

Universidade do Minho
Escola de Engenharia

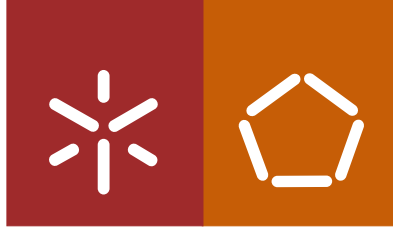
Rui Tiago Oliveira Ferraz

**Uma infra-estrutura virtual de nível 2
para interligação de máquinas virtuais**

Rui Tiago Oliveira Ferraz **Uma infra-estrutura virtual de nível 2 para interligação de máquinas virtuais**

UMinho | 2011

Outubro de 2011



Universidade do Minho
Escola de Engenharia

Rui Tiago Oliveira Ferraz

Uma infra-estrutura virtual de nível 2 para interligação de máquinas virtuais

Dissertação de Mestrado
Mestrado em Engenharia de Comunicações

Trabalho realizado sob a orientação do
Professor Doutor António Costa
e da
Professora Doutora Maria João Nicolau

Outubro de 2011

É AUTORIZADA A REPRODUÇÃO PARCIAL DESTA DISSERTAÇÃO APENAS PARA EFEITOS DE INVESTIGAÇÃO, MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE;

Universidade do Minho, ___/___/_____

Assinatura: _____

Agradecimentos

Obrigado a todos aqueles que permitiram a facultaram este momento:

Aos meus pais que suportaram os meus estudos, aos companheiros de luta que sempre estiveram ao meu lado a ajudar-me a ultrapassar problemas e a lutar pelo objectivo.

Aos orientadores, que sempre me brindaram com conselhos sábios.

À Catarina, que sempre esteve ao meu lado.

Resumo

O conceito de virtualização de redes está a expandir-se. A capacidade de oferecer, a partir de uma única infraestrutura de rede, várias redes virtuais distintas interessa tanto a operadoras como a académicos. Esta característica permite criar um "mundo" virtual, auto-contido, com parâmetros controlados e alteráveis, onde podem ser conduzidas experiências com novas tecnologias ou mesmo, olhando sob um ponto de vista empresarial, oferecer serviços específicos a clientes onde os parâmetros da rede são dinâmicos e contratualizados com um revendedor.

Neste trabalho descreve-se a pesquisa, desenvolvimento e teste de uma implementação de rede virtual, onde aborda os temas relacionados com a descoberta de recursos e as diferentes metodologias para comunicação das entidades virtuais. As ligações criadas permitem virtualizar uma arquitectura de nível 2 considerando cenários de conexão entre dois ou mais nós. São utilizados vários mecanismos de modo criar a rede virtual, conexões baseadas em *Sockets* e *RawSockets* permitem assegurar uma comunicação controlada sobre uma infra-estrutura de rede operacional. Os parâmetros que estas ligações oferecem foram avaliados numa situação real comprovando a possibilidade de disponibilização desta tecnologia sobre a infra-estrutura pública.

A proposta foi implementada num protótipo funcional que permitiu avaliar as opções tomadas durante o desenvolvimento do projecto. Os resultados obtidos comprovam a capacidade da abordagem para criar uma infra-estrutura de rede virtual de nível 2 criando conexões entre nós virtuais com características diferentes e com uma largura de banda similar à oferecida pela rede de suporte. Foram implementadas 4 formas de criação de *links* virtuais que se dividem no número de nós que permitem interligar e na forma como permitem que a informação seja trocada.

Abstract

Network virtualization is expanding. The capacity to deploy various networks using the underlying infra-structure concurrently without the pressures applied by legacy protocols is seen by network operators and academics as a point of interest and evolution.

The characteristics of a virtual network allow the creation of various virtual worlds self contained and with controlled parameters where experiences can be maintained with new technologies, or in an enterprise context, where it can offer specific conditions to clients, where the network parameters are contracted.

This work describes a proposal of a virtual network implementation, where the discovery of resources and the exchange of information between nodes are some of the key aspects.

The virtual connections, allow the virtualization a layer 2 network architecture considering links between 2 or more virtual nodes. Various approaches were used in order to create the virtual network links, connections based in Sockets and RawSockets ensure a controlled communication through an operational substrate network.

The capabilities of these connections where tested in a real situation, ensuring the possibility to deploy this technology in a public infrastructure.

The proposal was implemented in a functional prototype, that allowed the evaluation of the choices made during the development. The results obtained demonstrate the capacity to create a layer 2 virtual network infrastructure capable of creating connections with diferent parameters between virtual hosts, with a bandwidth similar to the support network. 4 approaches to create of the virtual links were implemented, that can be divided by the number of virtual hosts they are capable of connecting, and the way they share the information.

Conteúdo

Agradecimentos	iii
Resumo	v
Abstract	vii
Lista de figuras	xi
Lista de tabelas	xv
Acrónimos	xvii
1 Introdução	1
1.1 Enquadramento	1
1.2 Objectivos	3
1.3 Estrutura da dissertação	3
2 Virtualização	5
2.1 Emulação de <i>Hardware</i>	7
2.2 Virtualização total	8
2.3 Paravirtualização	9
2.4 Virtualização ao nível do Sistema operativo	10
2.5 Virtualização ao nível da aplicação	11
2.6 Comparação das características	11
2.7 Virtualização da rede	12
2.7.1 Classificação quanto ao nível de segurança	12
2.7.2 Classificação quanto ao nível lógico	15
2.7.2.1 Infra-estrutura virtual de nível 3	15
2.7.2.2 Infra-estrutura virtual de nível 2	17
3 Soluções para construção de redes virtuais	19
3.1 <i>X-bone</i>	20
3.2 <i>Vine</i>	26
3.3 <i>Violin</i>	30
3.4 Rede Virtual PT inovação	33

3.5	4WARD	38
3.6	Síntese	45
4	Proposta de uma infra-estrutura virtual de nível 2	47
4.1	Descoberta de recursos	48
4.2	Mapeamento das entidades virtuais	50
4.2.1	Mapeamento de ligações ponto a ponto	51
4.2.2	Mapeamento de ligações multi-ponto	52
4.2.3	Manutenção e portabilidade da rede virtual	54
5	Implementação da proposta de infra-estrutura virtual de nível 2	55
5.1	Procura de hosts	55
5.2	Ponto a ponto - <i>RawSockets</i>	60
5.2.1	<i>RawSockets</i>	60
5.2.1.1	Jpcap	62
5.2.2	Negociação de Links virtuais	63
5.2.3	Criação do link virtual	68
5.2.3.1	Interface virtual	68
5.2.3.2	Interface externa	71
5.3	Ponto a ponto - <i>Sockets Unicast</i>	72
5.3.1	Interface externa	74
5.3.2	Interface virtual	74
5.4	Ponto a multi-ponto - <i>RawSocket</i>	75
5.5	Ponto a multi-ponto - <i>MulticastSockets</i>	85
5.6	Portabilidade da rede virtual	86
5.7	Configuração e manutenção das máquinas virtuais	89
6	Resultados	91
6.1	Largura de banda virtual com uma conexão <i>TCP</i>	93
6.2	Largura de banda virtual utilizando uma conexão <i>UDP</i>	94
6.3	Atraso introduzido pela conexão virtual	98
7	Conclusão	101
	Referências	105
	Anexo A	111

Lista de Figuras

2.1	Evolução da tecnologia de virtualização desde 1960 - Imagem retirada de [1]	6
2.2	A virtualização permite vários sistemas sobre um único suporte	7
2.3	Esquema representativo da Emulação de Hardware - adaptado de [2]	8
2.4	Esquema representativo da virtualização total - adaptado de [2]	9
2.5	Esquema representativo da paravirtualização - adaptado de [2]	9
2.6	Esquema representativo da virtualização ao nível do Sistema operativo - adaptado de [2]	10
2.7	Esquema representativo da virtualização ao nível do Sistema operativo - adaptado de [2]	11
2.8	<i>Trusted VPN</i> : linha de comunicação separada da infra-estrutura pública	13
2.9	<i>Secure VPN</i> : Linha de comunicação com privacidade, onde os dados são cifrados	14
2.10	<i>Híbrid VPN</i> : Linha de comunicação com privacidade baseada em duas técnicas diferentes	14
2.11	Encapsulamento das informações a enviar	16
3.1	Imagem retirada de [3] que ilustra a base lógica do funcionamento do <i>X-bone</i>	21
3.2	Imagem adaptada de [3], que permite compreender a arquitectura do projecto <i>X-bone</i>	23
3.3	Imagem adaptada de [3] que esquematiza os passos realizados pela procura de nós para o <i>overlay</i>	24
3.4	Imagem adaptada de [3], que permite verificar de que forma os nós comunicam os <i>overlay</i> criados	24
3.5	Imagem adaptada de [3], que esquematiza os cabeçalhos introduzidos na informação a enviar	25
3.6	Imagem adaptada de [3], que permite verificar o <i>IPsec</i> presente em cada um dos cabeçalhos	26
3.7	Figura retirada de [4], que representa a arquitectura de um <i>overlay Vine</i> dividido pelas suas áreas lógicas de endereçamento	28
3.8	Imagem adaptada de [4] que permite compreender a forma como o <i>Vine</i> comunica	29
3.9	Imagem retirada de [5], que pretende esquematizar a infra-estrutura <i>Violin</i> , com 3 níveis de rede distintos	31
3.10	Imagem adaptada de [6], que permite compreender de que forma se relacionam os módulos	34
3.11	Figura retirada de [6] que permite compreender o algoritmo de procura de <i>links</i> virtuais	38
3.12	Imagem retirada de [1] que permite esquematizar as relações entre os módulos de criação da rede virtual	40

3.13	Imagem retirada de [1], que permite compreender de que forma se relacionam os protocolos de reserva de recursos utilizados	42
3.14	Imagem retirada de [1] que descreve a forma como os links são criados e os recursos são reservados no caminho entre os extremos	43
3.15	Imagem retirada de [1], que descreve a forma como os <i>InP</i> se relacionam com as entidades de rede	44
4.1	Representação do funcionamento da rede virtual a desenvolver, onde a camada superior troca datagramas de nível 2 e a rede de suporte IP	48
4.2	Passos realizados na procura de nós disponíveis na rede de suporte	49
4.3	Troca de informações entre nós virtuais numa abordagem ponto a ponto	52
4.4	Esquema de um <i>link</i> multi-ponto	53
5.1	Cada nó ligar-se-à e escutará um endereço multicast de troca de mensagens de controlo	56
5.2	Mensagem genérica trocada entre os <i>hosts</i> que suportam a rede virtual	56
5.3	Mensagem do tipo 1, enviada de forma a requisitar um pedido de recursos	57
5.4	Procura de nós com um <i>TTL</i> inicial	57
5.5	Procura de nós disponíveis na rede, utilizando um <i>TTL</i> mais elevado	59
5.6	Formato da mensagem de tipo 3 enviada, que permite divulgar localmente os nós conhecidos	59
5.7	Formato da mensagem de tipo 4, que permite actualizar o estado dos nós vizinhos	59
5.8	Troca de informação no modelo <i>OSI</i>	61
5.9	Transporte da informação da camada de ligação para a de aplicação	62
5.10	Captura de informação recorrendo ao <i>libpcap</i>	63
5.11	Campos presentes no cabeçalho <i>IPv4</i>	64
5.12	Formato da mensagem 5, que permite divulgar o <i>link</i> virtual que se pretende criar	65
5.13	Formato da mensagem 6, com o campo argumento a indicar que o protocolo sugerido não foi aceite	65
5.14	Mensagem de resposta ao pedido de criação de um link virtual onde os parâmetros foram aceites	66
5.15	Diagrama temporal que reflecte a forma como os parâmetros da ligação são negociados	66
5.16	As conexões virtuais funcionarão transparentemente sobre o <i>link</i> físico	67
5.17	Exemplo de como os protocolos poderiam ser negociados numa rede com mais do que dois nós	67
5.18	Descoberta do endereço mac num sistema <i>Linux</i>	69
5.19	Descoberta do endereço mac num sistema <i>Windows</i>	69
5.20	Envio de informações do link virtual para nós na mesma rede ou em redes distintas	70
5.21	Encadeamento das informações da máquina virtual até serem enviadas	70
5.22	Esquema lógico de como se processa a comunicação entre dois nós	72
5.23	Representação lógica de como as <i>Threads</i> se posicionam nas interfaces que pretendem escutar	73
5.24	Forma como os <i>DatagramSockets</i> trocarão informação entre os processos	73
5.25	Representação de uma troca de mensagens utilizando o <i>multicast</i>	76
5.26	O grupo de controlo permitirá anunciar a criação de um endereço para troca de mensagens entre nós virtuais	76

5.27	Formato da mensagem <i>Membership Query</i> , enviada pelos routers que suportam <i>Multicast</i>	77
5.28	Formato da mensagem <i>Membership Report</i> , enviada pelos nós de forma a definir a sua junção ao grupo <i>multicast</i>	78
5.29	Imagem adaptada [7]. Junção a um grupo <i>multicast</i> , utilizando <i>IGMPv3</i> , onde a mensagem é enviada para o grupo <i>224.0.0.22</i>	80
5.30	Imagem adaptada [7]. Alteração dos nós que se pretende escutar do grupo <i>multicast</i>	81
5.31	Imagem adaptada [7]. Resposta dos nós que pertencem ao grupo <i>multicast</i> a uma <i>Membership Queries</i>	81
5.32	Imagem adaptada [7]. Saída implícita de um grupo <i>multicast</i>	81
5.33	Cálculo do endereço <i>ethernet</i> a partir do endereço <i>IP</i> de um grupo <i>multicast</i> . .	82
5.34	Troca de mensagens num <i>link</i> multi-ponto, onde uma mensagem enviada para o grupo é passada para todos os nós	84
5.35	Formato da mensagem de tipo 7, que permite anunciar os <i>links multicast</i> criados	85
6.1	Infra-estrutura de suporte utilizada para realização dos testes	92
6.2	Resultados de largura de banda efectiva utilizando uma comunicação <i>TCP</i> entre os nós virtuais	93
6.3	Resultados de largura de banda efectiva utilizando uma comunicação <i>UDP</i> entre os nós virtuais	95
6.4	Resultados do teste de controlo de perdas utilizando uma conexão <i>UDP</i>	97
6.5	Representação gráfica dos atrasos registados em cada uma dos link virtuais criados	98
1	Janela inicial da aplicação	111
2	Janela de criação de uma conexão <i>LVSU</i>	112
3	Janela de alteração das características da máquina virtual	113
4	Janela de criação de <i>link</i> virtual do tipo <i>LVSU</i>	114
5	Janela de parametrização da conexão <i>RMSM</i>	115
6	Janela de criação de uma conexão <i>RMR</i>	115
7	Janela que permite consultar as características dos links criados	116

Lista de Tabelas

2.1	Síntese das características de virtualização	11
6.1	Características dos nós utilizados	91
6.2	Acrónimos utilizados para as abordagens de criação de <i>link</i> virtual	92

Acrónimos

CPU	C entral P rocessing U nit
GT-ITM	G eorgia T ech - I nternet T opology M odels
IGMP	I nternet G roup M anagement P rotocol
IP	I nternet P rotocol
LVRU	L ink V irtual com R awSockets U nicast
LVSU	L ink V irtual com S ockets U nicast
MAC	M achine A ddress C ode
NIC	N etwork I nterface C ard
OSI	O pen S ystems I nterconnection
QoS	Q uality of S ervice
RMR	R ede M ulti-ponto com R awSockets
RMSM	R ede M ulti-ponto com S ocket M ulticast
TCP	T ransmission C ontrol P rotocol
TTL	T ime T o L ive
UDP	U ser D atagram P rotocol
UML	U ser M ode L inux
XML	E xtensible M arkup L anguage

Capítulo 1

Introdução

O capítulo servirá para introduzir o projecto desenvolvido, apresentando a motivação para que este se tenha realizado, os objectivos em que pretende focar e finaliza-se com a descrição da estrutura do documento.

1.1 Enquadramento

O aumento da competitividade no desenvolvimento de sistemas de informação leva a que várias empresas procurem soluções que as diferenciem num mercado em constante crescimento. Sistemas actualmente comercializados para tarefas específicas são disponibilizados como uma imagem para uma plataforma virtual que pode ser descarregada rapidamente, apta a cumprir as necessidades específicas de um projecto, evitando alterações significativas, mantendo o sistema hospedeiro seguro enquanto são realizadas alterações no sistema virtual.

Esta tecnologia revelou-se muito interessante uma vez que permite testar sistemas sem grande dificuldade, pondo à prova o funcionamento de vários serviços nos mais diversos ambientes, por este motivo é aproveitada por várias empresas na área das tecnologias de informação permitindo a redução de custos, aumento de eficiência e fiabilidade. A capacidade de simular sistemas veio melhorar a capacidade de desenvolvimento, facultando, por um lado, o corte de custos com maquinaria e recursos humanos e por outro facilitar a reposição de um serviço numa situação de falha[8].

O conceito de virtualização globalizou-se, e apresenta conceitos que se estendem aos vários domínios, tal como a rede. A capacidade de criar uma infra-estrutura de comunicação manobrável, pode ser visto como o próximo passo na evolução da virtualização e da *Internet*[9], oferecendo às operadoras a possibilidade de fornecer serviços diferenciados, criando uma abstracção entre a infra-estrutura física e aquela que os nós virtuais vêem como disponível.

Uma arquitectura de rede virtual permite que vários nós se possam interligar segundo uma infra-estrutura virtual facultando a realização de testes com tecnologias em desenvolvimento sem a necessidade de um elevado custo em equipamentos e ligações. Algumas implementações vêem até esta tecnologia como um suporte de serviços de emergência, na medida em que permitem assegurar a comunicação sobre qualquer tipo de rede, por mais retalhada que esta se encontre[3].

Uma rede virtual permite a troca de informações num ambiente controlado e contido, sobre uma infra-estrutura pública. A topologia observada pelos nós que compõem a infra-estrutura virtual pode ser completamente distinta da topologia física, contudo a rede virtual utiliza os recursos disponíveis da infra-estrutura de suporte de forma concorrente.

Esta tecnologia apresenta algumas implementações disponíveis, que demonstram a capacidade de desenvolvimento que as redes virtuais pode oferecer. Projectos como o *PlanetLab*[10], *Vini*[11], *Geni*[12], *AWARD*[1] e *Akari*[13] são alguns dos que demonstram a motivação global para abordar este tema, que permite a criação de diversas implementações de rede com parâmetros distintos sobre uma infra-estrutura física ou mesmo sobre outra infra-estrutura virtual[5].

A capacidade de diversificação nos métodos de comunicação usados faz com que se veja a virtualização de rede como uma plataforma que potencia crescimento[9] pois permite estabelecer múltiplas redes isoladas, assentes sobre uma única plataforma física, abrindo um leque de novas possibilidades para operadoras. A tecnologia é, também, útil no contexto académico, uma vez que possibilita o desenvolvimento de novos protocolos e algoritmos, fornecendo uma separação da rede real e oferecendo uma diversidade de cenários de teste consoante a topologia de rede necessária.

A criação de uma rede virtual é abordada na maior parte dos projectos como uma série de passos encadeados, procura de recursos, mapeamento e instanciação das entidades, manutenção e portabilidade apresentam-se como as principais dificuldades a resolver. Estes projectos cingem-se a conexões de nível 3 entre nós virtuais, trocando apenas pacotes *IP* e mantendo a comunicação ponto a ponto. O projecto desenvolvido foca as dificuldades comuns de criação de

uma rede virtual, acrescentando-lhe a comunicação entre dois ou mais nós no nível de ligação de forma a permitir a construção de topologias de nível 2.

1.2 Objectivos

Os principais objectivos desta dissertação são:

- Avaliação das tecnologias de criação de redes virtuais, e as principais técnicas para realizar esta tarefa;
- Definir uma arquitectura que possibilite a criação de topologias virtuais de nível 2 sobre *IP*;
- Propor protocolos de comunicação entre máquinas virtuais;
- Implementar o protocolo proposto num protótipo funcional;
- Testar o protótipo implementado em várias situações.

1.3 Estrutura da dissertação

No capítulo 1 é feita uma breve introdução ao tema desta dissertação. É descrita a motivação para a realização deste trabalho, juntamente com os respectivos objectivos a alcançar. Este capítulo termina com a descrição da estrutura adoptada para esta dissertação.

O capítulo 2 aborda as tecnologias usadas. Inicialmente a virtualização, quais as motivações que fizeram com que a sua expansão fosse tão rápida. Depois é apresentada uma visão sobre algumas tecnologias utilizadas na construção do sistema e os conhecimentos considerados importantes para a compreensão da dissertação.

O capítulo 3 apresenta o estado da arte das redes virtuais. Descrevem-se os vários projectos estudados nos parâmetros mais importantes para a construção de uma rede virtual.

No capítulo 4 é apresentada a solução prevista para o problema apresentado, dividindo-o nas várias parcelas que o regem. Desde a escolha do software de virtualização, à forma como a

procura de entidades é realizada e a forma como comunicam entre si.

O capítulo 5 permite aprofundar a visão dada no capítulo anterior, abordando as soluções escolhidas e a forma como foram inseridas no projecto.

No capítulo 6 são apresentados os resultados experimentais. Aqui são descritas as ferramentas utilizadas para obter estes resultados, juntamente com a respectiva implementação destas ferramentas. É também efectuada uma comparação entre as diferentes características das conexões virtual implementadas, de modo a realçar as suas vantagens.

Por fim, o capítulo 7 apresenta as conclusões desta dissertação, resumindo os objectivos cumpridos, as possíveis modificações a efectuar ao sistema e propostas de trabalho futuro.

Capítulo 2

Virtualização

A virtualização não é um assunto recente e não está cingido a uma área. É um tema que vem sendo amadurecido com o aparecimento de novas tecnologias e desenvolvimentos. A origem da virtualização remonta ao ano de 1960[1], e refere-se ao conjunto de técnicas usadas para que um sistema lógico possa abstrair-se de um sistema físico que o suporta. Nesta técnica, comumente chamada de virtualização de plataformas, uma ferramenta de virtualização fornece aos *hosts* virtuais uma emulação do *hardware* do sistema.

Este tipo de tecnologia permitiu um avanço nas *TIC* a nível internacional, de que é exemplo o aparecimento da memória virtual[14] em 1972. Esta evolução pode ser comprovada na figura 2.1.

A proliferação da tecnologia trouxe grandes benefícios tanto para o mundo académico como para o mundo empresarial, sendo neste último que foi aproveitado todo o seu potencial permitindo uma revolução na forma como as aplicações empresariais são construídas.

A virtualização permite melhorias significativas em alguns dos pontos cruciais que afectam o funcionamento das empresas[8]:

- **Melhoria da eficiência do sistema** - A eficiência sofreu um aumento com a utilização da virtualização. O que era, sem virtualização, um conjunto de sistemas difusos preparados apenas para um serviço específico tornou-se, com a inclusão desta tecnologia, num ponto centralizado, capaz de conter vários serviços diferenciados, com uma elevada eficiência.

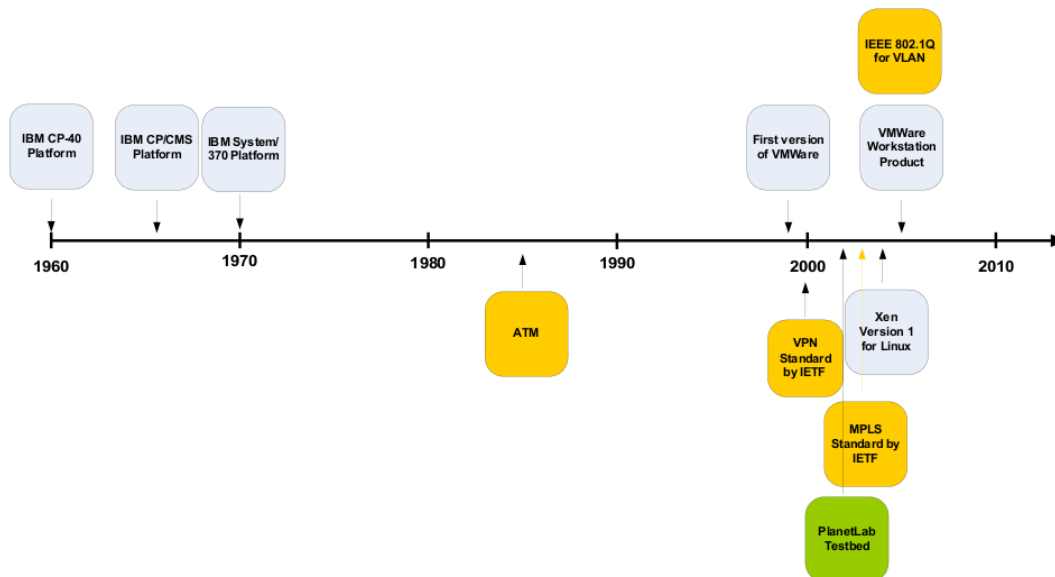


Figura 2.1: Evolução da tecnologia de virtualização desde 1960 - Imagem retirada de [1]

- **Complexidade de manutenção** - Com a evolução da área das *TIC* a velocidade de expansão das empresas aumentou exponencialmente. De forma a acompanhar esta evolução foi necessário o aumento da complexidade na manutenção dos sistemas. A virtualização vem combater esta dificuldade, reduzindo a complexidade e o número de servidores, permitindo ter apenas numa máquina vários serviços disponíveis e facilmente alteráveis.
- **Disponibilidade de serviço** - Quando uma máquina falha, os serviços podem ser facilmente reiniciados uma vez que as máquinas virtuais são representadas por ficheiros, sendo fácil o transporte de uma máquina virtual de um sistema para outro, poupando o tempo necessário que a instalação e configuração levariam num sistema normal.
- **Custos** - A redução de custos é o ponto principal para a rápida disseminação da virtualização. A capacidade de passar serviços dispersos por vários servidores para um único, permite aumentar a eficiência e reduzir o custo com equipamentos. Esta diminuição proporciona também a possibilidade de redução de espaço que é necessário para os acomodar.
- **Recuperação de falhas** - A utilização de vários serviços no mesmo sistema faculta a implementação de redundância na disponibilização dos serviços, permitindo a recuperação de falhas facilmente, sem que com isto se comprometa todo o sistema.
- **Independência do Sistema operativo** - Com a divergência e proliferação de sistemas operativos, é imperativo que um serviço esteja disponível, seja qual for o sistema que esteja

a ser utilizado. A virtualização permite que sejam acomodados serviços diferenciados pelo sistema operativo, tal como está esquematizado na figura 2.2.

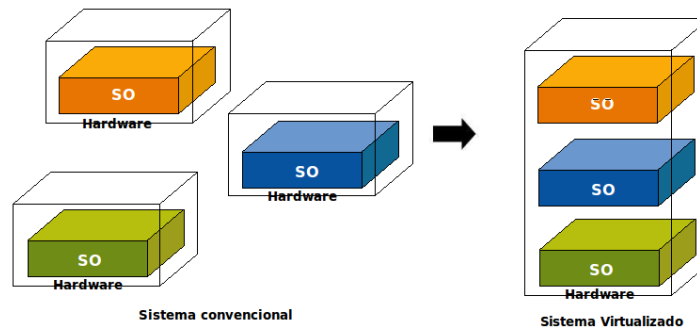


Figura 2.2: A virtualização permite vários sistemas sobre um único suporte

Estas vantagens, aliadas à capacidade de desenvolvimento que a tecnologia apresenta, faz com que a virtualização seja um ponto de interesse para o desenvolvimento das empresas. Grandes avanços têm sido feitos nesta área proporcionados, também, pelo meio académico que se voltou para esta tecnologia, utilizando todo o seu potencial.

A camada de virtualização, que controla o acesso aos vários recursos disponíveis permite restringir o processamento, memória, ou a largura de banda dado a uma máquina virtual permitindo mais facilmente gerar testes de stress.

Da mesma forma que existem várias vantagens com o uso da virtualização, existem diferentes maneiras de o conseguir obtendo resultados distintos consoante o nível de abstracção utilizado[2].

2.1 Emulação de *Hardware*

Este tipo de virtualização é o considerado mais complexo. Neste método, todo o *hardware* utilizado pela máquina virtual é emulado. Como todos os processos necessitam de ser emulados pela camada de emulação, a velocidade do sistema virtual é reduzida, diminuindo consoante os recursos do sistema que são emulados(Ciclos de relógio do *CPU*, *Caching*, etc ...)[2].

A emulação do *hardware* permite testar sistemas sem os modificar, alterando as características do hardware que é virtualizado de acordo com as necessidades.

O esquema do conceito apresentado está apresentado na figura 2.3.

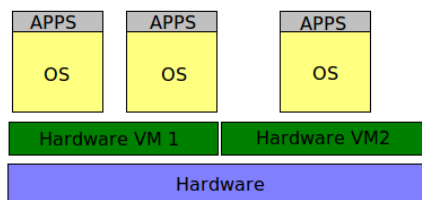


Figura 2.3: Esquema representativo da Emulação de Hardware - adaptado de [2]

Todo o sistema é emulado criando de uma camada de abstracção que vai funcionar como o *hardware* nativo. Podem ser até criadas várias camadas de abstracção diferenciadas, testando vários cenários.

Este tipo de virtualização é muito utilizada por equipas de desenvolvimento de drivers. Em vez de se esperar que o hardware esteja preparado, pode ser usada a sua virtualização, simulando o código que está a ser desenvolvido.

São exemplos deste tipo de virtualização projectos como o *BOCHS*[15], *VirtualPC*[16] ou o *QEMU*[17].

2.2 Virtualização total

A virtualização total, é um método de virtualização também conhecido por virtualização nativa. Este modelo conta com uma camada de virtualização que media a utilização do *hardware* pelos sistemas operativos virtualizados. Esta camada de abstracção controla as instruções que devem ser tratadas e as que devem ser ignoradas, uma vez que o *hardware* nativo não vai ser utilizado por um sistema virtualizado, mas sim partilhado usando este mediador. A abstracção deste conceito está apresentada na figura 2.4

A virtualização total vai ter um desempenho superior à emulação de *hardware*, contudo possui ainda com um grande *overhead* que se deve principalmente à camada de mediação, que trata os pedidos de utilização do hardware.

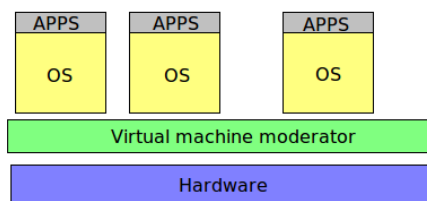


Figura 2.4: Esquema representativo da virtualização total - adaptado de [2]

A principal vantagem deste tipo de virtualização é a possibilidade de utilizar o sistema operativo sem qualquer tipo de alteração, pois o *hardware* que vai ser utilizado é o que a máquina realmente possui. Contudo, para que isto aconteça o sistema operativo virtualizado necessita de suportar o hardware a ser usado.

São exemplos deste tipo de virtualização projectos como o *VMWare Workstation*[18], *Parallels*[19] e *VirtualBox*[20].

2.3 Paravirtualização

A paravirtualização é também uma técnica popular de virtualização que possui algumas similaridades com a virtualização total. Esta forma de virtualização utiliza um supervisor para permitir a partilha dos recursos. É incluído nos sistemas operativos uma *API* especial que permitirá o aumento de velocidade da virtualização. Este tipo de funcionamento, descarta a necessidade de recompilação do sistema, ou a mediação de todos os pedidos de *hardware* uma vez que os sistemas operativos virtuais também vão participar em todo no processo de virtualização, ou seja, terão o conhecimento que estão a ser virtualizados, este conceito está apresentado na figura 2.5.

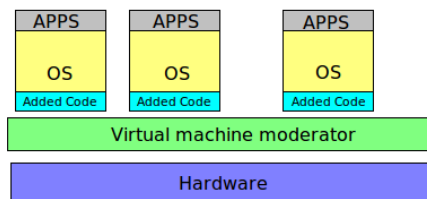


Figura 2.5: Esquema representativo da paravirtualização - adaptado de [2]

A paravirtualização necessita que os sistemas operativos virtuais sejam configurados para interagir com a camada de virtualização e um sistema que não esteja preparado para isto não pode ser executado em cima do mediador de paravirtualização.

São exemplos da paravirtualização projectos como o *XEN*[21] e o *VMWare*[18].

2.4 Virtualização ao nível do Sistema operativo

A virtualização ao nível do sistema operativo usa técnicas diferentes das focadas neste capítulo, pois ao invés de ser utilizada uma camada de abstracção que realiza todo o trabalho de mediar e controlar as diferentes máquinas e pedidos, todos os sistemas virtuais são construídos no sistema operativo nativo e todas as máquinas virtuais são criadas de forma a que sejam isoladas umas das outras. Este mecanismo de virtualização é apresentado na figura 2.6.

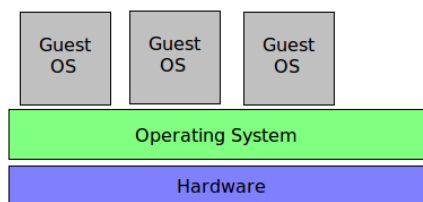


Figura 2.6: Esquema representativo da virtualização ao nível do Sistema operativo - adaptado de [2]

Esta forma de virtualização, devido ao facto dos sistemas virtuais utilizarem chamadas ao sistema do sistema operativo nativo, não acarreta o *overhead* que caracteriza outro tipo de técnicas. Isto ocorre em grande parte pela possibilidade de se colocar de parte a camada de mediação que era utilizada nas outras técnicas abordadas. Contudo, necessita que o sistema operativo nativo seja preparado para suportar as máquinas virtuais que irão ser iniciadas.

São exemplos deste tipo de virtualização projectos como o *Solaris Containers*[22], *BSD Jails*[23], *Linux Vservers*[24].

2.5 Virtualização ao nível da aplicação

A virtualização ao nível da aplicação permite que uma aplicação execute, utilizando o sistema operativo confinada a um ambiente isolado. Um utilizador que pretenda utilizar esta abordagem tem acesso a uma implementação de uma camada que executará no sistema operativo, as aplicações farão uso da camada de mediação que tratará dos pedidos. A representação desta abordagem está apresentada na figura 2.7.

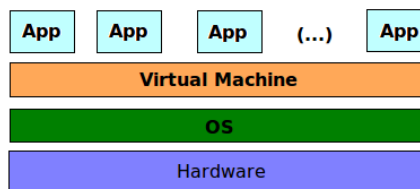


Figura 2.7: Esquema representativo da virtualização ao nível do Sistema operativo - adaptado de [2]

A *Java Virtual Machine*[25] é um exemplo deste tipo de virtualização.

2.6 Comparação das características

A tabela 2.1 apresenta uma comparação entre as abordagens de virtualização apresentadas[26].

Emulação	Desempenho	Alteração no OS virtual	Independência	Suporte <i>CPU</i>
Emulação de <i>Hardware</i>	Razoável	N	S	S
Virtualização total	Boa	N	S	S
Paravirtualização	Variável	S	N	N
Virtualização ao nível do S.O.	Boa	N	N	N
Virtualização ao nível da aplicação	Não aplicável	N	N	N

Tabela 2.1: Síntese das características de virtualização

As várias formas de virtualização estudadas apresentam contrapartidas, por um lado existe a paravirtualização e a virtualização ao nível do sistema operativo que tiram o máximo dos sistemas virtualizados, permitindo obter um maior desempenho que as restantes abordagens, mas necessitam de alterações no sistema operativo e suporte do *hardware*. Estes dois pontos tornam-se

especialmente importantes quando o sistema que se pretende desenvolver se pretende escalável, uma vez que nem todos os nós estão preparados para a utilização deste tipo de virtualização.

A virtualização total, por outro lado, permite virtualizar todo o *hardware* que o sistema operativo irá utilizar, possibilitando a utilização de qualquer tipo de nó. Esta faceta acarreta uma quebra no desempenho, que poderá ser aceitável caso o sistema a desenvolver não apresente este requisito.

2.7 Virtualização da rede

O trabalho desenvolvido propõe a construção de uma infra-estrutura virtual, é por este motivo importante o estudo das tecnologias que assentam sobre a mesma filosofia, utilizando uma conexão virtual como um mecanismo de comunicação entre dois pontos.

Uma *VPN* é um exemplo de uma conexão virtual, construída sobre uma infra-estrutura de rede física onde podem ser utilizados mecanismos de segurança de forma a complementar a comunicação.

A flexibilidade destes mecanismos, aliados ao custo que apresentam em comparação com linhas de comunicações privadas, tornam as *VPN* uma tecnologia muito usada por empresas que pretendem privacidade na troca de informações uma vez que permitem uma comunicação segura, usando um meio partilhado por um custo menor que aquele que seria necessário usando uma conexão dedicada.

Existem vários tipos de *VPN* e diferentes formas de as classificar, sejam *trusted VPN*, *secure VPN* ou *hybrid VPN*, dependendo do tipo de privacidade que oferecem ou segundo o nível lógico onde actuam[27].

2.7.1 Classificação quanto ao nível de segurança

No início do desenvolvimento da infra-estrutura da *Internet*, uma *VPN* podia ser considerada um circuito privado alugado por uma operadora de comunicações a um cliente[27]. Onde a segurança da comunicação era baseada numa linha dedicada, sem qualquer tipo de segurança ou

encapsulamento, ou seja, o fornecedor do serviço assegurava-se que a linha não seria usada por mais nenhum cliente. Tinha de existir uma confiança mútua entre o utilizador e o fornecedor. O utilizador confiava que o fornecedor do serviço manteria a integridade dos circuitos utilizando as melhores técnicas para evitar a captura de informação, por esta razão, este tipo de *VPN* são chamados *trusted VPN*. A comunicação entre os extremos é realizada sobre uma ligação dedicada, como a que está esquematizada na figura 2.8.

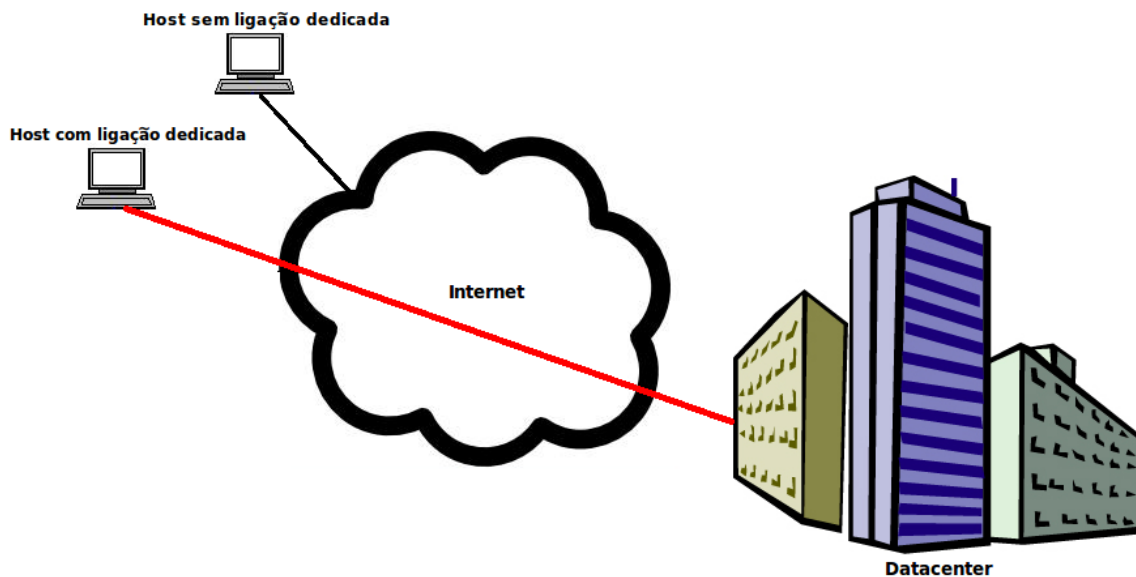


Figura 2.8: *Trusted VPN*: linha de comunicação separada da infra-estrutura pública

A massificação da infra-estrutura comunicações e a sua utilização por cada vez mais empresas, fez com que a segurança nas comunicações se tornasse um assunto de extrema importância, já não sendo fiável aceitar uma ligação apenas com base apenas na confiança. Este facto permitiu que vários vendedores pudessem olhar para este mercado como uma possibilidade, desenvolvendo protocolos que permitem realizar a troca segura de informação utilizando métodos criptográficos, proporcionando um nível de segurança não só no canal de comunicação, como nos dados que são enviados, este tipo de *VPN* são chamados de *secure VPN*. O meio de comunicação é partilhado e a troca de informação utiliza a infra-estrutura pública, como está representado na imagem 2.9.

Uma *VPN* pode ainda ser constituída por *secure VPN* e *trusted VPN*, criando o terceiro tipo de *VPN*, designado por *hybrid VPN*. Nesta abordagem, as ligações entre pontos intermédios da

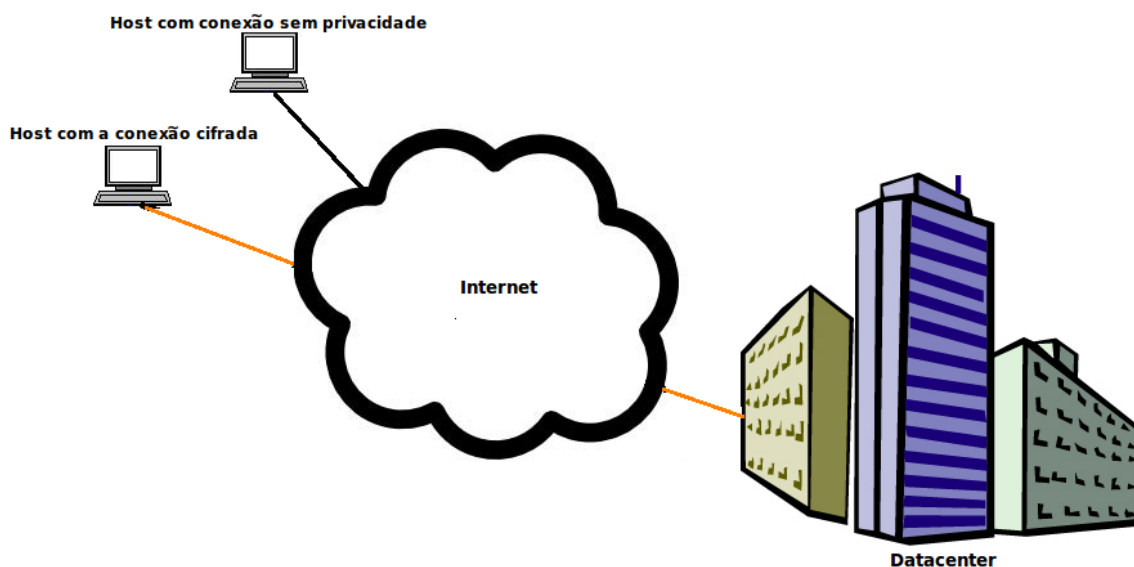


Figura 2.9: *Secure VPN*: Linha de comunicação com privacidade, onde os dados são cifrados

rede podem possuir diferentes características, fazendo com que entre os extremos, sejam utilizadas ambas as tecnologias, tal como apresentado na figura 2.10.

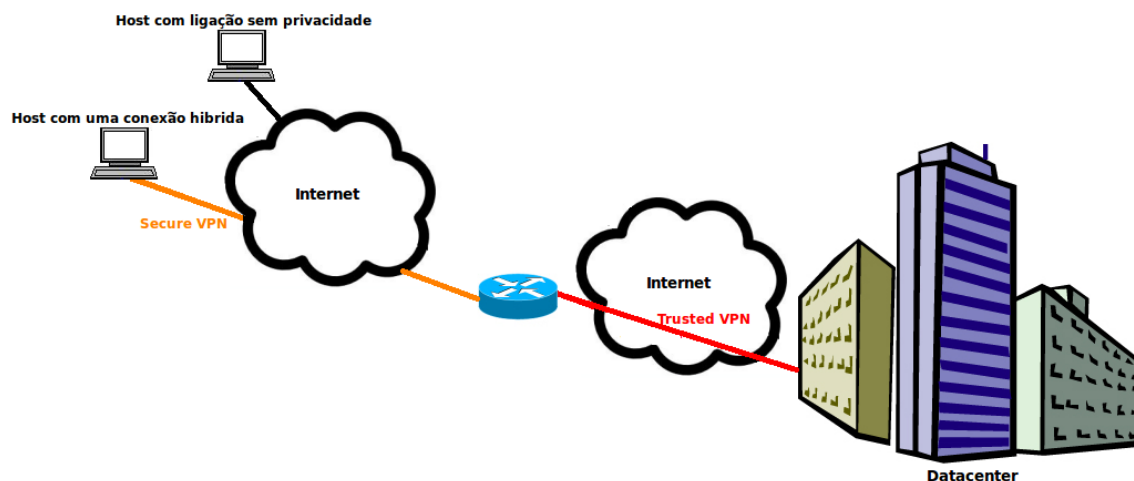


Figura 2.10: *Híbrid VPN*: Linha de comunicação com privacidade baseada em duas técnicas diferentes

Estes tipos de *VPN* são utilizados com base nas necessidades dos clientes que as requisitam. Entre as principais características que estas apresentam, é possível verificar as seguintes[27]:

- Troca segura de informação usando uma infra-estrutura partilhada;

- Certeza de que informação é originária do outro ponto da conexão;
- Controlo sobre o caminho que o tráfego segue, determinando quais as propriedades do caminho que é tomado, não permitindo que um utilizador mal intencionado altere esta rota;
- Capacidade de associar um nível de *QoS* a uma rota;
- Abstracção da rede de suporte.

Os *secure VPN* fornecem a segurança inerente à criptografia, mas não permitem assegurar a rota que a informação vai tomar. Com as *trusted VPN* é possível controlar as propriedades dos caminhos tomados, mas não é possível assegurar que os dados não sejam alterados. A inclusão das duas formas de *VPN* permite caracterizar as *hybrid VPN*, onde em espaços controlados utilizam as melhores características que cada uma destas pode oferecer.

2.7.2 Classificação quanto ao nível lógico

Para além da classificação baseada na privacidade, é possível separar as *VPN* pelos níveis da camada *OSI* onde actuam[28]. Separando as características das linhas de comunicação privadas por *VPN* de nível 2 ou 3.

2.7.2.1 Infra-estrutura virtual de nível 3

As *Layer 3 VPN* são caracterizadas pelo uso dos protocolos do nível de rede para a troca de informação entre os extremos do *link* virtual. Existem dois tipos de *VPN* de nível 3 que são baseados no tipo de interacção que o fornecedor da rede tem na ligação:

- Numa abordagem orientada ao cliente, a informação é transportada transparentemente, sendo da responsabilidade dos clientes a criação, manutenção e posterior destruição dos túneis entre eles. O cliente que pretende enviar informação, encapsula a informação e envia-a para a rede sobre o túnel criado. Quando a cápsula atinge o outro extremo, a informação contida na cápsula é extraída e o pacote é passado para a rede do receptor.

- Numa abordagem centrada no fornecedor de serviço, a rede está responsável pela configuração e manutenção do *VPN*. Podendo até ser um recurso intermédio, controlado pelo fornecedor o mediador da comunicação, criando os túneis entre os extremos.

Túnel *IP-IP*

Um túnel *IP-IP* permite exemplificar o funcionamento de um *link* virtual de nível 3. Um túnel não é mais que um acordo de ligação entre dois pontos, que registam o início e o fim de um segmento controlado, um extremo que encapsula a informação a ser enviada e um outro onde a informação é desencapsulada. Sumariamente, são *links* ponto a ponto onde o envio dos dados é conduzido pela infra-estrutura numa política *best-effort*. Esta técnica descrita em [29] permite que dois pontos inicialmente não contactáveis troquem informação.

Os dados ao serem enviados para o destino são encapsulados, num outro datagrama *IP* que funcionará como uma cápsula. O cabeçalho do pacote a ser transportado é mantido pelo encapsulador. O campo *Source* e *Destination* do cabeçalho da cápsula identificam os extremos do túnel, enquanto que os do interior identificam o emissor e receptor do datagrama tal como descrito na figura 2.11.

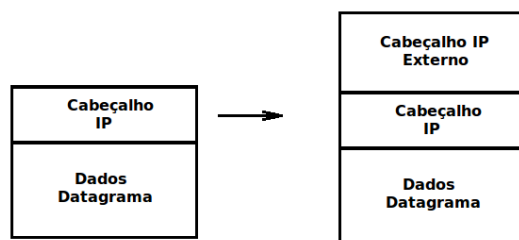


Figura 2.11: Encapsulamento das informações a enviar

Este mecanismo, ainda que simples, serve o propósito de troca de informação entre *hosts* inicialmente incontactáveis, recorrendo à capacidade do *IP* para suportar vários serviços, identificando-os com protocolos distintos. A cápsula de transporte é marcada com o protocolo 4, permitindo-a circular na rede até que atinja o outro extremo da ligação.

IPsec

Uma solução para as *VPN* de nível 3 que permite a inclusão de segurança é o *IPsec*. O *IPsec* proporciona um conjunto de protocolos que permitem controlar a integridade e confidencialidade das comunicações numa rede *IP*[30]. A combinação destes protocolos permite que se estabeleça um canal de comunicação seguro entre dois extremos, como são exemplo, o *ESP*¹ e o *AH*²[31].

ESP - Encapsulating Security Payload

O protocolo *ESP* permite dois modos de funcionamento, modo transporte e modo de túnel. Em modo túnel, o *ESP* cria um novo cabeçalho *IP* para cada pacote enviado, colocando no campo destino e origem as duas extremidades do túnel *ESP*, assegurando a integridade e confidencialidade do pacote de dados.

Em modo de transporte, o *ESP* usa o cabeçalho *IP* original, em vez de criar um novo, cifrando apenas os dados a serem enviados, e não os cabeçalhos *IP*.

AH - Authentication Header

O *AH* é um dos protocolos utilizados na arquitectura *IPsec* e oferece aos seus utilizadores integridade nos pacotes trocados, facultando um mecanismo de autenticação que permite bloquear ataques por repetição. Nas primeiras versões do *IPsec*, o *ESP* não suportava autenticação, apenas a encriptação. Por esta razão, o *AH* e o *ESP* eram normalmente utilizados em conjunto de forma a fornecer confidencialidade e integridade nas comunicações. Posteriormente foi adicionado ao *ESP* a capacidade de autenticação, o que fez com que o *AH* começasse a ser menos usado, desaparecendo de algumas distribuições do *IPsec*.

2.7.2.2 Infra-estrutura virtual de nível 2

Uma *VPN* de nível 3 está projectada de forma a fornecer conectividade entre sistemas terminais que pretendam trocar, a maior parte das vezes, pacotes *IP*. Contudo, esta tecnologia continua a limitar o teste e desenvolvimento de novos protocolos ou aplicações onde o nível de virtualização necessita de ser baixado para a camada de ligação. É aqui que as *VPN* de nível 2 se

¹Encapsulating Security Payload

²Authentication Header

enquadram e onde se coloca o nível de virtualização pretendido pela dissertação. Um sistema que permite a troca de tramas de nível 2 sobre uma rede de suporte que comunica sobre o nível 3.

As *VPN* de nível 2 são ligações entre dois ou mais *hosts* que permitem a troca de tramas de nível 2[32]. Este tipo de *VPN* são implementados sobre protocolos de camadas superiores, como *IP* ou *MPLS (Multiprotocol Label Switching)*. Permitindo que se simule uma conexão utilizando protocolos conhecidos sobre todo o percurso da informação.

A vantagem da utilização deste método é a capacidade de poder ser utilizado qualquer protocolo de nível 2 na comunicação entre extremos enquanto que o encaminhamento se realiza com protocolos conhecidos. É possível os extremos utilizarem *ethernet*, *frame-relay* ou *ATM* para a comunicação e a informação percorrer o seu percurso recorrendo ao *MPLS* ou *IP*. Esta tecnologia permite a unificação da estrutura de rede, permitindo a migração de protocolos de nível 2 para uma abordagem de comunicação sobre *IP* ou *MPLS*, ligando redes onde os protocolos de comunicação são distintos. A utilização deste tipo de redes, permite uma melhoria na capacidade de desenvolvimento de novos protocolos, facultando a evolução e amadurecimento de tecnologias que não poderia ser realizado na rede pública.

A separação entre a rede de suporte e os extremos permite uma maior simplicidade, os extremos comunicam entre si, transparentemente, utilizando tramas de nível 2 enquanto que a rede que faz o encaminhamento utiliza um protocolo de nível 3[32]. Os nós extremos realizam o encaminhamento até à camada de ligação, enquanto que a rede de suporte trata até ao nível de rede.

Capítulo 3

Soluções para construção de redes virtuais

Neste capítulo serão descritas algumas das técnicas usadas em projectos onde o problema é similar ao apresentado nesta dissertação. Serão apresentados 5 projectos, que abordam soluções distintas para a construção de redes virtuais.

O *X-bone*[3, 33], *Vine*¹[4], o trabalho desenvolvido pela PT inovação e Universidade de Aveiro nesta área[6, 34, 35], o *Violin*²[5] e o projecto *4WARD*[1] permitem analisar o futuro desta tecnologia, qual o rumo que deve tomar e a evolução que pode ter.

As redes virtuais são uma tecnologia com a qual existe contacto diário, sem que muitas vezes isso seja apercebido, *VPN*, *P2P*, *VOIP*, são alguns dos exemplos das tecnologias, que permitem a utilização da rede de suporte sobre uma camada de transporte.

Esta camada de comunicação permite a utilização de recursos a pedido, onde o *host* possui o conhecimento que está a usar uma tecnologia sobre a rede física. A rede virtual ultrapassa este conceito, apresentando a rede como um serviço que funciona sobre a rede de suporte, construído por nós, *links* e interfaces virtuais que utilizam os recursos de forma concorrente, transparentemente ao utilizador.

¹*Virtual Network Architecture for Grid Computing*

²*Virtual Internetworking on overlay Infrastructure*

3.1 *X-bone*

O *X-bone*[33] é um projecto desenvolvido na Universidade de *South Carolina* que pode ser compreendido como um organizador de *overlays* de rede sobre *IP*, facultando a descoberta, configuração e manutenção dos recursos da rede física.

O utilização de *overlays* não é recente, sistemas que necessitam de ligações seguras sob uma ligação de rede não confiável, trocas de informação com foco apenas nos extremos como *P2P*, são algumas das tecnologias que permitem uma abstracção da rede física, permitindo manter o foco apenas na comunicação entre os extremos. Este tipo de comunicação necessita, na maior parte dos casos de alterações profundas tanto nas aplicações como nos sistemas operativos que as suportam.

Ao contrário do que acontece nas redes convencionais, o *X-bone* permite uma abordagem completamente transparente na criação e manutenção da rede *overlay*, oferecendo a possibilidade do utilizador apenas apresentar a forma como pretende a rede, fazendo com que a aplicação procure os *nós* aptos a participar no *overlay*, os configure e permita a manutenção e observação dos seus recursos. A abordagem utilizada no *X-bone* permite que se crie um *overlay* de rede sem que seja necessária a alteração dos componentes físicos que a suportam. Para atingir este fim o *X-bone* baseia-se apenas na troca de informação em pacotes *IP*, recorrendo ao encapsulamento[36]. O encapsulamento pode ser visto como a base da aplicação uma vez que os *links* virtuais são baseados nesta tecnologia e permitem uma utilização recursiva, fazendo com que um *link* físico possa ser atravessado várias vezes na mesma rede virtual, utilizando vários níveis de encapsulamento.

Segundo o autor[3], um *overlay* de rede é uma rede isolada, construída sob uma rede existente e é composta por *hosts*, *routers* e túneis entre os extremos da ligação. A forma como estes túneis são vistos pela rede moldará o *overlay*, definindo a sua forma, tal como apresentado na figura 3.1.

Um *overlay* virtual deve ser regido por 3 princípios fundamentais:

1. Deve ser contido, não permitindo que os conteúdos trocados sejam contextualizados externamente;

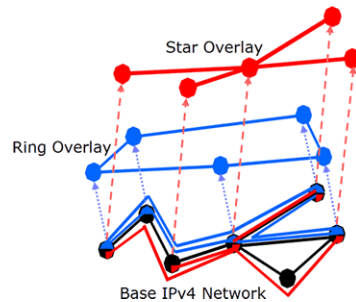


Figura 3.1: Imagem retirada de [3] que ilustra a base lógica do funcionamento do *X-bone*

2. Deve ser consciente dos recursos que está a utilizar, permitindo testar novos sistemas utilizando redes com capacidades e características diferentes;
3. Deve ser capaz de se abstrair da rede física, fornecendo uma infra-estrutura sem que seja conhecida, para os nós virtuais, a rede que a suporta.

A criação de um *overlay*, usando os métodos convencionais, necessita de uma intervenção manual, em cada um dos pontos de alteração ou manutenção da rede: A topologia pretendida deve ser mapeada nos componentes físicos da rede e os parâmetros que a regem devem ser determinados manualmente e alterados de acordo com o que é pretendido. Este processo pode-se arrastar, podendo demorar dias até que um *overlay* de rede esteja configurado. Após o *overlay* estar disposto e configurado, existe ainda a possibilidade de falha, podendo a falta de manutenção e monitorização levar a que erros na configuração possam por em causa todo o sistema. O *X-bone* apresenta uma abordagem para combater esta dificuldade, propondo uma aplicação que apenas necessita da configuração inicial que pode ser realizada utilizando uma interface gráfica ou uma *API* que permite a criação do *overlay*.

Partilha de recursos

O *X-bone* aborda os vários constituintes da rede como recursos a ser usados pelos diferentes *overlays* criados. Os recursos serão usados em concordância com as suas capacidades. *Routers*, *links* e *hosts*, serão utilizadas em diversas camadas e de forma concorrente.

Os *links* virtuais são criados recorrendo a encaminhamento pré-definido ou túneis pré-configurados. O primeiro requer que a informação do caminho que o datagrama deve tomar seja pré-calculada

e introduzida no cabeçalho, esta abordagem, *source routing*, é limitada pela quantidade de informação que pode ser colocada. A segunda abordagem requer que o datagrama seja encapsulado num novo cabeçalho. Esta abordagem não limita o comprimento do *link* virtual pelos saltos que deve dar na rede. Introduzindo, em contra-partida, a necessidade de desencapsulamento da informação no destino, gerando overhead.

Os *routers* usados pelas redes *overlay* são partilhados entre as várias instâncias criada. Estes são compostos por interfaces, tabelas de encaminhamento e algoritmos de selecção de rotas. As interfaces terão o funcionamento das interfaces de um *host*, usadas para o envio dos datagramas. As tabelas de encaminhamento permitem guardar a informação utilizada na fase do envio enquanto que o algoritmo de envio permite computar a forma como a informação recebida no pacote *IP* é usada para escolher a melhor interface para o envio da informação, alterando caso seja necessário, as informações da tabela de encaminhamento.

A tarefa para qual os *routers* são concebidos já é por si partilhada, e por esta razão, não existe necessidade de controlar os recursos do *router* de forma especial, existindo apenas a necessidade de garantir que não existe uma utilização repetida de endereços de rede. A utilização de *links* de suporte partilhados com endereços de interfaces locais, acarreta a necessidade de endereçamento virtual, de forma a saber qual a máquina virtual que recebe os dados enviados.

Arquitectura

O *X-bone* é um sistema distribuído, que apresenta uma arquitectura baseada em três componentes principais: a interface de controlo da aplicação (*GUI*), o *overlay manager* (*OM*) e o *resource daemon* (*RD*), todos estes processos terão uma importância no algoritmo e estarão interligados de forma a criar e manter o *overlay*. A forma como os processos se interligam está representada na figura 3.2.

O *OM* permite a criação do *overlay* em fases e está integrado na aplicação que fornece o *GUI*, o conjunto destes dois processos é denominado *xd*. O *xd* recorre às capacidades do *multicast* para descobrir recursos dispersos na rede e a *Sockets TCP* para configuração e monitorização dos *Resource Daemon* encontrados. Os pedidos de descoberta de recursos são enviados utilizando um datagrama *IP* e são repetidos com um *TTL* cada vez mais alto, até que um número

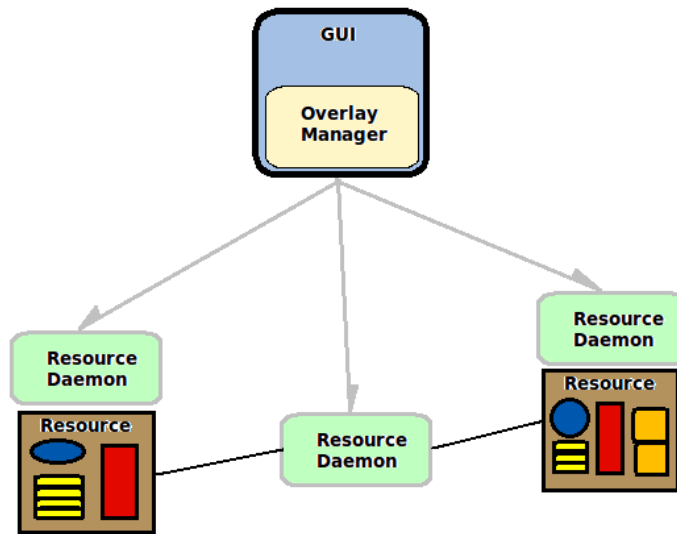


Figura 3.2: Imagem adaptada de [3], que permite compreender a arquitectura do projecto *X-bone*

de nós definido seja encontrado, ou até que o *TTL* atinja o valor limite. O pedido é enviado publicamente, mas encontra-se assinado/encriptado de forma a que apenas um conjunto restrito de componentes possa responder. Os *hosts* que recebem os pedidos decidirão se pretendem participar na rede, baseando-se nos recursos que possuem disponíveis.

Os *RD* são processos que executam nos recursos que podem ser requisitados para a rede, tal como *routers*, *links* e *hosts*. Estes terão a tarefa de monitorizar o estado do recurso e a forma como este é partilhado por todos os *overlays* criados.

O *OM* irá então seleccionar um conjunto da lista de nós que responderam, criando uma conexão *TCP/SSL* com cada *RD*. A comunicação com os *RD* permitirá a criação do novo *overlay* de rede, configurando os recursos associados aos *RD* a cada um deles. São configurados os túneis entre recursos, os mecanismos de encapsulamento e desencapsulamento e são alterados os endereços das interfaces virtuais e a forma como vão comunicar. Os *RD* neste ponto também terão uma grande importância na medida em que serão os gestores do espaço de endereçamento.

O *OM* vai computar as informações de configuração, como os endereços dos extremos dos túneis e entradas nas tabelas de encaminhamento, enviando-o para cada *RD*. Uma vez que o *overlay* é criado, as conexões *TCP/SSL* são destruídas. O esquema deste algoritmo pode ser consultado na figura 3.3.

O protocolo de controlo usado no *X-bone* está construído sobre um sistema de dois endereços

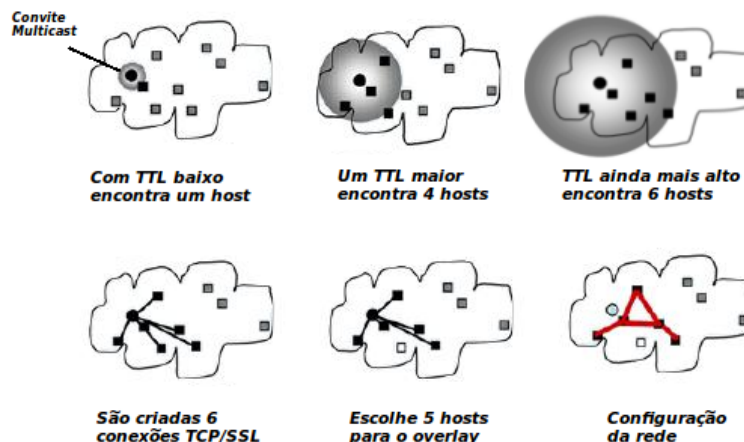


Figura 3.3: Imagem adaptada de [3] que esquematiza os passos realizados pela procura de nós para o *overlay*

multicast, o primeiro é utilizado para a troca de mensagem de controlo, onde será distribuído o endereço que permitirá a troca de informação entre os nós do *overlay*. Cada *overlay* terá o seu próprio endereço *multicast* de controlo, que será anunciado usando o primeiro. Como esquematizado na imagem 3.4.

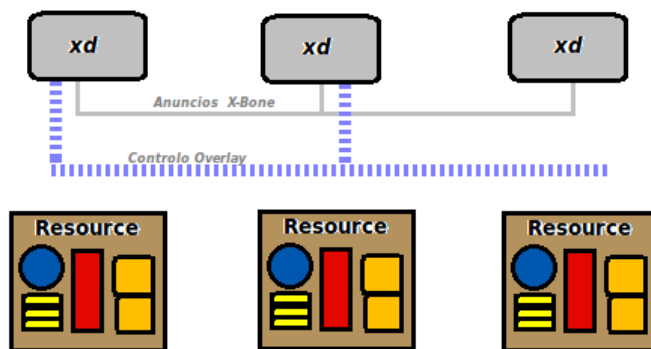


Figura 3.4: Imagem adaptada de [3], que permite verificar de que forma os nós comunicam os *overlay* criados

Os canais são usados para a descoberta de recursos. Os pedidos enviados pelo canal *multicast* principal são enviados com um *TTL* limitado de forma a bloquear a divulgação da rede de forma descontrolada. Caso o pedido inicial não obtenha respostas suficientes, o *TTL* é incrementado, e uma nova mensagem é enviada. Isto ocorre até que os recursos necessários sejam

obtidos. Quando são obtidos recursos suficientes é criado o canal de controlo do *overlay*, cujo alcance é limitado aos *hosts* que responderam ao pedido inicial.

O *X-bone* utiliza para a comunicação entre os *hosts*, um mecanismo de encapsulamento, para cada nível de *overlay*. Criando um total de 3 níveis de cabeçalhos *IP* para cada *overlay*. O cabeçalho mais interno representa o *link* do *overlay* e permite indicar os seus extremos. O segundo funcionará como a camada de ligação do *overlay*, e indicará os pontos do túnel sobre o qual o pacote está a ser enviado. O último cabeçalho, indica os extremos do túnel na rede de suporte que poderá também ser um *overlay*, criando uma rede recursiva. Os endereços utilizados pertencem a um conjunto privado, separado da rede de suporte. O datagrama que é enviado pela aplicação está apresentado na figura 3.5.

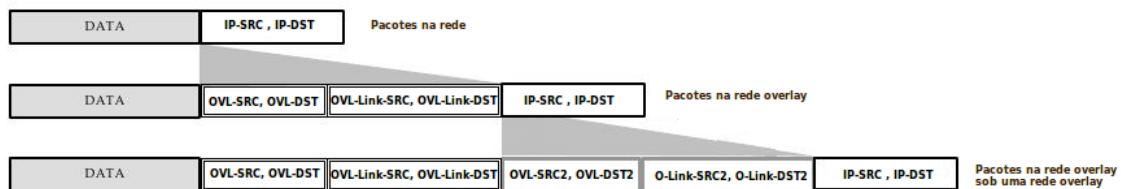


Figura 3.5: Imagem adaptada de [3], que esquematiza os cabeçalhos introduzidos na informação a enviar

O *X-bone* permite a criação de *links* virtuais com troca de informação segura utilizando o *IPsec*, proporcionando segurança no cabeçalho do *link* do *overlay*. Este mecanismo permite ser independente do *IPsec* da rede, que pode até não estar implementado, ou do *IPsec* da aplicação que pode não estar acessível. Esta abordagem permite, caso seja necessário e esteja disponível, a implementação de *IPsec* em todos os cabeçalhos utilizados no envio do pacote *IP* tal como demonstrado na imagem 3.6.

O *X-bone* oferece também uma robustez assinalável na criação dos *overlays* possuindo uma capacidade de detecção e recuperação de falhas. Para cada acção executada, é possível retornar ao ponto prévio, uma vez que todas as alterações de estado que ocorrem, são guardadas num ficheiro de recuperação. O *OM* envia também *pings* periódicos, que permitem atestar o estado de conectividade dos diversos *RD*.

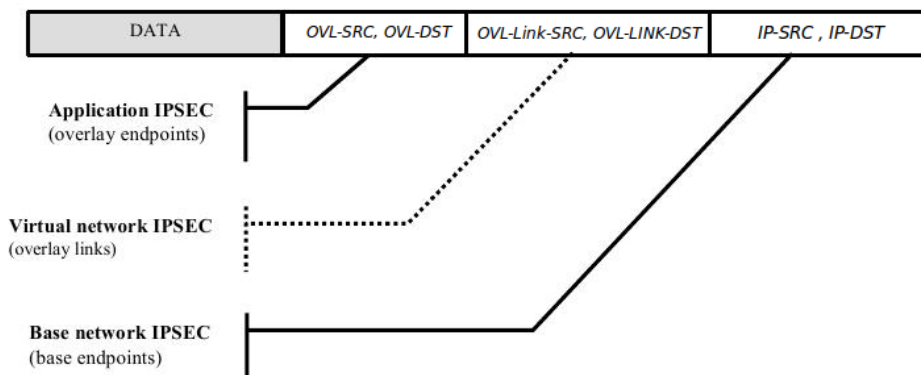


Figura 3.6: Imagem adaptada de [3], que permite verificar o *IPsec* presente em cada um dos cabeçalhos

3.2 *Vine*

O *Vine*[4] é um projecto desenvolvido na Universidade da Flórida[4] e pretende oferecer uma abordagem de criação de rede virtual que permite conectividade simétrica entre os recursos da rede.

Um dos pontos fundamentais numa arquitectura de rede distribuída é a conectividade entre os recursos que nela estão contidos. Contudo a comunicação na *Internet* é assimétrica no sentido em que só ocorre quando parte de um dos extremos. O projecto propõe uma solução para esta dificuldade, permitindo conectar recursos que se encontram em domínios lógicos distintos.

Existem tecnologias na arquitectura da *Internet* que contribuem para que esta assimetria seja reforçada, tais como redes privadas e *firewalls*. Um nó numa rede privada, é capaz de ultrapassar facilmente uma *Network Address Translation (NAT)* ou *proxy* de forma a conectar-se a um endereço na rede pública, contudo o contrário não é verdadeiro, os servidores com endereço público não podem comunicar com nós dentro de uma rede privada, sem que esta seja alterada.

A limitação de conectividade provém também do uso de *firewalls*. Este mecanismo, criado inicialmente para impedir o tráfego mal intencionado de atingir recursos, filtra muitas das vezes o tráfego de rede normal tornando a comunicação entre os *hosts* impossível.

A arquitectura *Vine*, de forma a permitir a criação de uma rede virtual simétrica, transparente e portátil, pauta-se pelo cumprimento dos seguintes requisitos:

- Comunicação simétrica entre entidades, ultrapassando as dificuldades impostas por *NAT* e *proxies*;
- Serviços inalterados, os serviços que funcionam na rede devem manter-se inalterados com o aumento da rede *Vine*;
- Fácil configuração dos recursos que pretendam agregar-se ao *overlay*;
- Não devem existir impactos nas políticas de segurança implementadas na rede;
- Disponibilização de mecanismos que permitam automatizar a criação do *overlay* de rede;
- Possibilidade das aplicações serem executadas sem que para isso sejam alteradas para o *overlay*;
- Não devem existir alterações no sistema operativo nativo;
- A infra-estrutura de suporte ao *overlay* não deve ser alterada;
- Independente da plataforma usada, o sistema operativo não deve ser factor para o funcionamento do *overlay*;
- Deve ser escalável.

Arquitectura

O principal objectivo do *Vine* é a criação de uma rede virtual com capacidade de comunicação bi-direccional entre *hosts*, onde as entidades como *routers* e *links* são virtualizados. Os *routers* normais não são capazes de suportar o tráfego gerado pelo *Vine*, por esta razão cada *overlay* de rede necessita de um *Virtual Router (VR)* que possa encaminhar a informação. Os *hosts* são configurados para reencaminhar o tráfego pelos *VR* e não necessitam de *software* adicional instalado.

Para cada nova rede virtual, é alocado um novo espaço de endereçamento e um novo *VR* que o permitirá gerir. Os *hosts* são configurados com interfaces usadas pelo tráfego gerado pelo *Vine*. Por sua vez a comunicação entre os *VR* processa-se com a ajuda de túneis, que recorrem à infra-estrutura de suporte, criando a arquitectura descrita pela figura 3.7.

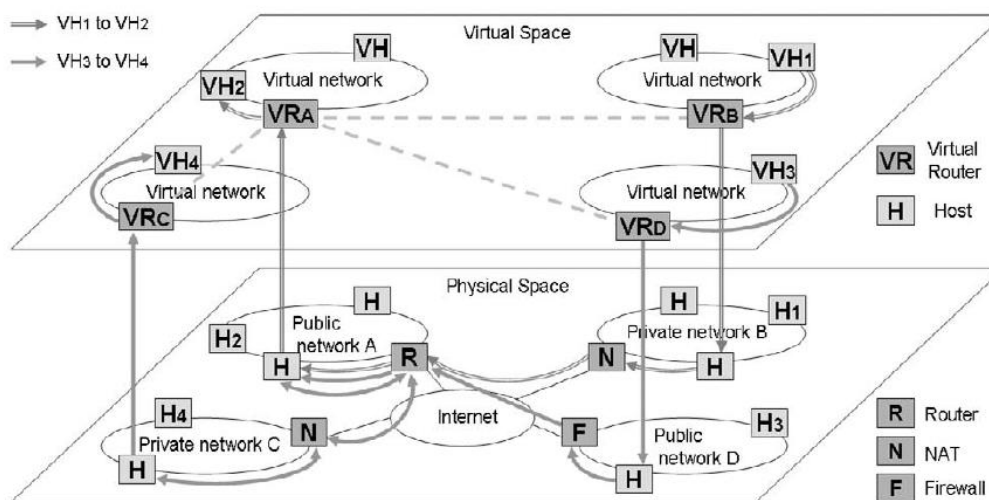


Figura 3.7: Figura retirada de [4], que representa a arquitetura de um *overlay Vine* dividido pelas suas áreas lógicas de endereçamento

Existem várias formas de simular interfaces de rede: A captura dos pacotes antes de estes atingirem a rede; A interceptação das chamadas de sistema de rede, alterando-as; Ou a utilização uma nova implementação de um *Network interface Card (NIC)*, emulando os físicos. A todas estas opções podem ser apontados pontos negativos, que vão desde a necessidade de alteração da *stack* de rede do sistema operativo até à alteração de *drivers* e recompilação de *kernel*. Uma outra opção é o uso de múltiplos endereços em cada *NIC* físico, configurando uma rota estática que permite o encaminhamento dos pacotes *Vine* para os *VR*.

O espaço de endereçamento é pensado de forma a não impedir o normal funcionamento dos *hosts* que pretendam utilizar a rede de suporte. Desta forma, cada *overlay* possuirá um espaço de endereçamento privado que pode ser *10.0.0.0/8*, *172.16.0.0/12* e *192.168.0.0/16*.

Podem ser criados *overlays* em locais de rede privada e onde não existe a possibilidade de configuração de um *VR* público. Nestes casos, o *VR* não é capaz de trocar informação de forma normal. Necessitando de ser o *VR* dentro da *NAT* o iniciador da comunicação, pois logo um pacote é enviado e a comunicação é iniciada, a resposta é normalmente possível. O *Vine* aproveita este facto para permitir a troca de informação, recorrendo a um *VR* intermédio, conectado à rede física, que serve como um armazenador de informação, onde os pacotes que são destinados para os *VR* incontactáveis são guardados. Os *VR* limitados iniciam uma conexão com os *VR* armazenadores e toda a informação que possuem é reenviada para o *VR* inicialmente incontactável. O *VR* apenas necessita de funcionar como armazenador quando esta conexão é terminada.

Os *VR* têm um papel muito importante na arquitectura *Vine*, pois possibilitam a comunicação entre os *overlays* criados, representando um ponto de entrada para cada *overlay*, por esta razão necessitam de ter acesso à rede partilhada mas não necessitam de possuir um endereço público.

Os *VR* gerem um par de tabelas de encaminhamento, uma *local network description table (LNDT)*, e um *global network description table (GNDT)*. Para cada *overlay Vine* activo o *LNDT* guarda as informações relativas à área local e o *GNDT* armazena as informações da estrutura da rede virtual, onde cada entrada identifica um *VR*.

Cada pacote gerado por um *host* é enviado para um *VR*, que verificará, utilizando o campo destino do pacote recebido, o *Virtual network ID* na tabela *LNDT* de forma a saber qual a tabela *GNDT* a consultar, que permitirá indicar o *VR* para onde devem ser endereçados os dados, e utilizando um túnel, são enviados para o *VR* correcto, que entregará a informação ao nó correspondente.

O protocolo de rede está centrado nos *VR* e nas capacidades de reconhecimento dos *hosts* nas tabelas, como é possível verificar pelo exemplo citado de [4] e apresentado na figura 3.8:

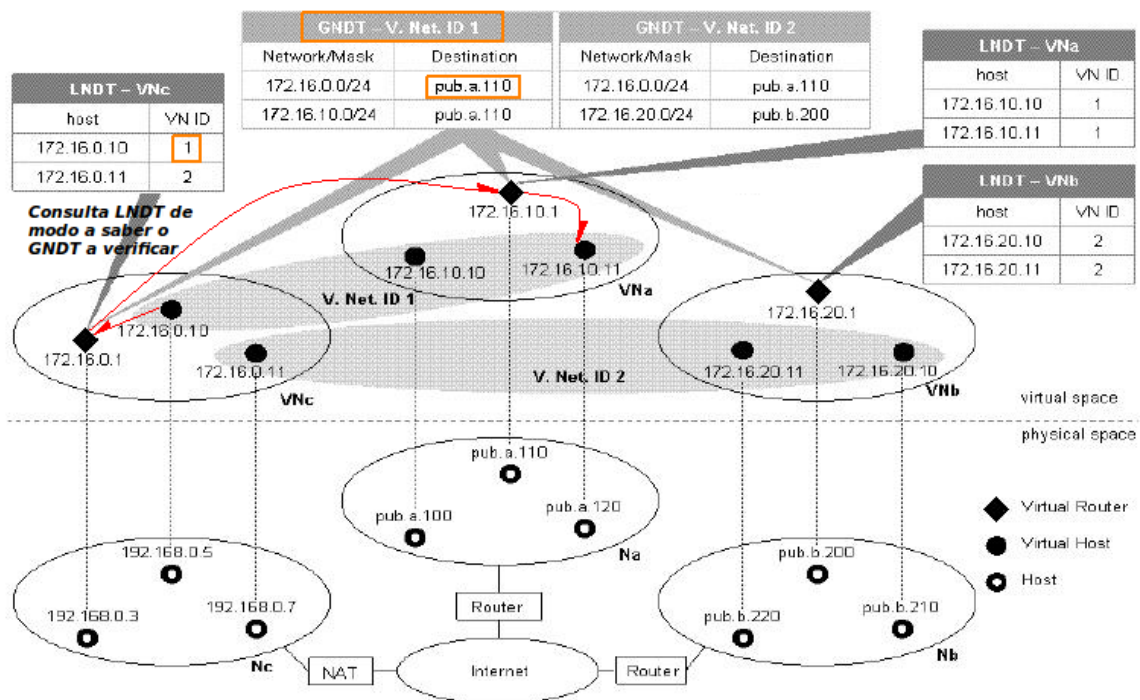


Figura 3.8: Imagem adaptada de [4] que permite compreender a forma como o *Vine* comunica

A figura 3.8 exemplifica a forma como o *Vine* permite a comunicação entre os *overlays* criados. A rede virtual 1 (*V.Net.ID1*) conecta os *hosts* participantes que pertencem às redes *VNa*

e *VNc*, enquanto que a rede virtual 2 (*V.Net.ID2*) conecta os *hosts* nas sub-redes *VNb* e *VNc*. Todos os *VR* (representados por losangos na imagem), recebem uma cópia do *GNDT* contendo informação sobre todas as redes *Vine* existentes, ou seja recebem a tabela que corresponde à *V.Net.ID1* e *V.Net.ID2*. Quando um pacote é enviado de *172.16.0.10* para *172.16.10.11*, é consultada a *LNDT-VNc* que informa que o *host* de origem é um membro da rede virtual 1. Com esta informação, o *VR* da rede local, (*172.16.0.1*) consulta o *GNDT-V.Net.ID1* e envia o pacote para *pub.a.110*, que entrega a informação.

Os pacotes destinados a *172.16.0.0/24*, cujo *VR* se encontra numa *NAT*, são guardados pelo *pub.a.110*. O *VR* da rede *VNc*, abre uma conexão *TCP* para o *VR* armazenador que o permitirá receber todas as mensagens guardadas.

Implementação

O protótipo de *Virtual Router* foi implementado em *Java*, onde a captura dos pacotes de rede é realizada pela infra-estrutura *netfilter* e o envio dos pacotes, após serem tratados, recorre às funções do *libnet*[37]. Os *hosts* que pretendem participar na rede *Vine*, não necessitam de instalar qualquer tipo de *software*, sendo apenas necessário uma intervenção administrativa para configurar a interface virtual e as respectivas rotas.

Existem três pontos que introduzem *overhead* na abordagem *Vine*: A inserção de *VR* na rede local, aumentando em dois o número de saltos até um pacote atingir o seu destino; O software utilizado em cada um dos *VR* necessitará de tratar os pacotes recebidos; O *overhead* introduzido pelo túnel no encapsulamento e posterior desencapsulamento das informações.

3.3 *Violin*

O *Violin*[5] é um projecto desenvolvido na *Purdue University* que propõe a criação de uma rede virtual sobre uma infra-estrutura virtual já criada como é exemplo o *PlanetLab*[10].

Um *Violin*, consiste em *routers*, *switches* e *hosts* virtuais que executam nos nós do *overlay* de primeiro nível.

A principal particularidade do *Violin* é o isolamento que oferece a cada *overlay*, disponibilizando-lhe os seus próprios endereços e confinando todas as comunicações. Os *Violin*³ são um conjunto de aplicações distribuídas que permitem uma elevada flexibilidade na alteração no desenho da rede ou nas características desta. Esta capacidade de confinar o tráfego numa arquitectura controlada, permite testar serviços que não podem ainda ser disponibilizados na rede pública.

Arquitectura

O *Violin* pretende ser uma rede virtual sobre uma rede virtual. O primeiro nível de rede virtual é criado sobre uma rede de suporte e o segundo sobre o primeiro, facultando uma estrutura hierárquica de virtualização, este conceito é apresentado na imagem 3.9.

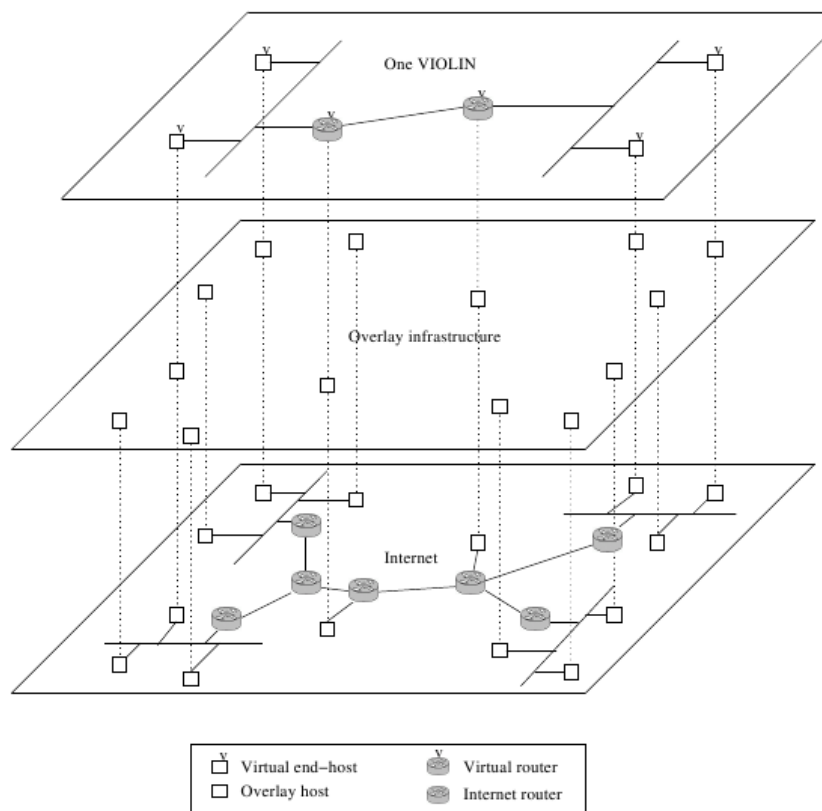


Figura 3.9: Imagem retirada de [5], que pretende esquematizar a infra-estrutura *Violin*, com 3 níveis de rede distintos

³O autor, utiliza o termo *Violin* para o projecto realizado e para as redes virtuais criadas.

As entidades *Violin*, executam em *hosts* da rede virtual e simulam o funcionamento dos congêneres físicos. Estes podem ter o comportamento de um *host* (*vHost*), representado por uma máquina virtual, cada nó pode executar vários *vHosts*, que podem pertencer a diferentes *Violin*. Podem comportar-se como uma *LAN* virtual (*vLAN*), criando um *switch* virtual, que permite conectar múltiplos *vHosts*. É também possível a virtualização de um *router* virtual (*vRouter*), que é representado, por uma máquina virtual com várias interfaces virtuais, podendo comunicar com várias *vLANs*.

Cada *Violin* é uma rede isolada e por este motivo os endereços de dois *Violin* podem facilmente ser sobrepostos, ou até utilizar diferentes formas de endereçamento sem se prejudicarem. Cada instância da rede *Violin*, possui um administrador, que tem a capacidade de alterar as características do *Violin*.

Este nível de virtualização vem apresentar um meio controlado, seguro e dinâmico, capaz de testar aplicações distribuídas facilmente e sem necessidade de recorrer directamente à rede física onde a informação gerada pelas aplicações poderia não ser suportada.

Como todos os participantes de uma *Violin* são aplicações, a reconfiguração das entidades presentes na rede é simples. É possível adicionar, remover, ou migrar os nós dinamicamente, introduzindo alterações nos *vHosts* e nos *vRouters* de acordo com o teste que se pretende realizar.

As entidades *Violin* são implementadas como máquinas virtuais em *hosts* no primeiro nível do *overlay*, e utilizam o *User Mode Linux (UML)*[38] como abordagem para a virtualização.

O *UML* apenas permite a troca de informação entre o *host* virtual e o *host* físico, mantendo a comunicação apenas local. Para isto é introduzido no *UML* uma extensão ao seu comportamento, permitindo a troca de informação entre *hosts* virtuais recorrendo a túneis de transporte.

As entidades *Violin* são configuradas de forma específica para a tarefa que lhes é proposta. Um *vSwitch* é criado para cada *vLAN* e é responsável pelo envio dos pacotes ao nível da camada de ligação, simulando o nível 2 do modelo *OSI*, este é emulado por um *daemon UDP* no *host* que permite receber o tráfego da rede e armazena-lo, reenvia-lo ou rejeita-lo.

Um *router* virtual possui uma implementação muito similar a um *vHost*, a principal diferença reside no facto do *vRouter* possuir capacidades de endereçamento e tratamento dos pacotes recebidos, necessitando, por esta razão, de um suporte ao nível do *kernel* que permita o reencaaminhamento dos pacotes e o acesso a chamadas ao sistema de modo a configurar as interfaces para a comunicação. O *vRouter* implementa um mecanismo de encaminhamento baseado em

zebra[39]. Uma outra forma de implementação do *router* seria a utilização do sistema *Click*[40].

3.4 Rede Virtual PT inovação

O projecto desenvolvido pela *PT Inovação* em colaboração com a universidade de Aveiro[6, 35] propõe uma abordagem de criação, descoberta, monitorização e manutenção de redes virtuais, baseando-se no projecto *4WARD*[1]. Focando posteriormente a investigação, num algoritmo de descoberta de topologias de rede, tanto físicas como virtuais. Recorrendo para isto ao *multicast* de nível 2 e ao algoritmo *spanning tree* como forma de reduzir o *overhead* na troca de informações entre os nós.

A virtualização de redes pode ser vista como uma tecnologia impulsionadora, que vem permitir que sejam testados novos protocolos e arquitecturas, tendo sido vista nos últimos anos com grande potencial pelas operadoras, permitindo oferecer aos clientes redes feitas à medida.

A virtualização de redes oferece vários pontos de desenvolvimento, apresentando vários desafios que devem ser solucionados de forma a criar a infra-estrutura. Nomeadamente, como devem os recursos de rede como *links*, *hosts* e *routers* ser virtualizados e de que forma o desempenho pode ser equiparado aos equipamentos reais. Existem projectos que abordam este tema[39, 40] mas nenhum é capaz de atingir o desempenho observado em equipamentos reais. O outro desafio prende-se com a monitorização e manutenção das redes virtuais construídas. Este projecto apresenta uma abordagem para a criação e manutenção de redes virtuais assim com uma proposta de um algoritmo de descoberta de nós, virtuais e físicos que se baseia no *multicast* e no conceito de vizinhança para otimizar o processo de troca de mensagens entre nós. A abordagem aproveita também o algoritmo *spanning tree* para reduzir o número de mensagens trocadas.

Arquitectura

A arquitectura do projecto é composta por três módulos, o agente, o *manager* e o centro de controlo, estes realizarão tarefas complementares, e relacionam-se segundo arquitectura descrita na

imagem 3.10.

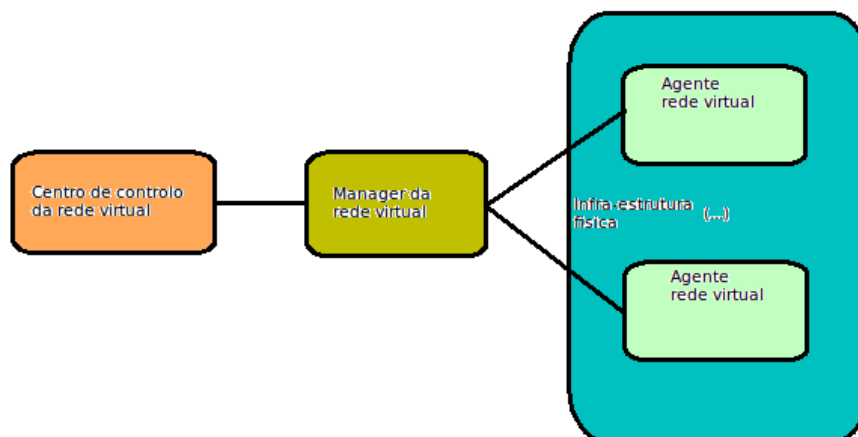


Figura 3.10: Imagem adaptada de [6], que permite compreender de que forma se relacionam os módulos

O centro de controlo da rede virtual é a interface que permite ao utilizador realizar alterações na rede e analisar os dados obtidos. O *manager* da rede virtual possui a capacidade de agregar a informação dos agentes, enviando comandos e agregando a informação recebida de forma a construir uma visão da rede de suporte e virtual, mantendo toda esta informação actualizada e fornecendo-a ao centro de controlo. O agente executará em cada um dos nós da rede, enviando as informações acerca do estado dos recursos locais ao *manager*.

A arquitectura proposta permite apresentar uma forma de descoberta de recursos e topologias. Apresentando aos administradores a topologia da rede física e virtual. Isto é possível pois os nós trocam informação de forma a descobrirem quem são os seus vizinhos. Os agentes periodicamente actualizam o estado dos recursos enviando a informação ao *manager*, possibilitando a identificação de diversos pontos de falha ou recursos onde a carga computacional é elevada. Cada recurso de rede possui a informação sobre a utilização da memória e disco, a carga do *CPU*, o estado dos *links* e interfaces e o número de máquinas virtuais dos seus vizinhos.

Mapeamento de nós virtuais

O mapeamento dos nós virtuais, está baseado nos conceitos de *stress* nos nós e nos *links*, apresentado em [35], permitindo levar em consideração o efeito do *CPU* e memória na carga de cada um dos nós. O mesmo acontece com a carga sobre os *links* virtuais, que em [41] apenas considera o número de *links* virtuais criados sobre um *link* físico, sem ter em conta a largura de banda que estes reservam.

O algoritmo proposto determina inicialmente a carga sobre os nós e os *links*. Os que apresentam menor carga são mais propensos a aceitar a criação de novos recursos virtuais.

São definidos $k_j = 0 \dots (L_{V_j} - 1)$ e $i = 0 \dots (L_S - 1)$ onde k_j é o número de um dado *link* virtual que pertence à rede virtual j . L_{V_j} é o número de *links* virtuais na mesma rede virtual, i representa um dos *links* físicos e L_S é o número de *links* da rede de suporte[35].

É inicialmente definido que o stress de um *link* virtual, $(S_{L_{V_j}})$ do *link* k_j , que pertence à rede virtual j é igual à largura de banda que tem alocada. Ou seja: $S_{L_{V_j}} = BW(k_j)$

Após todas as cargas dos *links* virtuais serem calculadas, é calculada a carga do *link* físico, $S_{L_S}(i)$ é a carga do i *link* físico, sendo definido da seguinte forma:

$$S_{L_S}(i) = \sum_j^{N_V} \sum_k^{L_{V_j}} ((S_{L_{V_j}}(k_j) | k_j \supseteq i))$$

Onde N_V é o número de redes virtuais criadas.

O próximo passo consiste na determinação da carga em cada um dos nós (S_N), que corresponde à combinação dos recursos disponíveis da rede de suporte, máquinas virtuais activas, *RAM* disponível (*MB*), número de *CPU*, a frequência e a carga que estes possuem.

Com estes valores é calculada a função Λ que permite determinar se um nó virtual, pertencente à j rede virtual, está activo e a executar no i nó.

$$\Lambda(n_j, i) = \begin{cases} 1 & \text{se } n_j \supseteq i \wedge n_j \text{ está activo} \\ 0 & \text{se não se verificar} \end{cases}$$

Assim, a carga sobre o i nó terá o valor:

$$S_{N_i} = \frac{\sum_j^{N_V} \sum_n^{N_{V_j}} \Lambda(n_j, i)}{\delta + \text{FreeRAM} \times \text{CPUFreq} \times (N \times \text{CPU} - \text{Load})}$$

Onde δ é representado por uma constante que evita a divisão por zero. N_{V_j} é o número de nós virtuais, na rede virtual j .

Os valores obtidos permitirão a determinação dos candidatos a suportar a criação do nó virtual. Para isto é seleccionado um conjunto de nós, baseado na localização, número de *CPU*, frequência, memória livre e espaço no disco disponível. Sobre o conjunto é executado o algoritmo de selecção que ordena os nós a mapear pelo número de candidatos, criando em primeiro lugar nos que possuem mais recursos disponíveis.

O algoritmo termina com o mapeamento do último nó virtual e selecção da rota entre eles. Para cada possível candidato v , um algoritmo *CSPF*⁴ calcula a melhor rota para os nós dados como vizinhos, u , utilizando os valores de carga nos *links* obtida anteriormente. O potencial de um nó suportar um nó e *links* virtuais é dado da seguinte forma:

$$\pi(v) = \sum_{u \in V_C} D(v, u) \times S_{N_v}$$

Onde V_C é o conjunto dos candidatos dos nós vizinhos. Após estes valores serem calculados para todos os nós, aquele que possui o valor mais baixo é escolhido. O algoritmo termina quando todos os nós virtuais forem mapeados.

⁴*Constraint Shortest Path First*

Descoberta das topologias

De forma a mapear a rede virtual é necessário o reconhecimento da topologia de suporte, abordando algoritmos que permitam a descoberta, mantendo a carga computacional no *manager* o mais reduzida possível.

O algoritmo de procura está baseado no conceito de vizinhança, onde cada nó conhece exactamente quais são os seus vizinhos e quais os recursos virtuais que estão a ser utilizados. Os agentes ligam-se a um grupo *multicast* pré-definido e trocam mensagens de controlo entre si. O grupo *multicast* criado é apenas local, ou seja possui um *TTL* de 1. Fazendo com que os nós apenas conheçam directamente os seus vizinhos, limitando o envio de informação para nós que não estejam interessados na informação, a agregação da informação em cada um dos agentes possibilita conhecer toda a topologia de rede.

Em cada segmento de rede um dos agentes é escolhido como o *designated router (DR)*, que tem como função passar a informação que descreve a rede a novos nós que entrem no segmento. O *DR* é escolhido, baseando-se no *ID* do agente, um inteiro que é atribuído pelo *manager* quando este inicia. O agente com o *ID* mais baixo na rede é elegido *DR* e terá a responsabilidade de enviar a informação aos recém-chegados ao segmento. Ao iniciar, cada agente é o seu próprio *DR*, após receber uma mensagem com um *ID* mais baixo actualiza a informação sobre o *DR*.

Cada agente que se ligue vai enviar uma mensagem para o grupo *multicast*, indicando a sua actividade. A mensagem é enviada por todas as interfaces e possui a informação da interface que o origina.

A descoberta do *link* virtual é menos clara, isto porque um *link* virtual pode representar vários saltos na rede física. Os agentes trocam entre si a informação sobre dois tipos de recursos, os recursos locais e aqueles que são utilizados pelos vizinhos que utilizam um nó físico como um salto na rede virtual, a figura 3.11 tenta explicar este conceito.

A figura 3.11 permite descrever a forma como os recursos virtuais são anunciados. O recurso virtual *A*, executa no nó físico *a*. Este sabe que o nó virtual *A* possui duas interfaces virtuais conectadas a diferentes redes virtuais, numa das suas interfaces. O nó físico, vai então anunciar *A* através das suas interfaces. As mensagens serão recebidas pelo nó físico *b*, que registará *A* e as suas interfaces como um potencial vizinho. Será então verificado se alguns dos seus recursos virtuais está conectado a *A*, sobre uma das suas interfaces, *eth0* ou *eth1*. Como isso não acontece,

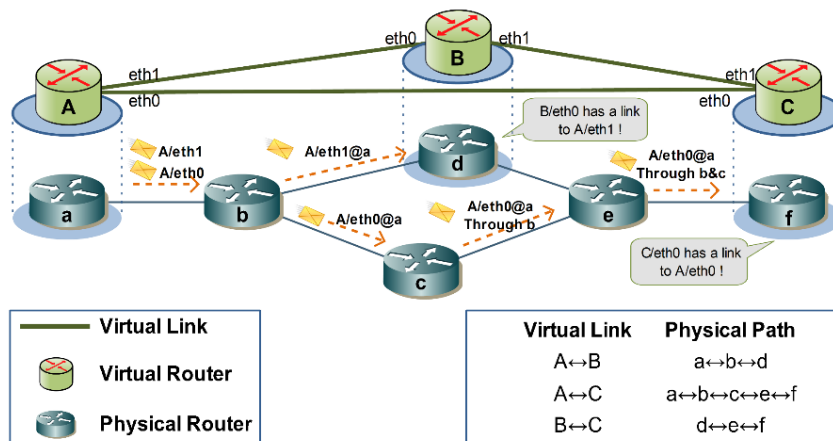


Figura 3.11: Figura retirada de [6] que permite compreender o algoritmo de procura de *links* virtuais

o nó *b* irá verificar se é um salto para algum *link* de *A*, e irá identificar duas possibilidades, uma que liga a interface *eth0* de *A* e outra que permite conectar a interface *eth1*. Após identificar-se como um salto, *b* irá divulgar *A* e as suas interfaces aos links físicos correspondentes. O nó físico *d*, vai receber o anúncio e verificar que *a* possui uma ligação com um dos seus recursos virtuais, *d* fica ciente que o seu recurso *B* está ligado à interface *eth1* de *A* pela interface *eth0*.

Estas informações são repassadas pelos nós físicos, até que a estrutura virtual seja conhecida pelos nós que fornecem recursos virtuais para a rede virtual.

3.5 4WARD

O projecto *4WARD*, é um projecto europeu de rede virtual, que pretende o aumento da competitividade da indústria de comunicação Europeia[1].

As redes de futuro, são criadas de forma a serem rapidamente adaptadas às necessidades presentes e às que podem surgir. O objectivo do projecto *4WARD* é o desenvolvimento de aplicações de forma mais rápida e fácil, guiando a infra-estrutura para uma rede de serviços de comunicação mais organizada e orientada para o utilizador.

O conceito de virtualização, no projecto *4WARD* refere-se à virtualização da rede como um todo, desenvolvendo a investigação em torno de duas áreas principais: O desenvolvimento de um *framework* para a construção e manutenção de uma rede virtual ponto a ponto a pedido,

permitindo passar da virtualização de recursos, para a virtualização da rede como um todo; A Virtualização eficiente de recursos da rede, incluindo *routers*, *links* e *links* sem fios.

São definidos três papéis principais que estão interligados como representado na figura 3.12:

- Um *InP*(*Infrastructure provider*), que permite manter os recursos virtuais;
- Um *VNP*(*Virtual Network Provider*), que constrói as redes virtuais usando os recursos virtuais que foram disponibilizados pelos *InP* ou outros *VNP*;
- O *VNO*(*Virtual Network Operator*), permite operar, controlar e gerir as redes virtuais.

O processo de configuração da rede virtual inicia-se com a introdução do pedido de rede virtual, disponibilizado na forma de grafos, ao que o sistema inicia a fase de descoberta de recursos, mapeamento (Seleção de recursos) e instanciação das entidades virtuais (alocação de recursos e ligação com a rede física).

A figura 3.12 permite compreender de que forma se interligam os papéis descritos na abordagem [1]. Estes cooperam entre si, de forma a criar e manter a rede virtual. Para isto, descrevem passos e fases que permitem atingir o objectivo pretendido.

A fase de descoberta e tratamento de pedidos é realizada pelo *VNP* e é baseada nas relações de similaridade entre os pedidos de criação de rede virtual, especificados pelos utilizadores e as descrições da rede de suporte, disponibilizada pelos *InP*. O processo de *matching* baseia-se num *framework* de descrição de recursos, técnicas de *clustering* e algoritmos de similaridade apresentados em [1]. Utilizando estes recursos, os fornecedores de rede virtual, dividem o grafo de rede virtual, entre múltiplos *InP* de acordo com o pedido.

O mapeamento da rede virtual será realizado pelo *InP* e consiste na selecção dos constituintes da rede de suporte para a alocação dos recursos virtuais. Os recursos serão posteriormente instanciados pelo *InP* que os reserva e aloca, de forma a criar a rede virtual. Após a rede virtual estar criada e operacional, deve ser possível manter os parâmetros que foram pedidos. Neste passo, é disponibilizado um mecanismo de alteração, monitorização e manutenção dinâmica.

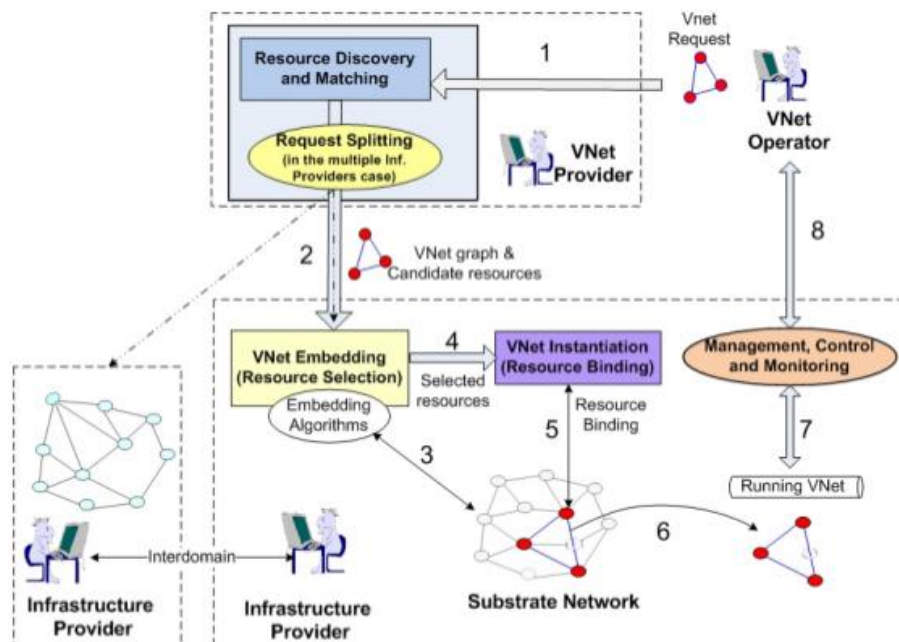


Figura 3.12: Imagem retirada de [1] que permite esquematizar as relações entre os módulos de criação da rede virtual

Mapeamento da rede virtual

O mapeamento da rede virtual consiste em considerar os pedidos de criação de rede virtual, convertendo-os num conjunto específico de nós e caminhos virtuais obtidos a partir da rede de suporte.

A tarefa de saber qual o melhor *host* para instanciar um nó virtual, é um problema NP-completo[42] e por esta razão o mapeamento dos nós virtuais, tem sido combatido utilizando algoritmos heurísticos[35, 43, 44], algoritmos customizáveis[43], processos de mapeamento iterativos [45] e mapeamento coordenado de nós e *links* em [46]. Como o suporte da rede virtual pode não ser estável é necessário a implementação de algoritmos mapeamento adaptativo para além do mapeamento inicial, os presentes em [47] permitem assegurar que isto ocorra, utilizando um algoritmo inicial para criar e alocar os recursos virtuais e logo que a rede virtual esteja instanciada, um algoritmo distribuído de controlo de falhas, mantém a topologia virtual.

A implementação do algoritmo de mapeamento utiliza uma abordagem com vários agentes, distribuídos sobre uma plataforma de máquinas *GRID5000*⁵[48] que emulam os nós virtuais. A

⁵O *GRID5000* é uma instalação de cerca de 5000 processadores, partilhados sobre 9 *clusters*, localizados em várias regiões de França e inter-conectados por uma rede de fibra óptica de 10Gb/s

Java Agent Development Framework (JADE)[49] é usada para implementar os agentes autónomos da rede de suporte, estes trocam mensagens e cooperam para executar os algoritmos distribuídos. As mensagens trocadas entre os agentes são definidas pela *Agent Communication Language (ACL)* [50] e a ferramenta *GT-ITM*(Georgia Tech - Internetnetwork Topology Models)[51] é usada para criar topologias atendendo aos pedidos de criação de rede virtual recebidos.

De forma a criar, manter e adaptar as redes virtuais são usados algoritmos que permitem a negociação do mapeamento distribuído e a sincronização entre a rede de suporte e os nós virtuais. A *framework* de mapeamento multi-agente, *JADE*, permite que estes colaborem e comuniquem entre si de forma a criar e manter as redes virtuais.

No mapeamento inicial os pedidos de rede virtual devem ser divididos em séries de *clusters* com topologia em estrela. Estes *clusters* são compostos por um nó central, como por exemplo um *hub*, onde múltiplos nós são conectados. A interligação de *hubs* permite também a conexão entre vários *clusters*. O mapeamento da topologia para a rede de suporte é realizado designando sequencialmente os *clusters* de nós e *hubs*.

O algoritmo de mapeamento utilizado para criar os *clusters* é distribuído, onde um *root node*, o nó que possui mais recursos disponíveis, será responsável por seleccionar e mapear um *cluster* na rede de suporte. Este determina o conjunto de nós físicos que suportarão os nós virtuais, baseando-se no caminho mais curto para cada um deles. Os *root node* comunicam e colaboram entre si de forma a decidir qual o melhor mapeamento possível para os *clusters*.

Algoritmo de mapeamento dinâmico

O algoritmo de mapeamento tolerante a falhas mantém as topologias de rede virtual escolhendo novos nós e *links* que permitam tratar as falhas das entidades mapeadas com o algoritmo inicial. O algoritmo está baseado na monitorização e detecção de falhas e usa o *framework* multi-agente para assegurar a troca dos dados entre os agentes.

Os agentes trocam mensagens e cooperam de forma a planear as decisões de re-mapeamento em caso de falhas. Quando um nó da rede de suporte falha, o *root node* é incumbido de seleccionar um novo nó para manter a topologia. Se o *root node* falha, um dos nós que estavam já conectados na rede é seleccionado como o novo *root node*.

Configuração dos links virtuais

Os *links* virtuais, são criados entre os nós da rede de suporte que executam um nó virtual. Um protocolo de configuração de *link* virtual, aloca os recursos necessários no caminho entre os dois nós físicos, conectando as interfaces virtuais. O protocolo é baseado numa implementação do protocolo *Next Steps in Signalling (NSIS[52])*. Foi ainda utilizado o *NSIS Signalling Layer Protocol (QoS NSLP)* que executa sobre o *General Internet Signalling Transport Protocol (GIST[53])*.

A figura 3.13 permite compreender de que forma a arquitectura de protocolos *NSIS* se relaciona. O *QoS NSLP* é um protocolo de reserva de recursos, que permite o controlo de admissão ao meio e o encaminhando dos pacotes *IP* com base no valor de *QoS*. A extensão criada no âmbito do projecto *4WARD*, permite que este transporte as informação de endereços necessária para a criação do *link* virtual entre os nós virtuais.

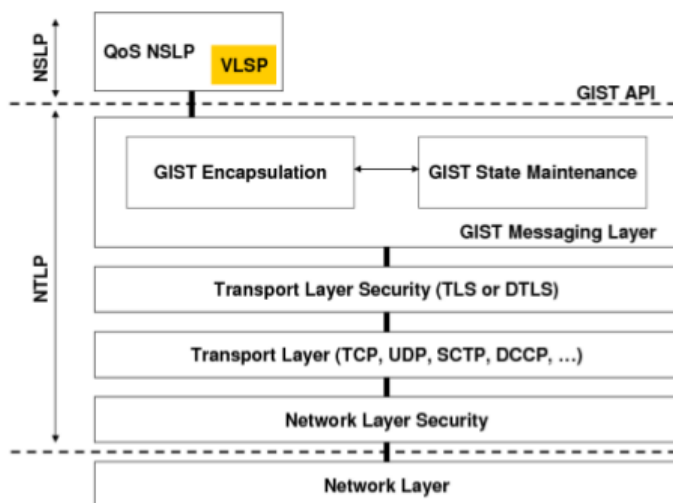


Figura 3.13: Imagem retirada de [1], que permite compreender de que forma se relacionam os protocolos de reserva de recursos utilizados

A criação de um *link* virtuais passa por uma série de passos que podem ser representados pela figura 3.14:

- Os nós de suporte necessitam de obter o endereço do nó no ponto contrário do *link* virtual. Isto significa que para que se forme o *link* virtual entre dois pontos, sejam conhecedores dos extremos;

- Os nós virtuais devem ser instanciados quando o *link* virtual é inicializado pois as mensagens de sinalização necessitam que os nós virtuais estejam activos antes da configuração do *link* virtual;
- A troca de mensagens de sinalização entre os nós físicos, permite que se confirme a existência de um caminho entre os dois nós. Podendo o protocolo de reserva de recursos ser utilizado entre os *Resource Management Functions (RMFs)* de forma a assegurar *QoS* entre os *links* virtuais se tal for necessário;
- As mensagens de sinalização devem atingir o nó no outro extremo de forma a iniciar o estado necessário para a conexão dos *links* virtuais na interface correcta do nó virtual. A informação necessária para este fim é transportada dentro das mensagens de sinalização para os *RMF* envolvidos;
- O passo final consiste na iniciação da conexão entre os extremos do túnel do *link* virtual usando a informação recebida.

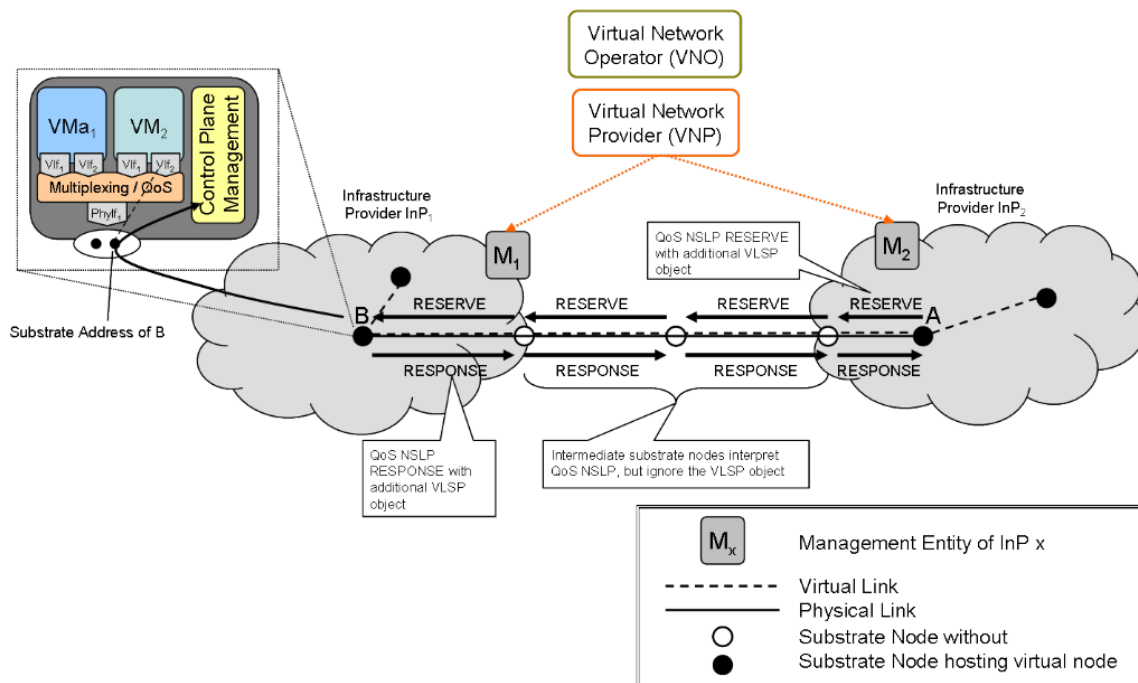


Figura 3.14: Imagem retirada de [1] que descreve a forma como os links são criados e os recursos são reservados no caminho entre os extremos

Descoberta de recursos, manutenção e controlo

Na abordagem de virtualização de rede, o *InP* é responsável pela configuração, manutenção e controlo da rede física interagindo com os recursos de rede através da consola de gestão, tal como representado na imagem 3.15. Esta permite uma visão global sobre os recursos de rede. Sempre que um pedido de estabelecimento ou modificação da rede virtual é recebido, o gestor da rede baseia-se nos recursos disponíveis para decidir se deve aceitar o novo pedido. Se este pedido for aceite, deve também decidir como mapeá-lo na rede. O gestor deve ser responsável por comunicar com os agentes localizados em cada nó físico. Obtendo informação sobre os recursos da rede, guardando-os numa base de dados.

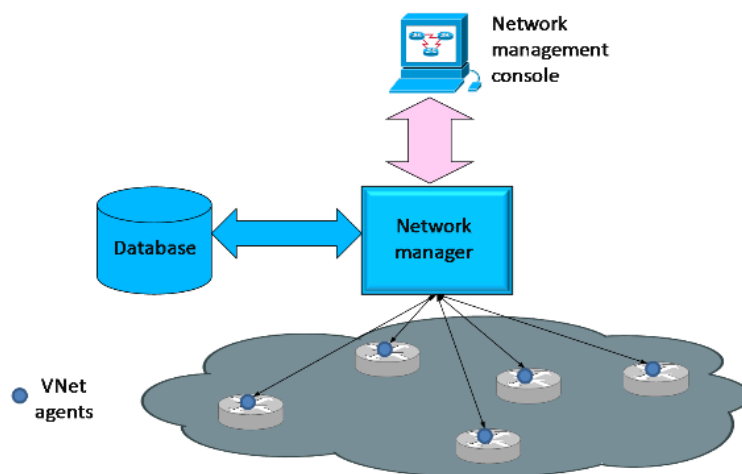


Figura 3.15: Imagem retirada de [1], que descreve a forma como os *InP* se relacionam com as entidades de rede

As abordagens do projecto *4WARD* foram implementadas utilizando um sistema de virtualização baseado no *Xen hypervisor*[21] e foi possível a configuração da rede virtual e posterior criação com base num *XML* que permite descrever os parâmetros da rede. Após a recepção de um pedido para a criação de uma nova rede virtual, são procurados os recursos físicos necessários para a acomodação dos respectivos nós e *links* virtuais.

3.6 Síntese

Os projectos que foram estudados apresentam soluções distintas para a construção de uma rede virtual. Contudo, os módulos utilizados para a construção do sistema são similares: procura de recursos, mapeamento e manutenção integram grande parte dos algoritmos estudados.

A procura de nós disponíveis para a construção da rede virtual é realizada de formas distintas nos projectos estudados, enquanto que o *X-Bone*[3, 33] aborda este tema realizando uma procura expansiva, aumentando o *TTL* em cada mensagem de procura enviada, projectos como o da PT inovação[6, 34, 35] e *4WARD*[1] apostam numa comunicação constante entre os *hosts*, obtendo a informação que necessitam de uma forma constante e actualizada.

O mapeamento de entidades virtuais é utilizado de forma diferenciada pelos projectos estudados. Este problema é abordado exaustivamente no projecto da PT inovação que sugere várias abordagens para que seja resolvido, agarrando-o como um problema NP-completo[42] e, com base nos conceitos defendidos em [35] propõe uma solução que utiliza as capacidades físicas do nós para saber qual o melhor para mapear o nó virtual. O projecto *X-bone*, por outro lado, utiliza um *RD* que permite, a qualquer altura, obter a carga de um recurso e com isto decidir se deve ou não aceitar a criação de um novo nó virtual.

A capacidade da rede se manter depois de ser criada é importante, uma vez que permite manter a transparência que caracteriza uma rede virtual. O projecto *X-bone* aborda este tema utilizando um ficheiro que armazenará todas as alterações que ocorreram na rede, permitindo contrariar uma acção que faça com que a rede deixe de funcionar. Por outro lado, projectos como o *4WARD* propõem algoritmos de mapeamento dinâmico, que permitem obter alternativas caso exista a falha de uma entidade na rede.

Os projectos *Vine*[4] e *Violin*[5] abordam o problema de criação de rede de uma outra forma. O primeiro foca o desenvolvimento numa rede centralizada num *VR* que fará todas as tarefas necessárias da rede virtual, configurando os *hosts* que dela fazem parte com interfaces à medida que apenas são capazes de comunicar com o *VR*. Este projecto oferece uma solução para ultrapassar mecanismos de *NAT* e *proxies*. O projecto *Violin* aborda em grande parte, a criação das entidades virtuais e os comportamentos que devem possuir, sugerindo uma arquitectura hierárquica de conexão entre *vHosts*, *vLANs* e *vRouters*.

Capítulo 4

Proposta de uma infra-estrutura virtual de nível 2

O projecto realizado no âmbito da dissertação propõe a criação de uma infra-estrutura de rede virtual, capaz de interligar nós virtuais, recorrendo a algumas tecnologias bem sedimentadas na estrutura da rede pública.

De forma a que um sistema aspire a ser escalável é necessário que tome como bases as tecnologias que vigoram na rede pública. Protocolos como o *IP* no nível 3 do modelo *OSI* e *ethernet* no nível 2, controlam a forma como a informação é trocada.

A rede virtual é construída sobre a rede de suporte, *links* e nós são virtualizados, utilizando os recursos físicos e criando cenários específicos com condições determinadas pelo utilizador. O sistema deverá ser capaz de permitir que qualquer nó possa pertencer à rede que se pretende criar, sem necessitar de configurações extensas e alterações do sistema operativo. A portabilidade é também um ponto importante no desenvolvimento do projecto uma vez que deve ser possível a construção da rede virtual sobre qualquer infra-estrutura.

A virtualização permite que cada nó criado para a rede virtual seja um sistema alterável, capaz de executar tarefas distintas. Um nó virtual deve ser capaz de emular o funcionamento de um *router*, ou um *switch*, dependendo apenas do sistema que esteja a executar em cada um dos pontos da rede e por esta razão, cada nó da rede será tratado como um sistema genérico, capaz de receber e enviar informação.

Uma rede virtual de nível 2 é composta por um conjunto de *links* e nós virtuais, mapeados numa infra-estrutura de suporte. Onde a comunicação pode ser realizada entre dois ou mais nós.

O projecto desenvolvido apresenta uma arquitectura representada pela figura 4.1.

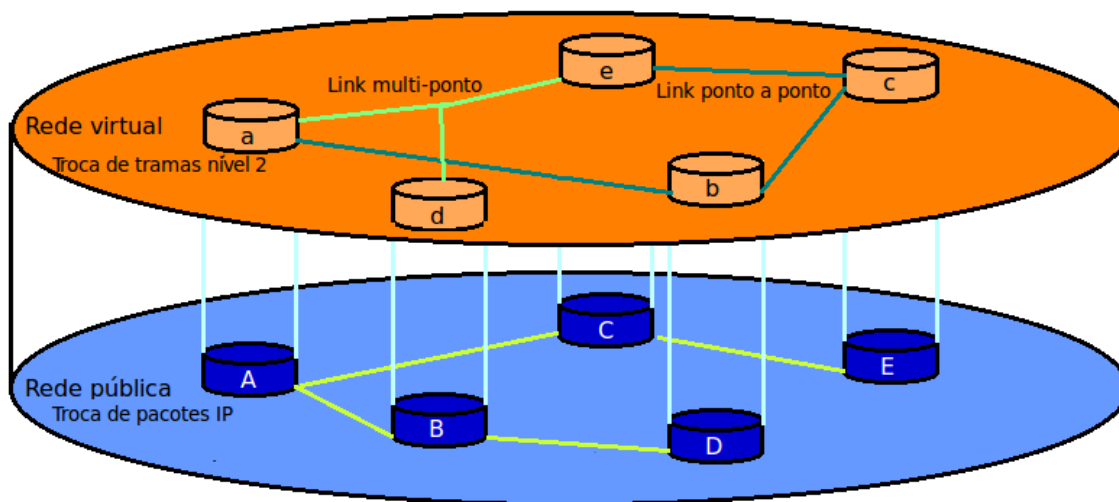


Figura 4.1: Representação do funcionamento da rede virtual a desenvolver, onde a camada superior troca datagramas de nível 2 e a rede de suporte IP

Os nós virtuais poderão ter ligações ponto a ponto, ou multi-ponto. Estas entidades poderão simular o papel de *switches*, *routers* e nós dependendo do teste que se pretenda realizar e do sistema operativo que se encontra virtualizado.

A capacidade de criação de *links* com parâmetros distintos, permite simular várias arquitecturas de rede, tal como topologias em estrela, barramento ou *token ring*, permitindo testar o funcionamento de novos protocolos e aplicações num meio controlado e isolado.

A criação da infra-estrutura de rede pode ser condensada numa série de passos e modelos de funcionamento que se complementarão. Permitindo que a agregação das informações que se obtêm proporcione a construção da rede.

4.1 Descoberta de recursos

De forma a ser possível a construção da rede virtual, é inicialmente necessário descobrir que recursos da rede de suporte estão disponíveis. Entidades como nós, interfaces e máquinas virtuais serão recursos que a rede virtual poderá utilizar. Abordagens baseadas em *multicast* [3][6], possibilitam a comunicação distribuída de uma forma simples. Enquanto que abordagens como

[4] se focam na infra-estrutura como vários *overlays* interligados, onde os recursos da aplicação se restringem a uma entidade capaz de encaminhar as informações.

O problema foi abordado de forma similar ao que ocorreu em [3], definindo uma forma de comunicação que permite detectar todos os *hosts* disponíveis para a criação da infra-estrutura virtual. A procura de recursos para a rede virtual terá um alcance definido pelo utilizador, que recorrendo à interface gráfica da aplicação deve introduzir o valor máximo de saltos que a mensagem de procura deve dar. Este valor servirá para contactar os recursos dispersos na rede tal como representado na figura 4.2.

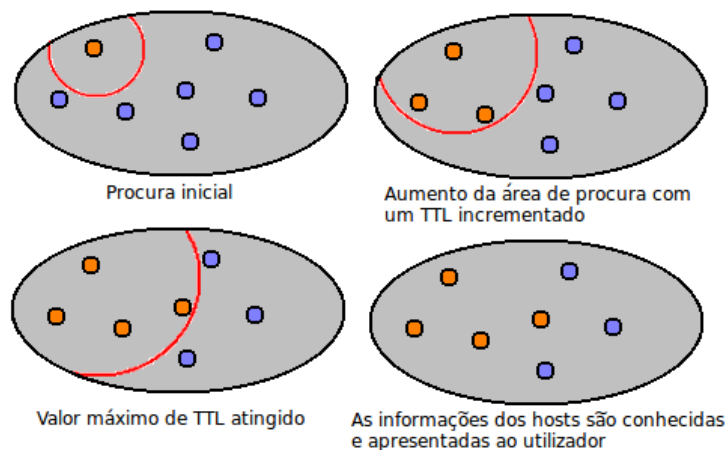


Figura 4.2: Passos realizados na procura de nós disponíveis na rede de suporte

A figura 4.2 permite esquematizar a procura de nós. Inicialmente é enviada uma mensagem com o *TTL* 1 para o grupo *multicast*, que será apenas respondida por *hosts* locais. O *TTL* vai sendo aumentado, até que atinja se valor passado como parâmetro, ou sejam encontrados os nós necessários para a construção da rede.

Quando uma mensagem de criação de rede é recebida, as características do nó são compiladas e disponibilizadas ao nó que requisitou a informação. A mensagem de resposta incluirá os dados que permitirão interligar as instâncias das máquinas virtuais:

- Informações do nó de suporte:
 - IP* das interfaces;
 - Endereço *Mac* das interfaces;

Carga local;

IP do *gateway* de saída.

- Informações das máquinas virtuais:

Nome das máquinas virtuais;

Sistema operativo;

Estado de funcionamento;

Endereço *ethernet* que cada interface virtual possui;

Interface, no sistema operativo nativo, que fará a ligação com a interface virtual.

Os dados obtidos pela procura de nós são apresentados ao utilizador, que caso não esteja satisfeito com as informações retornadas, tem a capacidade de requisitar uma nova procura, repetindo o processo.

De acordo com a abordagem em [6] a procura dos nós poderia ser realizada de forma constante, enviando periodicamente uma mensagem com as características de cada *host*. Contudo, esta aumentaria a carga na rede com o massificação dos nós trazendo problemas de escalabilidade. A abordagem sugerida propõe que todos os parâmetros da procura sejam customizáveis, possibilitando a existência de várias procuras de nós com endereços de controlo *multicast* distintos, na mesma rede, sem que interfiram entre si.

4.2 Mapeamento das entidades virtuais

A criação da rede virtual será baseada nas informações recebidas na fase inicial de procura de recursos, que permitirão controlar a forma como a rede deve ser criada, identificando as características das conexões entre os nós virtuais.

Uma arquitectura virtual de nível 2 permite que comunicação seja ponto a ponto, ou multiponto, ocasionando trocas de informação entre dois ou mais nós. O projecto requer que sejam pensadas formas de criação de conexões capazes de trocar informações entre vários nós de forma transparente.

Alguns trabalhos estudados tratam o mapeamento de instâncias virtuais com grande importância, sugerindo métodos de mapeamento automático que recorrem a algoritmos heurísticos[35, 43, 44], algoritmos customizáveis[43], processos de mapeamento iterativos [45] e mapeamento coordenado de nós e *links* em [46]. Contudo, estas abordagens tiram o poder ao utilizador de mapear as entidades virtuais da forma que pretende.

A abordagem sugerida neste trabalho oferece o controlo de criação da rede virtual ao utilizador, permitindo que este construa a camada de rede de acordo com o teste pretendido.

4.2.1 Mapeamento de ligações ponto a ponto

A criação de um *link* virtual recorrerá aos dados que foram obtidos pela procura de recursos no primeiro passo do algoritmo. O utilizador define os extremos da conexão na rede de suporte e as características do *link*, que permitirão o delineamento de quais os nós que hospedarão as entidades virtuais.

Esta escolha é da competência do utilizador, que verificando a carga sobre cada um dos nós, saberá se quer um teste utilizando um nó com uma carga elevada ou não. A definição dos extremos físicos, acompanhará a definição dos pontos virtuais de comunicação definindo as máquinas virtuais que devem ser arrancadas e quais as interfaces que devem ser os pontos de ligação entre as entidades criadas. Existe também a possibilidade de atribuir aos *links* virtuais criados um valor de *QoS*, uma largura de banda máxima e a compressão das informações. As características dos *links* ponto a ponto passadas pelo utilizador, são agregadas e trocadas entre os extremos físicos do *link* virtual, negociando as suas características.

O canal de comunicação entre os nós virtuais deve ser capaz de oferecer duas características muito importantes:

- Assegurar-se que os dados enviados pelos nós virtuais não devem ter qualquer contexto fora do ambiente virtual;
- Os dados enviados deverão ser entregues apenas ao extremo do *link* virtual.

A primeira característica pode ser obtida configurando as máquinas virtuais, para que comuniquem exclusivamente com o nó que as suporta, o sistema hospedeiro, isto permitirá criar uma

nova interface no nó de suporte que servirá para trocar informações. O segundo ponto já não é tão trivial como o primeiro, e existem várias soluções que propõem abordagens para o problema do encaminhamento da informação, apresentando propostas como a interceptação das chamadas do sistema, alterando-as para que possuam um funcionamento de acordo com o que o projecto requer, implementações que recorrem a alterações das interfaces de rede ou a inclusão de novos endereços, que permitirão enviar a informação para um nó preparado para realizar o papel de *proxy*.

A abordagem apresentada pela dissertação propõe um mecanismo de captura e envio de informação baseado em *sniffers* e encapsulamento. Após os parâmetros da ligação serem acordados entre os dois pontos, são iniciados os nós virtuais que corresponderão aos extremos da conexão. A interface do nó virtual criado, será escutada e os datagramas que envia são capturados e tratados. São depois colocados numa cápsula de transporte endereçada ao nó físico remoto que estará preparado para os receber. A imagem 4.3 permite descrever sumariamente, a forma como a comunicação se processa.



Figura 4.3: Troca de informações entre nós virtuais numa abordagem ponto a ponto

A informação que provém da máquina virtual é processada pela aplicação desenvolvida, colocando-a num pacote *IP* que atravessará a rede de suporte até o outro extremos da conexão.

4.2.2 Mapeamento de ligações multi-ponto

Os *links* multi-ponto terão uma implementação diferente da comunicação entre dois nós virtuais. Isto ocorre pois o paradigma de comunicação é alterado, não sendo possível a negociação dos

parâmetros entre os *hosts*. Uma vez que teria o potencial para se tornar uma tarefa demorada com o aumento de nós na rede e a junção de novos nós.

A junção à rede virtual, utilizando um *link* multi-ponto é uma tarefa que cada nó deverá realizar e onde cada *host* deve controlar as redes virtuais a que se pretende juntar.

Um *link* multi-ponto necessita da criação de um mecanismo que permita a troca de mensagens por um conjunto difuso de nós virtuais. O *multicast* oferece exactamente estas características, uma difusão das informações enviadas pelos *hosts* que as pretendam receber.

A abordagem proposta implementa um canal de controlo que permitirá anunciar as informações dos *links* multi-ponto utilizados, cada um destes *links* é identificado pelo seu endereço *multicast*, que é anunciado no canal de controlo e disponibilizado, com os parâmetro que possui, para que os outros nós virtuais se possam juntar dinamicamente.

A junção de um nó à utilizando uma conexão multi-ponto já criada, pressupõe que o nó aceita os seus parâmetros, podendo apenas costumizar a máquina virtual a arrancar.

O esquema lógico de funcionamento de um *link* multi-ponto, pode ser observado na figura 4.4.

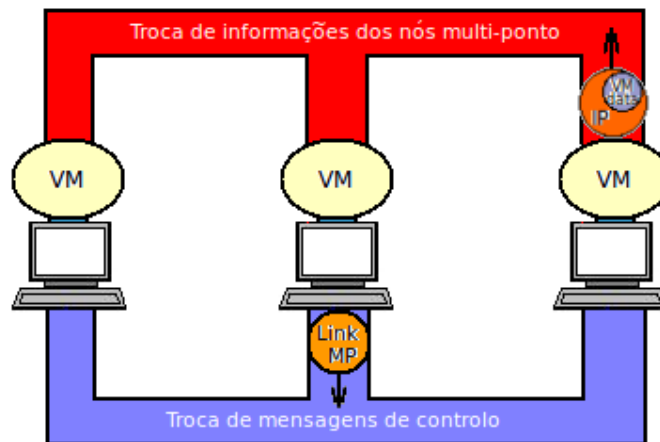


Figura 4.4: Esquema de um *link* multi-ponto

A informação trocada sobre o *link* multi-ponto é encaminhada sobre a rede de suporte utilizando pacotes *IP*. No destino a informação é desencapsulada e disponibilizada ao nó virtual. É possível existirem vários grupos de controlo e troca de informação, sendo possível a criação de vários *links* virtuais multi-ponto sobre a mesma infra-estrutura de rede. Por um motivo de

escalabilidade e controlo, apenas os nós criados localmente são anunciados pelo canal de controlo. Esta abordagem faz com que caso o gerador do *link* multi-ponto deixe de estar activo, seja cessada a divulgação do *link* multi-ponto. Este controlo de divulgação assegura que a divulgação dos *links* virtuais seja controlada e não anunciada por todos os *hosts* o que resultaria num *flooding* da rede com mensagens de controlo.

A criação do mecanismo de controlo de envio de mensagens não tem qualquer controlo sobre a eliminação das redes que são criadas. Caso o nó criador de um *link* multi ponto, cesse o seu funcionamento, os nós virtuais que a compõem continuarão a ser capazes de comunicar entre si utilizando o *link* virtual.

4.2.3 Manutenção e portabilidade da rede virtual

A manutenção da rede virtual apresenta uma importância tão elevada como a criação dos *links*, a capacidade de modificar a forma como as entidades se comportam é importante pois vem permitir a alteração das variáveis do teste que se pretende realizar. Factores como endereços das interfaces que se pretendem capturar e valor de *QoS* da cápsula de transporte, devem poder ser alterados de forma dinâmica, permitindo potenciar a capacidade de configuração da rede virtual.

De forma a ser possível conseguir este parâmetro, cada nó virtual necessitará de uma janela de configuração, onde as características tanto da conexão, como da máquina virtual poderão ser alteradas, de acordo com o que é pretendido.

Esta manutenção alia-se à portabilidade da rede virtual, onde os dados de cada um dos *links* e nós virtuais, são carregados para um ficheiro *XML*, que permitirá instanciar a rede criada sobre uma outra infra-estrutura de suporte. As informações no ficheiro *XML* são transformadas em informações da rede virtual e mapeadas na rede sobre a forma de entidades virtuais.

A portabilidade entre redes distintas necessita na grande parte dos casos de alterações no ficheiro *XML* originalmente criado pela aplicação, uma vez que os nós de suporte terão *IPs* distintos. Sendo necessária a mudança dos extremos físicos nas ligações ponto a ponto, para que se possa carregar a infra-estrutura de rede correctamente.

Capítulo 5

Implementação da proposta de infra-estrutura virtual de nível 2

Este capítulo servirá para apresentar os passos dados na implementação da proposta apresentada de rede virtual. Serão explicados os módulos que compõe a solução, desde a procura de recursos disponíveis, o protocolo de negociação de parâmetros e a criação dos túneis que permitirão a conexão entre os nós.

O problema foi dividido em vários de menor magnitude e foram testadas várias abordagens de forma a comparar o seu comportamento em cenários reais.

A comunicação entre os nós virtuais criados foi projectada recorrendo a *Sockets* e *RawSockets*. Foi ainda implementada a conexão multi-ponto, que recorrendo a *MulticastSockets* e a *RawSockets* permite a comunicação entre dois ou mais nós.

5.1 Procura de hosts

A rede virtual vai ser composta por várias entidades da rede de suporte que disponibilizarão os seus recursos para a sua criação. De modo a descobrir os recursos disponíveis, foi implementado um mecanismo de procura baseada na estudada em [3], que permite procurar e obter informações dos nós que estão interessados e preparados em participar na criação da infra-estrutura virtual.

Cada *host* possuirá uma aplicação que o permitirá conectar-se a um endereço *multicast*, mu-tável, de onde obterá as informações de controlo da rede, como representado na figura 5.1.

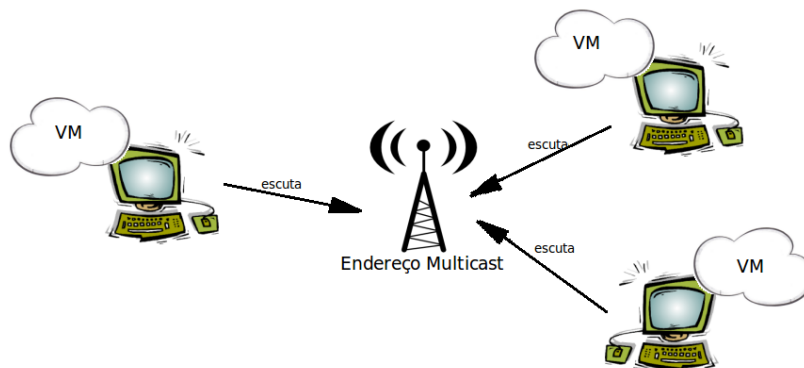


Figura 5.1: Cada nó ligar-se-à e escutará um endereço multicast de troca de mensagens de controlo

A comunicação entre os nós que disponibilizam os seus recursos para a criação da rede virtual, vai ser realizada usando um formato de mensagem pré-definido. A mensagem, representada na figura 5.2, terá os campos:

- Tipo de mensagem - (*int*) Permite descrever os parâmetros contidos na mensagem, de forma a trata-los convenientemente;
- Argumentos - (*String*) Campo de controlo da mensagem, onde podem ser transportadas descrições do protocolo;
- Dados - (*Object*) Neste campo são transportados os dados que permitem descrever as entidades, tanto da rede física como da virtual.

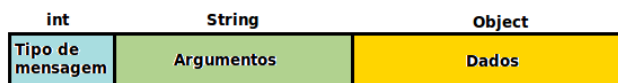


Figura 5.2: Mensagem genérica trocada entre os *hosts* que suportam a rede virtual

A ligação entre dois nós utilizando uma conexão ponto a ponto, requer que sejam inicialmente procurados os recursos que estão disponíveis na rede. O utilizador define o número

máximo de nós que pretende na rede virtual, ou apenas o valor máximo de *TTL* que será colocado nas mensagens. A procura de recursos é iniciada com o envio da mensagem representada na figura 5.3.



Figura 5.3: Mensagem do tipo 1, enviada de forma a requisitar um pedido de recursos

Esta mensagem é enviada inicialmente com um *TTL* de 1, fixando a procura à rede local, onde apenas os nós ligados directamente ao endereço *multicast* receberão a mensagem. A figura 5.4 pretende ilustrar esta comunicação.

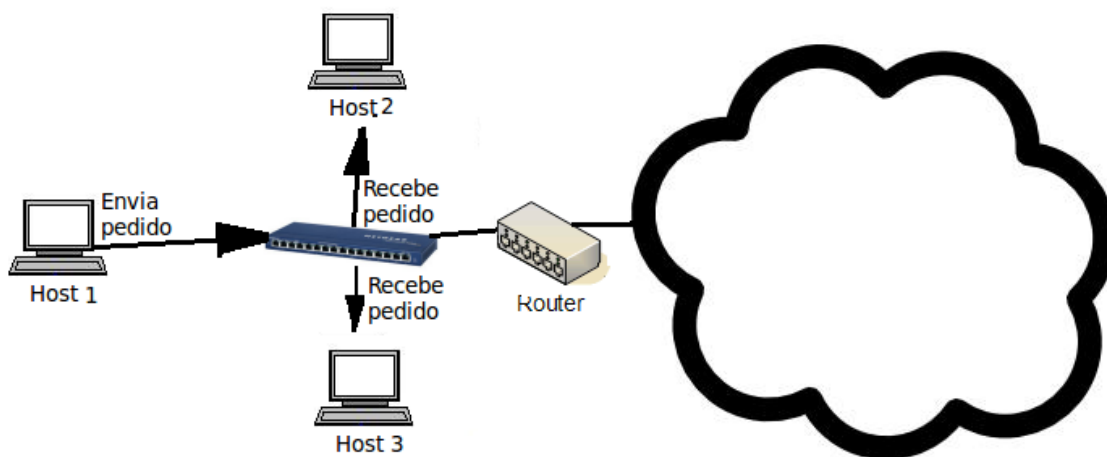


Figura 5.4: Procura de nós com um *TTL* inicial

O nó que envia a mensagem vai receber as respostas dos *hosts* preparados para pertencer à rede virtual. Nestas mensagens estarão as características de cada recurso. As interfaces que possui disponíveis, a carga do sistema e as máquinas virtuais que tem ao dispor são enviados na mensagem de forma a informar o nó que envia o pedido.

A carga vai ser obtida utilizando os parâmetros do sistema, calculando a carga de cada *host*, usando a biblioteca *SIGAR*[54] para obter os valores e a fórmula apresentada em [35] para o

cálculo.

$$C_n = \frac{N_{vm}}{\delta + R_f \times F_{cpu} \times (N_{cpu} \times F_{cpu} - C_{cpu})}$$

- C_n - Carga de cada *host*;
- N_{vm} - Número de máquinas virtuais em funcionamento no *host*;
- δ - Variável usada para impedir que ocorra uma divisão por zero;
- R_f - RAM livre no sistema em *Mbyte*;
- F_{cpu} - Frequência do CPU em *MHz*;
- N_{cpu} - Número de CPUs que o sistema possui;
- C_{cpu} - Carga que o CPU tem no momento, o valor pode variar entre 0 e $N_{cpu} \times F_{cpu}$;

O valor calculado é colocado no Objecto que formará o campo dados da mensagem. Para cada mensagem de criação de rede é também criada uma lista das interfaces do nó, guardada sobre a forma de um Objecto do tipo *VnetHost*. Neste Objecto ficarão guardadas, relativamente a cada interface, a nomenclatura que possui no sistema, o IP, o endereço *mac* que o identifica e o *gateway* por defeito. As máquinas virtuais e as suas características, tal como os endereços *ethernet* das interfaces virtuais e a interface do sistema operativo a que estão ligadas, são também enviadas na mensagem.

A procura de nós apenas termina no momento em que o nó que envia a mensagem do tipo 1, atinge o valor de *TTL* definido como parâmetro, ou quando o número de respostas é suficiente para a criação da rede virtual. Com o envio da primeira mensagem é iniciado um contador, que ao atingir os 2 segundos incrementa o valor do *TTL* e envia uma nova mensagem. O valor sugerido é baseado no tempo que biblioteca de captura normalmente demora a obter as informações do sistema. Esta procura de recursos está apresentada na imagem 5.5.

Cada *host* ao receber uma mensagem do tipo 1, vai compilar a carga que possui e agrupar as informações sobre as suas interfaces e máquinas virtuais, enviando a informação para o cliente

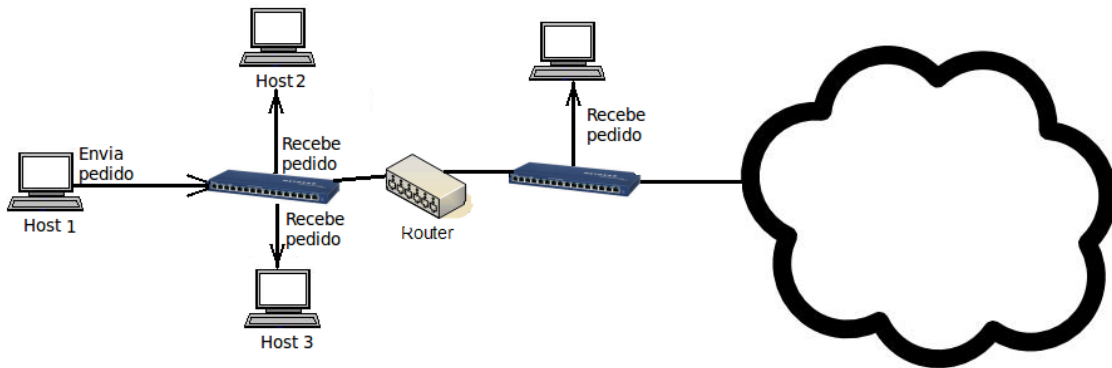


Figura 5.5: Procura de nós disponíveis na rede, utilizando um *TTL* mais elevado

que requisitou o pedido de criação de rede, este, ao recebe-la, substitui-la-à, caso já exista.

O valor do *TTL* da mensagem vai continuar aumentar a cada dois segundos até atingir uma das condições que o permitirá passar para o próximo ponto do protocolo. Quando isto ocorre, o *host* uniformiza o conhecimento na rede, enviando-o numa mensagem para o grupo *multicast*. A mensagem enviada está representada pela figura 5.6.



Figura 5.6: Formato da mensagem de tipo 3 enviada, que permite divulgar localmente os nós conhecidos

A informação na lista pode ser actualizada a pedido do utilizador, quando isto acontece é enviada uma mensagem de tipo 4, representada na figura 5.7, para o endereço *multicast* à qual os *hosts* responderão com uma actualização dos seus parâmetros. A actualização permitirá ao *host* substituir os dados por mais recentes.



Figura 5.7: Formato da mensagem de tipo 4, que permite actualizar o estado dos nós vizinhos

Esta mensagem permite obter os valores da carga de cada *host* no instante, permitindo actualizar as características de cada nó para a criação dos *links* virtuais. Esta mensagem não possuirá uma semântica diferente da enviada com tipo 1, contudo, foi necessário realizar esta separação pois os processos que desencadeiam no *host* que a recebe não são os mesmos.

5.2 Ponto a ponto - *RawSockets*

A ligação ponto a ponto recorrendo a *RawSockets*, permitirá criar um *link* virtual que recorrerá ao envio e recepção de informação directamente. Cada uma destas conexões interligará duas interfaces de máquinas virtuais e usará a infra-estrutura de rede para o encaminhamento de informação.

A rede pública possui um grande volume e variedade de tráfego, de modo a ser possível que o tráfego enviado por um extremo do *link* virtual se distinga é necessário a existência um mecanismo que permita marcar os datagramas enviados. Por este motivo foi também introduzido no sistema um protocolo de negociação dos parâmetros da ligação. Inicialmente será explicado o funcionamento do *RawSocket*, assim como a análise da biblioteca que foi usado para a sua criação, o *Jpcap*[55].

5.2.1 *RawSockets*

Um *RawSocket* pode ser visto como uma ligação que permite o acesso vertical a toda pilha *OSI*. Estes possibilitam que uma aplicação escute a camada de comunicação que lhe interessa, obtendo a informação da forma que o sistema operativo os recebe, ainda sem o processamento que teria em cada uma das camadas do modelo *OSI*. Isto permite que uma aplicação obtenha um datagrama *ethernet*, da mesma forma que este circula na rede, com endereço *ethernet* de origem e destino, tipo de informação que contém etc.. Curto-circuita-se desta forma as várias etapas de obtenção de informação que vão ocorrendo em cada nível da implementação do modelo *OSI*,

que teriam a representação da figura 5.8 num sistema onde todas as camadas estivessem implementadas.

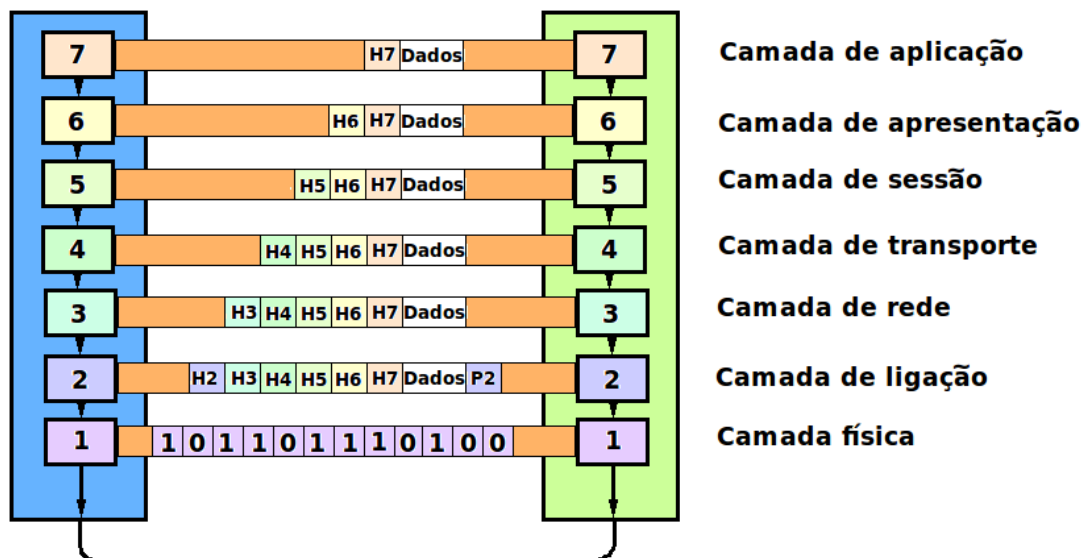


Figura 5.8: Troca de informação no modelo OSI

No tipo de modelo apresentado na figura, é possível verificar que o datagrama que chega à camada de ligação contém mais informação que aquela que chega ao nível superior. Dando-se assim a possibilidade ao programador de controlar o que é enviado e qual o caminho que toma.

O princípio base dos *RawSockets* é a ligação directa a qualquer uma das camadas de passagem com a camada de aplicação criando uma abstracção como a representada na figura 5.9.

Estas ligações necessitam de permissões especiais para serem criadas, pois permitem um controlo de toda a camada de comunicação, da responsabilidade do sistema operativo. Por esta razão muitas linguagens não os suportam nativamente, necessitando de apoio de outras bibliotecas para realizar esta tarefa.

Foram estudadas algumas destas bibliotecas disponíveis em *Java*, o *Jpcap*[55], o *RockSaw*[56] e o *JPacket*[57]. Foi ainda estudada a possibilidade de implementação deste mecanismo usando ferramentas que permitem o desenvolvimento de métodos em *C*, executando-os em *Java*, com recurso ao *JNI*(*Java Native Interface*)[58].

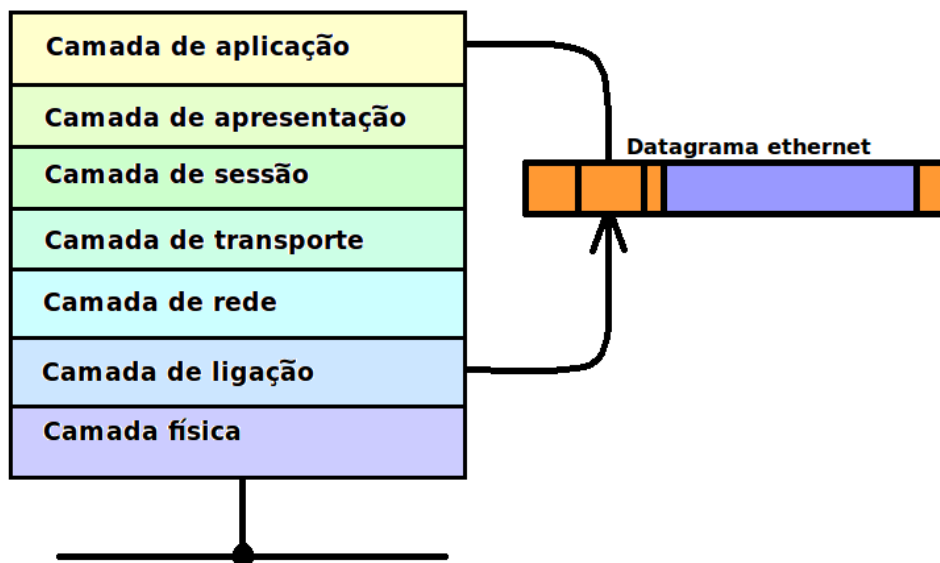


Figura 5.9: Transporte da informação da camada de ligação para a de aplicação

5.2.1.1 Jpcap

O *Jpcap* é uma biblioteca *Java*, que faz uso do *JNI* que permite a captura e envio de pacotes de rede a partir de aplicações desenvolvidas em *Java*[55]. Permite a leitura de pacotes de dados da camada de ligação, obtendo-os com toda a informação necessária para o envio. É possível ainda a criação de ficheiros que contêm a informação que foi trocada na rede e o seu posterior estudo. O *Jpcap* permite que qualquer datagrama, ao ser recebido, seja tratado como um objecto, permitindo trata-lo de forma simples, obtendo facilmente todas as informações que os cabeçalhos dos vários níveis possam fornecer. Os datagramas recebidos podem ainda ser filtrados, mediante regras propostas pelos utilizadores, permitindo um maior uso real dos recursos, pois apenas se processam os pacotes com informação que realmente interessa à aplicação desenvolvida.

O *Jpcap* está baseado no *libpcap/winpcap*. O *libpcap* é uma biblioteca *open source*, que permite a criação a captura dos pacotes em interfaces. Este projecto, desenvolvido em 1994, por dois investigadores, foi inicialmente pensado como meio de aumentar o desempenho de vários protocolos. O principal objectivo deste seria a criação de uma *API* independente da plataforma[59]. O funcionamento do *libpcap*, assim como o do *Jpcap* pode ser descrito pela figura 5.10.

Quando um datagrama atinge a interface de rede, é capturado pelo *hardware* e processado pelo *device driver* do dispositivo. A informação obtida é então duplicada, deixando o original

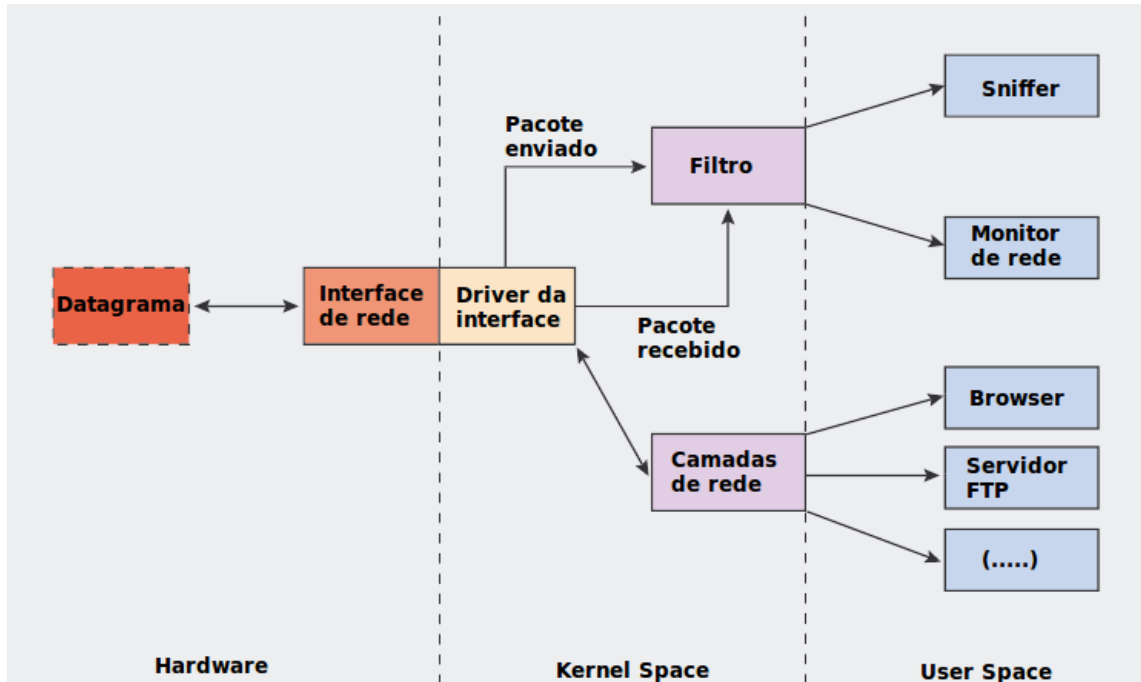


Figura 5.10: Captura de informação recorrendo ao *libpcap*

seguir o seu caminho no modelo de comunicação, enquanto que uma cópia é processada pelo filtro definido pelo programador, passando-o à aplicação caso este seja aceite.

A biblioteca *Jpcap*, permite o controlo da recepção e envio de informação facultando mecanismos que permitem interagir com a rede, facilitando a construção de aplicações que controlam o envio e captura de tráfego, utilizando para isto os filtros que o *libpcap* possui disponíveis.

5.2.2 Negociação de Links virtuais

Os recursos descobertos pela procura são neste momento apresentados ao utilizador, que os utiliza para definir os parâmetros do *link*, indicando os extremos do *link* virtual a ser criado. O utilizador fornecerá à aplicação os dados que serão utilizados para a criação do *link* virtual:

- *IP* da interface física local;
- Endereço *ethernet* que se pretende na máquina virtual, caso seja necessário um valor específico;

- *IP* da interface física remota;
- Endereço *ethernet* da máquina virtual a criar na máquina remota;
- Largura de banda do *link* virtual;
- Compressão de dados nos extremos;
- Valor de *QoS* a colocar nos datagramas *IP* a serem trocados.

Estes dados são avaliados pela aplicação que irá validá-los, após a verificação, é criado um Objecto do tipo *Tunel* que os irá conter. É então calculado um valor que permitirá identificar *link* localmente.

A conexão virtual será identificada pelo valor que será colocado em todos os pacotes IP enviados, e é combinado pelas duas máquinas. Para se compreender a alteração que vai ser efectuada no cabeçalho *IP* é necessário estudar os campos que o contém.

O cabeçalho *IPv4* possui a forma[60] descrita na figura 5.11:

	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
0	Version				IHL				Type of service				Total length																			
1	Identification								Flags				Fragment Offset																			
2	TTL				Protocol				Header Checksum																							
3	Source Address																															
4	Destination Address																															
5	Options																Padding															

Figura 5.11: Campos presentes no cabeçalho *IPv4*

O campo protocolo tem grande relevância para a aplicação e será o campo que permitirá reconhecer que informações pertencem a um *link* virtual. O campo possui 8 *bits* e normalmente indica o protocolo de próximo nível no *payload* do pacote *IPv4*. Este varia para os diversos protocolos e pode ser consultado em [61]. Existe uma porção bastante grande de protocolos que não são usados e a aplicação utiliza-os para enumerar os *links* virtuais criados. Também o *IPv6* faz uso deste campo, alterando-lhe apenas a nomenclatura para *Next Header*, mas mantendo os 8 bits do campo[62].

O valor de protocolo é calculado de entre aqueles que estão livres, ou seja, de 143 a 252. Dos

109 valores possíveis, a aplicação escolhe um ao acaso, utilizando a função *random* e colocando-o no Objecto *Tunel* juntamente com os restantes parâmetros. O Objecto é enviado para o endereço *multicast* numa mensagem com o tipo 5, que terá a estrutura presente na figura 5.12.



Figura 5.12: Formato da mensagem 5, que permite divulgar o *link* virtual que se pretende criar

Cada nó que recebe a mensagem, verificará se algum dos extremos indicados no Objecto recebido contém um endereço de uma das suas interfaces. Se esta condição não for verificada, a mensagem recebida é descartada. Caso contrário, a informação recebida é importante e permitirá a criação do canal de comunicação. O nó verifica se algum outro *link* virtual com um extremo seu, partilha o número de protocolo com o que se pretende criar. Se sim, é computado um novo número de protocolo, da mesma forma que é feito no remetente da mensagem de criação. Este novo número de protocolo irá substituir o computado originalmente, alterando o objecto que descreve o *link* virtual. O objecto alterado é colocado numa mensagem do tipo 6 com o argumento “*NOK*” como está descrito na figura 5.13.



Figura 5.13: Formato da mensagem 6, com o campo argumento a indicar que o protocolo sugerido não foi aceite

O *host* ao receber a mensagem com o argumento “*NOK*” testa se o novo protocolo proposto está livre localmente. Se isto acontecer é enviada uma nova mensagem do tipo 5, com o Objecto contendo o protocolo sugerido, repetindo o processo inicial. O processo termina quando uma mensagem do tipo 6 com o argumento “*OK*” é entregue. O Objecto recebido permitirá descrever o *link* virtual, já com o protocolo a ser usado. Esta mensagem permite terminar o protocolo de negociação e é apresentada na figura 5.14.



Figura 5.14: Mensagem de resposta ao pedido de criação de um link virtual onde os parâmetros foram aceites

Após esta mensagem ser enviada/recebida, os extremos iniciam o processo de criação do *link*, explicado na secção seguinte. As informações das entidades virtuais criadas é inserida de imediato numa estrutura de dados que permite controla-los.

O esquema temporal de negociação do valor do campo protocolo está apresentado na figura 5.15 e permite compreender a forma como o protocolo é negociado entre os *hosts*.

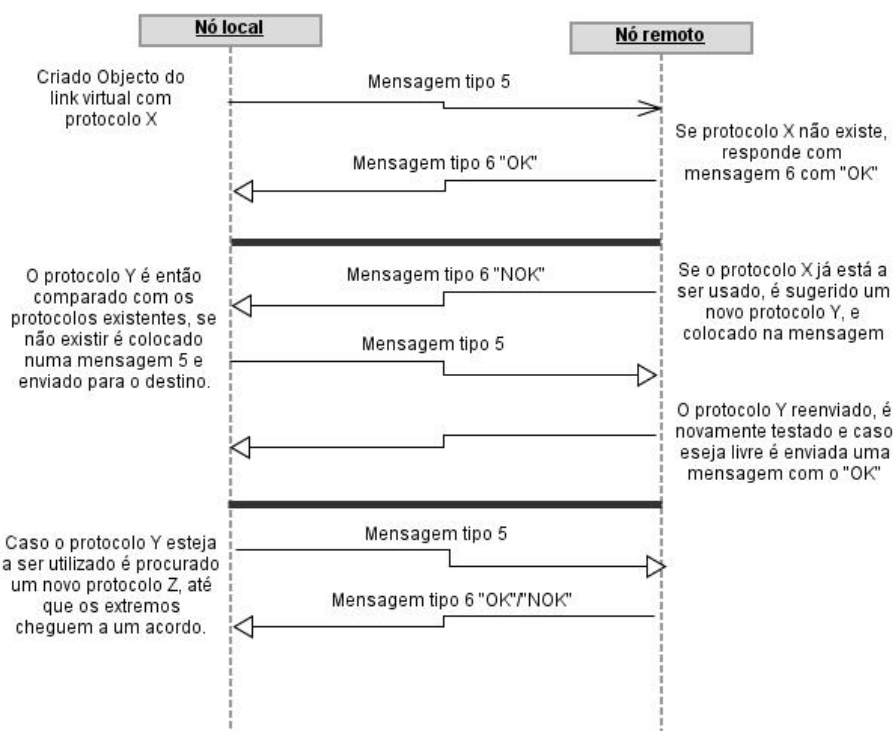


Figura 5.15: Diagrama temporal que reflecte a forma como os parâmetros da ligação são negociados

O processo de numeração de *links* vem facilitar a multiplexação e desmultiplexação das informações nos extremos. Um *link* virtual será caracterizado pelo número de protocolo que lhe está atribuído, permitindo que um nó de suporte possa ter vários nós virtualizados com vários *links* virtuais a operar sobre o mesmo *link* físico. A figura 5.16 tenta esquematizar este conceito.

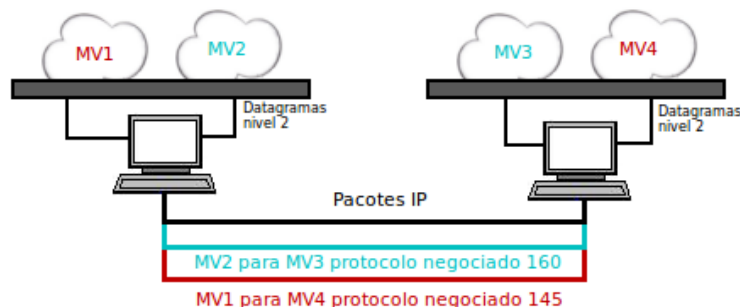


Figura 5.16: As conexões virtuais funcionarão transparentemente sobre o *link* físico

Os *links* negociados terão apenas uma validade local, ou seja, podem existir vários *links* virtuais na mesma rede virtual com o protocolo que vai ser escolhido desde que nenhum possua uma ligação com um outro nó que já possui esse protocolo. O método de endereçamento de *links*, permite na prática a construção de 109 conexões virtuais por cada nó real. Este conceito pode ser explicado pela figura 5.17.

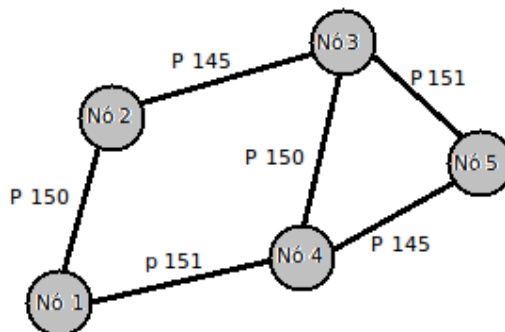


Figura 5.17: Exemplo de como os protocolos poderiam ser negociados numa rede com mais do que dois nós

A figura 5.17 pretende exemplificar um dos casos possíveis, todavia muito improváveis, de como os protocolos poderiam ser negociados. A utilização da função *random* para gerar os valores de protocolos assegura que uma situação de negociação de protocolos onde este tenha que ser renegociado é muito pouco provável. Fazendo com que a primeira mensagem do tipo 5 enviada seja na maior parte dos casos aceite de forma peremptória.

5.2.3 Criação do link virtual

O primeiro passo para a criação da conexão é a iniciação do nó virtual com as características pedidas, criando posteriormente os módulos que a permitirão trocar informação. Um dos processos recolherá o tráfego que cada nó virtual gera de forma a possibilitar o envio para o destino. É então configurada a interface de comunicação com o exterior, que permitirá receber informações geradas pelo nó remoto e marcadas com o protocolo esperado.

5.2.3.1 Interface virtual

De forma a criar o *link* virtual, é necessário a captura do tráfego gerado na interface do *host* virtual, identificado pelo endereço *mac* da interface indicada pelo utilizador. O processo inicia-se com a criação de uma *Thread* que inicia a máquina virtual e a captura de informação enviada pela interface virtual indicada pelo utilizador.

A captura de informação é possível recorrendo à biblioteca *Jpcap* e aos filtros que esta permite criar. A instância que permitirá capturar a informação será conectada à interface da máquina virtual, fazendo com que este capture a informação que é enviada. Esta procura de qual a interface a escutar recorre à biblioteca *vboxjws*, um recurso da aplicação de virtualização *VirtualBox* que permite controlar e obter informações das máquinas virtuais. Após ser criado o Objecto que estará ligado à interface pedida, é criado um filtro que permite capturar os dados enviados pela interface virtual. O filtro é conjugado com as informações fornecidas pelo utilizador, "*ether src [mac]*", e permitirá capturar os pacotes de informação enviados com o endereço *ethernet* da máquina virtual. O filtro é de extrema importância pois permite capturar o tráfego enviado pela interface e não aquele que é recebido.

O tráfego recebido do nó virtual irá, dependendo das opções do utilizador, ser comprimido e colocado dentro de um datagrama que servirá de cápsula. A cápsula terá como *IP* de destino o endereço do nó de suporte da máquina virtual remota e será marcada com o protocolo combinado entre os extremos. É também criada uma instância da classe *JpcapSender*, que recorrendo às capacidades do *JNI*[58] invocará métodos nativos que permitirão criar um *RawSocket* que enviará a cápsula com a informação recebida do nó virtual.

O encaminhamento é realizado pela rede de suporte e recorrendo às suas capacidades. O envio é realizado para o *router* que é indicado pelo sistema operativo como o primeiro salto, sendo necessário a descoberta do endereço *ethernet* do recurso que servirá de *gateway*. Esta informação é obtida recorrendo a ferramentas do sistema operativo. É pedido que o sistema execute o comando "*arp -a*" e as linhas de informação obtidas são tratadas e comparadas com os dados recolhidos sobre as interfaces existentes. A linha que possua o *gateway* de saída enumerado, é recolhida e a informação sobre o endereço *mac* que esta possui é capturada. Esta abordagem foi pensada de forma a poder ser utilizada em vários sistemas operativos, assim como indicam as figuras 5.18 e 5.19, que demonstram a utilização do comando num sistema *Windows* e *Unix*.

```

ruiferraz@ruiferraz:~$ arp -a
Ferraz-PC.lan (192.168.1.65) em 00:22:15:82:c6:b4 [ether] em eth1
dsldevice.lan (192.168.1.254) em 00:26:44:9b:21:a2 [ether] em eth1
ruiferraz@ruiferraz:~$
    
```

Figura 5.18: Descoberta do endereço mac num sistema *Linux*

```

Microsoft Windows [Versão 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. Todos os direitos reservados.

C:\Users\Ferraz>arp -a

Interface: 192.168.1.65 --- 0xb
Endereço Internet      Endereço físico      Tipo
192.168.1.64           00-26-5e-54-d3-56    dinâmico
192.168.1.253         02-26-44-9b-21-a2    dinâmico
192.168.1.254         00-26-44-9b-21-a2    dinâmico
192.168.1.255         ff-ff-ff-ff-ff-ff    estático
224.0.0.22            01-00-5e-00-00-16    estático
224.0.0.251           01-00-5e-00-00-fb    estático
224.0.0.252           01-00-5e-00-00-fc    estático
239.192.152.143       01-00-5e-40-98-0f    estático
255.255.255.250       01-00-5e-7f-ff-fa    estático
255.255.255.255       ff-ff-ff-ff-ff-ff    estático
    
```

Figura 5.19: Descoberta do endereço mac num sistema *Windows*

O conhecimento do endereço *Ethernet* do *gateway* vai permitir que os pacotes criados sejam enviados para um recurso capaz de reencaminhar a informação para o local correcto da rede. Esta medida, contudo, faz com que a informação enviada para nós de suporte que se encontrem na mesma rede realize um salto adicional no *router* marcado como *gateway*. Este método de envio, poderá no futuro ser alterado, de forma a possibilitar uma encaminhamento directo quando os nós se encontram na mesma rede. O esquema deste funcionamento está descrito na figura 5.20

A captura de informação vinda do nó virtual vai utilizar um método de *callback*, utilizado pelo *Jpcap*. Sempre que um novo datagrama é capturado, uma nova *Thread* de atendimento é criada, preparada para tratar as informações recebidas.

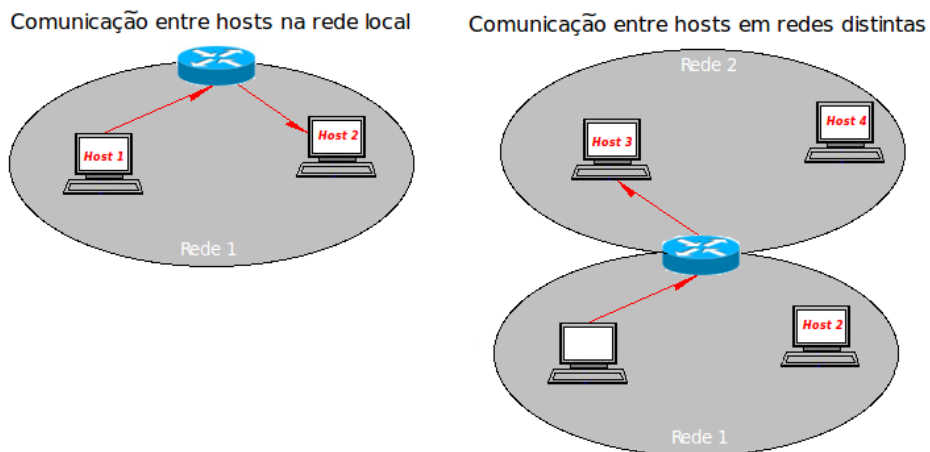


Figura 5.20: Envio de informações do link virtual para nós na mesma rede ou em redes distintas

As *Threads* criadas, utilizam o mínimo de instâncias possível e recorrem a métodos cuja execução seja rápida, tentando evitar que se criem mais *Thread* de atendimento a datagramas do que aquelas que o sistema é capaz de aguentar.

O processo de tratamento dos datagramas recebidos será incumbido de os tratar e enviar para o destino, utilizando um mecanismo de encapsulamento. As características da cápsula são alteráveis de acordo com a experiência que se pretende realizar. A figura 5.21 apresenta a forma como os pacotes são enviados pela máquina virtual e posteriormente pelo nó de suporte.

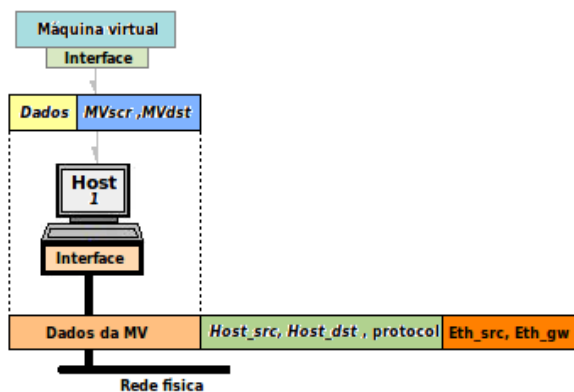


Figura 5.21: Encadeamento das informações da máquina virtual até serem enviadas

A cápsula terá como *payload* o pacote de dados recebido da máquina virtual e será enviada utilizando a instância do *JpcapSender* criada. O datagrama é enviado com o endereço *mac* do *gateway* por defeito, o *IP* do extremo e o valor do protocolo que identificará o *link*. Esta

procura do endereço ethernet tem como finalidade ultrapassar o encaminhamento realizado pelo nó, realizando a entrega.

5.2.3.2 Interface externa

De forma a completar a criação do *link* virtual é necessária a captura da informação enviada pelo *host* virtual remoto que será recebida no pacote *IP* marcado com o protocolo definido.

Será assim criada uma outra instância do tipo *JpcapCaptor* que escutará a interface de comunicação com o exterior. De forma a assegurar que apenas será capturado o tráfego enviado pelo outro extremo do *link* virtual, é criado um filtro com a semântica *"ip proto [protocolo] and not src host [IP_{interface}]"* este assegurará que apenas será capturado o tráfego marcado com o protocolo acordado. É de imperativa importância a colocação da segunda cláusula do filtro, na medida em que irá impedir que a *Thread* de tratamento dos pacotes recebidos da ligação externa, capture os pacotes enviados pela captura das informações das máquinas virtuais, o que criaria um *loop* de envio e recepção. Os pacotes capturados serão tratados e os dados que transportam no *payload* são retirados. É aqui que estarão as tramas enviadas pela máquina virtual remota e que serão passados para a máquina virtual local. Este pacote obtido do campo de dados da cápsula *IP* é então desencapsulado, descomprimido, caso seja necessário, e enviado para a máquina virtual, recorrendo a um Objecto *JpcapSender* criado previamente. Esta última acção termina a tarefa da *Thread* de atendimento.

Cada *link* virtual vai ser representado por duas *Threads*, em cada nó, que permitirão receber os pacotes marcados da ligação externa enviando-os para a máquina virtual e capturar informação da máquina virtual, que após fazer o encapsulamento, faz o seu envio para a rede.

Graficamente, a comunicação entre os dois nós, poderá ser representada pela figura 5.22.

Em cada *host*, estão dispostas duas *Threads*, representadas em verde e azul na figura 5.22, cada uma terá duas funções: Ligação à interface física, que irá capturar as informações recebidas da rede (CF - Captura da interface física), enviando-as para a máquina virtual (EF - Envio da interface física), com a cor verde; Captura da máquina virtual (CV -Captura na interface virtual) e o envio destas informações pela rede física (EV - Envio da interface virtual), com a cor azul.

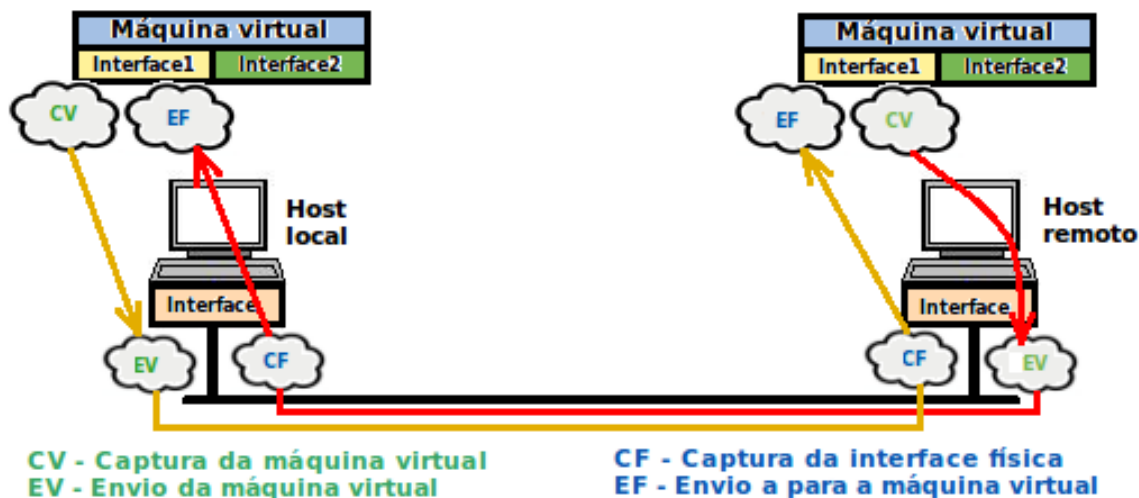


Figura 5.22: Esquema lógico de como se processa a comunicação entre dois nós

5.3 Ponto a ponto - *Sockets Unicast*

A ligação ponto a ponto poderá ser também criada usando *Sockets unicast* recorrendo às capacidades que o *Java* pode oferecer. Nesta abordagem são criadas duas *Threads* de controlo que irão controlar as informações que provêm da máquina virtual. Neste tipo de abordagem, não é necessária a criação implícita da cápsula de transporte, pois todo o processo de encaminhamento e recepção de informação é controlado pela camada de virtualização *Java*.

O modelo nesta abordagem será similar ao anterior, com a alteração do meio de comunicação entre as aplicações. Cada nó fará uso de um *DatagramSocket* que permitirá a conexão entre as duas aplicações distribuídas. Sendo possível representar cada *host* pela figura 5.23:

Cada *host* identificado pela procura de nós e que seja indicado para a criação de um *link* virtual vai iniciar dois módulos, um que escuta a máquina virtual e envia os dados recebidos para o destino, utilizando uma instância *JpcapCaptor*, e outro que recebe a informação do nó remoto, enviando-a para a máquina virtual. A comunicação entre as duas aplicações vai então acontecer implicitamente entre dois pontos criados para o efeito, de acordo com o que está representado na figura 5.24.

De forma a instanciar a conexão o utilizador fornece à aplicação, utilizando os dados obtidos

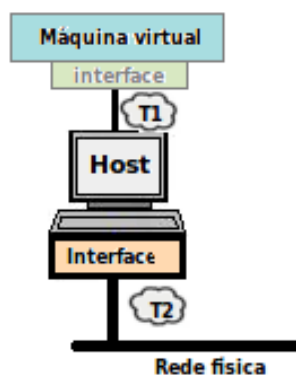


Figura 5.23: Representação lógica de como as *Threads* se posicionam nas interfaces que pretendem escutar

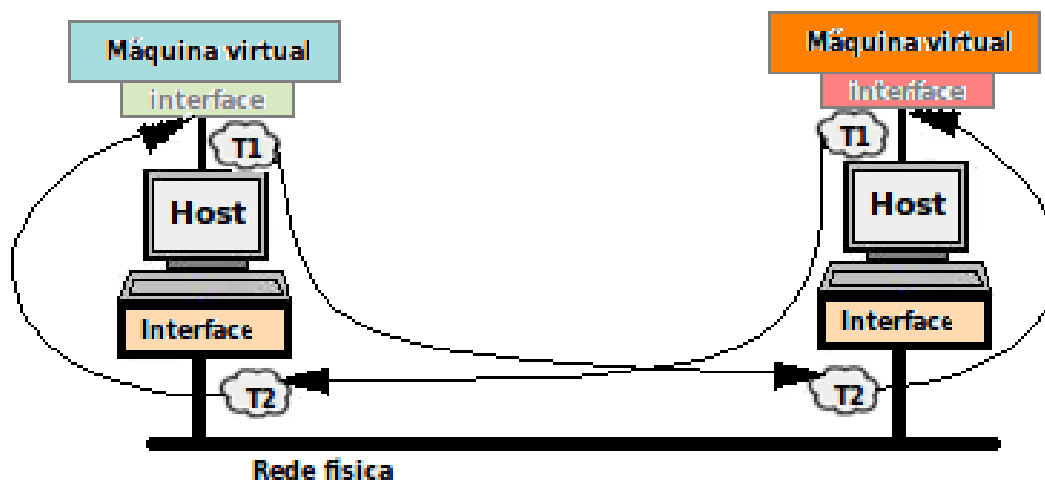


Figura 5.24: Forma como os *DatagramSockets* trocarão informação entre os processos

com a procura de recursos, as características do *link* que permitirão a criação das *Threads* para a comunicação entre os processos de cada *host* e para a troca de informação entre as máquinas virtuais.

Os *IP* dos nós extremos, o nome e a interface da máquina virtual e a necessidade de compressão de informações são inseridos na interface gráfica. Com esta informação é criado um Objecto do tipo *Tunel*, à imagem do que ocorria na abordagem com *RawSockets*, a principal diferença é não necessitar de negociação de protocolos, pois os dados são encaminhados pelo *DatagramSocket*.

Após as características do *link* serem avaliadas pela aplicação, testando a conectividade de cada um dos extremos, são enviados, recorrendo a um *Socket TCP* que é conectado ao nó remoto que será o extremo do *link* virtual. Após esta troca de dados o *Socket* de controlo é terminado e dará lugar aos *DatagramSockets* da conexão apresentados na figura 5.24.

5.3.1 Interface externa

A interface externa será responsável pela captura dos pacotes enviados pelo nó remoto, colocando-os ao dispor da máquina virtual.

A *Thread* que trata da interface externa instanciará um *DatagramSocket* que permitirá receber informação do nó de suporte remoto e estará à escuta numa porta específica. O *host* remoto enviará *DatagramPackets* para a porta onde o *DatagramSocket* está à escuta.

Sempre que um datagrama é recebido o seu conteúdo é individualizado, descomprimido, caso seja necessário, e convertido num objecto do tipo *Packet*. O objecto é então enviado para a máquina virtual, utilizando uma instância do classe *JpcapSender*, previamente conectada à interface da máquina virtual.

5.3.2 Interface virtual

A interface virtual implementa um método de captura das informações enviadas pela máquina virtual, escutando a interface que foi indicada para a conexão.

O processo que realiza a captura recorre a uma instância da classe *JpcapCaptor* que será conectada à interface da máquina virtual, capturando as informações que esta envia.

Para que a captura se processe da forma pretendida, tratando apenas os datagramas enviados pela interface do nó virtual, é necessário filtrar os que são recebidos. É configurada a instância *JpcapCaptor*, definindo os parâmetros de captura pretendidos e configurando o filtro com a semântica "*ether src [mac]*" onde *[mac]* corresponde ao endereço *ethernet* passado pelo utilizador, ou caso este não tenha sido especificado, o endereço da interface pedida.

A informação recebida pela máquina virtual será comprimida, caso o utilizador assim o pretenda, e enviada pelo *DatagramSocket* para uma porta pré-definida. O *host* remoto, estará pronto

a receber estes dados, descomprimindo-os, caso seja necessário, e passando-os para a máquina virtual.

5.4 Ponto a multi-ponto - RawSocket

Uma rede de nível 2 é normalmente um meio partilhado entre vários nós, onde as mensagens enviadas para o meio de comunicação são partilhadas por todos os *hosts* na rede. Para que esta abordagem ao problema fosse possível, foi projectada a melhor forma de virtualizar esta ligação, utilizando as máquinas virtuais como extremos de uma conexão de rede multi-ponto.

Recorrendo às tecnologias conhecidas, é possível criar uma rede partilhada de difusão por vários utilizadores, onde as mensagens enviadas por um, são partilhadas por todos os que as pretendam receber. É desta forma que o problema foi abordado, criando uma rede baseada no *multicast IP* e partilhada entre os *hosts* que pretendam participar.

A forma de por em prática esta solução foi abordada de duas maneiras, recorrendo a um *RawSocket* com a criação de todas as mensagens que são trocadas e utilizando um *Multicast-Socket*. A troca de mensagens utilizando esta tecnologia, está representada na figura 5.25

Com a iniciativa de se criar um novo *link* virtual multi-ponto, a aplicação junta-se a um grupo *multicast* de controlo, que permite a comunicação entre os *nós* físicos. Neste grupo são anunciados os endereços de outros grupos que representam *links* multi-ponto activos.

Na abordagem multi-ponto recorrendo a *RawSockets* a comunicação com o grupo de controlo é implementada utilizando uma instância *MulticastSocket*, enquanto que os *links* de comunicação recorrem a *RawSockets* para o envio e recepção de informação. Esta ideia está esquematizada na figura 5.26.

A criação do *link* multi-ponto iniciar-se-à com a junção ao grupo *multicast* passado por parâmetro. É então criada uma instância da classe *JpcapSender*, que permite o envio de informações, utilizando um *RawSocket*.

Esta abordagem necessita que se compreenda de que forma se processa a criação, manutenção e eliminação de um grupo *multicast*, de forma a que se consiga simular o protocolo.

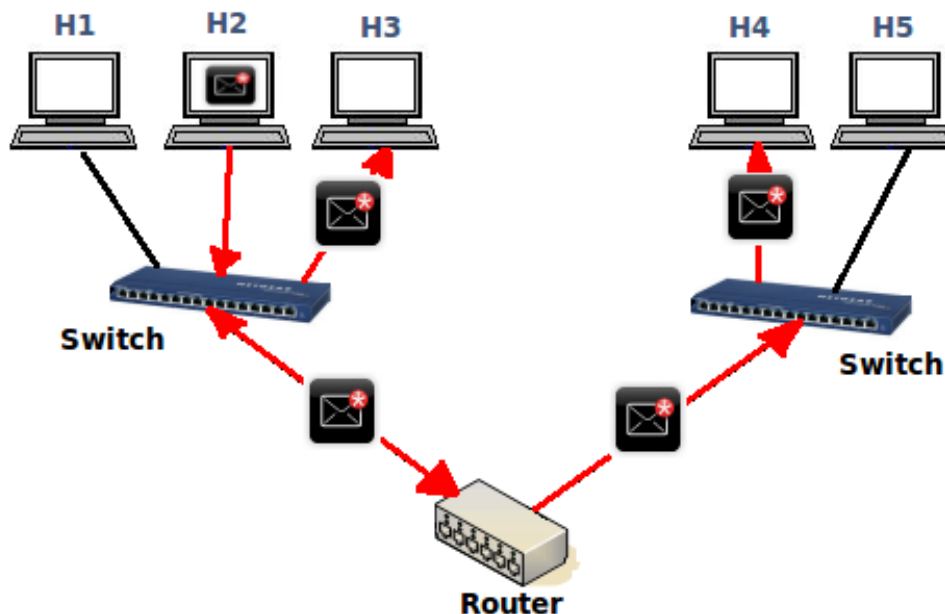


Figura 5.25: Representação de uma troca de mensagens utilizando o *multicast*

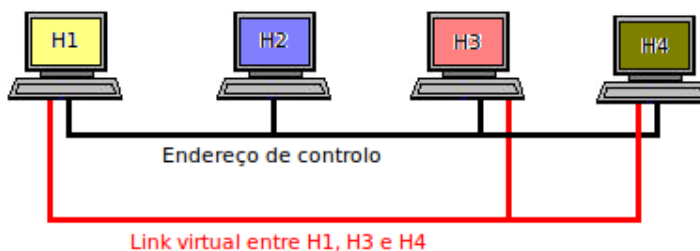


Figura 5.26: O grupo de controlo permitirá anunciar a criação de um endereço para troca de mensagens entre nós virtuais

As mensagens trocadas serão definidas pelo protocolo *IGMPv3*[63], e serão usadas, tanto pelo *router*, como pelo *host* para criar o grupo *multicast*.

As mensagens *IGMP* são encapsuladas em pacotes *IP* marcados com o número de protocolo 2. As mensagens trocadas pelo protocolo, na óptica do nó que se junta a um grupo, poderão apresentar duas formas:

- *Membership query* - Onde é enviado o pedido de participação no grupo *multicast*;
- *Membership report* - Que controla o estado do nó no grupo *multicast*.

A finalidade e formato da mensagem *IGMPv3* enviada é determinada pelo valor que possui no campo *type*:

- *0x11* - *Membership Query*
- *0x22* - *Version 3 Membership Report*
- O *IGMPv3* deve também ser capaz de receber e tratar os tipos de mensagem enviados pelas versões anteriores:
 - *0x12* - *Version 1 Membership Report*
 - *0x16* - *Version 2 Membership Report*
 - *0x17* - *Version 2 Leave Group*

A mensagem *Membership Query* será utilizada pelo *router* para questionar as interfaces vizinhas acerca do seu estado de recepção, as *queries* têm o formato descrito na figura 5.27:

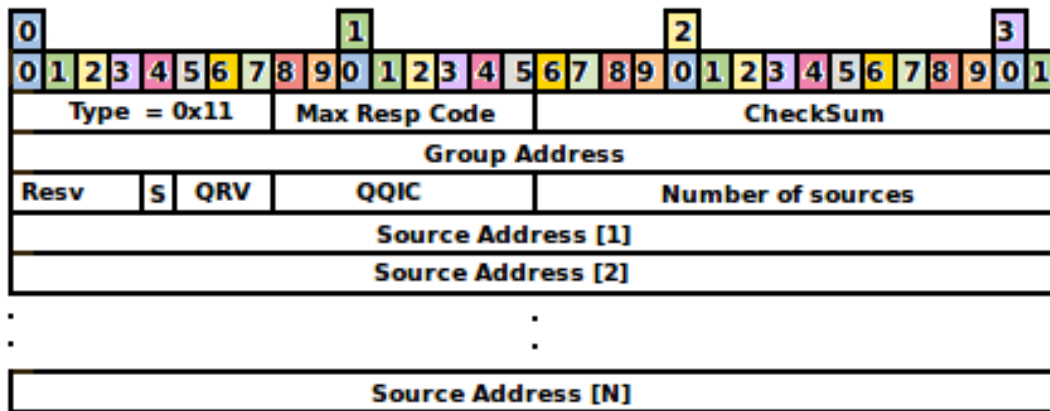


Figura 5.27: Formato da mensagem *Membership Query*, enviada pelos routers que suportam *Multicast*

Os campos da mensagem apresentada na figura 5.27 terão o seguinte significado:

- *Max Resp Code* - Campo que especifica o tempo máximo permitido até que seja enviado um relatório;

- *Checksum* - Campo que permite validar o datagrama *IGMP* recebido;
- *Group Address* - Endereço *multicast* visado aquando do envio de *queries Group-Specific*, ou *Group-and-Source-Specific*;
- *Resv* - Campo reservado, colocado a zero no envio e ignorado na recepção;
- *S - Flag* de indicação de supressão da actualização dos temporizadores;
- *QRV* - Variável de robustez usado pelo emissor da *Query*;
- *QQIC (Querier's Query Interval Code)* - Usado para especificar o intervalo de tempo entre *queries*;
- *Number of sources* - Especifica o número de endereços presentes na *query*;
- *Source Address [i]* - Conjunto de endereços *unicast*.

Os *Membership report* são enviados pelos sistemas para reportar aos *routers* vizinhos, o estado de recepção *multicast*, ou alterações nas interfaces. Estes relatórios possuem o formato apresentado na figura 5.28:

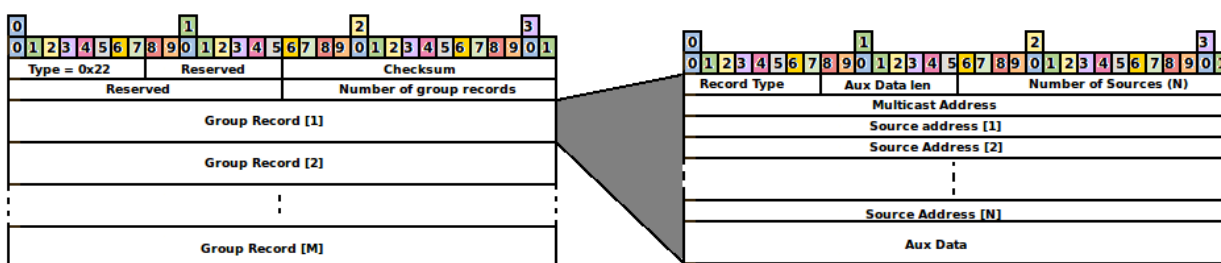


Figura 5.28: Formato da mensagem *Membership Report*, enviada pelos nós de forma a definirem a sua junção ao grupo *multicast*

Onde os campos terão o seguinte significado:

- *Reserved* - Campos marcados a zero na transmissão, e ignorados na recepção;
- *Checksum* - Campo de controlo da mensagem, permitindo validar a sua integridade;

- *Number of Group Records* - O número de *Groups Records* presentes no *report*;
- *Group Record* - Bloco de informação que contém informação de um *host* num grupo *multicast*.

O campos presentes no *group record* terão o seguinte significado:

- *Record Type* - Campo que vai permitir diferenciar a tarefa do *group record* e pode ter os seguintes tipos:
 - *Current-State Record* - Enviado por um sistema em resposta a uma *query* recebida ao qual responde com o estado da interface para cada grupo a que esta pertence;
 - *Filter-Mode-Change Record* - Enviado por um sistema quando uma invocação local de *IPMulticastListen* causa uma alteração no modo de filtragem da interface para um grupo *multicast*;
 - *Source-List-Change Record* - Enviado por um sistema quando uma invocação de *IPMulticastListen* causa a alteração de um parâmetro;
- *Aux Data length* - Campo que contém o tamanho do campo *Aux Data*;
- *Number of Sources* - Campo que especifica o número de endereços presentes;
- *Multicast Address* - Endereço *multicast* ao qual este *group report* pertence;
- *Source Address* - Conjunto de endereços *IP unicast*;
- *Aux Data* - Contém informação adicional pertencente ao *Group Record*, o *RFC* que contém a descrição do protocolo não define este campo, logo não deve ser usado devendo ser referenciado para fins futuros.

Ao contrário do que acontecia com as versões 1 e 2 do *IGMP*, na versão 3 foi criado um endereço *multicast* onde todos os relatórios são trocados. O endereço 224.0.0.22, tornou-se um ponto comum que todos os *routers* capazes de processar *IGMPv3* escutam. Um sistema que opere com a versão 1 e 2 do *IGMP*, envia os relatórios para o grupo *multicast* presente no *Group Address* do *Membership report*.

Com este conhecimento, é possível compreender a forma como o *IGMPv3* processa o seu protocolo de junção, manutenção e saída de um grupo *multicast*[7].

Membros que se pretendam juntar a um grupo *multicast* enviam um relatório *IGMPv3* para o endereço *224.0.0.22*. Isto permite que a junção seja assíncrona, não sendo necessário que estes recebam uma mensagem para se juntar ao grupo. A figura 5.29 apresenta um exemplo de uma junção a um grupo *multicast*.

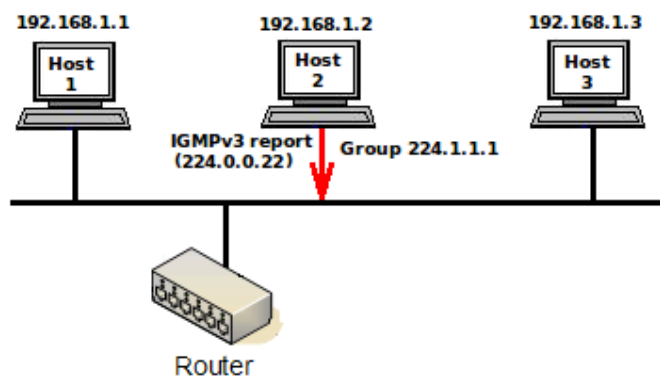


Figura 5.29: Imagem adaptada [7]. Junção a um grupo *multicast*, utilizando *IGMPv3*, onde a mensagem é enviada para o grupo *224.0.0.22*

O *report* poderá conter os *hosts* que se pretende ou não escutar, isto permite que se controle o tráfego escutado, filtrando fontes de tráfego indesejadas. A operação está ilustrada na figura 5.30 e é realizada utilizando a *include list* e a *exclude list* presente no *group record* do relatório enviado.

O *router* envia para o grupo de todos os *hosts*, *224.0.0.1*, *Membership Queries* periódicos, ao que todos os *hosts*, responderão com um *IGMPv3 membership report* que contém o estado da interface nos vários grupos. Este processo é apresentado na figura 5.31.

Ao contrário do que ocorre no *IGMPv1*, onde a saída de um grupo não necessita de ser comunicada, na versão 2 e 3 do *IGMP* a saída do grupo é comunicada pelo *host*. Enviando a mensagem para o grupo na versão 2, ou para o grupo de *reports* na 3^a, ao que o *router* responde com uma *query* para o grupo de forma a verificar se ainda existem nós no grupo, eliminando-o

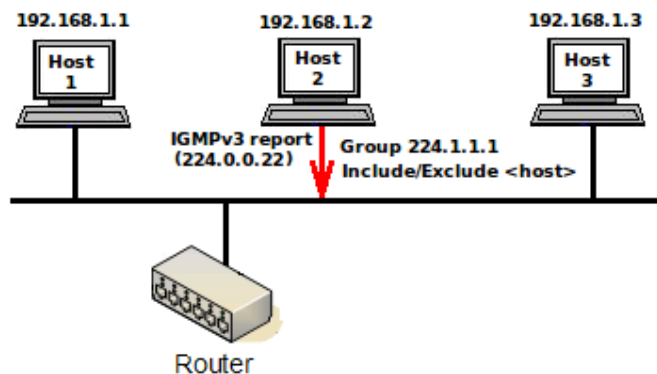


Figura 5.30: Imagem adaptada [7]. Alteração dos nós que se pretende escutar do grupo *multicast*

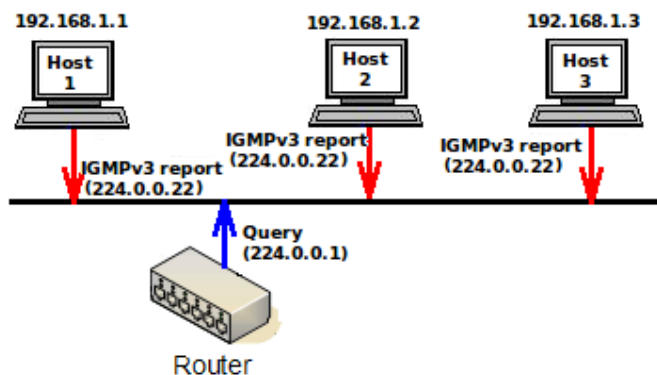


Figura 5.31: Imagem adaptada [7]. Resposta dos nós que pertencem ao grupo *multicast* a uma *Membership Queries*

se não existirem interessados. Tal como representado pela figura 5.32

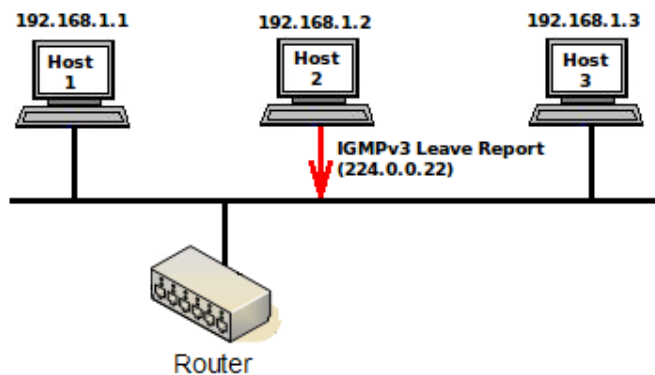


Figura 5.32: Imagem adaptada [7]. Saída implícita de um grupo *multicast*

Esta explicação pretende apresentar a forma como é criado e mantido um grupo *multicast* uma vez que, com a utilização de *RawSockets*, todas as mensagens que seriam enviadas pelo protocolo *multicast* terão de ser preparadas e enviadas directamente pela aplicação.

A junção a um grupo *multicast* é o primeiro passo, para isto é necessário que seja enviado um datagrama de criação do grupo. Deve-se criar uma instância da classe *JpcapSender*, que permite o envio de informações para o canal de comunicação. É também instanciado um pacote *IP*, com um sub-cabeçalho *UDP*, marcado com o protocolo 2, que conterà a mensagem *IGMP*. As mensagens de relatórios enviadas por um *host* que queira pertencer a um grupo, são enviadas para o endereço *224.0.0.22*, qualquer que seja o endereço *multicast* escolhido para a criação do *link* virtual.

A mensagem que se pretende enviar para o grupo de controlo terá de possuir os endereços, tanto *IP* como *ethernet*, que o identificam. O endereço *ethernet* para onde a mensagem deve ser enviada é computado com base no *IP* do grupo, o cálculo de um a partir do outro é simples e está representado na figura 5.33.

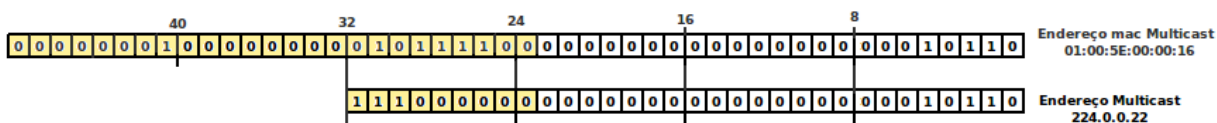


Figura 5.33: Cálculo do endereço *ethernet* a partir do endereço *IP* de um grupo *multicast*

Os primeiros 25 *bits* do endereço *multicast* são fixos, e informam que o endereço *ethernet* pertence a um grupo *multicast*, os restantes serão os 23 *bits* menos significativos do endereço *multicast*. Para o endereço *IP* 224.0.0.1 o endereço *ethernet* correspondente será 01:00:5e:00:00:01. Após ser construído o datagrama *ethernet*, instanciando um Objecto do tipo *EthernetPacket*, é ligado à instância do Objecto *IPPacket* ambos previamente configurados com os endereços. Resta construir a mensagem *IGMP* que será transportada pelo datagrama.

Tendo em conta o modo de funcionamento do *IGMP*, deve ser enviada inicialmente uma mensagem *IGMP* do tipo 22. Esta necessitará de ser construída utilizando um *array* de *bytes*, controlando o sítio onde cada informação deverá ser colocada, assegurando que se segue o esquema de mensagem apresentado anteriormente. A mensagem enviada servirá para informar o *router* da criação do grupo *multicast*. Desta forma, o primeiro *byte* a ser colocado no *array* é o tipo de mensagem que se pretende enviar, o *byte* 22. O próximo *byte* é um campo reservado e

é colocado, a zero[63]. O campo *checksum* da mensagem *IGMP* será o último a ser colocado e será representado por 16 *bits* que serão calculados pela soma do complemento para um do complemento para um de toda a mensagem *IGMP* com o campo *checksum* colocado a zero[64]. Para realizar esta tarefa, é usado um *buffer* de *bytes*, ao qual vão sendo retirados dois *bytes* em cada ciclo. Estes são colocados, recorrendo a máscaras, numa variável e desta é calculado o complemento para um e somado ao valor de *checksum* já existente. Em cada soma, é retirado o excesso, sendo que no fim de todo o ciclo é calculado o complemento para um da soma obtida.

O próximo campo é o grupo ao qual o nó se vai juntar colocando, na mensagem *IGMPv3*, o endereço *multicast* em *bytes*. O conjunto de *bytes* que formam a mensagem é então colocado no *payload* do pacote *IP*, enviando-o pelo *RawSocket*.

Neste momento o processo está ligado ao grupo *multicast* e preparado para passar para a segunda parte do algoritmo. De forma a que a ligação ao grupo *multicast* se mantenha activa é também criada uma *Thread* que permite o envio de *Membership reports* periódicos para o grupo de *224.0.0.22*.

Neste ponto é criada a instância da máquina virtual e a *Thread* que permite capturar o tráfego que desta provém. A captura recorre a uma instância da classe *JpcapCaptor* à qual é aplicado um filtro que permite que sejam capturados os datagramas que provém do nó virtual, o filtro possuirá a semântica "*ether src [mac]*". Os datagramas que provém da máquina virtual são então colocados dentro de um pacote *IP*, enviando-o para o endereço *multicast*. É colocado no *IP* de destino o grupo que facultará a troca de informação e o endereço *ethernet* que será obtido usando a mesma conversão explicada na imagem 5.33. Os datagramas poderão ser ainda marcados com um protocolo, que à imagem do que acontecia com os *links* virtuais *unicast*, permitem a criação de várias conexões virtuais, utilizando o mesmo endereço *multicast*. Esta medida vem maximizar a utilização dos grupos *multicast*, permitindo que várias redes virtuais funcionem sobre o mesmo grupo.

Ao mesmo tempo é iniciado o processo de captura da ligação externa. Este vai filtrar os datagramas que chegam ao *host*, permitindo capturar apenas os que interessam à aplicação. Para isto é criado um filtro com a semântica "*ip proto [IP_protocol] dst host [IP_multicast]*", caso exista um número de protocolo atribuído, ou apenas "*dst host [IP_multicast]*" caso este não seja proposto. Isto fará com que apenas sejam disponibilizados para a aplicação datagramas que são enviados para o grupo *multicast* e marcados com o protocolo escolhido. A aplicação vai analisar os datagramas recebidos, retirando-lhes o *payload* e enviando-o para a máquina virtual usando uma

instância do objecto *JpcapSender*. O funcionamento do *link* multi-ponto está esquematizado na figura 5.34.

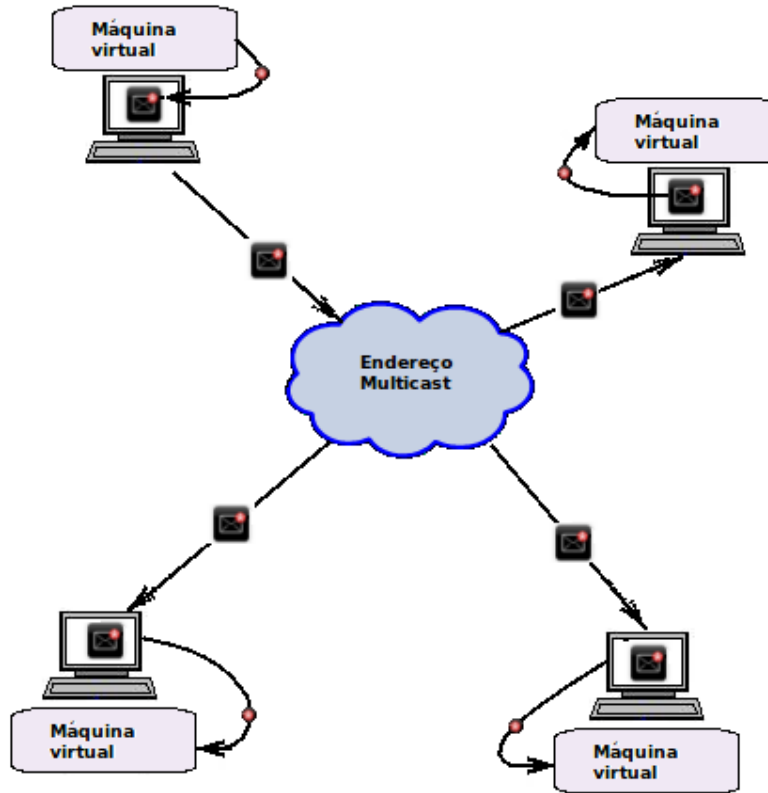


Figura 5.34: Troca de mensagens num *link* multi-ponto, onde uma mensagem enviada para o grupo é passada para todos os nós

Uma mensagem enviada por uma máquina virtual é encapsulada num pacote *IP* e encaminhada para o endereço *multicast*. Todos os outros nós, conectados ao grupo e utilizando o mesmo protocolo, capturarão o datagrama. Retirando o *payload*, que será entregue à máquina virtual local.

5.5 Ponto a multi-ponto - *MulticastSockets*

Uma conexão multi-ponto pode ser também criada usando uma instância do *MulticastSocket* disponibilizada pelo *Java*. Esta implementação será similar à solução que recorre ao *RawSocket*, executando o protocolo de ligação a um grupo *multicast* de forma transparente.

É instanciado um Objecto do tipo *MulticastSocket* que permitirá receber os anúncios no grupo de controlo, de onde se receberão mensagens periódicas dos outros nós sobre os grupos criados.

Sempre que um novo *link multicast* é criado, as suas informações são adicionadas ao gestor de túneis. Estas informações serão divulgadas por uma *Thread* incumbida de anunciar os endereços *multicast* criados localmente, que serão enviados numa mensagem do tipo 7, representada na figura 5.35, para o endereço *multicast*.



Figura 5.35: Formato da mensagem de tipo 7, que permite anunciar os *links multicast* criados

As mensagens do tipo 7 terão a lista dos grupos criados localmente, este mecanismo tenta diminuir a quantidade de informação e mensagens que cada máquina envia para o meio.

Um nó pode escolher entre juntar-se a um grupo previamente criado, ou criar um novo, introduzindo os parâmetros do novo *link* virtual. A aplicação processa a junção ao grupo utilizando uma instância do *MulticastSocket*. É também iniciado o nó virtual pedido, alterando as suas características de acordo com o que foi requisitado pela interface gráfica.

Os dados enviados pelo nó virtual serão capturados da mesma forma que nas abordagens anteriores, recorrendo a uma instância *JpcapCaptor*, configurada com um filtro com uma semântica que o permitirá obter as informações enviadas pela interface escolhida.

A abordagem não necessita da criação de um mecanismo de captura da interface física, pois o tráfego que se pretende receber será trocado apenas a partir do *MulticastSocket*. Para cada datagrama recebido por este meio é criada uma *Thread* de tratamento para o qual são passados o datagrama recebido e uma instância da classe *JpcapSender*, previamente criada e ligada ao nó virtual. Cada trama recebida iniciará uma *Thread* que estará incumbida de obter o *payload* do datagrama capturado, trata-lo e envia-lo para a máquina virtual.

As diferenças entre a abordagem descrita neste ponto e a anterior são a transparência de processos para o programador e a velocidade de envio de informação para o meio. Na abordagem que recorre aos *MulticastSockets* a junção a um grupo não necessita que seja conhecido o protocolo de junção, nem tão pouco as mensagens trocadas.

5.6 Portabilidade da rede virtual

Uma das mais valias oferecidas pela rede virtual é a portabilidade, a capacidade de utilizar uma configuração numa outra rede de suporte torna a criação de uma infra-estrutura de rede virtual mais rápida e oferece ao utilizador a capacidade de criar a rede parametrizada e com as características esperadas num outro suporte.

De forma a obter esta funcionalidade foi adicionado ao sistema a capacidade de guardar a configuração da rede num ficheiro *XML* que a permitirá arrancar no estado em que se encontrava.

Quando se cria um dos *links* que a rede virtual oferece, as suas características são guardadas numa estrutura que permitem controlar os recursos usados. A aplicação, após a criação do ficheiro de rede, inicia a sua escrita. Utilizando as listas de *links* ponto a ponto e multi-ponto para guardar cada uma das instâncias virtuais sobre a forma de *XML*. O ficheiro guardará os dados numa forma equivalente à que foi introduzida na interface gráfica, podendo ser definidos de 4 tipos diferentes dependendo da forma como o *link* é criado.

Os atributos para os *links* ponto a ponto serão os seguintes:

- **Tipodelink** - Define qual o tipo de *link* virtual a que pertencem os atributos, possui os valores "lvsu" ou "lvru";
- **hostLocal** - Indica o *IP* do nó de suporte de um dos extremos;
- **hostDestino** - Indica o *IP* do outro extremo do *link* virtual;
- **macVmL** - Indica o endereço *ethernet* da interface do nó virtual que será um dos extremos do *link* virtual;

- **macVmR** - Endereço *ethernet* da interface do nó virtual que será o outro extremo do *link* virtual;
- **proto** - Número do protocolo a ser utilizado no *link* virtual ponto a ponto, este valor vem a zero caso o tipo de *link* seja "lvsu";
- **compress** - Permitirá indicar se o *link* possui compressão de informação;
- **LB** - Largura de banda máxima no *link* criado;
- **VML** - Indicação de qual a máquina virtual que será um dos extremos do *link* virtual;
- **VMR** - Qual a máquina virtual a arrancar no outro extremo do *link*;
- **QOS** - Valor de *QOS* a colocar nas cápsulas que permitirão transportar os dados dos nós virtuais;
- **interfacelocal** - Número da interface do nó virtual de um dos extremos que será ligada ao *link* virtual e que terá o atributo *macVmL*;
- **interfaceremota** - Número da interface do nó virtual do outro extremos que estará conectada ao link e que terá o endereço *ethernet macVmR*.

As informações dos *links* multi-ponto, poderão também ser guardadas no ficheiro, de forma a ser possível inicia-los numa outra rede de suporte:

- **Tipodelink** - Permite indicar se o link será do tipo "rsm"(multi-ponto utilizando *MulticastSockets*) ou "rsm"(multi-ponto recorrendo a *RawSockets*), os dois tipos de *link* multi-ponto que a aplicação é capaz de oferecer;
- **iphost** - Indica qual o *IP* do nó de suporte que se conectará ao *link* multi-ponto;
- **enderecomulticast** - Indica qual o grupo *multicast* que proporciona a troca de informação entre os nós virtuais;
- **porta** - Qual a porta que o *MulticastSocket* deverá utilizar;
- **compress** - Informa se se devem comprimir os dados recebidos do nó virtual;
- **macVmL** - Endereço *ethernet* da interface do nó virtual;

- **numInter** - Número da interface do nó virtual, que deve possuir o atributo *macVmL*;
- **maqVirt** - Qual a máquina virtual que deve ser arrancada localmente;
- **LB** - Largura de banda a atribuir à conexão virtual;
- **qos** - Valor de *QoS* a atribuir às cápsulas que serão enviadas;
- **proto** - Número do protocolo que será atribuído à conexão.

O ficheiro terá as indicações dos *links* virtuais, criando uma representação legível e compreensível da rede criada. As informações neste ficheiro poderão ser posteriormente alteradas de acordo com o que o utilizador pretende criando uma rede virtual à medida da rede de suporte e de acordo com as características que deve possuir.

Num momento em que se pretenda reiniciar a rede virtual, é pedido à aplicação que receba o ficheiro e que crie as instâncias virtuais a partir dele. Os campos do ficheiro são individualizados utilizando a biblioteca *Jdom*[65] e após serem tratados, atestando a sua validade, e capacidade de comunicar de cada um dos extremos, são criados os objectos que representarão cada uma das entidades virtuais.

Para cada conexão virtual encontrada no ficheiro são verificados os endereços que identificam os nós da rede de suporte, e caso o *IP* que este parâmetro apresenta não seja o mesmo do nó que recebeu o ficheiro de configuração, é iniciado o envio para o *host* identificado pelo endereço *IP* no parâmetro *hostLocal*, caso uma comunicação ponto a ponto, ou *iphost*, numa multi-ponto. O envio da informação será realizado recorrendo a um *Socket TCP*. A aplicação possui um *ServerSocket* à escuta da porta 7000 de forma a receber informações de rede. O nó que recebe este pedido será um dos extremos do *link* virtual e iniciará a sua criação, não necessitando de realizar uma procura de recursos, pois os *hosts* do ficheiro são conhecidos previamente.

O envio da informação das entidades de rede entre os nós extremos é realizado utilizando *Sockets TCP*, da mesma forma que o envio explicado no parágrafo anterior, conectando-se com o *IP* do *host* remoto presente no Objecto de tipo *Tunel* recebido do *host* que tratou o ficheiro de criação da rede virtual.

5.7 Configuração e manutenção das máquinas virtuais

As máquinas virtuais serão o ponto central da aplicação desenvolvida, uma vez que serão os extremos dos *links* virtuais e permitirão construir a infra-estrutura de rede virtual pretendida.

Para que as máquinas virtuais inicializadas, com o pedido de criação de um novo *link*, possam utilizar a rede virtual de forma transparente, é necessário que sejam configuradas previamente. Um nó virtual terá de possuir um ambiente controlado, onde o tráfego enviado apenas terá de fazer sentido no contexto criado.

Cada nó virtual terá as suas interfaces de rede conectadas a interfaces criadas no sistema operativo de suporte, desta forma qualquer informação que seja enviada do nó virtual é entregue no sistema de suporte que tratará a informação recebida de acordo com o tipo de *link* criado. Este tipo de conexão comumente designada "*Host only connection*" permite manter o sistema operativo convidado confinado à comunicação com um sistema hospedeiro, fazendo com que apenas obtenha informação que lhe é passada, numa abordagem contida de rede.

De forma a ser possível virtualizar os nós, recorreu-se à ferramenta *Open Source VirtualBox*. Esta ferramenta, permite uma virtualização total do sistema e disponibiliza uma *API* que permite a alteração dos parâmetros utilizados pela camada de controlo.

A biblioteca disponibilizada pela *VirtualBox* permite a alteração de alguns dos parâmetros da camada de virtualização. Criando uma instância do Objecto *VirtualBoxManager* que após se conectar ao *webservice* da camada de virtualização que dá acesso à *API*, permitirá alterar e consultar os parâmetros que esta fornece ao nó virtual.

A alteração dos parâmetros da camada de virtualização pode por vezes não ser suficiente. Por esta razão, foi implementado um sistema de configuração interna do nó virtual. Um *daemon* que fica à escuta numa porta no sistema operativo virtualizado. Que permite que a aplicação se conecte ao nó virtual e faculte a alteração de alguns parâmetros que a camada de virtualização da *VirtualBox* não permite. Para isto o *daemon* possuirá um *DatagramSocket* à escuta na porta 15000 que será iniciado com o arranque do sistema. A aplicação enviará informação utilizando um *Socket* que será criado recorrendo à interface gráfica de controlo do nó virtual, ou seja, a ligação com a máquina virtual recorrendo a este método só acontecerá se o utilizador assim o pretender.

Capítulo 6

Resultados

Este capítulo permitirá apresentar os resultados obtidos para os testes realizados sobre cada uma das abordagens de criação de *links*. A criação de uma rede virtual foca-se na forma como os *links* são criados e nas características que são capazes de oferecer, largura de banda efectiva, perdas e latência na conexão são parâmetros importante que os testes vão permitir avaliar.

Os testes foram realizados utilizando a aplicação desenvolvida em *Java* e três nós, representados por portáteis que a executam. Os nós utilizados situam-se em redes distintas, com um *router* a servir de encaminhador. Cada suporte possui o sistema operativo *Ubuntu* enquanto que os nós virtualizados utilizam o *FreeBSD 8.2*.

As características dos nós utilizados pode ser consultada na tabela 6.1:

Processador	Número de <i>CPU</i>	Memória	Disco rígido	Sistema Operativo
<i>Intel(R) Core(TM)2 Duo 2.10GHz</i>	2	6 Gb	320 Gb	<i>Ubuntu 10.10</i>
<i>Intel(R) Core(TM)2 Duo 2.20GHz</i>	2	2 Gb	160 Gb	<i>Ubuntu 11.04</i>
<i>Intel(R) Core(TM)2 Duo 1.73GHz</i>	2	2 Gb	120 Gb	<i>Ubuntu 10.10</i>

Tabela 6.1: Características dos nós utilizados

A rede de suporte utilizada para os testes possui a forma apresentada na figura 6.1, e utiliza a infra-estrutura de rede disponível nos laboratórios de Comunicações do Departamento de Informática da Universidade do Minho.

Com este cenário foram realizados os vários testes sobre a rede virtual, abordando cada uma das formas de criação de *links* e testando os parâmetros de largura de banda utilizando conexões

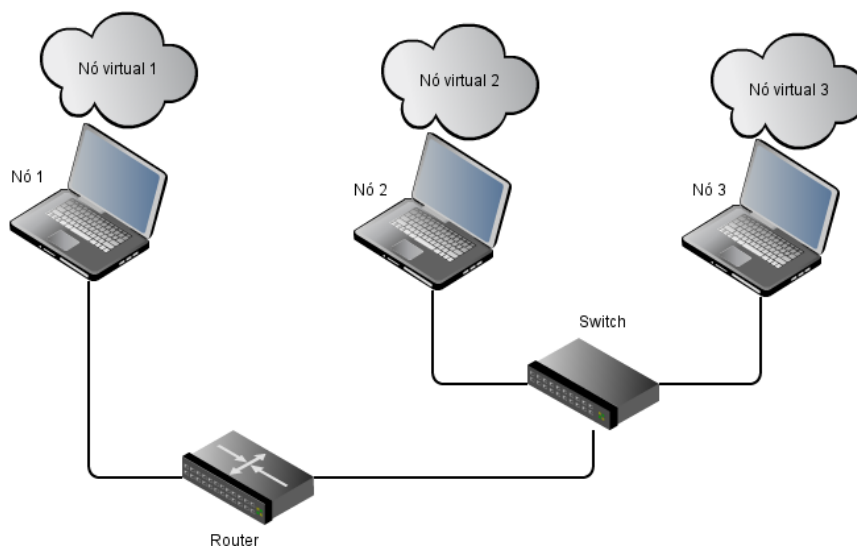


Figura 6.1: Infra-estrutura de suporte utilizada para realização dos testes

TCP e *UDP* de prova, avaliando as perdas e atrasos registados nesses fluxos.

Foram utilizadas as 4 abordagens de criação de *link* para os testes, as quais poderiam ou não utilizar compressão no envio de informação. De forma a medir as capacidades dos *links* virtuais, foi utilizada a ferramenta *Iperf*. Esta permite criar uma *stream* de dados entre dois pontos na rede, e avaliar a comunicação. Os diferentes tipos de *links* são designados de acordo com a legenda da tabela 6.2.

Inicialmente os testes foram realizados na rede de suporte, de forma a obter um valor comparativo para as medida obtidas.

Os testes multi-ponto por seu lado, recorreram a toda a infra-estrutura de suporte disponível para a realização dos testes, utilizando os três nós de suporte para a conexão.

<i>LVRU</i>	Link virtual utilizando <i>RawSockets Unicast</i>
<i>LVSU</i>	Link virtual utilizando <i>Sockets Unicast</i>
<i>RMSM</i>	Rede Multi-ponto utilizando <i>Sockets Multicast</i>
<i>RMR</i>	Rede Multi-ponto utilizando <i>RawSockets</i>

Tabela 6.2: Acrónimos utilizados para as abordagens de criação de *link* virtual

6.1 Largura de banda virtual com uma conexão *TCP*

O primeiro teste realizado pretende criar uma conexão *TCP* entre dois nós sobre o *link* virtual, testando a velocidade máxima de troca de informação. Foram realizadas 10 medições em cada um dos *links* virtuais implementados, utilizando a aplicação desenvolvida para realizar a troca de informação. Os resultados obtidos deste teste podem ser observados pela figura 6.2.

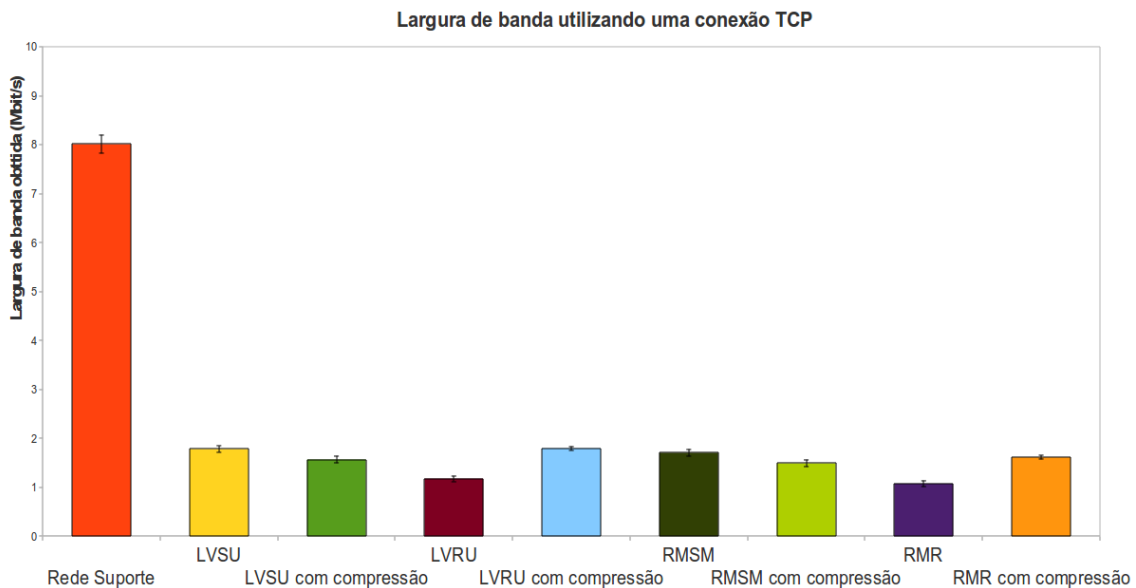


Figura 6.2: Resultados de largura de banda efectiva utilizando uma comunicação *TCP* entre os nós virtuais

Como é possível verificar, as abordagens implementadas utilizam uma percentagem da largura de banda da suportada pela rede. Esta situação pode ser explicada pela forma como o envio de informação é realizado. A biblioteca utilizada, *Jpcap*, necessita que os objectos se encontrem serializados aquando do envio e recepção. Por esta razão, não é possível aproveitar ao máximo o espaço disponível num datagrama *Ethernet* uma vez que alguma da informação transportada não é utilizável pela máquina virtual.

A abordagem *LVRU*, apresenta uma grande discrepância na largura de banda com e sem compressão. A informação que é enviada utilizando esta abordagem, é colocada num pacote IP e posteriormente numa trama *ethernet*, esta última define um tamanho máximo para os dados enviados, por esta razão, é necessário a divisão da informação a enviar, que é conseguida

reduzindo o *MTU* das máquinas virtuais que utilizam esta abordagem. Isto faz com que a informação seja fragmentada, aumentando o *overhead* gerado na conexão e baixando o desempenho. Contudo, a abordagem com compressão já não necessita deste mecanismo, pois as informações que são enviadas, podem ser colocadas numa única trama *ethernet*.

Os resultados obtidos com o *LVSU*, equiparam-se à abordagem com *RawSockets*, contudo também aqui é registada fragmentação quando o tamanho de informação a enviar aumenta.

Os valores registados na utilização da conexão *TCP* sobre as abordagens multi-ponto regista uma largura de banda similar à que foi obtida nas outras abordagens. Estes valores podem ser explicados pela forma como o envio da informação é realizado.

As abordagens baseadas na troca de datagramas *UDP*, como são as implementações *RMSM* e *LVSU*, apresentaram resultados muito baixos inicialmente, o que fez com que se aumentasse o tamanho máximo dos *buffers UDP* do sistema operativo de suporte. Isto fez com que os valores obtidos inicialmente fossem largamente ultrapassados apresentando, após a alteração, valores aceitáveis para uma comunicação como é possível verificar pela figura 6.2.

É ainda interessante verificar que as implementações que utilizam as mesmas bases tecnológicas para troca de informação apresentam larguras de banda similares. As abordagens que utilizam como suporte os datagramas *UDP* apresentam uma taxa de fragmentação elevada o que pode ser combatido com a utilização da compressão da informação, abordagens que recorrem a trocas de pacotes de dados criados pela aplicação apresentam a quebra que a redução do *MTU* acarreta.

6.2 Largura de banda virtual utilizando uma conexão UDP

O segundo teste realizado permitiu obter a largura de banda efectiva de cada uma das ligações, utilizando um fluxo de prova *UDP* sobre o *link* virtual. O *UDP* ao contrário do *TCP* não oferece controlo de tráfego, ou seja, não oferece mecanismos de reenvio de tráfego, aceitando as perdas de pacotes que possam ocorrer. Este teste permite medir a capacidade de disponibilização de largura de banda que os *links* virtuais são capazes de oferecer. Iniciou-se o teste pedindo a transferência de dados entre nós virtuais a 1 *Mbit/s*, aumentando este valor até atingir a largura de banda máxima que a rede é capaz de oferecer, 10 *Mbit/s*. Este teste permite obter a informação

da largura de banda máxima obtida em cada teste e as perdas que ocorrem. Cada teste é realizado 10 vezes para cada largura de banda, de forma a obter uma média final dos valores.

O gráfico da figura 6.3 permite esquematizar a capacidade de largura de banda de cada *link* virtual.

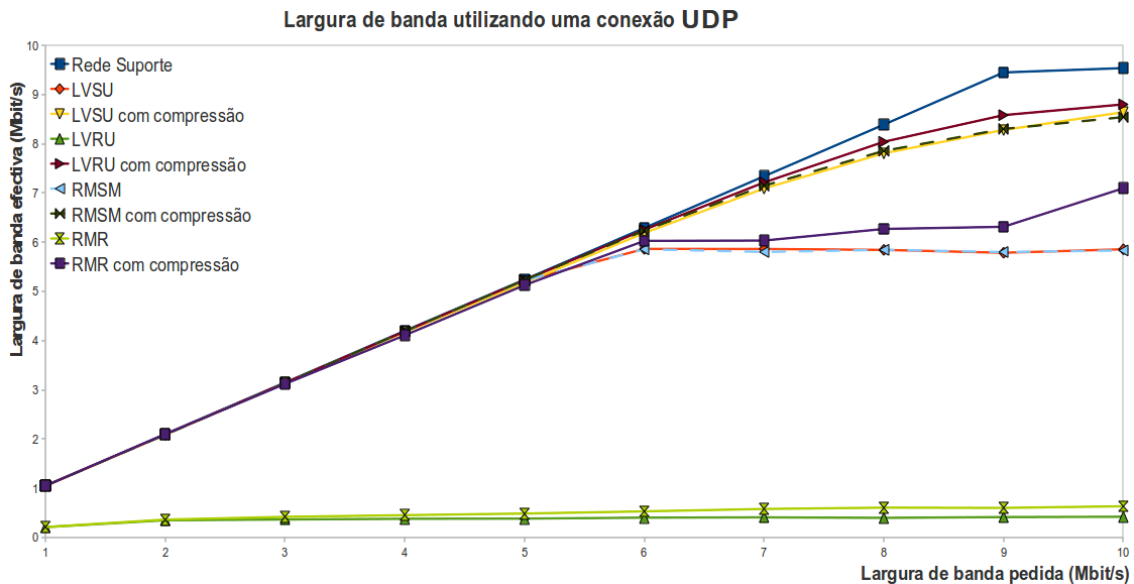


Figura 6.3: Resultados de largura de banda efectiva utilizando uma comunicação *UDP* entre nós virtuais

O gráfico apresentado na figura 6.3 permite verificar as diferenças entre a largura de banda efectiva que cada uma das abordagens virtuais é capaz de oferecer. Estes resultados estão ligados aos valores de perdas que moldam a capacidade máxima de transferência de cada uma. Os valores obtidos com as abordagens *RawSocket* (*LVRU* e *RMR*) sem compressão foram extremamente baixos em comparação com o que foi obtido nos outros testes, isto pode ser explicado pela forma como o envio de informação é realizado nestas abordagens. O encapsulamento da informação numa trama de nível 2 faz com que se limite a informação que se pode enviar, isto é feito com o controlo do *MTU* das máquinas virtuais, diminuindo a quantidade de informação que estas enviam. Esta forma de controlo provoca um aumento na fragmentação da informação que provém do nó virtual e no *overhead* gerado para o tratamento dos dados. Esta necessidade de baixar o *MTU* provém da utilização da biblioteca *Jpcap* que utiliza a serialização de Objectos para o envio e recepção da informação, aumentando a quantidade de dados que são enviados em

cada cápsula e que não são recebidos da máquina virtual. As mesmas abordagens com compressão permitirão, por outro lado, a utilização de uma percentagem de largura de banda mais próxima da suportada pela rede. Isto acontece pois os dados recebidos são comprimidos até um tamanho capaz de ser transportado por uma trama *ethernet*, evitando a necessidade de fragmentação.

O envio da informação comprimida recorrendo a *Sockets UDP* (*LVSU* com compressão) acompanha de perto a evolução da abordagem *LVRU*, sendo apenas de salientar que a mesma abordagem sem compressão (*LVSU*) não é capaz de atingir valores tão elevados uma vez que o *Socket* que realiza o envio dos datagramas fragmenta a informação aquando do envio, provocando uma limitação no desempenho do *link* virtual.

A abordagem multi-ponto que recorre a *RawSocket(RMR)* apresenta uma largura de banda intermédia, mas superior ao valor obtido na rede multi-ponto cuja troca de informação utiliza o *MulticastSocket* do *Java*. O valor da largura de banda com a abordagem *RMR* é mais elevada quando é utilizada a compressão, isto pode ser explicado pela necessidade de fragmentação dos dados que estão a ser enviados, uma vez que a biblioteca utilizada requer que as informações capturadas e enviadas sejam serializadas provocando um degrading do desempenho.

Em algumas abordagens foi obtido um valor aproximado ao obtido pela rede de suporte. Isto comprova que os *links* virtuais possuem a capacidade de suportar uma largura de banda efectiva próxima da rede de suporte, aproveitando as capacidades que esta é capaz de oferecer. Com o aumento da largura de banda pedida, o valor efectivo obtido foi similar, permitindo obter uma visão limite do que o *link* virtual é capaz de oferecer. Ainda relativo ao teste realizado com um conexão *UDP*, podemos observar o valor das perdas associadas a cada ligação, apresentado na figura 6.4.

Os resultados apresentados na figura 6.4 apresentam uma lógica similar ao que é constatado no gráfico de largura de banda efectiva *UDP*. Com o evoluir da largura de banda pedida, a capacidade do *link* virtual vai sendo testada. Quando é atingido o limiar do que a rede de suporte em conjunto com o *link* virtual são capazes de oferecer, alguns dos pacotes enviados são perdidos. A perda de alguns datagramas de informação vem limitar o aumento dos valores no gráfico de largura de banda efectiva, uma vez atingido o ponto onde não é possível aumentar a capacidade de transferência de informação pelo *link* virtual é aumentado o valor de perdas, fazendo com que

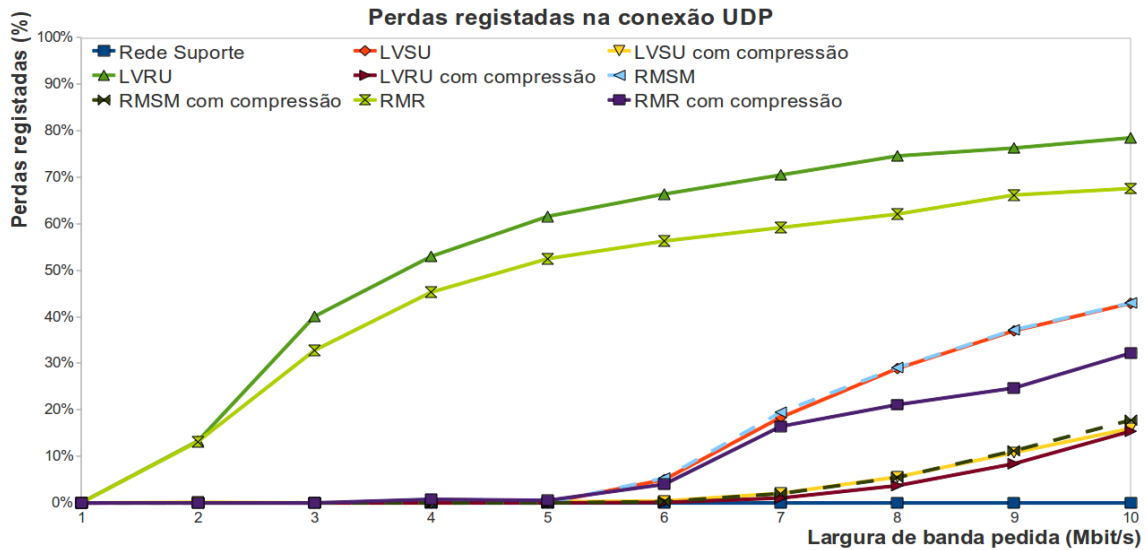


Figura 6.4: Resultados do teste de controlo de perdas utilizando uma conexão *UDP*

a largura de banda efectiva se mantenha constante, tal como é possível observar pelo gráfico 6.3.

No gráfico 6.4 é possível verificar que a necessidade de redução do *MTU* devido à serialização e subsequente fragmentação, registado nas abordagens *RawSocket* sem compressão, faz com que o valor das perdas aumente com a progressão da largura de banda pedida, isto acontece pois o elevado pedido de recursos aliado à redução do *MTU* fará com que o nó virtual necessite de limitar a informação que envia em cada datagrama, dividindo-o, o que resultará na restrição da largura de banda oferecida por estes *links* e um aumento no valor de perdas.

O envio da informação comprimida recorrendo a *Sockets UDP* (*LVSU* com compressão) acompanha de perto a evolução da abordagem *LVRU* com compressão, sendo apenas de salientar que a mesma abordagem sem compressão (*LVSU*) não é capaz de atingir valores tão elevados uma vez que o *Socket* que realiza o envio dos datagramas fragmenta a informação aquando do envio, provocando uma limitação no desempenho do *link* virtual.

A abordagem *LVSU* apresenta os valores com uma variação bastante acentuada com a utilização da compressão. Este facto deve-se ao facto do *IP* realizar a fragmentação da informação de forma a conseguir transmitir os dados, por outro lado a compressão permite fazer com que um maior volume de dados possam ser colocados num datagrama, eliminando na grande parte dos casos, a necessidade de fragmentação. Este facto pode ser constatado tanto no gráfico 6.4 como

6.3 onde o valores atingidos pela abordagem são melhores com a utilização da fragmentação.

6.3 Atraso introduzido pela conexão virtual

Foi medido o atraso na ligação utilizando as opções de *link* virtual implementadas. Para este cenário de testes foram criados dois nós virtuais em redes distintas pedindo que fossem trocados 200 *pings* em intervalos periódicos de 200 *ms*. Foi registada a diferença temporal entre os *requests* e os *replies* de forma a ser possível saber qual a carga temporal que a aplicação acarreta em cada uma das conexões virtuais.

Os valores obtidos podem ser consultados na figura 6.5.

Como é possível verificar pelos atrasos registados na figura 6.5 as abordagens implementa-

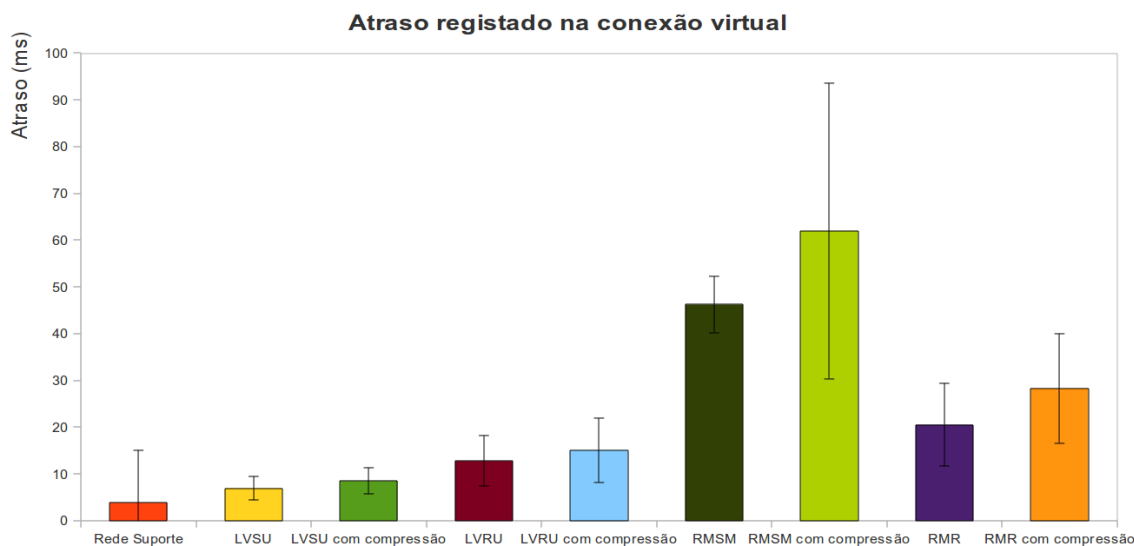


Figura 6.5: Representação gráfica dos atrasos registados em cada uma dos link virtuais criados

das que permitem a comunicação sobre o *link* virtual acrescentam um atraso adicional na conexão, apresentando valores mais elevados que a sobre a rede de suporte. Este facto é facilmente compreensível se for levado em conta o algoritmo que a aplicação executa. A necessidade de capturar o tráfego e encapsula-lo acarreta um *overhead* adicional e sem o qual a rede virtual não poderia ser criada. O encapsulamento da informação recebida da máquina virtual apresenta necessidade de mais um patamar de tratamento, o que faz com que o *overhead* temporal registado na conexão seja mais alto em todas as abordagens implementadas.

As implementações permitem verificar que a compressão do fluxo de dados vai acarretar um atraso adicional à conexão fazendo com que seja necessário que o fluxo de dados a enviar passe por um patamar de tratamento adicional. É possível verificar que a abordagem que apresenta o atraso mais reduzido recorre a *Sockets UDP* para a troca de informação, seguida da abordagem que recorre a *RawSockets*. Esta diferença pode ser explicada pela forma como os datagramas são enviados na abordagem *LVRU*, onde a cápsula é enviada para o endereço *ethernet* do *gateway* indicado pelo sistema operativo, acrescentando mais um salto na ligação, fazendo com que a abordagem apresente um atraso maior relativamente à abordagem que utiliza uma entrega directa.

A abordagem multi-ponto *RMSM* apresenta um atraso na conexão extremamente elevado quando comparado com as outras abordagens, tornando-o de difícil explicação, pois deveria possuir um atraso similar ao registado na abordagem *LVSU* uma vez que o envio de informação recorre também a datagramas *UDP*, contudo, este teste necessita de ser realizado novamente de forma a comprovar o valor obtido.

Capítulo 7

Conclusão

A virtualização de redes é neste momento visto como um catalisador para o desenvolvimento da *Internet* do futuro. Tanto operadoras como académicos olham para esta ferramenta promissora como uma possibilidade de evolução no ramo da investigação e no desenvolvimento de novas soluções e serviços. A capacidade de instanciar várias redes virtuais sobre uma infra-estrutura partilhada, oferecendo serviços diferenciados a clientes distintos permite disponibilizar arquitecturas de rede com parâmetros de acordo com as necessidades. A tecnologia de virtualização de redes permite a capacidade de amadurecimento de novos protocolos e serviços, colocando-os sobre um ambiente controlado.

Vários avanços foram conseguidos nos últimos anos nesta área e projectos como o *4WARD*, *VINI* e *PlanetLab* permitem comprovar o investimento e a importância que esta tecnologia pode apresentar no futuro. Uma infra-estrutura virtual é capaz de proporcionar o desenvolvimento de serviços e protocolos cada vez mais especializados e robustos.

O trabalho desenvolvido na dissertação aborda a criação de um sistema que possibilita o desenvolvimento de uma rede virtual de nível 2. Para isto foram pensadas e implementadas as formas como uma topologia de nível 2 é capaz de trocar informação. Cada uma das formas de troca de informação foi implementada utilizando duas abordagens com níveis de abstracção distintos. A comunicação entre os nós virtuais é realizada sobre conexões com características à medida, permitindo a troca de informação numa rede dinâmica onde as suas entidades possuem apenas uma percepção dos recursos virtuais.

A proposta descrita na dissertação foi implementada num protótipo funcional que permite testar a abordagem, analisando as características que cada uma das funcionalidades oferecem a

uma infra-estrutura virtual. Os algoritmos de procura de recursos, a negociação dos parâmetros da conexão e posterior criação e manutenção das entidades virtuais, cooperam para possibilitar a construção do suporte de comunicação.

A comunicação entre os nós recorre a *Sockets* e *RawSockets* para a troca de informação entre dois extremos. Os dados são colocados em cápsulas configuradas de acordo com as necessidades do utilizador. Uma abordagem similar foi utilizada para a comunicação multi-ponto, recorrendo às mesmas estruturas de comunicação para uma troca de informação sobre *multicast*. A capacidade de comprimir os *streams* de dados das máquinas virtuais, permite realizar uma troca entre largura de banda e processamento. Os datagramas enviados são comprimidos, de forma a gastar o mínimo de recursos, assegurando a comunicação com um débito efectivo superior ao que seria utilizável, em alguns casos, pela comunicação sem compressão.

Os testes realizados permitem comprovar que a solução proposta assegura a troca de informação, contendo-a numa infra-estrutura dinâmica e controlada. Os resultados obtidos em cada uma das abordagens atestam a capacidade de comunicação que a implementação apresenta sob pressão, onde obteve características próximas às registadas na rede de suporte mesmo considerando o *overhead*.

A possibilidade de realizar a multiplexagem/desmultiplexagem dos canais de comunicação com recurso ao campo protocolo permite retirar um nível de *overhead* que é normalmente observado, resultando numa entrega da informação muito mais clara e simples entre as máquinas virtuais e mantendo a escalabilidade do sistema. As implementações estudadas propõem a criação de uma plataforma virtual de nível 3 sobre uma infra-estrutura de rede do mesmo nível, mantendo as mesmas limitações de protocolos registadas na rede pública, a proposta aqui apresentada tenta ultrapassar este pressuposto oferecendo um meio onde a comunicação virtual decresce um nível lógico, permitindo a virtualizar a camada de ligação, sobre uma infra-estrutura de nível 3.

As conexões criadas para a construção da rede virtual, são focadas na troca de informação, não sendo do âmbito do projecto a aposta no desempenho. Contudo, foram estudados os pontos na aplicação que poderiam induzir atraso. A utilização da biblioteca *Jpcap* trouxe consigo a serialização de Objectos e a necessidade de enviar informação que não está directamente associada aos dados enviados pelo nó virtual, provocando um degrading no desempenho da comunicação. De forma a poder combater este problema foi desenvolvida uma nova biblioteca que permitia a recepção e envio de informação como uma *stream* de *bytes*, eliminando o problema

da Serialização na comunicação virtual. A implementação desta nova biblioteca no protótipo desenvolvido não foi até de momento realizada tendo sido marcada, devido às alterações que produziria no sistema, como um trabalho futuro caso a necessidade de aumento do desempenho existisse.

Infelizmente o não existiu disponibilidade para o teste do sistema em larga escala, contudo no desenvolvimento da abordagem e da implementação foram utilizados pressupostos que permitem manter esta característica.

Um outro ponto que pode ser necessário limar é a possível necessidade que da rede virtual ser estendida a domínios privados, ou seja, pode ser necessário a construção de redes virtuais onde alguns dos nós que a compõem possam estar numa *NAT* ou *proxy*, alguns dos trabalhos estudados pressupõem este tipo de mecanismos, criando pontos de armazenamento e reencaminhamento de tráfego, contudo, esta solução poderá não ser a melhor para resolução do problema, que deverá ser melhor analisada.

O próximo passo no desenvolvimento do sistema poderá recair sobre a implementação de um mecanismo de segurança e autenticação dos nós que compõem a rede virtual, o protótipo que foi desenvolvido não utilizou qualquer mecanismo de segurança para a troca de informação, contudo existem pontos críticos, onde esta poderá de futuro ser incorporada, tal como a procura de recursos, a negociação e troca das informações dos links virtuais e o envio de informações recebidas através do ficheiro de configuração.

Referências

- [1] P. A. A. Gutiérrez, S. Baucke, R. Bless, M. Bourguiba, J. M. Cabero, L. Caeiro, J. Carapinha, M. Dianati, C. Görg, I. Houidi, L. I. Gamir, Y. Lemieux, W. Louati, O. Maennel, L. Mathy, M. Melo, J. Nogueira, P. Papadimitriou, S. P. Sánchez, A. Serrador, I. Seskar, J. Tiemann, A. Udugama, C. Werle, F. Wolff, A. Wundsam, Y. Zaki, D. Zeglache, and L. Zhao, “4ward – architecture and design for the future internet,” *4WARD*, vol. D-3.2.1, Jun. 2010.
- [2] M. T. Jones, “An overview of virtualization methods, architectures, and implementations,” <http://www.ibm.com/developerworks/linux/library/l-linuxvirt/>, Dec. 2006, visitado em 03/04/2011. [Online]. Available: <http://www.ibm.com/developerworks/linux/library/l-linuxvirt/>
- [3] J. Touch, “Dynamic internet overlay deployment and management using the x-bone,” *Network Protocols, 2000. Proceedings. 2000 International Conference*, Nov. 2000.
- [4] M. Tsugawa and J. A. Fortes, “A virtual network (vine) architecture for grid computing,” *In Proc. of 20th International Parallel and Distributed Processing Symposium (IPDPS-2006)*, 2006.
- [5] X. Jiang and D. Xu, “Violin: Virtual internetworking on overlay infrastructure,” *in Proc. of the 2nd intl. symp. on parallel nd distributed processing and applications*, 2003.
- [6] J. Nogueira, M. Melo, J. Carapinha, and S. Sargento, “A distributed approach for virtual network discovery,” *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*, 2010.
- [7] “Ip multicasting at layer 2 - module 2,” Cisco Systems, Sep. 2001.

- [8] A. Shroff and V. R. Donthireddy, “Virtualization imperatives and performance,” <http://www.infosys.com/IT-services/application-services/white-papers/Documents/virtualization-imperatives-performance.pdf>, 2009.
- [9] J. Carapinha, P. Feil, P. Weissmann, S. E. Thorsteinsson, Ç. Etemoglu, Ó. Inbórsson, S. Çiftçi, and M. Melo, “Network virtualization - opportunities and challenges for operators,” *Lecture Notes in Computer Science*, 2010.
- [10] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, and M. Wawrzoniak, “Planetlab : An overlay testbed for broad-coverage services,” *ACM SIGCOMM Computer Communication Review*, Jul. 2003.
- [11] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, “Vini veritas: realistic and controlled network experimentation,” in *Proc. ACM SIGCOMM*, 2006. [Online]. Available: <http://www.vini-veritas.net/>
- [12] J. P. Herron, “Geni meta-operations center,” *eScience, 2008. eScience '08. IEEE Fourth International Conference*, Dec. 2008. [Online]. Available: <http://www.geni.net/>
- [13] H. Harai, “Designing new-generation network - overview of akari architecture design,” *Communications and Photonics Conference and Exhibition (ACP), 2009 Asia*, Nov. 2009. [Online]. Available: <http://akari-project.nict.go.jp/eng/index2.htm>
- [14] A. Bensoussan, C. T. Clingen, and R. C. Daley, “The multics virtual memory: Concepts and design,” *Communications of the ACM*, vol. Volume 15 Issue 5, May 1972.
- [15] K. Lawton, G. Alexander, D. Becker, C. Bothamy, B. Denney, V. Ruppert, and S. Shwartsman, “Bochs ia-32 emulator project,” <http://bochs.sourceforge.net/>, 2001.
- [16] “Windows virtual pc,” <http://www.microsoft.com/windows/virtual-pc/>, 2006.
- [17] “Qemu - open source processor emulator.”
- [18] “Vmware,” <http://www.vmware.com/>, VMware, 2002.
- [19] “Parallels virtualization,” <http://www.parallels.com/>, Parallels, 1999.
- [20] “Virtualbox,” <https://www.virtualbox.org/>.

-
- [21] P. Barham, B. Dragovic, K. Fraser, S. H., T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *SOSP '03 Proceedings of the nineteenth ACM symposium on Operating systems principles*, Dec. 2003.
- [22] "Oracle solaris containers," <http://www.oracle.com/technetwork/server-storage/solaris/containers-169727.html>.
- [23] "Freebsd jails," <http://www.freebsd.org/doc/handbook/jails.html>.
- [24] "Linux-vserver," <http://linux-vserver.org/>.
- [25] "Java virtual machine," <http://java.sun.com/docs/books/jvms/>, Oracle.
- [26] *Understanding Full Virtualization, Paravirtualization, and Hardware Assist*, VMware, Nov. 2007.
- [27] *VPN Technologies: Definitions and Requirements*, VPN Consortium, Jul. 2008.
- [28] P. Knight and C. Lewis, "Layer 2 and 3 virtual private networks: Taxonomy, technology, and standardization efforts," *IEEE Communications Magazine*, Jun. 2004.
- [29] W. Simpson, "Ip in ip tunneling (rfc 1853)," <http://tools.ietf.org/html/rfc1853>, Oct. 1995.
- [30] S. Frankel, K. Kent, R. Lewkowski, A. D. Orebaugh, R. W. Ritchey, and S. R. Sharma, "Guide to ipsec vpns," <http://csrc.nist.gov/publications/nistpubs/800-77/sp800-77.pdf>, Dec. 2005.
- [31] S. Kent and K. Seo, "Security architecture for the internet protocol (rfc4301)," <http://tools.ietf.org/html/rfc4301>, Dec. 2005. [Online]. Available: <http://tools.ietf.org/html/rfc4301>
- [32] A. Abdelhalim, "Ip/mppls-based vpns," Foundry Network, Tech. Rep., 2002.
- [33] J. Touch and S. Hotz, "The x-bone," *Proc. Global Internet Mini-Conference (Globecom)*, 1998.
- [34] J. Carapinha and J. Jiménez, "Network virtualization - a view from the bottom," *ACM SIGCOMM Conference 2009: Barcelona, Spain - VISA Workshop*, Aug. 2009.

- [35] J. Nogueira, M. Melo, J. Carapinha, and S. Sargento, "Virtual network mapping into heterogeneous substrate networks," *Computers and Communications (ISCC), 2011 IEEE Symposium on*, Jul. 2011.
- [36] C. Perkins, "Ip encapsulation within ip (rfc2003)," <http://tools.ietf.org/html/rfc2003>, IBM, Oct. 1996. [Online]. Available: <http://www.ietf.org/rfc/rfc2003.txt>
- [37] G. Foot and C. Catlett, "The libnet packet construction library," <http://libnet.sourceforge.net/>, Feb. 2007. [Online]. Available: <http://packetfactory.openwall.net/projects/libnet/>
- [38] "The user-mode linux kernel home page," <http://user-mode-linux.sourceforge.net/>. [Online]. Available: <http://user-mode-linux.sourceforge.net/>
- [39] A. Dickmeiss, H. Levanto, M. Cromme, M. Taylor, and S. Hammer, "Zebra - user's guide and reference," Nov. 2009. [Online]. Available: <http://www.zebra.org/>
- [40] K. E., M. R., C. B., J. J., and K. M. F., "The click modular router project," <http://read.cs.ucla.edu/click/click>. [Online]. Available: <http://read.cs.ucla.edu/click/>
- [41] Y. Zhu and M. Ammar, "Algorithms for assigning substrate network resources to virtual network components," *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, Apr. 2006.
- [42] D. G. Andersen, "Theoretical approaches to node assignment," Jan. 2002, unpublished Manuscript.
- [43] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: Substrate support for path splitting and migration," *Computer Communication Review*, Apr. 2008.
- [44] R. Ricci, C. Alfeld, and J. Lepreau, "A solver for the network testbed mapping problem," *SIGCOMM Computer Communications Review*, vol. 33, Jan. 2003.
- [45] J. Lu and J. Turner, "Efficient mapping of virtual networks onto a shared substrate," *Department of Computer Science and Engineering Washington University in St Louis Technical Report WUCSE2006*, 2006.

- [46] N. M. M. K. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," *INFOCOM 2009, IEEE*, 2009.
- [47] I. Houidi, W. Louati, and D. Zeghlache, "A distributed virtual network mapping algorithm," *Communications, 2008. ICC '08. IEEE International Conference*, May 2008.
- [48] F. Cappello, E. Caron, M. Dayde, F. Desprez, Y. Jegou, P. Primet, and E. Jeannot, "Grid'5000: a large scale and highly reconfigurable grid experimental testbed," *Grid Computing, 2005. The 6th IEEE/ACM International Workshop*, Nov. 2005.
- [49] F. Bellifemine, G. Caire, A. Poggi, and G. Rimassa, "Jade: A white paper," Telecom Italia Lab. [Online]. Available: <http://jade.tilab.com/>
- [50] "Foundation for intelligent physical agents," 1997. [Online]. Available: <http://www.fipa.org/>
- [51] M. Thomas, E. Edwards, and S. Bhattacharjee, "Modeling topology of large internetworks," <http://www.cc.gatech.edu/projects/gtitm/>. [Online]. Available: <http://www.cc.gatech.edu/projects/gtitm/>
- [52] R. Hancock, G. Karagiannis, J. Loughney, and S. V. den Bosch, "Next steps in signaling (nsis): Framework (rfc4080)," <http://www.ietf.org/rfc/rfc4080.txt>, 2005.
- [53] H. Schulzrinne and R. Hancock, "Gist: General internet signalling transport (rfc5971)," <http://tools.ietf.org/html/rfc5971>, Oct. 2010.
- [54] "SIGAR API (System information gatherer and reporter)," <http://www.hyperic.com/products/sigar>, Hyperic. [Online]. Available: <http://www.hyperic.com/products/sigar>
- [55] K. Fujii, "Jpcap tutorial," <http://netresearch.ics.uci.edu/kfujii/Jpcap/doc/>, 2007. [Online]. Available: <http://netresearch.ics.uci.edu/kfujii/Jpcap/doc/tutorial/index.html>
- [56] "Rocksaw," <http://www.savarese.org/software/rocksaw/>, Savarese.
- [57] I. Sendin, "jpacket project," <http://jpacket.sourceforge.net/>, 2001.
- [58] S. Liang, *The Java Native Interface - Programmer's Guide and Specification*, <http://java.sun.com/docs/books/jni/download/jni.pdf>, 1999.

- [59] L. M. Garcia, *Programming with Libpcap - Sniffing the network from our own application*. Hakin9, Feb. 2008, pp. 38–46.
- [60] J. Postel, “Internet protocol (rfc 791),” <http://tools.ietf.org/html/rfc791>, September 1981.
- [61] J. Arkko and S. Bradner, “Protocol numbers (rfc5237),” <http://tools.ietf.org/html/rfc5237>, Jul. 2011. [Online]. Available: <http://www.iana.org/assignments/protocol-numbers/protocol-numbers.txt>
- [62] S. Deering and R. Hinden, “Internet protocol, version 6 (ipv6) specification (rfc2460),” <http://www.faqs.org/rfcs/rfc2460.html>, Dec. 1998.
- [63] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan, “Internet group management protocol, version 3 (rfc3376),” <http://www.ietf.org/rfc/rfc3376.txt>, Oct. 2002.
- [64] R. Braden and D. Borman, “Computing the internet checksum (rfc1071),” <http://tools.ietf.org/html/rfc1071>, Sep. 1988.
- [65] W. Biggs and H. Evans, “Simplify xml programming with jdom,” May 2001. [Online]. Available: <http://www.ibm.com/developerworks/java/library/j-jdom/>

Anexo A

Utilização da aplicação

Logo que a aplicação é iniciada, é apresentado um *front-end* que oferece as opções de conexões que podem ser criadas. As opções apresentadas permitem construir ligações entre dois nós da rede utilizando *Sockets (LVSU)*, ou *RawSockets (LRVU)*, ou entre dois ou mais nós utilizando um link multi-ponto construído com um *MulticastSocket (RMSM)*, ou com *RawSockets (RMR)*.

A interface representada em 1 também permite que se altere parâmetros como o *TTL* e número de nós a procurar na rede, assim como o endereço de controlo usado para a troca de mensagens em cada uma das ligações criadas.



Figura 1: Janela inicial da aplicação

Criação de *link* virtual utilizando *RawSockets*

Um dos links que pode ser criado interliga os nós virtuais, recorrendo a uma implementação *RawSocket*. O primeiro passo para a sua criação é a instanciação dos processos auxiliares, que

irão também permitir a conexão a um endereço *multicast* que servirá para a troca de mensagens de controlo.

Logo que o utilizador pressione o botão de iniciar a criação do link, a aplicação inicia o processo de procura de *hosts* preparados para a criação. Esta procura de *hosts*, utilizará como valores máximos, os introduzidos pelo utilizador na interface gráfica ou utilizará um valor por defeito de 5.

Quando a procura de *hosts* atinge o seu limite, a aplicação inicia uma nova janela que irá permitir obter as informações da procura, podendo repeti-la, caso seja necessário. A janela apresentada na figura 2 também permitirá, de uma forma intuitiva, a criação do link entre duas máquinas virtuais, que serão iniciadas em tempo de compilação, ligando o *link* virtual a uma das suas interfaces directamente. Para isto, o utilizador seleccionará as características do link: a interface de saída do nó local e remoto; o endereço *ethernet* da máquina virtual local e remota¹, o valor de *QoS* que se pretende utilizar nos pacotes *IP* que irão servir de cápsula, a largura de banda máxima que se pretende na interface da máquina virtual e por fim se vai ser utilizada a compressão da informação recebida da máquina virtual.



Figura 2: Janela de criação de uma conexão *LVSU*

Estes dados serão recebidos pela aplicação que os irá tratar, agregando-os num e negociando as suas características com o *host* remoto. Quando a negociação terminar, é iniciada a máquina

¹este campo deve ser precedido de dois bytes fixos com o valor *08* e *00*, para que possam se identificados como endereços da máquina virtual)

virtual, assim como a janela representada em 3 que permite controlar as características da máquina virtual.

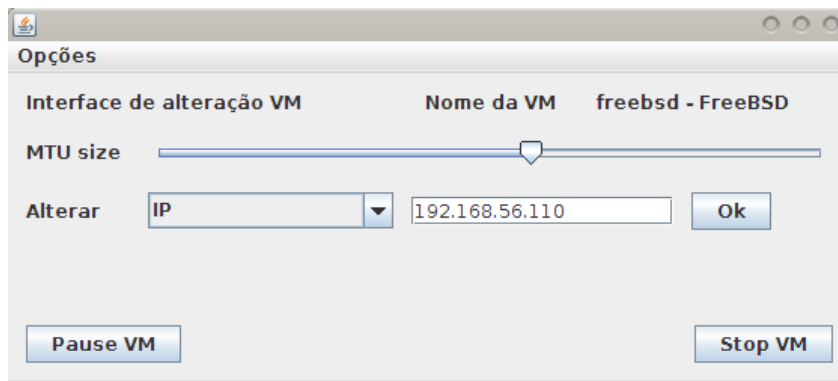


Figura 3: Janela de alteração das características da máquina virtual

Criação de *link* virtual utilizando *Sockets unicast*

A criação de uma ligação ponto a ponto com recurso a sockets *unicast*, permitirá interligar as duas máquinas virtuais, recorrendo a *Sockets UDP*.

Inicialmente são instanciados os processos que permitirão a participação na rede virtual sendo para isto necessário a conexão a um grupo *multicast* que permitirá a troca de informações entre os *hosts*. Uma vez que esta tarefa está terminada, é iniciada uma nova janela que permitirá estabelecer os parâmetros do novo *link* virtual. *IP* origem, *IP* destino, máquina e interface virtual e as suas características, são alguns dos parâmetros alteráveis que permitem criar o *link* entre as duas máquinas. A interface gráfica de gestão, possuirá a forma apresentada na figura 4.

Criação de *link* multi-ponto utilizando *Sockets Multicast*

A ligação multi-ponto permitirá simular uma ligação de nível 2, recorrendo a uma comunicação *multicast*, desta forma cada datagrama enviado pela máquina virtual é difundido pela rede física, de forma a que os *hosts* interessados sejam capazes de a obter.

A criação de uma ligação deste tipo inicia-se com a junção a um endereço de controlo, que

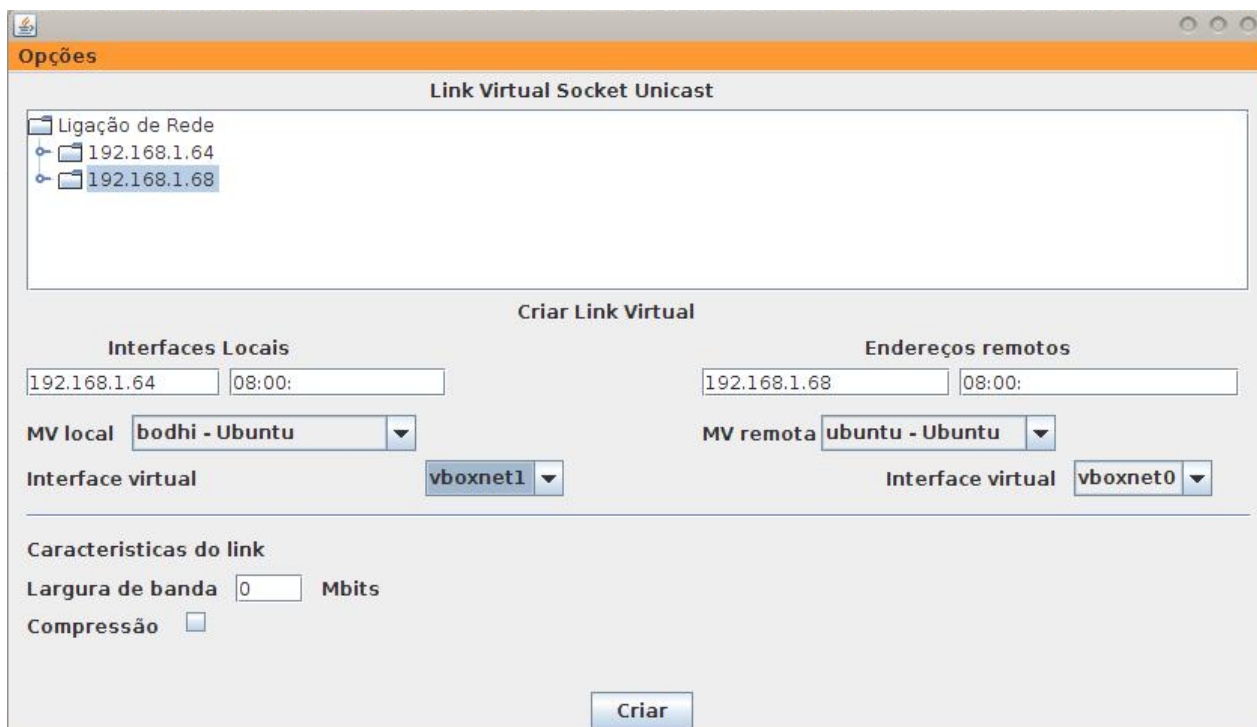


Figura 4: Janela de criação de *link* virtual do tipo *LVSU*

permitirá receber informações sobre as redes multi-ponto criadas. Sendo primeiro passo a iniciação das instâncias para a criação da rede e a indicação do endereço de controlo. Para isto é utilizado o menu superior da interface inicial da aplicação, será então apresentada a janela apresentada na figura 5 que permitirá a introdução do endereço. A partir deste momento a aplicação inicia a recepção de mensagens de anuncio de redes multi-ponto.

Ao requisitar, a criação do *link*, é apresentado um novo interface, que permite obter a informação de todos os *links* multi-ponto existentes que foram anunciados no endereço *multicast* de controlo. É possível a conexão a estes grupos, indicando também qual a máquina e interfaces virtuais que se pretende ligada à rede.

A interface gráfica permitirá a ligação a um novo grupo *multicast* que será difundido periodicamente nas mensagens de anuncio, ou a um já existente escolhendo-o na lista e clicando na *checkbox*.



Figura 5: Janela de parametrização da conexão *RMSM*

Criação de *link* multi-ponto utilizando *RawSockets*

A criação do *link* multi-ponto utilizando *RawSockets*, possuirá uma interface gráfica muito similar à que é utilizada pela abordagem com *MulticastSockets*. Após se conectar a um endereço *multicast* onde escutará os anúncios de redes virtuais dos outros utilizadores, é possível iniciar a interface de configuração da rede *multicast* com *RawSockets* pressionando o botão "*Rmr*".

A interface da figura 6 permitirá definir os parâmetros da ligação, permitindo ligar a um grupo que foi anunciado pelo grupo *multicast* de controlo, ou criar uma nova rede *multicast*.

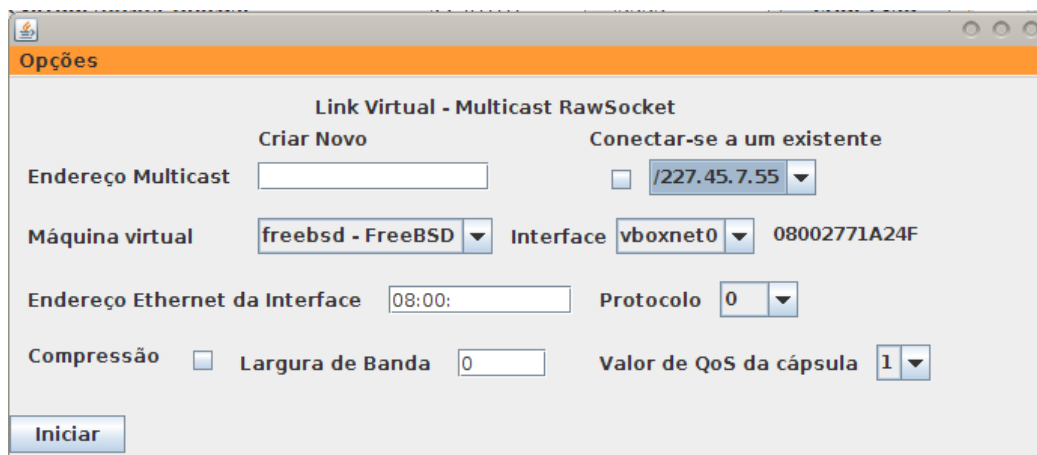


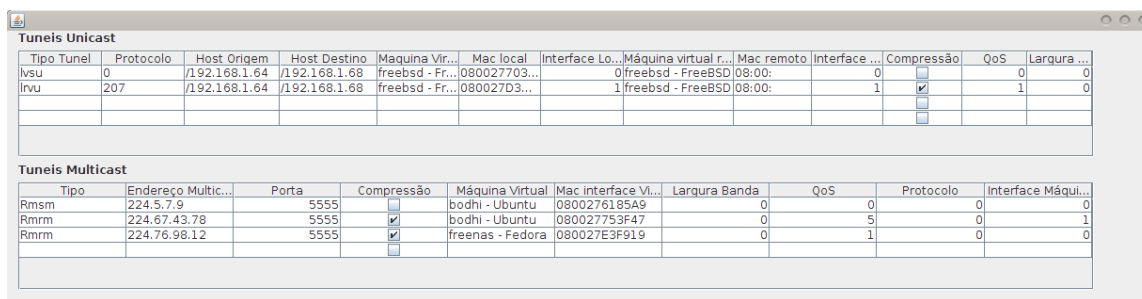
Figura 6: Janela de criação de uma conexão *RMR*

Caso se pretenda a conexão com um grupo *multicast* previamente criado, alguns dos parâmetros que o caracterizam, são automaticamente utilizados, como é exemplo a compressão dos

dados. A máquina virtual, é automaticamente iniciada e as suas interfaces são configuradas de acordo com o que é parametrizado pelo utilizador.

Janela de controlo de links criados

Foi também implementada uma janela que permite controlar as características que cada link está no momento a utilizar, esta será iniciada utilizando o menu de opções da janela inicial e terá a forma da figura 7.



The screenshot shows a window with two tables. The first table, 'Tuneis Unicast', has columns: Tipo Tunel, Protocolo, Host Origem, Host Destino, Máquina Vir..., Mac local, Interface Lo..., Máquina virtual r..., Mac remoto, Interface..., Compressão, QoS, and Larqura... The second table, 'Tuneis Multicast', has columns: Tipo, Endereço Multic..., Porta, Compressão, Máquina Virtual, Mac interface Vi..., Larqura Banda, QoS, Protocolo, and Interface Máqui... Both tables contain several rows of data with checkboxes in the 'Compressão' column.

Tuneis Unicast												
Tipo Tunel	Protocolo	Host Origem	Host Destino	Máquina Vir...	Mac local	Interface Lo...	Máquina virtual r...	Mac remoto	Interface...	Compressão	QoS	Larqura ...
lvsv	0	/192.168.1.64	/192.168.1.68	freebsd - Fr...	080027703...		0 freebsd - FreeBSD	08:00:	0	<input type="checkbox"/>	0	0
lvvu	207	/192.168.1.64	/192.168.1.68	freebsd - Fr...	080027D3...		1 freebsd - FreeBSD	08:00:	1	<input checked="" type="checkbox"/>	1	0

Tuneis Multicast									
Tipo	Endereço Multic...	Porta	Compressão	Máquina Virtual	Mac interface Vi...	Larqura Banda	QoS	Protocolo	Interface Máqui...
Rmsm	224.5.7.9	5555	<input type="checkbox"/>	bodhi - Ubuntu	0800276185A9	0	0	0	0
Rmrm	224.67.43.78	5555	<input checked="" type="checkbox"/>	bodhi - Ubuntu	080027753F47	0	5	0	1
Rmrm	224.76.98.12	5555	<input checked="" type="checkbox"/>	freenas - Fedora	080027E3F919	0	1	0	0

Figura 7: Janela que permite consultar as características dos links criados