.2}$$

Depois de algumas tentativas de representação em um número menor de classes, resolveu-se separar os dados em 1000 classes no intervalo de [0.000;1.000] para se obter uma boa representação e para que as pequenas diferenças entre os fluxos ficassem claras para a rede neural. A figura 5.3 mostra um exemplo de entradas para o sistema neural.

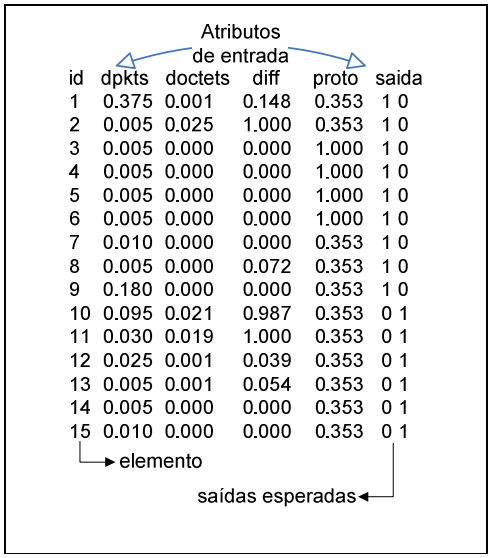


Figura 5. 3 – Exemplo de entrada da rede neural

Na figura 5.3 também é possível observar o formato binário com que as saídas esperadas serão apresentadas à rede neural. A saída possui dois campos que podem assumir os valores *0* ou *1*. Para o experimento se convencionou que as saídas representariam os protocolos segundo o que se mostra na tabela 7.

Tabela 7 – Representação usada como saída de cada elemento

Saída / Protocolo	**Campo 1**	**Campo 2**
Regular	0	1
Peer-to-peer	1	0

5.6.2 - Criação dos arquivos de treinamento

Neste ponto, toda a etapa de pré-processamento dos dados foi realizada. Porém, alguns aspectos devem ser considerados antes da divisão dos 5000 fluxos em diferentes arquivos.

No ambiente de coleta podem ocorrer diversas variações no comportamento dos fluxos por diversas razões:

- Entrada de novos computadores na rede;
- Variações no *link* ADSL;
- Início de novas transferências, etc.

Muitos desses fatores alteram o comportamento dos fluxos por certos períodos de tempo. Portanto, para garantir que todas as nuâncias da comunicação estejam representadas nos arquivos de treinamento, validação e teste, os 5000 fluxos foram embaralhados aleatoriamente entre si. Dessa forma, nenhuma tendência momentânea dos fluxos fica isolada em um único arquivo utilizado no processo.

A tabela 8 apresenta os três arquivos gerados para cada protocolo e as respectivas quantidades de fluxos presentes em cada um. Com os arquivos separados está finalizada a etapa 4 mostrada na figura 5.1.

Tabela 8 – Fluxos em cada arquivo de protocolo

Arquivo	**Número de fluxos**
Treinamento	3500
Validação	1000
Teste	500

A proporção dos fluxos em cada arquivos foi estabelecida em 70% dos fluxos para treinamento, 20% para validação cruzada e 10% dos fluxos para o conjunto de testes, desta forma seguindo recomendações e outros modelos encontrados na literatura que trata de processos de treinamento de redes neurais [27].

5.7 - Aplicação do modelo neural

Conforme descrito no item 3.2, a rede neural utilizada para o experimento é um modelo MLP, *Multi Layer Perceptron* ou modelo multicamada. Sendo um modelo neural muito conhecido e utilizado em meios acadêmicos para várias aplicações, o MLP foi escolhido para esta dissertação.

A partir deste ponto é utilizado o *framework* Matlab, fornecido pela MathWorks, para implementar as etapas 5 e 6 do diagrama mostrado na figura 5.1.

5.7.1 - Apresentação dos dados ao modelo neural

A figura 5.4 mostra, de forma geral, todo o mecanismo que é utilizado para o processo de treinamento, validação e testes para reconhecimento de protocolos regulares e *peer-to-peer*.

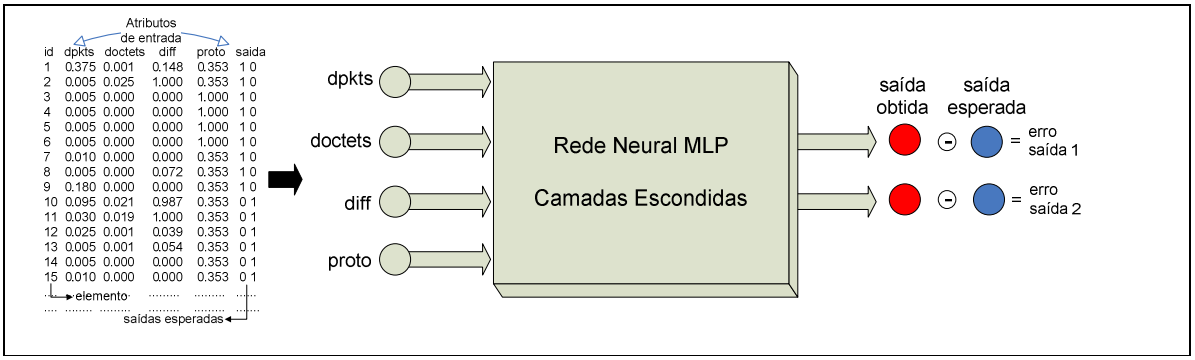


Figura 5. 4 – Diagrama do sistema

O sistema possui quatro entradas para os atributos normalizados *dpkts, doctets, diff* e *proto* e duas saídas que são comparadas com as saídas esperadas para se obter o erro na classificação do fluxo da comunicação.

A arquitetura interna da MLP, implementada pelo Matlab, é abstraída neste momento. É suficiente conhecer que são utilizados modelos com uma ou duas camadas escondidas com variadas combinações de neurônios nessas camadas. Os neurônios das camadas escondidas utilizam a função logística sigmoidal como função de limiar, e a camada de saída utiliza a função puramente linear. O resultado dos melhores arranjos serão mostrados no capítulo 6.

O algoritmo de treinamento por *backpropagation* é utilizado com a função de ativação logística sigmoidal para cada neurônio das camadas escondidas, conforme discutido nos itens 3.2 e 3.3.

A figura 5.4 também mostra as duas saídas da rede neural. Essas saídas são as responsáveis por sinalizar as saídas obtidas para cada um dos fluxos presentes nos arquivos de treinamento, validação cruzada e teste. As medidas de desempenho do treinamento, o critério de parada pela avaliação do comportamento do erro do conjunto de validação e a taxa de

acerto para o conjunto de testes são computados através da comparação entre a saída obtida e a saída esperada. Este processo de comparação é descrito detalhadamente no item 5.7.3.

5.7.2 - Comparação dos protocolos em pares

Devido ao caráter investigativo deste estudo e aos aspectos técnicos do treinamento de sistemas neurais, optou-se por realizar em todos os ensaios uma comparação entre dois protocolos, sendo sempre um regular (http ou ftp) e um protocolo *peer-to-peer* (eDonkey2000 ou BitTorrent). O fato do treinamento ser sempre realizado com pares de protocolos facilita o processo de aprendizado da rede neural [27], pois ela necessita distinguir entre apenas dois tipos de fluxos de comunicação de dados.

Nesse ponto está pautada a escolha dos protocolos para esta investigação. Pois os comportamentos dos protocolos http e *peer-to-peer* são muito parecidos [21], portanto conhecer o desempenho do processo de detecção entre estes protocolos é importante. O ftp é conhecidamente o protocolo regular mais utilizado para transferências de grandes arquivos, da mesma forma que os protocolos *peer-to-peer* vêm sendo utilizados. Logo, uma comparação entre esses protocolos fornece um resultado válido para o objetivo geral proposto no estudo.

Com essa consideração, é possível determinar o tamanho total dos arquivos de treinamento, validação e teste, que são mostrados na tabela 9.

Tabela 9 – Fluxos dos arquivos do processo de treinamento

Arquivo	Número de fluxos
Treinamento	7000
Validação	2000
Teste	1000

5.7.3 - Medida de desempenho

Para iniciar o processo de treinamento é necessário estabelecer um critério que permita a avaliação do aprendizado da rede neural. A medida mais comum de desempenho encontrada na bibliografia é a taxa de erro [27]. A taxa de erro consiste na medida do módulo da

diferença entre os valores esperados como saída para um determinado elemento do conjunto de treinamento e o valor sinalizado para aquele fluxo na saída da rede.

O erro pode ser computado de várias formas como, por exemplo, erro médio, erro absoluto, etc. Para o sistema neural implementado em Matlab, a medida do erro escolhida é somatório dos erros ao quadrado (*Sum of Square Error*), que daqui para frente será tratado como SSE. Esta medida de erro fornece uma informação mais qualitativa de como está se comportando o processo de treinamento. A equação de cálculo do erro para o modelo proposto é:

$$= \sum_{i=}^{m} ((\quad - \quad) + (\quad - \quad)) \qquad (5.3)$$

Considerando o SSE, quando existe uma leve diferença entre o valor esperado e o valor obtido, o erro para no aprendizado daquele elemento sofre uma pequena penalização, pois quando elevado ao quadrado um erro pequeno (menor que um) torna-se menor ainda. Em contrapartida, quando um elemento apresenta um erro alto (maior que um) ele possui uma penalização maior, já que quando elevado ao quadrado o valor do erro aumenta. Durante o processo de treinamento esta medida é mais útil, pois permite identificar mais rapidamente quando o treinamento não está adequado, já que a taxa de erro aumenta drasticamente, e interromper o processo.

5.7.4 - Parâmetros de treinamento

A tabela 10 apresenta os parâmetros utilizados pelo processo de treinamento. Através dela são estabelecidos dois critérios de parada do processo: número de épocas e limite de erro para o conjunto de treinamento. A forma como a rede neural utiliza esses parâmetros é discutida no item 3.2.

O parâmetro limite de erro do conjunto de treinamento foi calculado convencionando-se o valor do SSE em 0,01 para cada um dos fluxos do conjunto de treinamento, considerando que este valor de erro é satisfatório para cada elemento. Portanto, para 7000 fluxos do conjunto de treinamento o limite de erro SSE é 70.

Contudo, durante o procedimento, o treinamento da rede neural nunca foi encerrado, pelo alcance dos valores estipulados para o número de épocas ou limite de erro do conjunto de

treinamento. Todos os experimentos foram encerrados pelo aumento no erro do conjunto de validação, segundo os critérios utilizados pelo método de treinamento supervisionado (validação cruzada).

Não foi estipulado um tempo limite para o treinamento da rede neural, pois o objetivo deste estudo é a investigação da possibilidade de utilização do modelo neural na detecção de tráfego *peer-to-peer*. Logo, buscou-se o melhor desempenho do sistema neural sem se importar com a variável tempo.

Tabela 10 – Parâmetros do treinamento

Parâmetros	**Valor**
Número de épocas	1000
Taxa de aprendizagem inicial	0,001
Limite de erro	70
Tempo limite	Infinito

A escolha da taxa de aprendizado depende da função a aproximar. Valores muito pequenos tornam o treinamento lento, enquanto valores muito grandes podem provocar divergência do processo de treinamento [27]. Portanto, a taxa de aprendizado inicial foi configurada para 0,001, de acordo com o valor do menor incremento possível para os atributos dos fluxos de entrada da rede neural, e à medida que o processo de treinamento se encaminha, sendo alterada para que fosse encontrado o resultado melhor e de maneira mais rápida.

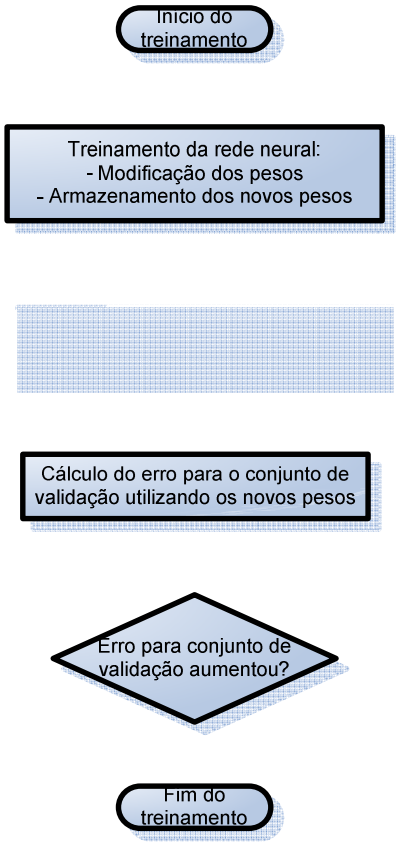
5.7.5 - Processo de treinamento

O processo de validação cruzada, etapa 5 da figura 5.1, discutido no item 3.3, em termos práticos, foi o critério de parada para o processo de treinamento do modelo proposto. A figura 5.5 ilustra de forma simples o algoritmo utilizado.

A primeira ação é o fornecimento dos arquivos de treinamento (7000 fluxos) e validação (2000 fluxos) para a rede. Pela ferramenta de redes neurais do Matlab, os pesos iniciais da rede são gerados aleatoriamente pela função mostrada no item 3.3, de acordo com o discutido na seção de redes neurais no item 3.2. Em seguida, tem início a modificação dos pesos da rede neural de acordo com os fluxos presentes no conjunto de treinamento. Finalizada a época de treinamento, os novos pesos são armazenados.

Pelos critérios de validação cruzada, é calculado o valor do erro quando se realiza a classificação dos fluxos do conjunto de validação com os pesos gerados na última época do treinamento. No caso da primeira época do processo, o erro de classificação do conjunto de validação é apenas armazenado para ser utilizado posteriormente.

Prossegue-se para a próxima época do processo, novos pesos são obtidos através do aprendizado dos fluxos contidos no arquivo de treinamento. Novamente é feita a classificação do conjunto de validação com os pesos gerados na última época. Caso o erro de classificação do conjunto de validação seja menor que o erro de classificação da época anterior, prossegue-se para um novo treinamento da rede neural, uma nova época.



5.7.6 - Rotina dos experimentos

Encerrado o processo de treinamento e validação, os pesos finais obtidos para a rede neural podem ser utilizados para a classificação dos fluxos contidos no conjunto de testes.

No sistema proposto, o conjunto de testes foi formado a partir dos fluxos que estavam contidos nos conjuntos inicialmente obtidos, após a coleta de dados dos fluxos. A classificação desses fluxos é então utilizada para mensurar o desempenho do sistema de detecção de tráfego *peer-to-peer*. A forma de avaliação e os resultados obtidos são detalhados no próximo capítulo.

CAPÍTULO 6 – RESULTADOS DA APLICAÇÃO DO MODELO

6.1 - Forma de avaliação dos resultados obtidos

A medida de erro SSE fornece uma visão qualitativa geral de como se comportou a classificação do conjunto de testes. Contudo, este dimensionamento de erro não fornece o detalhamento necessário para a análise de desempenho desejada. Portanto, para avaliação do modelo proposto na classificação do conjunto de testes é utilizado o sistema mostrado na figura 6.1.

Conforme comentado anteriormente, a medida do erro sempre será computada em função da diferença entre o valor de saída esperada e o valor de saída obtida para cada um dos fluxos dos conjuntos de treinamento, validação e teste.

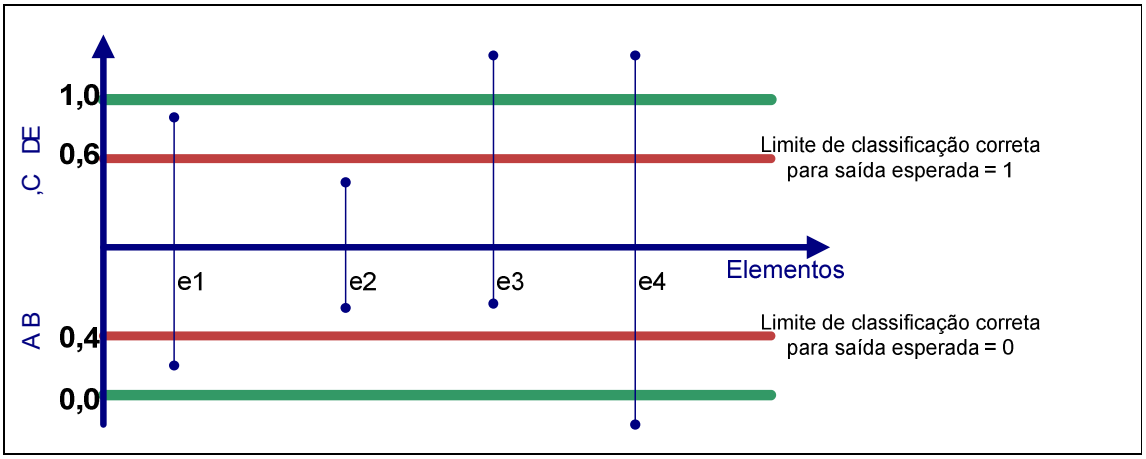


Figura 6. 1 – Classificação dos fluxos dos conjuntos

Em um sistema de classificação neural existe um valor limite para o erro, que determina quando um padrão é ou não classificado corretamente [27]. Para o modelo proposto, o valor do limite de erro de classificação escolhido foi 0,4 e 0,6. Assim se a saída esperada for 1 e os valores obtidos para a saída forem superiores a 0,6, a saída obtida é considerada correta. No caso da saída esperada igual a 0, valores inferiores a 0,4 serão considerados corretos.

Como exemplo, na figura 6.1, são ilustradas as saídas obtidas de quatro elementos, que foram classificados conforme descrito. O elemento *e1* possui as duas saídas situadas nas

regiões de classificação admitidas como certas, portanto e1 será considerado corretamente classificado. O elemento *e2* possui ambas as saídas obtidas fora das regiões de classificação correta, logo será classificado como incorreto. Para *e3*, uma saída obtida permaneceu dentro da região de classificação correta para a saída esperada 1, porém para a saída esperada 0 a classificação foi fora da região admitida, nesses tipos de caso a classificação será incorreta. Considerando o elemento *e4* temos que ambas as saídas obtidas estão muito acima dos valores esperados, porém estão dentro da região considerada correta, logo a classificação é correta.

Agora, após a descrição do sistema de avaliação do erro de classificação, é possível mostrar os resultados obtidos com a aplicação do modelo descrito no capítulo 4.

6.2 - Resultados obtidos

Conforme dito anteriormente, os experimentos foram realizados utilizando os protocolos agrupados dois a dois, sempre um protocolo regular e outro peer-to-peer. Quatro experimentos foram realizados. No primeiro, o modelo neural foi encarregado de distinguir entre tráfego http e tráfego P2P gerado pelo eMule, aplicativo da rede ed2k. No segundo experimento, o modelo neural foi encarregado de diferenciar entre tráfego http e tráfego P2P gerado pelo Azureus, aplicativo da rede BitTorrent. No terceiro experimento, o modelo neural foi encarregado de distinguir entre tráfego ftp e tráfego P2P gerado pelo eMule. No quarto experimento, o modelo neural foi encarregado de diferenciar entre tráfego ftp e tráfego P2P gerado pelo Azureus.

Nos subitens que se seguem são apresentadas as arquiteturas que mostraram melhor desempenho na classificação dos elementos dos fluxos de comunicação contidos no conjunto de testes.

6.2.1 - Experimento 1

As tabelas 11 e 13 apresentam os resultados obtidos para os processos de treinamento, validação e teste dos pesos da rede neural com a função de classificar os padrões de comportamento de fluxos gerados pelas aplicações que utilizam os protocolos http e eDonkey2000. Nessas tabelas estão os erros obtidos para cada etapa do treinamento e o número de neurônios utilizados em cada uma das camadas escondidas nas arquiteturas escolhidas e o número de épocas utilizadas no treinamento.

Tabela 11 – Resultados do processo de treinamento http-ed2k MLP de 1 camada

Conjunto	Erro de classificação (sse)	Erro percentual absoluto médio (mape)	Arquitetura da MLP utilizada	
Treinamento	1211,08	39,51	Número de camadas	1
Validação	411,67	43,12	Neurônios 1ª camada	250
Teste	224,84	43,30	Neurônios 2ª camada	-
			Número de épocas de treinamento	141

O resultado do processo de testes dos pesos da rede neural, através da classificação de um conjunto de fluxos específico para esta finalidade, é mostrado no formato de tabela de confusão (

A mesma analogia pode ser feita para a análise de todas as tabelas de confusão que se seguem incluindo a tabela 14 que apresenta os resultados da classificação dos protocolos http-ed2k utilizando uma rede MLP de duas camadas.

Tabela 13 – Resultados do processo de treinamento - http-ed2k - MLP 2 camadas

Conjunto	Erro de classificação (SSE)	Erro percentual absoluto médio (mape)	Arquitetura da MLP utilizada	
Treinamento	1154,45	37,35	Número de camadas	2
Validação	409,68	41,90	Neurônios 1ª camada	40
Teste	189,60	41,45	Neurônios 2ª camada	30
			Número de épocas de treinamento	94

Tabela 14 – Classificação do conjunto de teste - http-ed2k - MLP 2 camadas

	http	ed2k	indefinido	%
HTTP	386	71	43	77,2%
ed2k	8	477	15	95,4%
%	98,0%	87,0%	-	86,3%

6.2.2 - Experimento 2

São apresentados neste item os resultados obtidos para a criação do classificador de fluxos http e BitTorrent. As tabelas 15 e 17 apresentam os resultados dos processos de treinamento, validação e teste dos pesos da rede neurais MLP de uma e duas camadas, respectivamente.

As tabelas 16 e 18 apresentam as tabelas de confusão para a classificação dos fluxos do conjunto de testes, utilizando os pesos da rede neurais MLP de uma e duas camadas respectivamente.

Tabela 15 – Resultados do processo de treinamento - http-bittorrent - MLP 1 camada

Conjunto	Erro de classificação (sse)	Erro percentual absoluto médio (mape)	Arquitetura da MLP utilizada	
Treinamento	1239,52	40,21	Número de camadas	1
Validação	402,64	45,36	Neurônios 1ª camada	450
Teste	180,22	44,82	Neurônios 2ª camada	-
			Número de épocas de treinamento	72

Tabela 16 – Classificação do conjunto de teste - http-bittorrent - MLP 1 camada

	http	bittorrent	Indefinido	%
HTTP	406	50	44	81,2%
Bittorrent	10	444	46	88,8%
%	97,6%	89,9%	-	85,0%

Tabela 17 – Resultados do processo de treinamento - http-bittorrent - MLP 2 camadas

Conjunto	Erro de classificação (SSE)	Erro percentual absoluto médio (mape)	Arquitetura da MLP utilizada	
Treinamento	867,45	28,06	Número de camadas	2
Validação	330,53	35,07	Neurônios 1ª camada	50
Teste	149,66	34,70	Neurônios 2ª camada	40
			Número de épocas de treinamento	187

Tabela 18 – Classificação do conjunto de teste - http-bittorrent - MLP 2 camadas

	http	bittorrent	indefinido	%
HTTP	420	55	25	84,0%
bittorrent	13	456	31	91,2%
%	97,0%	89,2%	-	87,6%

6.2.3 - Experimento 3

São apresentados neste item os resultados obtidos para a criação do classificador de fluxos ftp e ed2k. As tabelas 19 e 21 apresentam os resultados dos processos de treinamento, validação e teste dos pesos da rede neurais MLP de uma e duas camadas respectivamente.

As tabelas 20 e 22 apresentam as tabelas de confusão para a classificação dos fluxos do conjunto de testes, utilizando os pesos da rede neurais MLPs de uma e duas camadas respectivamente.

Tabela 19 – Resultados do processo de treinamento - ftp-ed2k - MLP 1 camada

Conjunto	Erro de classificação (SSE)	Erro percentual absoluto médio (mape)	Arquitetura da MLP utilizada	
Treinamento	901,42	29,85	Número de camadas	1
Validação	250,42	32,70	Neurônios 1ª camada	250
Teste	110,21	32,02		

Tabela 22 – Classificação do conjunto de teste - ftp-ed2k - MLP 2 camadas

	ftp	ed2k	indefinido	%
FTP	480	18	2	96,0%
ed2k	36	454	10	90,8%
%	93,0%	96,2%	-	93,4%

6.2.4 - Experimento 4

São apresentados neste item os resultados obtidos para a criação do classificador de fluxos ftp e BitTorrent. As tabelas 23 e 25 apresentam os resultados dos processos de treinamento, validação e teste dos pesos da rede neurais MLP de uma e duas camadas respectivamente.

As tabelas 24 e 26 apresentam as tabelas de confusão para a classificação dos fluxos do conjunto de testes, utilizando os pesos da rede neurais MLP de uma e duas camadas respectivamente.

Tabela 23 – Resultados do processo de treinamento - ftp-bittorrent - MLP 1 camada

Conjunto	Erro de classificação (SSE)	Erro percentual absoluto médio (mape)	Arquitetura da MLP utilizada	
Treinamento	1042,32	32,48	Número de camadas	1
Validação	298,79	33,07	Neurônios 1ª camada	100
Teste	149,85	31,95	Neurônios 2ª camada	-
			Número de épocas de treinamento	208

Tabela 24 – Classificação do conjunto de teste - ftp-bittorrent - MLP 1 camada

	FTP	BITTORRENT	INDEFINIDO	%
FTP	481	11	8	96,2%
BITTORRENT	84	405	11	81,0%
%	85,1%	97,4%	-	88,6%

Tabela 25 – Resultados do processo de treinamento - ftp-bittorrent - MLP 2 camadas

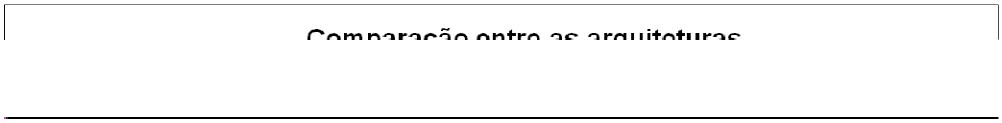
Conjunto	Erro de classificação (SSE)	Erro percentual absoluto médio (mape)	Arquitetura da MLP utilizada	
Treinamento	956,46	28,91	Número de camadas	2
Validação	278,19	29,93	Neurônios 1ª camada	30
Teste	135,94	28,88	Neurônios 2ª camada	20
			Número de épocas de treinamento	300

Tabela 26 – Resultados do conjunto de teste ftp-bittorrent MLP de 2 camadas

	ftp	bittorrent	indefinido	%
FTP	492	5	3	98,4%
Bittorrent	87	409	4	81,8%
%	85,0%	98,8%	-	90,1%

6.3 - Comparação dos resultados obtidos

Pelas tabelas de resultados se observa que as arquiteturas de dupla camada apresentam um melhor desempenho na classificação dos fluxos dos tráfegos que atravessam o roteador, isto é mostrado na figura 6.2. Este fato está de acordo com o discutido no item 3.2.



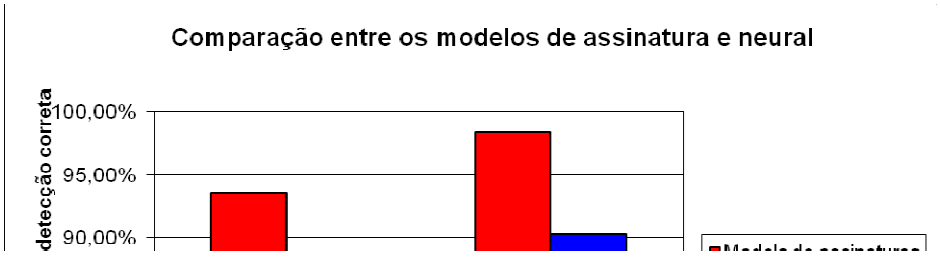
tráfego, que em alguns momentos é confundido com o perfil do tráfego *peer-to-peer*. Entretanto os valores finais para a classificação entre os protocolos alcançaram um nível considerado satisfatório para justificar a continuidade da aplicação de redes neurais para este fim, com posterior implementação de um sistema real.

Como comparação dos resultados obtidos pela utilização do modelo neural proposto é mostrada, na tabela 27, a precisão da detecção alcançada pelo modelo de assinaturas, discutido no item 2.4, retirado de [6]. A precisão de detecção do modelo neural foi obtida pelo cálculo da média entre as porcentagens de classificações corretas dos protocolos *peer-to-peer* em relação os protocolos regulares utilizando a MLP de duas camadas escondidas, que apresentou o melhor resultado. A tabela 27 não tem fins de comparação científicos, já que existe uma diferença fundamental entre o modelo baseado em assinaturas e o modelo proposto, o primeiro utiliza pacotes e o segundo, informações de fluxos. Desta forma a tabela 27 fornece apenas uma noção de desempenho do modelo proposto em relação ao modelo baseado em assinaturas.

Tabela 27 – Comparação do modelo neural com assinaturas

Protocolo	**Precisão da Detecção (%)**	
	Modelo Assinaturas	**Modelo Neural**
eDonkey2000	93,63%	89,85%
BitTorrent	98,43%	88,85%

Graficamente, a tabela 27 é mostrada na figura 5.3.



O fato é que o modelo baseado em assinaturas possui melhor desempenho em relação ao modelo proposto, porém as limitações que o modelo baseado em assinaturas possui (mudança no pacote, criptografia, etc) não são encontradas no modelo proposto. Por exemplo, caso o tráfego P2P seja criptografado, a precisão de detecção do modelo de assinatura se reduziria a praticamente zero, enquanto o modelo proposto continuaria apresentando o mesmo desempenho mostrado na tabela 27.

No item 2.5 foi mostrado o modelo baseado em topologia, que é capaz de detectar 90% do tráfego *peer-to-peer* que é detectado pelo modelo baseado em assinaturas [7]. Com esses dados pode-se afirmar que o resultado do modelo neural proposto possui um desempenho compatível com o modelo baseado em topologia. Ambos os modelos são comparáveis até no aspecto do alto número de falsos positivos [7] que são encontrados nas classificações entre http-bittorrent e ftp-bittorrent realizadas pelo modelo neural proposto.

Infelizmente, a literatura não traz dados que possibilite a comparação entre o modelo proposto e modelo *crawler*, discutido no item 2.6.

6.4 - Relevância dos resultados

Após toda a discussão em torno do tema do problema causado pela transferência de arquivo em aplicativos *peer-to-peer*, as soluções existentes e seus problemas, este estudo apresenta um modelo que utiliza um paradigma diferente dos encontrados nas aplicações comerciais. A proposição de um modelo neural com a finalidade de disponibilizar uma ferramenta para auxiliar os administradores de rede no controle de aplicativos *peer-to-peer* alcançou resultados satisfatórios. As porcentagens de classificações corretas encontradas em todos os testes estão dentro do esperado e justificam a criação de uma aplicação para o mundo real, utilizando o processo descrito.

6.5 - Vantagens e benefícios do modelo proposto

Para o monitoramento e controle das comunicações de dados de aplicativos *peer-to-peer* os administradores não possuem ferramentas adequadas que forneçam as informações apropriadas para rápida análise ou processamento. As ferramentas comerciais atuais, baseadas em filtros da camada 4, são inadequadas diante das técnicas de camuflagem utilizadas pelos

aplicativos P2P. O modelo baseado em assinaturas de pacotes torna-se ineficaz quando a comunicação é criptografada.

A oportunidade que o modelo proposto apresenta está em construir um sistema capaz de identificar e marcar os fluxos de conexões P2P para posterior filtragem ou outro tipo de processamento nos *backbones* das redes, não importando as portas de comunicação ou criptografia dos dados utilizadas pelos aplicativos.

Outro aspecto positivo do modelo, consiste em sua capacidade de trabalhar com dados consolidados do protocolo Netflow, sem execução de qualquer tipo de análise sobre o próprio pacote da comunicação. Isto se refletiria na aplicação do modelo em *backbones* com conexões de grande velocidade, quando um método rápido e eficiente para a identificação seria essencial, pois não são gerados atrasos adicionais na comunicação. Um sistema real, que utiliza o desenvolvimento proposto não realizaria nenhum tipo de processamento sobre o pacote da comunicação em si, não gerando atrasos adicionais. A arquitetura do sistema em um ambiente real é discutida no capítulo 7.

6.6 - Limitações do modelo

O mesmo aspecto que contribui para as vantagens do sistema produz também aspectos negativos. Como a análise é realizada sobre o fluxo de informações já consolidado no roteador, ou seja, quando o usuário da rede efetuar um tráfego de dados P2P, o sistema irá detectar somente após o fim da comunicação, quando ocorreu a troca de arquivos *peer-to-peer*.

Contudo, em uma aplicação prática, esse aspecto pode ser facilmente contornado, pois quando o fluxo for identificado pela rede neural, todas as próximas tentativas de conexão do usuário (endereço IP de origem) com o de destino (endereço IP de destino) serão bloqueadas ou estarão sujeitas às ações determinadas pelas políticas de gestão da rede. Isto é possível armazenando os endereços IPs de origem e destino contidos no elemento de fluxo, classificados como P2P.

CAPÍTULO 7 - CONCLUSÕES

7.1 - Conclusões

Apesar dos avanços proporcionados pelas redes *peer-to-peer*, tanto na forma de compartilhar as informações, quanto na criação de redes colaborativas, permitindo que recursos computacionais sejam compartilhados através da Internet, elas trouxeram consigo problemas para os sistemas de comunicação de dados. As livres trocas de arquivos entre usuários criam problemas para a comunicação de dados. A abertura na transferência de informações disponibilizou acesso livre a materiais protegidos por direitos autorais como, por exemplo, filmes, músicas, livros, softwares, etc [2-11].

Outro aspecto negativo da comunicação P2P foi o grande impacto causado nos *link*s de comunicação de dados. A característica colaborativa das redes P2P obriga que a transferência de dados seja bidirecional para todos os *hosts* que a compõem. Logo, a troca de grandes arquivos (2 Gbytes ou mais) tem causado uma mudança no perfil do tráfego da Internet, prejudicando a comunicação de dados como um todo, pois os *link*s de dados têm ficado estrangulados, prejudicando o desempenho de muitas outras aplicações [2-10]. Outra característica nociva da comunicação P2P são os vírus de computador e *spywares* que utilizam as redes para se propagarem [7].

Portanto, este estudo se concentra, em seu objetivo geral, na busca de uma solução alternativa aos sistemas atuais de controle de tráfego dos aplicativos *peer-to-peer*, como, por exemplo, os filtros de pacotes.

Na busca da solução para esse objetivo geral, é proposto um sistema baseado em comportamento do tráfego, que utiliza o reconhecimento de padrões dos fluxos gerados pela comunicação P2P nos dados exportados pelo protocolo Cisco Netflow. Para a solução do objetivo específico é empregado um modelo neural de redes perceptron multicamadas (MLP) permite inferir sobre: os tempos de conexões destes aplicativos, número de fluxos gerados, número de pacotes trafegados e número de bytes trafegados, desta forma identificando as conexões P2P.

Através dos resultados mostrados no item 6.2 se conclui que tanto o objetivo geral quanto o objetivo específico, apresentados na introdução do estudo, foram alcançados. Na

busca de um modelo neural capaz de discernir entre os protocolos regulares e *peer-to-peer*, os resultados confirmam as informações encontradas na bibliografia do assunto. O perfil de tráfego do protocolo HTTP é detectado em relação ao P2P e apresenta valores satisfatórios. Já na comparação entre o protocolo de transferência de arquivos FTP e os protocolos *peer-to-peer*, a distinção detectada pela rede neural é ainda maior.

Em relação aos outros modelos de solução, o sistema proposto pode ser comparável ao modelo de assinaturas, que, porém, sobre determinadas circunstâncias, é ineficiente na classificação do tráfego. O desempenho do modelo proposto é bastante parecido com o modelo de topologia e uma combinação entre ambos poderia criar um modelo ainda melhor, conforme será discutido no item 7.2.3.

O modelo proposto, além de superar as deficiências encontradas nos sistemas de filtros, que se tornam ineficazes quando o aplicativo *peer-to-peer* utiliza técnicas de camuflagem, e nos sistemas de assinaturas, que são incapazes de detectar o tráfego criptografado apresenta outras vantagens. O modelo trabalha com dados consolidados do protocolo Netflow, não realizando análises sobre o próprio pacote da comunicação. No caso da aplicação do modelo em *backbones* com conexões de grande velocidade, o sistema atenderia as necessidades de velocidade desejadas, pois não seriam gerados atrasos adicionais sobre a comunicação.

Um aspecto importante a ser citado sobre o modelo é a análise realizada sobre o fluxo de informações já consolidadas no roteador, ou seja, quando o usuário da rede efetuar um tráfego de dados P2P, o sistema irá detectar somente após o fim da comunicação, que ocorreu a troca de arquivos pela rede *peer-to-peer*. Porém, em uma aplicação real, esse problema pode ser abstraído, pois quando o primeiro fluxo do aplicativo *peer-to-peer* for detectado todas seguintes tentativas de conexão serão monitoradas, pois é possível construir uma tabela com os IPs de origem e destinos dos fluxos P2P e em seguida tomar as ações determinadas pela política de gestão da rede.

Nesse sentido, a pesquisa contribui com uma nova forma de monitoramento da transmissão de dados, disponibilizando um estudo investigativo da possibilidade de se utilizar técnicas de inteligência artificial na detecção de tipos de protocolos, pois ela se adapta às novas características de troca de arquivos. Esta investigação consistiu, assim, na elaboração de um modelo de avaliação capaz de distinguir, através do comportamento do fluxo da comunicação, os tráfegos *peer-to-peer* em relação aos protocolos “convencionais” (http, ftp, dns, etc).

7.2 - Sugestões para trabalhos futuros

7.2.1 - Novos protocolos

Naturalmente, este estudo deixa em aberto a continuidade da pesquisa para a classificação entre protocolos *peer-to-peer* e outros protocolos regulares largamente utilizados, como o pop, smtp, dns, etc. Dado o caráter investigativo desde estudo, não cabe uma análise exaustiva do emprego do modelo proposto na classificação de muitos protocolos neste documento.

Uma análise similar pode ser feita para a quantidade de protocolos detectados por uma rede neural. Neste ponto, sugere-se a aplicação de uma única rede neural na detecção de mais de dois protocolos por rede. No caso da análise de muitos protocolos conhecidos, uma única rede neural forneceria uma classificação mais variada para um número maior de protocolos.

7.2.2 - Novas técnicas de inteligência artificial

O emprego da rede neural MLP apresentou um bom desempenho através do uso do modelo proposto. Contudo, antes dessa escolha, foram realizados testes com um modelo neural híbrido entre *self-organizing maps* e uma *single layer* perceptron na classificação de dez protocolos simultaneamente, não obtendo bons resultados. Porém, com base em toda a discussão feita sobre o tema neste estudo, alguns ajustes podem ser realizados ao modelo em busca de um melhor desempenho.

Outra técnica de inteligência artificial avaliada foi o emprego de um classificador *Rough Sets* [38]. O modelo testado é muito similar ao proposto na figura 5.4, apenas substituindo o classificador neural pelo classificador escrito em *Rough Sets*. Este modelo apresentou resultados satisfatórios, contudo não foi abordado neste texto devido à extensão proposta para um trabalho de dissertação.

7.2.3 - Consideração de uma abordagem com mais parâmetros

Com o Netflow ainda é possível extrair mais indícios que possam identificar um aplicativo *peer-to-peer*. Contudo, a consideração destes parâmetros descaracterizaria a

avaliação do protocolo exclusivamente pelo comportamento. Estes parâmetros adicionais podem ser obtidos diretamente do protocolo Netflow, sendo eles:

- Topologia da rede e distribuição dos *hosts*: pela avaliação do número de *hosts* e as portas de comunicação, pode-se determinar se ele faz parte de uma rede *peer-to-peer*. Uma análise considerando todos os IPs da rede local pode-se supor a topologia da rede *peer-to-peer* com seus servidores e *hosts* ativos.
- Evolução do tráfego no tempo: à medida que um cliente encontra mais *hosts* com o arquivo que ele deseja transferir, a tendência é que o número de conexões, volume do tráfego e tempo de conexão sinalizem um host realizando uma transferência P2P.

A proposição ideal seria o complemento do modelo neural proposto adicionando informações utilizadas no modelo de topologia, pois o Netflow fornece os endereços IP de origem e destino e as portas utilizadas por todos os fluxos que atravessam o roteador. A forma de análise do modelo de topologia associado com o comportamento da conexão poderia fornecer resultados melhores. Considerando também a duração do fluxo, sabe-se que o tráfego de um *peer* evolui à medida que este permanece na rede, pois ele encontra ou é encontrado por outros *peers* para transferir arquivos. A adição dessa informação poderia proporcionar ainda mais precisão ao sistema.

7.2.4 - Solução para ambiente real

A figura 7.1 ilustra a arquitetura de um sistema prático para utilização do modelo MLP proposto. No primeiro momento, o servidor contendo a aplicação que implementa o modelo neural proposto não executaria nenhuma ação. Após o primeiro envio dos fluxos do roteador de borda para o servidor com a aplicação, os fluxos seriam classificados. Aqueles que fossem rotulados como tráfego *peer-to-peer* teriam seus endereços IPs de origem e destino armazenados em uma tabela de controle. Com os endereços dessa tabela, o servidor realiza a marcação dos futuros pacotes IP, com endereços de origem e destino cadastrados na tabela. Pode-se alterar o byte ToS com um identificador P2P, dessa forma forçando que o pacote esteja sujeito às políticas de QoS determinadas no roteador de borda.

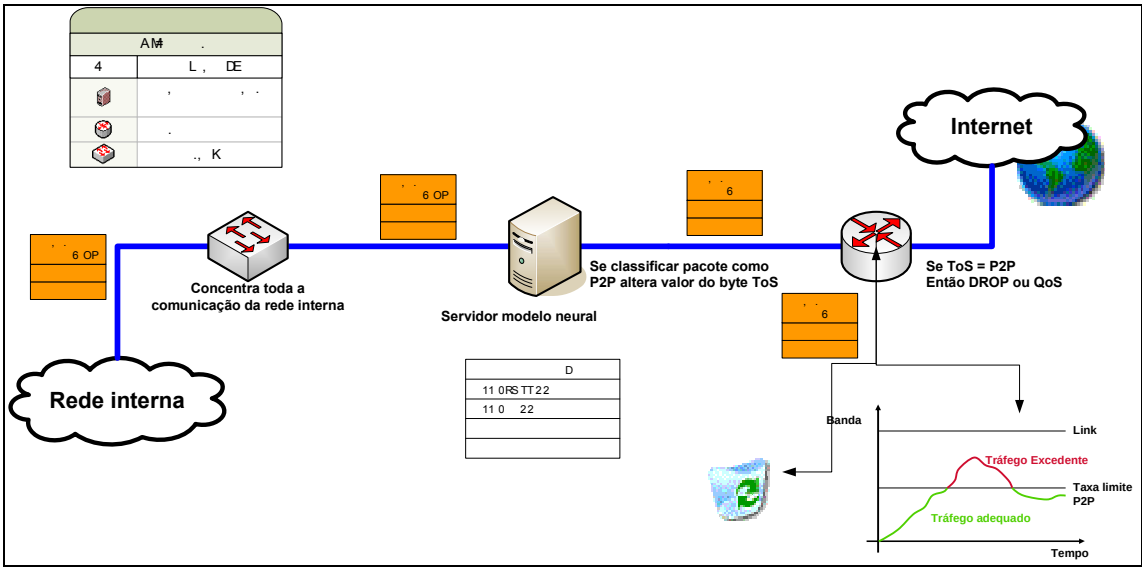


Figura 7. 1 – Arquitetura do sistema proposta para ambiente em tempo real

Para um sistema real, identificado o tráfego, existem três políticas para lidar com o tráfego P2P:

- Descarte;
- Aplicação de QoS; e
- Notificação e educação dos usuários de P2P.

A forma de tratar o tráfego depende da política de cada instituição. No entanto, a forma recomendada neste estudo é que o administrador da rede trabalhe com três níveis de classes de serviços aplicado através de QoS, sendo:

- Tráfego regular: seria dada a maior prioridade para a transferência de dados aos fluxos que fossem classificados como sendo regulares;
- Tráfego indefinido: possuiria um nível de prioridade intermediário para a transferência dos arquivos;
- Tráfego *peer-to-peer*: possuiria a menor prioridade na utilização do *link*.

Essa abordagem permitiria que mesmo um tráfego que seja classificado incorretamente possa ser realizado. No entanto, haveria de se considerar a atualização da tabela de IP que foram classificados como sendo pertencentes a redes *peer-to-peer*, que realizaria a reclassificação dos IPs de tempos em tempos. A utilização proposta seria capaz de melhorar o desempenho dos *link*s de comunicação de dados.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] B. Leiner, V. Cerf, D. Clark, R. Kahn, L. Kleinrock, D. Lynch, J. Postel, L. Roberts, S. Wolff (2000). “A Brief History of the Internet”. *Class Notes*, Novembro.

[2] T. Karagiannis, A. Broido, N. Brownlee, M. Faloutsos (2003). “File-sharing in the Internet: A characterization of P2P traffic in the backbone”. UC, Riverside CAIDA, SDSC, UCSD UCRiverside.

[3] S. F. L. Fernandes, G. S. Silvestre, K. L. Dias, J. B. Rocha Jr., D. Sadok, C. Kamienski (2004). “Análise de Tráfego P2P no Backbone da RNP”. 22° Simpósio Brasileiro de Redes de Computadores.

[4] N. Leibowitz, M. Ripeanu, A. Wierzbicki (2003) “Deconstructing the Kazaa Network”. *Proceedings of the the Third IEEE Workshop on Internet Applications* (WIAPP’03).

[5] A. Spognardi, A. Lucarelli, R. DiPietro (2005). “A Methodology for P2P File-Sharing Traffic Detection”. *Proceedings of the Second International Workshop on Hot Topics in Peer-to-peer Systems* (HOT-P2P'05).

[6] H. Bleul, E. P. Rathgeb, S. Zilling (2006). “Evaluation of an Efficient Measurement Concept for P2P Multiprotocol Traffic Analysis”. *Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications* (EUROMICRO-SEAA’06), IEEE.

[7] T. Karagiannis, A. Broido, M. Faloutsos, K. Claffy (2004). “Transport Layer Identification of P2P Traffic”. *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, Taormina, Sicily, Italy, IMC’04. ACM1-58113-821-0/04/0010

[8] A. Wagner, T. Dubendorfer, L. Hammerle, B. Plattner (2006). “Flow-Based Identification of P2P Heavy-Hitters”. *Proceedings of International Conference on Internet Surveillance and Protection*, ICISP '06. *Published in* 0-7695-2649-7/06, IEEE

[9] S. Subhabrata, J. Wang. “Analyzing Peer-to-peer Traffic Across Large Networks” (2004). *Published in IEEE/ACM Transactions on Networking,* 0-7695-2649-7/06, Vol. 12, No 2.

[20] S. Kirkpatrick, K. Yoram, B. Danny (2005). “The eMule Protocol Specification”. School of Computer Science and Engineering, The Hebrew University of Jerusalem, Jerusalém.

[21] J. Pouwelse, P. Garbacki, D. Epema, H. Sips (2004). “A Measurement Study of the

[29] FILE TRANSFER PROTOCOL (FTP). Request for Comments: 959. Disponível em <http://www.w3.org/Protocols/rfc959/>. Acessado em Novembro de 2006

[30] D. Comer, R. Droms (2003). “Computer Networks and Internets”. Prentice Hall; 4 ed.

[31] T. W. Shinder, R. J. Shimonski, D. L. Shinder (2003). “The Best Damn Firewall Book Period”. Rockland, USA: Syngress Publishing, Inc.

[32] A. S. Tanenbaum (2002). “Computer Networks”. Prentice Hall PTR; 4 edition.

[33] CISCO NetFlow. Disponível em <http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html>. Acesso em 22 de Abril de 2007.

[34] Introdução às redes Peer-to-peer. Notas de aula Redes de Computadores I. Universidade Federal do Rio de Janeiro. Disponível em <http://www.gta.ufrj.br/grad/06_1/p2p/emule.html> Acesso em 03 de Janeiro 2007.

[35] Cisco Systems (2006). “Managing peer-to-peer traffic with cisco service control technology”. *White Paper*, Cisco Systems, Inc.

[36] E. T. Nakamura, P. L. Geus, (2003). “Segurança de Redes em Ambientes Cooperativos”. 2. ed. São Paulo: Editora Futura.

[37] Cisco Systems (2002). “Software Configuration Guide For Cisco 2600 Series, Cisco 3600 Series, and Cisco 3700 Series Routers”. Manual de configuração, Cisco Systems, Inc.

[38] Z. Pawlak (1991). “Rough Sets: Theoretical Aspects of Reasoning About Data”. Dordrecht Kluwer Academic Publishing.

[39] S. Haykin (1999). “Neural Networks a Comprehensive Foundation. Second Edition”, Prentice Hall

APÊNDICE A – Configuração do roteador de borda

```
CoreFlow1# sh run
Building configuration...

Current configuration : 2015 bytes
!
! No configuration change since last restart
!
version 12.2
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname CoreFlow1
!
enable password class
!
username SanJose1
memory-size iomem 10
ip subnet-zero
ip flow-cache timeout active 1
ip cef
!
!
ip name-server 200.149.55.140
!
ip dhcp pool incit
   network 192.168.0.0 255.255.255.0
   default-router 192.168.0.1
   dns-server 200.149.55.140
!
vpdn enable
!
!
mta receive maximum-recipients 0
!
!
interface FastEthernet0/0
 description Interfce Lan
 ip address 192.168.0.1 255.255.255.0
 ip nbar protocol-discovery
 ip route-cache flow
 duplex auto
 speed auto
 no cdp enable
!
interface Serial0/0
 ip address 192.168.2.2 255.255.255.0
```

```
 shutdown
 no cdp enable
!
interface FastEthernet0/1
 description Interface WAN-ADSL
 bandwidth 1024
 ip address 192.168.254.2 255.255.255.0
 ip nbar protocol-discovery
 ip route-cache flow
 duplex auto
 speed auto
 no keepalive
!
interface Serial1/0
 no ip address
 shutdown
 no cdp enable
!
interface Serial1/1
 no ip address
 shutdown
 no cdp enable
!
interface Serial1/2
 no ip address
 shutdown
 no cdp enable
!
interface Serial1/3
 no ip address
 shutdown
 no cdp enable
!
interface Serial1/4
 no ip address
 shutdown
 no cdp enable
!
interface Serial1/5
 no ip address
 shutdown
 no cdp enable
!
interface Serial1/6
 no ip address
 shutdown
 no cdp enable
!
interface Serial1/7
```

```
 no ip address
 shutdown
 no cdp enable
!
ip flow-export version 5
ip flow-export destination 192.168.0.2 9996
ip classless
ip route 0.0.0.0 0.0.0.0 192.168.254.254
ip http server
```

APÊNDICE B – Sintaxe dos comandos flow-tools

- **Script de inicialização do flow-capture**

```
#!/bin/sh
# description: Start Flow-Capture

   case "$1" in
   'start')
   flow-capture -w /var/netflow/ft 192.168.0.2/192.168.0.1/9996 -S5 -V5 -E30G -n 287 -N 0 -
R /usr/local/netflow/bin/linkme
  ;;
   'stop')
   killall -9 flow-capture
   rm -f /var/lock/subsys/startflows
   ;;
   *)
   echo "Usage: $0 { start | stop }"
   ;;
   esac
   exit 0
```

- **Script de utilização das ferramentas flow-cat, flow-nfilter, flow-export**

```
#!/bin/bash
#Script para executar todos os comandos para gerar cada arquivo necessário
#Flow-cat concatena todos os arquivos de netflow
#Flow-nfliter filtra de acordo com a porta do serviço e joga em um arquivo

flow-cat ft* | flow-nfilter -ffiltro.http          -Ffoo > filtrado.http
flow-cat ft* | flow-nfilter -ffiltro.ftp       -Ffoo > filtrado.ftp
flow-cat ft* | flow-nfilter -ffiltro.emule       -Ffoo > filtrado.emule
flow-cat ft* | flow-nfilter -ffiltro.torrent       -Ffoo > filtrado.torrent

#Exportar os fluxos em ASCII
echo "http -->"
flow-export -f2 < filtrado.http > filtrado.http.ascii
echo "ftp -->"
flow-export -f2 < filtrado.ftp > filtrado.ftp.ascii
echo "emule -->"
flow-export -f2 < filtrado.emule > filtrado.emule.ascii
echo "torrent -->"
flow-export -f2 < filtrado.torrent > filtrado.torrent.ascii
```

- **filtro.ftp**

```
# Arquivo nfilter para protocolo ftp

filter-primitive port20
  type ip-port
  permit 20

filter-primitive port21
  type ip-port
  permit 21

#Servidor FTP passivo
filter-primitive ftppassivo
  type ip-address
  permit 143.106.10.150

filter-definition foo
  match ip-destination-address ftppassivo
or
  match ip-destination-port port20
or
  match ip-destination-port port21
```

- **filtro.http**

```
# Arquivo nfilter para protocolo http

filter-primitive port80
  type ip-port
  permit 80

filter-definition foo
  match ip-destination-port port80
```

- **filtro.emule**

```
# Arquivo nfilter para protocolo Emule
filter-primitive port4661
  type ip-port
  permit 4661

filter-primitive port4665
  type ip-port
  permit 4665

filter-definition foo
  match ip-destination-port port4661
or
  match ip-destination-port port4665
```

- **filtro.torrent**

```
#Arquivo nfilter para protocolo BitTorrent
filter-primitive port6881
  type ip-port
  permit 6881

filter-definition foo
  match ip-destination-port port6881
```

APÊNDICE C – Implementação da MLP no Matlab

```
echo on
clear

numEntradas   = 4;
numEscondidos1 = 15;
numEscondidos2 = 10;
numSaidas     = 2;
numTr         = 7000;
numVal        = 2000;
numTeste      = 1000;

echo off

arquivoTreinamento = fopen('train-torrent-ftp.txt','rt');
arquivoValidacao   = fopen('vali-torrent-ftp.txt','rt');
arquivoTeste       = fopen('teste-torrent-ftp.txt','rt');

dadosTreinamento    = fscanf(arquivoTreinamento,'%f',[(numEntradas + numSaidas),
numTr]);
entradasTreinamento = dadosTreinamento(1:numEntradas, 1:numTr);
saidasTreinamento   = dadosTreinamento((numEntradas + 1):(numEntradas + numSaidas),
1:numTr);

dadosValidacao      = fscanf(arquivoValidacao,'%f',[(numEntradas + numSaidas), numVal]);
entradasValidacao   = dadosValidacao(1:numEntradas, 1:numVal);
saidasValidacao     = dadosValidacao((numEntradas + 1):(numEntradas + numSaidas),
1:numVal);

dadosTeste          = fscanf(arquivoTeste,'%f',[(numEntradas + numSaidas), numTeste]);
entradasTeste       = dadosTeste(1:numEntradas, 1:numTeste);
saidasTeste         = dadosTeste((numEntradas + 1):(numEntradas + numSaidas), 1:numTeste);

fclose(arquivoTreinamento);
fclose(arquivoValidacao);
fclose(arquivoTeste);

for entrada = 1 : numEntradas;
    matrizFaixa(entrada,:) = [0 1];
end

%Para MLP de uma camada
%rede = newff(matrizFaixa,[numEscondidos1 numSaidas],
%{'logsig','logsig'},'trainlm','learngd','sse'); % {'logsig','purelin'},'trainlm','learngd'
%rede = init(rede);

%Para MLP de duas camadas
rede = newff(matrizFaixa,[numEscondidos1 numEscondidos2 numSaidas],
{'logsig','logsig','logsig'},'trainlm','learngd','sse'); % {'logsig','purelin'},'trainlm','learngd'
```

```
rede = init(rede);

rede.trainParam.epochs   = 1000;
rede.trainParam.lr       = 0.001;
rede.trainParam.goal     = 70;
rede.trainParam.max_fail = 30;
rede.trainParam.min_grad = 0;
rede.trainParam.show     = 1;
rede.trainParam.time     = inf;
rede.trainParam.mem_reduc = 4;

fprintf('\nInicio do treinamento: ')
DATESTR(NOW)

fprintf('\nTreinando ...\n')

conjuntoValidacao.P = entradasValidacao;
conjuntoValidacao.T = saidasValidacao;

[redeNova,desempenho,saidasRede,erros] =
train(rede,entradasTreinamento,saidasTreinamento,[],[],conjuntoValidacao);

fprintf('\nTestando ...\n');

[saidasRedeTeste,Pf,Af,errosTeste,desempenhoTeste] =
sim(redeNova,entradasTeste,[],[],saidasTeste);

fprintf('SSE para o conjunto de treinamento: %6.5f
\n',desempenho.perf(length(desempenho.perf)));
fprintf('SSE para o conjunto de validacao: %6.5f
\n',desempenho.vperf(length(desempenho.vperf)));
fprintf('SSE para o conjunto de teste: %6.5f \n',desempenhoTeste);

[maiorSaidaRede, nodoVencedorRede] = max (saidasRedeTeste);
[maiorSaidaDesejada, nodoVencedorDesejado] = max (saidasTeste);
classifinal=(abs(saidasRedeTeste-saidasTeste));
padraMalClassif=[];
classificacoesErradas = 0;
for padrao = 1 : numTeste;
  if classifinal(padrao)>=.3
        classificacoesErradas = classificacoesErradas + 1;
       padraMalClassif= [padraMalClassif; padrao];
    end
end

erroClassifTeste = 100 * (classificacoesErradas/numTeste);

fprintf('Erro de classificacao para o conjunto de teste : %g \n',erroClassifTeste);
```

```
if padraMalClassif~=0
fprintf('Padroes Mal Classificados:[');
fprintf(' %g',padraMalClassif);
fprintf(']; \n');
end

b=linspace(1,numTeste,numTeste);
plot(b,saidasRedeTeste,b,saidasTeste)
c=[saidasRedeTeste' saidasTeste'];
diary bprun.log
%c
fprintf('Erro de classificacao para o conjunto de teste : %g \n',erroClassifTeste)
fprintf('\nFim do treinamento: ')
DATESTR(NOW)
diary off
```

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo