Interoperabilidade e Otimização da Gestão de Redes com a Framework NSDL

TESE DE DOUTORAMENTO

Eduardo Miguel Dias Marques

DOUTORAMENTO EM ENGENHARIA INFORMÁTICA - SISTEMAS DISTRIBUÍDOS E CENTRADOS EM REDE

UNIVERSIDADE da MADEIRA

A Nossa Universidade

www.uma.pt

março | 2013

Interoperabilidade e Otimização da Gestão de Redes com a Framework NSDL

TESE DE DOUTORAMENTO

Eduardo Miguel Dias Marques

DOUTORAMENTO EM ENGENHARIA INFORMÁTICA - SISTEMAS DISTRIBUÍDOS
E CENTRADOS EM REDE

SUPERVISOR

Paulo Nazareno Maia Sampaio

Interoperabilidade e Otimização da Gestão de Redes com a *Framework* NSDL

Autor: Eduardo Miguel Dias Marques

Orientação: *Professor Doutor* Paulo Nazareno Maia Sampaio

Centro de Competências de Ciências Exatas e Engenharias
Universidade da Madeira

Júri

Presidente: *Professor Doutor* Luís Armando de Aguiar Oliveira Gomes
Professor Doutor Edmundo Heitor da Silva Monteiro
Professor Doutor Fernando Manuel R. Morgado Ferrão Dias
Professor Doutor Manuel Alberto Pereira Ricardo
Professor Doutor Paulo Nazareno Maia Sampaio
Professor Doutor Pedro Filipe Pereira Campos

Tese defendida a 20 de setembro de 2013, pelas 15 horas no Colégio dos Jesuítas, Funchal, para a obtenção do grau de Doutor em **Engenharia Informática**, especialidade de **Sistemas Distribuídos e Centrados em Redes**

submetida em março, 2013

A vida é para nós o que concebemos dela. Para o rústico cujo campo lhe é tudo, esse campo é um império. Para o César cujo império lhe ainda é pouco, esse império é um campo. O pobre possui um império; o grande possui um campo. Na verdade, não possuímos mais que as nossas próprias sensações; nelas, pois, que não no que elas vêem, temos que fundamentar a realidade da nossa vida.

Bernardo Soares, Livro do Desassossego

Gostava de dedicar este trabalho
a quem todos os dias está perto,
no coração, mesmo estando distante

Aos melhores pais, Agostinho e Docília,
ao meu irmão, Sérgio, e às suas meninas,
aos meus Avós, e a toda a minha família

E às minhas flores, Carol e Catarina, por
tornarem belo todo e cada um dos meus dias

Agradecimentos

Começo por aquele mais próximo deste trabalho, o meu orientador, Professor Doutor Paulo Sampaio. A sua compreensão e disponibilidade durante estes anos foram essenciais para conseguir terminar. Sem os seus conselhos e direções teria sido muito mais difícil avançar. O seu conhecimento e as discussões sobre o meu trabalho permitiram sempre solidificar o que estava feito e avançar com mais confiança. Por tudo deixo-lhe um Muito Obrigado.

Ao alunos de mestrado Ricardo Alexandre Silva Alves Plácido, Mílton de Castro Gouveia, Jesuíno da Costa Serra de Azevedo, José Jorge Fernandes Sousa e à Joana Patrícia Mendes Araújo um agradecimento especial por terem desenvolvido o seu trabalho de mestrado utilizando a ferramenta desenvolvida neste doutoramento e, assim, terem adicionado novos componentes ao projeto.

Ainda, um agradecimento para os alunos de cadeira de Tecnologias Avançadas de Redes pelo tempo dispendido a testar e a utilizar a ferramenta e no preenchimento de um inquérito, atividades fundamentais para a validação do trabalho feito no âmbito deste trabalho de doutoramento.

A todos os colegas do 'departamento' e da restante Universidade pelo bom e salutar ambiente para investigar e lecionar, e, em especial, ao grupo do café pelas discussões interessantes e intermináveis sobre tudo e sobre nada.

Aos centros de investigação *Centro de Ciências Matemáticas* e *Madeira Interactive Tecnhologies Institute* pelos apoios dados.

Por fim, aquelas a quem devo mais que um agradecimento no final desta caminhada: a minha esposa e a minha filha. Amo-vos muito e sem a vossa paciência e carinho em incontáveis noites e fins-de-semana não teria conseguido.

A todos, um bem haja.

Resumo

As redes de computadores cresceram nas últimas décadas em diversidade e complexidade, sempre procurando manter um nível elevado de qualidade na comunicação. Atualmente existe uma variedade de ferramentas para a gestão de redes que cobrem de forma completa ou parcial as diferentes etapas do ciclo de vida dessas redes. Dadas a dimensão e heterogeneidade dessas redes, o seu desenvolvimento e operação são tarefas que envolvem um número crescente de ferramentas, o que induz a uma maior complexidade. Além do mais, a maior parte das ferramentas existentes são independentes e incompatíveis, tornando a tarefa dos arquitetos e dos gestores de redes mais difícil. Dessa forma, é identificada a necessidade de uma abstração ou abordagem genérica que permita a interoperabilidade entre diferentes ferramentas/ambientes de rede de forma a facilitar e otimizar a sua gestão.

O trabalho apresentado nesta tese introduz a proposta e a implementação de uma *framework* para a integração de diferentes ferramentas heterogéneas de rede dando suporte à criação de um ambiente de gestão que cubra o ciclo de vida de uma rede de comunicação. A interoperabilidade proporcionada pela *framework* é implementada através da proposta de uma nova linguagem para a descrição de redes e de todos os seus componentes, incluindo a informação da topologia e dos contextos onde a rede pode existir. As demais contribuições desta tese estão relacionadas com (i) a implementação de algumas ferramentas de gestão para dar suporte à construção de cenários de rede utilizando a linguagem proposta, e (ii) a modelação de vários cenários de rede com tecnologias diferentes, incluindo aspetos de Qualidade de Serviço, para a validação da utilização da *framework* proposta para proporcionar a interoperabilidade entre diferentes ferramentas de gestão de redes.

A linguagem proposta para a descrição de redes preocupou-se com a descrição dos cenários de rede dando suporte às diferentes fases da existência dessa rede, desde o seu projeto até a sua operação, manutenção e atualização. Uma vantagem desta abordagem é permitir a coexistência de diversas informações de utilização da rede numa única descrição, mantendo cada uma independente das restantes, o que promove a compatibilidade e a reutilização das informações de forma direta entre as ferramentas, ultrapassando assim a principal limitação detetada nas linguagens e ferramentas existentes e reforçando as possibilidades de interoperabilidade.

Palavras-chave: gestão de redes, linguagem de descrição de redes, XML, simulação de redes, network simulator 2, network simulator 3, ambientes virtuais

Abstract

Computer networks have grown in recent decades in diversity and complexity, always trying to maintain a high level of quality in the communication. Currently there is a variety of management tools covering completely or partially the different stages of the networks' life cycle. Given the size and heterogeneity of these networks, their development and operation are tasks that involve a growing number of tools, which leads to a further complexity. Moreover, most of the existing tools are independent and incompatible, making the tasks of network architects and managers more difficult. Therefore, there is a clear lack of an abstraction or generic approach to enable interoperability between different tools/network environments in order to facilitate and optimize their management.

The work presented in this thesis introduces the proposal and implementation of a framework for the integration of different heterogeneous network tools supporting the creation of a management environment that covers the life cycle of a communication network. The interoperability provided by the framework is implemented through the proposal of a new language to describe a network and all its components, including information about the topology and the operational context of the network. Further contributions of this thesis are related to (i) the implementation of several management tools to support the construction of network scenarios using the proposed language, and (ii) the modeling of several network scenarios with different technologies, including Quality of Service aspects, to validate the use of the proposed framework and to provide interoperability between different network management tools.

The proposed language for describing networks is able to represent completely network scenarios supporting the different phases of its life cycle, from the design to operation, maintenance and upgrade. One advantage of this approach is to allow the coexistence of different information about the network utilization in a single description, keeping each one independent from each other, promoting the compatibility and reuse of information directly between tools, overcoming the major limitation identified in existing languages and tools, and promoting interoperability.

Keywords: network management, network description language, XML, network simulation, network simulator 2, network simulator 3, virtual environments

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Motivação

As redes de computadores não apenas cresceram em número nas últimas décadas, mas também em diversidade e complexidade. Os variados dispositivos de rede e locais onde se espera a existência de uma rede, e de preferência conectada à Internet, levaram ao aparecimento de muitas tecnologias de rede. Estas são muito distintas umas das outras, mas, essencialmente, todas foram desenvolvidas de forma a suportar maiores velocidades a uma cada vez maior distância, sempre procurando manter um nível elevado de qualidade na comunicação. Esta afirmação poderá ser aplicada a redes tradicionais e a novas redes, sejam elas com fios ou sem fios.

A boa operação de uma qualquer rede de comunicação depende da existência de ferramentas de suporte que agreguem e mantenham informação detalhada do seu estado. Algumas ferramentas, com funcionalidades específicas, podem ajustar o funcionamento da rede de forma que esta possa cumprir com os objetivos particulares traçados pelos seus responsáveis. Num momento anterior, durante o projeto da rede, houve outras ferramentas que suportaram o trabalho do arquiteto de rede, permitindo-lhe escolher todos os componentes de forma a respeitar os requisitos definidos para a sua futura rede. Durante a operação da rede, o administrador tem ainda a preocupação com a sua atualização, seja em termos de dimensão, seja em termos de novos serviços que deverá suportar. Nesse momento, mais uma vez, o responsável conta com ferramentas de rede para cumprir as suas tarefas, e, consequentemente, garantir o bom funcionamento da rede. Os trabalhos apresentados em [62, 122, 146, 172, 223] descrevem muitas ferramentas de redes presentes em vários domínios apontados nesta tese e verifica-se que o seu número e diversidade não têm parado de aumentar.

Assim, o desenvolvimento, a operação e a manutenção das redes de dados, pela dimensão e heterogeneidade destas, são tarefas que envolvem um número crescente de ferramentas. Estas induzem, nas várias tarefas, uma maior complexidade. As ferramentas de suporte em cada uma das fases ou em cada uma das tarefas são, em muitos dos casos, independentes e incompatíveis, tornando a tarefa dos arquitetos e dos gestores de redes mais difícil. No caso de um gestor de rede ser questionado sobre a possibilidade de a sua infraestrutura ter de suportar novos serviços, este poderá necessitar de utilizar novas ferramentas para conseguir determinar se a rede sob a sua responsabilidade é adequada ou se necessita de alguma atualização. Nesse caso, terá de descrever novamente a rede e as suas características para uma nova ferramenta e poderá então avaliar a inclusão dos novos serviços. As ferramentas têm por vezes especificações variadas para cada um dos

objetos de rede necessários e poderão permitir, ou não, a descrição similar da rede. Da mesma forma, ainda é provável que os objetos de rede de uma ferramenta não existam em outra ferramenta, sendo então necessário optar por objetos semelhantes que representem convenientemente as entidades da rede real. Estas dificuldades tornam complexa a realização de uma análise detalhada e fidedigna de uma rede para diversos cenários de operação. O trabalho apresentado em [10] destaca a necessidade de plataformas novas e avançadas para o teste e validação de novas soluções para a Internet.

Além das limitações das ferramentas, outro aspeto também importante é a aptidão dos utilizadores nas mais variadas ferramentas. Devido às características do domínio das redes, no caso de problemas complexos a aprendizagem da utilização de uma ferramenta pode ser longa. Os trabalhos [43, 211] referem o tempo demorado que é necessário para compreender a linguagem de uma determinada ferramenta. Se ainda for necessário utilizar várias ferramentas que não partilhem informações entre si, então a análise da rede pode ainda ser mais demorada e complexa.

Uma forma de abordar estas dificuldades para a gestão das redes é via a utilização alargada de linguagens/ontologias comuns entre as ferramentas. A informação mais relevante de uma rede é a sua topologia, onde são identificados todos os objetos de rede e as suas relações. Mas outras informações são ainda necessárias durante a operação de uma rede. Algumas serão específicas a um determinado domínio, como a informação para a execução de uma simulação de uma rede, ou a uma tecnologia, como a marcação de pacotes. Estes exemplos expõem o âmbito alargado das necessidades de descrição para uma rede.

Dessa forma, é identificada a utilidade de uma abstração que permita a construção da descrição de uma rede numa ferramenta e a sua reutilização por outras ferramentas de forma, o mais possível, direta. A possibilidade de repetir uma determinada análise de uma rede numa ferramenta e/ou plataforma diferente permitirá ao utilizador confirmar, ou não, as conclusões obtidas inicialmente, mas em ambos os casos a possibilidade de ter informações diferentes sobre a mesma rede permitirá uma melhor fundamentação sobre os resultados do seu funcionamento. No entanto, tal é raramente possível devido à pouca interoperabilidade existente entre ferramentas. A maior dificuldade – a aprendizagem das linguagens das várias ferramentas – é possível de ultrapassar, mas requer tempo, o que por vezes não é viável dentro dos objetivos de alguns projetos.

Apesar de existirem algumas normas para a descrição de redes, como o *Network Configuration* (NETConf) [78] e o *Common Information Model* (CIM) [69], a sua utilização fora dos seus domínios não é possível por a sua extensibilidade ser reduzida. Tal deve-se à sua especificidade num determinado tipo de rede e/ou tecnologia e que dificulta a sua reutilização por ferramentas para outros tipos de redes e ambientes.

Surge então assim a necessidade de uma linguagem genérica para a descrição de cenários de rede que consiga representar detalhadamente todos os pormenores de uma qualquer rede e essa descrição possa ser utilizada por diferentes ferramentas/ambientes, de forma a facilitar a gestão de uma rede.

1.2 Objetivos da tese

A realização deste projeto situou-se no domínio de gestão de redes e na procura de mecanismos e processos para a simplificação da utilização de diferentes ferramentas de redes por parte dos mais variados utilizadores, desde aqueles que se estão a iniciar no domínio das redes aos mais experientes

no contato com diversas tecnologias de rede.

A abordagem proposta consistiu na integração de algumas das ferramentas mais utilizadas pela comunidade de redes com novas bibliotecas de código e ferramentas. Para isso, foi necessário o estudo, a análise e a deteção das principais dificuldades na gestão de redes com as atuais ferramentas e, em seguida, a proposta de um grupo de componentes que permitam resolver e/ou minimizar essas limitações. Os componentes obtidos desse esforço devem permitir, comparativamente às ferramentas existentes, um processo de gestão de redes mais fiável, eficiente e simplificado. Assim, os principais objetivos desta tese são:

- A promoção da interoperabilidade entre ferramentas de diversos domínios de rede, em especial as ferramentas do domínio da simulação de redes, e que suportem as redes com fios, as redes sem fios e as redes com Qualidade de Serviço. Neste objetivo, o aspeto fundamental é fornecer aos utilizadores um conjunto de ferramentas simples que o ajudem na execução de análises avançadas sobre um determinado cenário de rede;

- A proposta de uma estrutura de dados que inclua todos os objetos de rede atuais e futuros, e que permita, de forma simples e completa, englobar as informações de rede contidas nas ferramentas de rede atuais e futuras. Além de ajudar a atingir o objetivo anterior, permitirá ainda a organização dos dados das redes e das suas análises, realizadas pelas ferramentas, de forma mais eficiente, e;

- A verificação dos componentes desenvolvidos em ambientes reais com utilizadores através de diferentes experiências de forma a garantir resultados variados e completos que validem o trabalho desenvolvido.

1.3 Contribuições desta tese

A partir dos objetivos genéricos propostos, as principais contribuições desta tese são as seguintes:

- A proposta e a implementação de uma nova *framework* para a integração de ferramentas de rede, estruturada em três camadas de forma a categorizar e organizar as suas funcionalidades. Além de incluir ferramentas de rede já utilizadas pela comunidade de investigação, foram ainda adicionadas na *framework* duas novas ferramentas para a modelação de cenários de rede, com particular ênfase na simulação de redes de dados. A *framework* é uma proposta para a criação de um ambiente de gestão que suporte o ciclo de vida de uma rede de comunicação;

- A proposta de uma nova linguagem para a descrição de redes e de todos os seus componentes, incluindo a informação da topologia e dos contextos onde a rede pode existir. A linguagem foi desenhada com base em três características fundamentais: simplicidade, permitindo que um humano ou máquina a possa compreender e utilizar; extensibilidade, estando incluída a possibilidade de adicionar novos objetos e características; e, independência, não estar limitada a uma ferramenta ou domínio de rede, podendo as suas descrições serem utilizadas por entidades variadas;

- A proposta de uma metodologia para a integração de todos os componentes, ferramentas e linguagem, na *framework* e para realização de operações variadas sobre uma qualquer rede.

Destaca-se, primeiro, a construção e a validação das descrições de cenários de rede na linguagem NSDL e, depois, a transformação para outras linguagens /ferramentas e execução de análises;

- O domínio da simulação de redes foi escolhido para as primeiras implementações da *framework* e sua validação. Assim, para os simuladores *Network Simulator 2* (NS2) e *Network Simulator 3* (NS3), foram definidos os objetos necessários para a construção de cenários na linguagem NSDL e as bibliotecas de código para a transformação para as linguagens nativas dos simuladores. Além das redes tradicionais, e no caso do NS2, implementaram-se as bibliotecas para o suporte a redes com Qualidade de Serviço. No âmbito do NS3 foram implementadas as bibliotecas para o suporte às redes sem fios, e;
- A integração da *Framework* NSDL em ambientes virtualizados. Mantendo ainda o enfoque na simulação de redes, foram desenvolvidas as bibliotecas para a tradução de descrições de redes NSDL para a execução de simulações de redes em ambientes virtualizados, procurando obter avaliações de redes mais próximas da realidade.

1.4 Notações

Nesta tese utilizam-se por vezes termos genéricos que podem levar a interpretações variadas. De seguida são apresentadas algumas das convenções que se utilizaram na redação desta tese de forma a clarificar a sua utilização.

Um 'cenário de rede' ou 'cenário' ou 'rede' referem sempre todos os aspetos de uma rede de dados, desde os seus objetos até ao seu contexto de operação. Quando é necessário referir um cenário de rede específico, por exemplo, de uma simulação, usa-se a forma 'cenário de simulação' ou 'simulação' e, neste caso, pretende-se apenas referir os dados de configuração de uma simulação para a rede.

O termo 'domínio' de rede é também usado de formas diversas nesta tese. Um domínio de rede é usado para englobar os objetos de rede de um determinado espaço e que partilham um conjunto de regras. Por exemplo, podem pertencer à mesma rede IP e ter ainda subdomínios. Nesta tese o termo 'domínio' será usado de uma forma mais abrangente e simples, indicando objetos de rede relacionados, como o domínio dos nós de uma determinada tecnologia, como no caso da Qualidade de Serviço e a arquitetura dos Serviços Diferenciados (DiffServ), com os nós DiffServ designados por domínio DiffServ.

O estilo <elemento> indica um elemento XML. O estilo @id indica um atributo. As referências a palavras ou partes de código de diversos exemplos surgem no formato de letra código. As ferramentas e linguagens são sempre referidas em *itálico*, exceto quando utilizadas siglas em maiúsculas.

1.5 Organização da Tese

Além deste capítulo, esta tese está organizada em sete outros capítulos, cada um descrito brevemente de seguida:

Capítulo 2: Estado da Arte – inclui o levantamento, a análise e a comparação das principais ferramentas de gestão de redes e das linguagens encontradas para a descrição de redes de

dados. Procuraram-se as principais características e são identificadas as principais limitações em cada grupo;

Capítulo 3: *Framework* NSDL – neste capítulo é descrita a *framework* proposta para o domínio das redes de dados e, mais especificamente, para o contexto da simulação. Cada uma das aplicações e das bibliotecas de código da *framework*, sejam as já existentes, bem como as que foram desenvolvidas no âmbito deste projeto, são apresentadas conjuntamente com as funcionalidades que trazem para a *framework*;

Capítulo 4: *Network Scenarios Description Language* (NSDL) – é feita a proposta de uma nova linguagem para a descrição de redes, a *Network Scenarios Description Language* (NSDL). Seguindo a estrutura da linguagem, é apresentada, primeiro, a forma de descrever os objetos de uma qualquer rede de comunicações e, por último, são apresentados alguns dos cenários onde esta descrição já é utilizada;

Capítulo 5: Implementação NSDL de Redes com Qualidade de Serviço com NS2 – apresenta um caso de estudo da integração do simulador de rede *Network Simulator 2* na *Framework* NSDL. É descrito o processo de integração da ferramenta na *framework* e são expostos alguns casos de estudo que serviram de validação para a implementação. Foi dada uma maior ênfase a alguns cenários de rede com Qualidade de Serviço;

Capítulo 6: Redes Sem Fios com o *Network Simulator 3* – este capítulo tem uma estrutura similar ao Capítulo 5, mas aplicada à ferramenta *Network Simulator 3*. Os cenários destacados na integração do simulador na *framework* foram os cenários das redes sem fios, em particular as redes WiMAX e LTE. São ainda discutidos alguns aspetos sobre a interoperabilidade de cenários entre os simuladores *Network Simulator 2* e *Network Simulator 3*;

Capítulo 7: Simulação de Redes em Ambientes Virtuais – descreve a integração de ambientes virtuais na *Framework* NSDL. São apresentados alguns casos de estudo sobre a implementação de cenários de rede em ambientes virtuais a partir da *Framework* NSDL, e;

Capítulo 8: Conclusões e Perspetivas – este capítulo finaliza esta tese com as conclusões finais sobre todo o projeto, um sumário das principais contribuições e algumas das linhas de investigação para trabalhos futuros.

Capítulo 2

Estado da Arte

2.1 Introdução

No capítulo anterior foram apresentados o enquadramento e a motivação deste trabalho, focando na necessidade de ferramentas cada vez mais complexas para a resolução dos problemas das redes de dados atuais. Um dos problemas identificados é a ineficiência na gestão de redes dada a falta de interoperabilidade entre as ferramentas heterogéneas existentes. Uma possível solução para esse problema seria a possibilidade de reutilização de informação de uma ferramenta numa outra ferramenta diferente, conseguindo assim uma otimização na gestão e uma melhor qualidade da informação sobre o funcionamento de uma determinada rede. No âmbito deste problema, este capítulo apresenta as principais ferramentas de gestão de rede existentes e várias das linguagens e ontologias utilizadas para descrever uma rede e seus componentes.

A primeira parte deste capítulo debruça-se sobre as ferramentas para a gestão e análise de redes de comunicação e que, em alguns casos, foram integradas no desenvolvimento deste trabalho. Estas permitem compreender melhor muitas das tarefas para a administração de uma rede, desde as mais simples, como catalogar os equipamentos, até às mais complexas e avançadas, como avaliação de desempenho. A análise às ferramentas detetou algumas limitações no processo de gestão.

Outro tema muito relevante para este trabalho foi a descrição de redes de dados, principalmente quais as linguagens e ontologias utilizadas para a caraterização da rede. De forma similar às ferramentas, as diversas linguagens encontradas são detalhadas e comparadas, esperando que dessa análise se destaquem os princípios mais importantes na definição de uma linguagem. De forma complementar, são ainda apresentadas algumas das *frameworks* onde são utilizadas algumas dessas linguagens.

Este capítulo está organizado na seguinte forma: a secção 2.2 introduz o tópico da gestão das redes, as suas arquiteturas principais e propostas mais recentes. A secção 2.3 apresenta, por grupos, algumas das ferramentas mais utilizadas para a gestão e análise das redes. A secção 2.4 aborda o tema da descrição de redes, quais as principais propostas existentes, as suas características e os problemas mais comuns. Por fim, a secção 2.5 apresenta as conclusões do estudo feito sobre as ferramentas e as linguagens de redes.

2.2 Gestão de Redes de Computadores

As redes de nova geração, com e sem fios, trazem novas abordagens e tecnologias para a conectividade entre terminais heterogéneos, e irão coexistir com muitas atuais redes por muitos anos. A procura por processos de gestão mais eficientes e eficazes é uma das áreas da investigação das redes sempre presente e cada vez mais necessária para garantir o bom funcionamento e aproveitamento das redes de comunicação. De seguida são apresentadas algumas das arquiteturas tradicionais e algumas das propostas mais recentes para a gestão de redes de comunicação.

As arquiteturas de Gestão OSI [125] e *Telecomunication Management Network* (TMN) [129] representam as abordagens tradicionais para a gestão de redes uma vez que são normas maduras e aceites por muitas entidades da área das redes.

Os princípios para a gestão de redes da organização OSI/ISO são definidos em dois documentos: o OSI *Management Framework* [124] e o OSI *Systems Management* [126]. Ambos surgem após o OSI *Basic Reference Model* [125] onde foi enquadrada a gestão da rede na arquitetura de camadas definida pela OSI e adicionadas algumas definições iniciais para a gestão de redes. Ambos refinaram e clarificaram os diversos componentes presentes no documento anterior, destacando-se a clarificação das áreas funcionais, a troca de informação de gestão e a gestão da informação.

As áreas funcionais, designadas como FCAPS, do acrónimo de *Fault*, *Configuration*, *Accounting*, *Performance* and *Security*, definidas para a gestão foram as seguintes [124]:

- **Falhas**: agrega o conjunto de funções e visam a deteção, isolamento e correção de erros durante a operação de uma rede. Os erros possíveis são: desenho, implementação, sobrecarga, distúrbios externos e longevidade da rede;
- **Configuração**: engloba o conjunto de funções que registam as configurações atuais, registas as alterações nas configurações; identifica os componentes de rede, inicia e encerra sistemas de rede e altera os parâmetros de rede;
- **Contabilidade**: são as funções que permitem cobrar e identificar custos pela utilização dos recursos da rede. Pode ainda ter funções para informar o utilizador sobre a sua utilização da rede e impor limites de utilização;
- **Desempenho**: é responsável pela otimização da Qualidade de Serviço (QoS) da rede. Necessita recolher dados de utilização e da configuração da rede de forma a obter informação estatística da rede e aferir sobre o desempenho da rede ao longo do tempo, e;
- **Segurança**: envolve as funções usadas pelo administrador de forma a prevenir o uso inadequado e não autorizado da rede. A gestão das chaves de rede, a gestão das *firewalls* e a criação de listagens de segurança são exemplos de atividades de segurança.

Os protocolos e funções para a troca de informação de gestão de uma rede foram previstos de três formas: sistemas, camadas e protocolos. A primeira forma envolve informações de gestão geral do sistema, não estando restrito a uma parte ou subsistema da rede. É a forma preferencial, mas não é a única utilizada. A segunda, inicialmente designado como *application*, lida com os serviços, funções e protocolos de cada camada de rede, suportado nos protocolos das camadas inferiores. A última forma está relacionada com cada instância de comunicação e associada a um determinado protocolo.

A gestão da informação (de *Management Information Base*) é a base de dados onde toda a informação pertinente para a gestão da rede reside. Conceptualmente organizada de forma hierárquica, inclui os objetos a gerir (traduzido de *Managed Object*) da rede e toda a informação de gestão associada a esses objetos.

A *Telecomunication Management Network* (TMN) foi introduzida pela organização *International Telecommunication Union*, sector de normalizações para as Telecomunicações (ITU-T) e os seus principais conceitos e elementos estão definidos na família de recomendações M.3000, com ênfase na recomendação M.3010 [129]. Ao contrário da Gestão OSI onde existe uma arquitetura global única, a TMN define três arquiteturas para a gestão de uma rede:

- **Funcional**: onde estão descritas as funções de gestão;
- **Física**; onde está definida a forma de implementação das funções de gestão nos equipamentos físicos, e;
- **Informação**: a partir de um dado momento a colaboração com o grupo de Gestão OSI levou a serem seguidos os mesmos princípios desse grupo para gestão da informação.

De seguida apresentam-se algumas abordagens existentes para a gestão das redes.

2.2.1 Abordagens para a Gestão de Redes

As arquiteturas apresentadas nesta secção destacam-se dadas as alterações significativas na forma de estruturar a gestão e controlo das redes, relativamente às arquiteturas tradicionais (OSI e TMN). Assim, as arquiteturas a apresentar de seguida são a 4D, a *Maestro* e a *In-Network Managment*.

A arquitetura 4D [99, 284] propõe uma redefinição completa da gestão e controlo da Internet, não considerando adequada qualquer abordagem que estenda as funções de gestão atuais, por tal tornar cada vez mais complexo o desenho atual das redes. Esta contribuição é uma sequência das ideias propostas no trabalho [60] e a ideia principal é separar a lógica da decisão sobre a rede, ou quais os requisitos definidos para a rede, dos protocolos subjacentes que irão garantir a operação da rede cumprindo os seus objetivos. Conduz também a uma abordagem centralizada da gestão, contrária a outras abordagens mais tradicionais onde a gestão é realizada de forma distribuída, suportada em diferentes protocolos de rede.

A arquitetura 4D toma o seu nome da existência de 4 componentes: Decisão, Disseminação, Descoberta e Dados. A camada de Decisão é responsável por todas as decisões para o controlo da rede, como abrangência, balanceamento de carga, controlo de acesso, segurança e configuração de interfaces. A camada de Disseminação garante comunicações eficientes entre os equipamentos de rede e os elementos de decisão. A camada de Descoberta pesquisa e identifica todos os elementos da rede. A camada de Dados gera cada um dos pacotes baseado no estado definido na camada de Decisão. Este estado inclui as tabelas de encaminhamento, os filtros de pacotes, o agendamento, a gestão das filas, os túneis e a tradução de endereços.

A arquitetura 4D busca as seguintes vantagens: maior robustez e segurança, acomodar mais adequadamente a heterogeneidade presentes nos ambientes de rede, e, por último, facilitar a evolução e inovação das redes.

Uma extensão ao modelo 4D surge com o *Complexity Oblivious Network Management* (*CONMan*) [18], reconhecendo as vantagens da sua estrutura, mas sendo mais flexível e abrangente nas

funções de gestão. Tem como objetivos conseguir desenhar redes: autoconfiguráveis para minimizar o erro humano, que tenham validações contínuas às configurações de forma a cumprirem os objetivos da rede, que suportem interfaces de gestão seguindo normas abstratas, e, operem sobre especificações declarativas, ou seja, instruções de alto nível que serão depois convertidas para implementações de baixo-nível.

Com uma diferente abordagem, mas partilhando o princípio de não evoluir os métodos tradicionais de gestão tal como o 4D e *CONMan*, surge a proposta *Maestro* [36, 37]. Este trabalho propõe para a gestão e controlo das redes os princípios dos sistemas operativos, onde num ambiente unificado existirão aplicações para o controlo dos elementos da rede e que orquestram todo o funcionamento da rede. O ambiente *Maestro* fornece um interface para a implementação modular de componentes que acederão e modificarão o estado da rede. É também responsável por manter a consistência dos estados de rede entre os diversos componentes ativos. Ou seja, é uma plataforma para o controlo de rede automático e programável (*Java*) através de aplicações modulares [38].

A plataforma *Maestro* tem sido testada conjuntamente com o projeto *OpenFlow* [170]. Neste é proposto para os *switchs* da rede um interface normalizado para a manipulação das tabelas internas de encaminhamento, colocando num controlador externo a complexidade e as decisões de gestão.

Enquanto as arquiteturas 4D e *Maestro* centralizam a gestão de uma rede, com algumas diferenças na flexibilidade com que o fazem, a proposta *In-Network Managment* (INM) [73, 88] é uma arquitetura que coloca em todos os elementos de uma rede alguma capacidade de gestão e, através da colaboração entre eles, visa conseguir uma gestão eficiente e eficaz da rede. As diferentes capacidades, ou funcionalidades, de gestão são embebidas em cada componente de rede são organizadas da seguinte forma [206]: inerente – indica as funções de gestão inerentes à própria lógica componente; integrada – as funções de gestão integram o componente, mas são independentes da lógica do componente; e, externa – são as funções de gestão localizadas em outro nó de rede.

A principal vantagem que a arquitetura INM procura atingir é ser muito escalável, comparativamente a arquiteturas centralizadas. A limitação desta solução está na dificuldade em definir os interfaces abstratos genéricos para a gestão. Ainda outra dificuldade presente surge na conversão de políticas de gestão globais para uma realidade de gestão distribuída.

Surgem ainda algumas reflexões sobre os desafios e dificuldades para a gestão de redes modernas. Em [9] são indicados alguns dos desafios para a gestão da Internet, como: o aparecimento de novas tecnologias, como o *OpenFlow*, e a sua gestão; conseguir uma gestão escalável, mas mantendo uma rigorosa gestão dos estados da rede; novos mecanismos para a automação das funções de gestão ; e, acesso facilitado a dados de configuração e da operação da rede de forma a reforçar a validação dos métodos de gestão. O trabalho apresentado em [247] reflete sobre outros aspetos das novas redes, como enquadrar na gestão da Internet as zonas de rede sem gestão e a dificuldade em manter a gestão escalável. Neste trabalho é feita a proposta de dois planos para a gestão: Informação e Conhecimento, onde o primeiro aborda os elementos para a aquisição da informação e o segundo é responsável pela análise da informação.

Nesta secção foram apresentadas as principais soluções para a gestão de redes tradicionais e futura Internet. As abordagens tradicionais têm sido adequadas em muitos ambientes, mas com grandes limitações em termos de automatização, suporte para a implementação de políticas globais dinâmicas e gestão das falhas. As abordagens mais recentes propõem através de novas arquiteturas e filosofias de gestão simplificar e agilizar a gestão de redes cada vez mais complexas. Algumas

mais extremas em termos de centralização, como a 4D, ou, no lado oposto, distribuindo por muitos componentes da rede a gestão, como a INM, existe ainda a *Maestro* como solução intermédia para a gestão das redes.

A tarefa de manter uma rede a operar corretamente é a responsabilidade principal de um administrador de rede. Em qualquer arquitetura de gestão as ferramentas de rede serão sempre fundamentais para apoiar o administrador a realizar as funções que garantem a operação adequada da rede. Na próxima secção são discutidas algumas das ferramentas mais utilizadas para a gestão de redes.

2.3 Ferramentas de Gestão de Redes

A gestão de uma rede realizada no âmbito de uma determinada arquitetura envolve um conjunto de atividades, métodos, procedimentos e ferramentas para a sua administração, operação e manutenção. Qualquer que seja a arquitetura, o número de tarefas é normalmente grande e, entre elas, existem algumas muito complexas e/ou demoradas de realizar. Além da dificuldade em realizar manualmente as tarefas, sem o apoio de ferramentas a probabilidade de cometer erros ou omissões nas configurações é maior [39, 53]. Como exemplo de tarefas temos o mapeamento, o controlo, a manutenção, a monitorização e a segurança da rede. Numa rede de média ou grande dimensão qualquer uma destas tarefas não pode ser executada unicamente pelo administrador e este necessita do apoio de ferramentas que incluam funcionalidades de suporte que o ajudem a realizar todos os passos requeridos para gerir convenientemente a sua rede.

A forma mais tradicional de agrupar as ferramentas poderá ser feita seguindo os grupos de áreas funcionais indicadas para a arquitetura da OSI. Assim, e seguindo o acrónimo FCAPS, teremos ferramentas para a gestão das falhas, das configurações, da contabilização, do desempenho e da segurança. No entanto, grande parte das ferramentas pesquisadas durante este trabalho contêm funções associadas a várias áreas funcionais, não sendo assim simples a sua organização seguindo este princípio. Por vezes encontram-se ferramentas para operações específicas e de âmbito limitado (e.g., descoberta de componentes de rede) e ferramentas genéricas com muitas das funções de gestão da rede, incluindo todas (ou quase todas) as áreas. Como seria de esperar, outras ferramentas aglomeram funções de apenas algumas das áreas funcionais. Como exemplo de alguns tipos de ferramentas de rede comuns podemos indicar: a geração de topologias, a geração de tráfego, a gestão de redes, a monitorização, a segurança, a simulação, e a visualização.

Assim, e para enquadrar cada uma das ferramentas apresentadas neste capítulo, foram escolhidas três categorias consideradas representativas das tarefas principais da gestão de redes. As categorias são as seguintes: *Modelação*, *Monitorização* e *Simulação*.

É possível considerar que a variedade de ferramentas e de áreas da gestão da rede presentes nas diversas arquiteturas permitiriam a escolha de outras categorias, em número e significado, mas sempre aproximadas das indicadas. As categorias apresentadas são, para o autor deste documento, representativas das funções mais importantes e comuns no domínio da gestão das redes e compõem uma divisão simples e fácil de compreender.

De seguida são apresentadas algumas das principais ferramentas de cada uma das categorias.

2.3.1 Modelação

A *Modelação* de uma rede envolve a identificação dos objetos de uma rede e dos seus parâmetros, bem como das relações que existem entre os vários objetos. O resultado desta atividade é uma base de dados com as entidades presentes numa rede e a informação de como elas estão interligadas, permitindo conhecer o mais fielmente possível o fluxo de informação. A estrutura dos nós, e as suas ligações, revelam a topologia física da rede, mas a interação entre as aplicações e protocolos fazem emergir outras topologias, neste caso, lógicas. Pode ainda incluir outros dados relacionados com a operação da rede, como os eventos que poderão ocorrer (por exemplo, ativar/desativar um equipamento num dado instante).

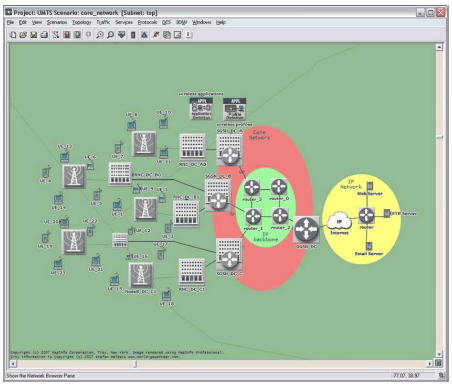
Portanto, as ferramentas de *Modelação* servem para criar uma base de dados de objetos que representem os componentes da rede, bem como de toda a informação necessária para a sua configuração. De forma complementar, podem ser usadas ferramentas automáticas que procurem na rede todos os elementos e as suas características, bem como as ferramentas que permitam ao administrador visualizar graficamente a topologia da rede e consultar os seus componentes. Uma última funcionalidade importante dessas ferramentas é o utilizador poder interagir com elas para editar/adicionar objetos e configurações. Estas ferramentas podem ser usadas sobre uma rede já existente e a operar e/ou sobre uma rede ainda não construída, podendo estar em fase de desenho e análise.

O processo de *Modelação* pode ser realizado manualmente pelo administrador da rede onde ele identifica e carateriza todos os equipamentos de uma rede ou pode obter o apoio de ferramentas neste processo. As ferramentas apresentadas de seguida estão divididas em três grupos: ambientes gráficos, geradores de topologias e descoberta da rede.

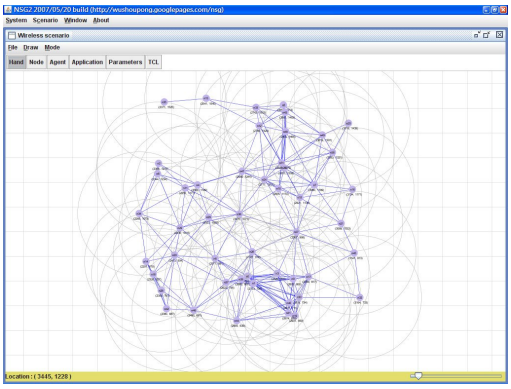
Os ambientes gráficos para a *Modelação* de uma rede são programas que permitem construir uma rede a partir de uma base de dados de objetos. A partir destes objetos o utilizador constrói a sua rede e realiza toda a sua configuração/parametrização. De seguida são apresentadas as aplicações *Opnet Modeler* [156] e NS2 *Scenarios Generator* (NSG2) [280], como exemplos de ferramentas para a modelação de redes para a simulação. A escolha desta área deve-se a esta ter ferramentas muito poderosas e que permitem a manipulação de muitos objetos e parametrizações longas e complexas. Serão também apresentadas duas ferramentas que não pertencem ao domínio de redes, mas que permitem a descrição de uma rede, o *Visio* [174] e *Dia* [67].

O *Opnet Modeler* está integrado na plataforma de simulação de eventos discretos *Opnet* (descrito na secção 2.3.3) e é um ambiente completo para a descrição de muitos componentes das redes, como as topologias de rede em diversos meios físicos, os interfaces de rede, os protocolos de todas as camadas e ainda muitas aplicações. A capacidade de parametrização dos objetos está relacionada com a capacidade do simulador *Opnet*, mas é bastante alargada e permite a simulação de, praticamente, todos os tipos de redes.

O NSG2 é uma ferramenta para a modelação de cenários de simulação para o simulador de redes *Network Simulator 2* (NS2) - descrito em detalhe na secção 5.2 - e é baseada na linguagem Java. Os objetos disponíveis para o utilizador permitem a definição flexível de redes com fios e sem fios, mas a quantidade de objetos é pequena e as configurações disponíveis são limitadas. À semelhança da ferramenta *Opnet Modeler* para o *Opnet*, o NSG2 apenas gera simulações para o NS2.



(a) *Opnet Modeler*



(b) NSG2

Figura 2.1: Ambientes gráficos das ferramentas para a Modelação de redes

A Figura 2.1 apresenta os ambientes do *Opnet Modeler* (2.1(a)) e NSG2 (2.1(b)) e, apesar das diferenças de funcionalidades e possibilidades entre ambos, o processo de construção de uma rede é similar.

Um diferente grupo de ferramentas que também permite a descrição de redes inclui as aplicações *Visio* e *Dia*. Ambas permitem, num ambiente gráfico poderoso, a construção de base de dados de objetos genéricos e relacionamentos entre eles. Ambas contêm grupos de objetos para as redes de computadores e, apesar de permitirem apenas descrições simples, podem ser estendidas com mais objetos e parâmetros. São independentes de outras ferramentas de redes, mas o seu formato está bem documentado e permitem a implementação de conversores para diferentes ferramentas. São aplicações para o *Desktop*, mas também já existem ferramentas similares a operar na Internet, como o *Gliffy* [97].

Um grupo diferente de ferramentas são os Geradores de Topologia. Estas ferramentas permitem a construção rápida de redes seguindo diferentes modelos bem definidos e que capturam o comportamento de uma determinada rede. Permitem tanto a construção de rede com pouco nós como redes com muitos milhares de nós. Este último processo seria demorado se fosse feito manualmente. Os geradores de topologias são usados principalmente para a investigação de novos protocolos e componentes de rede.

O gerador BRITE [171] é um dos mais usados por ter a capacidade de gerar topologias para alguns dos simuladores mais usados (NS2, OMNet++, J-SIM, SSFNET, descritos na secção 2.3.3). O BRITE agrega um número grande de modelos de topologias conhecidos e, assim, espera conseguir topologias que representem o mais aproximadamente a atual Internet. Além de um ambiente gráfico para a configuração dos modelos, uma funcionalidade importante do BRITE é a capacidade de reconhecer os formatos de outros geradores de topologias e poder traduzi-los para o seu formato, agregando assim muitas funcionalidades.

Apesar do BRITE ser dos geradores mais usados, outras ferramentas têm surgido com o propósito de ultrapassar algumas das suas limitações. Os trabalhos apresentados em [52] e [161] permitem de uma forma escalável gerar cenários de muito maior dimensão (milhões de nós) e mais realistas, oferecendo uma melhor representação da Internet. Outras abordagens mais específicas geram topologias mais fidedignas relativamente a um componente da rede. O trabalho descrito

em [255] apresenta uma melhor modelação das relações entre domínio a nível do encaminhamento, baseado em casos reais, e que permitirá uma avaliação mais real de muitos protocolos. Uma área emergente são as redes de sensores sem fios e no projeto desenvolvido em [40] os autores propõem um gerador de topologias para estas redes que captura as suas principais características e onde é possível adicionar os parâmetros mais relevantes, como aspetos da energia.

Por último nesta secção, a descoberta automática dos equipamentos e das topologias das redes a operar apoia o administrador no processo de conhecer detalhadamente as características da sua rede. Através de diferentes métodos, normalmente baseados nos protocolos de rede *Internet Control Message Protocol* (ICMP) [216] e *Simple Network Management Protocol* (SNMP) [105], estas ferramentas pesquisam toda a rede e recolhem informações sobre todos os equipamentos e do seu estado.

A aplicação *Network Mapper* (*Nmap*) [160, 190] usa pacotes IP de uma forma flexível para encontrar os nós da rede, quais os serviços ativos nesses nós, os seus sistemas operativos e muitas outras características. Existem ainda outras ferramentas semelhantes ao *Nmap* com maior ou menor enfoque em determinados detalhes da descoberta de características de uma rede e que incluem algumas ferramentas profissionais como o *Nessus* [233] e o *Open View* [292], ou ainda ferramentas específicas para certos tipos de rede como o trabalho de [24] para a descoberta de topologias em redes locais.

Um grupo recente de ferramentas tem tido um foco particular nas redes sem fios. Os projetos *Netstumbler* [175], *Netsurveyor* [183] e *Kismet* [135] identificam ativamente ou passivamente as características das redes sem fios (Wi-Fi) num determinado espaço, e determinam quais os protocolos de segurança a ser utilizados e outros parâmetros que determinam o grau de segurança/insegurança na operação de determinada rede.

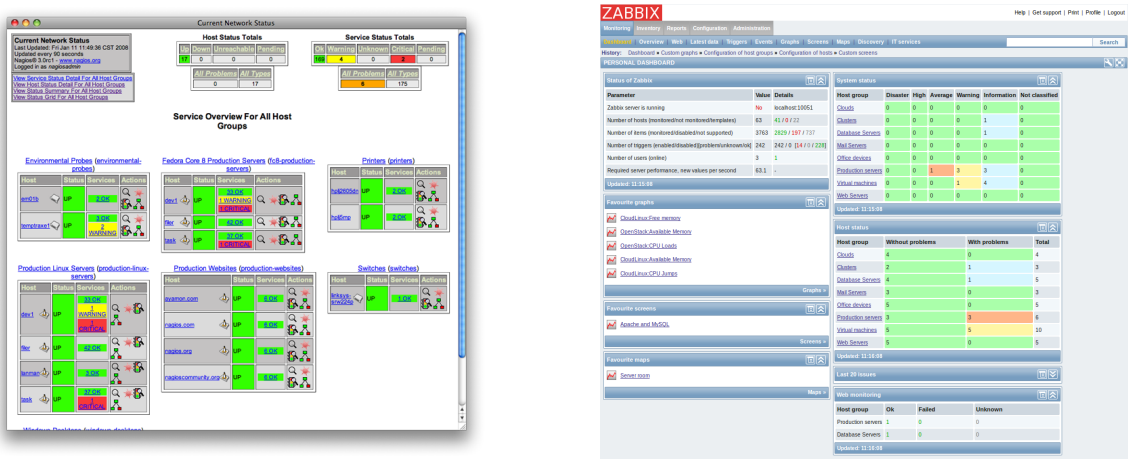
Surgem ainda por vezes ferramentas de um grupo diferente, mas importante para a gestão de redes e são as ferramentas de visualização. Por serem, em parte, redundantes com as ferramentas apresentados nos ambientes gráficos não tiveram um destaque nesta secção. Mas é importante referir que muitas ferramentas, como o BRITE, geram ficheiros de texto apenas e não tem aplicações para a visualização das redes geradas. Os projetos *Open Graph Viz* [23] o *aiSee* [8] e o *Flare* [106] são exemplos de ferramentas para a visualização de redes de qualquer tipo e podem complementar muitas das ferramentas de redes.

Um administrador conhecendo corretamente e de forma ampla os objetos da sua rede pode então avançar para um grupo de tarefas diferente, e também fundamentais, manter a rede a operar com o mínimo de falhas e de forma eficiente. A secção seguinte apresenta as ferramentas para a monitorização.

2.3.2 Monitorização

A *Monitorização* inclui as ferramentas que irão continuamente verificar o funcionamento da rede e gerar alertas quando surgirem falhas e/ou problemas de rede. Também irão obter o estado da rede em todos os momentos e guardá-lo em histórico, bem como as configurações e os eventos ocorridos. Estas ferramentas apoiam o administrador a cumprir a tarefa de manter a rede ativa e a operar como determinado nos seus requisitos.

A *Monitorização* de uma rede inclui todas as tarefas que visam garantir o bom funcionamento de



(a) *Nagios* (b) *Zabbix*

Figura 2.2: Ferramentas para a *Monitorização* de redes

uma rede de uma forma contínua. O agente monitor, constantemente, verifica todos os componentes da rede e comprova a sua boa operação ou não. Pode, de forma automática ou após comando do administrador, atuar sobre a rede que está a vigiar quando surgem os mais diversos eventos. Pode também gerar informação para a otimização do funcionamento e ainda prevenir problemas futuros.

A *Monitorização* é, por vezes, apenas referida como responsável pela recolha dos dados do funcionamento da rede, como por exemplo quais os protocolos, serviços e aplicações ativas e qual o seu estado. Nesta secção, para além dessa responsabilidade, a *Monitorização* foi ainda entendida como responsável por gerar alertas, respostas e relatórios, apoio à manutenção e ao planeamento da rede.

As ferramentas de *Monitorização* existentes são muito numerosas e uma lista mantida em [62] destaca-se por ter uma descrição sucinta e atualizada de um grande número de ferramentas criadas desde 1996, e com apontadores para mais informação sobre cada uma delas. As ferramentas de monitorização apresentadas de seguida foram escolhidas por incluírem as tarefas indicadas e por surgirem em diferentes comparações como projetos completos e de qualidade para a gestão das redes. Um primeiro grupo de ferramentas abrange plataformas de monitorização, ou seja, ferramentas que incluem muitas funcionalidades e apoiam todas, ou quase todas, as tarefas referidas para a monitorização de redes. O segundo grupo de ferramentas inclui exemplos que se destacam na realização de um grupo mais restrito de tarefas de monitorização.

A plataforma *Nagios* [104, 128, 178], apresentada na Figura 2.2(a), é uma das plataformas mais utilizadas para monitorização de redes. Engloba todas as tarefas de monitorização já referidas e ainda inclui ferramentas avançadas para a visualização global de toda a infraestrutura e/ou dos detalhes de cada componente. O sistema de alertas é muito flexível, permitindo rapidamente detetar as falhas e notificar o administrador quando algo está fora de limites predefinidos. Pode ainda tomar medidas automáticas para reativar os serviços. Os relatórios de estado permitem obter muitas informações, como por exemplo sobre os *Service Level Agreements* (SLA) acordados para a rede, e mantém um histórico completo sobre a atividade da rede.

A arquitetura da plataforma permite que a adição de novos componentes seja simples, garantindo que a plataforma se mantenha atualizada na tarefa de monitorização. Através desta ferra-

menta é simples aos administradores adicionarem novas regras de monitorização utilizando vários métodos, como uma linguagem declarativa ou uma linguagem de programação comum e ainda através de um ambiente gráfico. A grande comunidade de utilizadores partilha conhecimento, experiências e muitos componentes, garantindo a um administrador suporte extenso e rápido na utilização da plataforma. A plataforma é escalável, suportando a monitorização de milhares de nós.

A plataforma *Zabbix* [198, 252, 264], apresentada na Figura 2.2(b), é também uma solução com muitas funcionalidades para a gestão de redes. Permite a monitorização de muitos tipos de equipamentos e de serviços. Contém bibliotecas de código avançadas para a descoberta automática de objetos, permitindo descobrir muitas das suas características. Mantém uma arquitetura descentralizada para a monitorização, mas a administração é simples e centralizada. Os agentes monitores são instalados nos equipamentos e manterão uma base de dados local com todas as informações da operação deste. No caso de não ser possível a instalação, o *Zabbix* usa os métodos comuns, como o SNMP, e comandos *ping*, *traceroute*, etc., para verificar o estado dos serviços e do hardware de um determinado equipamento. A administração pode ser feita num ambiente multiutilizador, oferecendo a possibilidade de divisão de tarefas na monitorização.

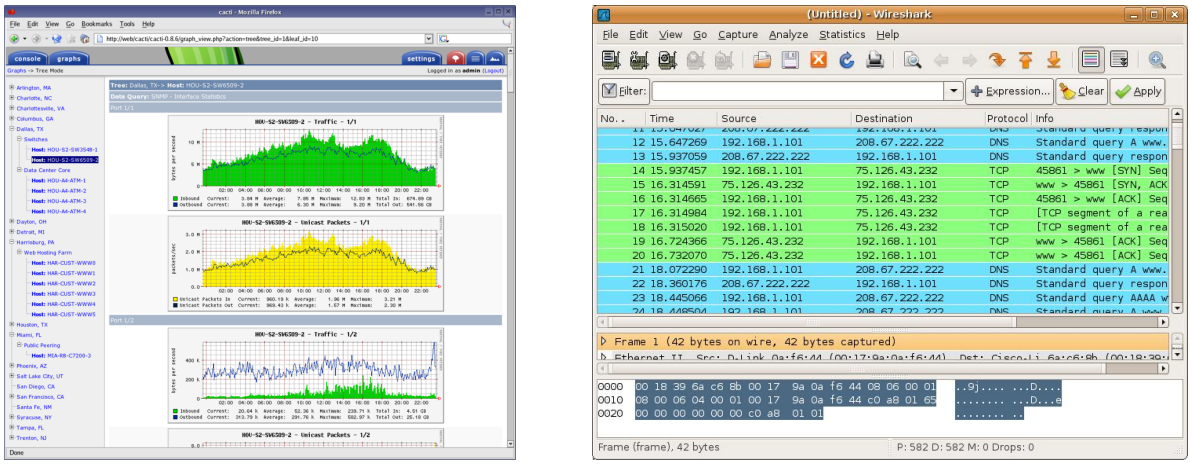
O *Nagios* e *Zabbix* são plataformas da comunidade de *software* livre e sem custos de aquisição, pelo menos para as versões base. O modelo de negócios é baseado na prestação de serviços de suporte e de desenvolvimento de componentes específicos para os seus clientes. As ferramentas *Spiceworks* [249] e *Ganglia* [168, 169, 236] são exemplos de outras ferramentas de monitorização de *software* livre que, de forma geral, oferecem o mesmo tipo de serviços a um administrador de rede que o *Nagios* e *Zabbix*, mas destacam-se numa ou várias características.

O *Spiceworks* além das funções de monitorização da rede contém funcionalidades avançadas para o inventário, para o processo de aquisição de equipamento e para o apoio a utilizadores (*helpdesk*). Integra também no software facilidades para interação e colaboração com a comunidade de utilizadores da ferramenta.

O *Ganglia* é um software especializado para a monitorização de sistemas de computação de alta velocidade, como *clusters* e *grids* de computadores. É muito escalável, mantendo uma estrutura rica de monitorização sem exigir muitos recursos a nível de processamento ao sistema graças a estrutura de dados e algoritmos eficientes. Utiliza linguagens normalizadas para a representação de dados e para a sua portabilidade de forma compacta com as linguagens XML [268] e XDR [76], respetivamente.

Ainda no grupo de plataformas de monitorização surgem muitas opções comerciais. Estes projetos, em comparação com os de *software* livre, são mais completos, simples de utilizar e com melhor documentação. O suporte, profissional e sob contratos comerciais, é normalmente mais rápido e competente. As plataformas *Bigbrother4* (BB4) [219] e *Zenoss* [290] são dois exemplos de plataformas de monitorização comerciais.

Ainda como ferramentas para a monitorização de redes, temos ainda ferramentas especializadas na apresentação do estado da rede ao longo do tempo, mostrando a largura de banda de uma determinada ligação ou o nível do processador de um equipamento. A ferramenta *Cacti* [150, 258], apresentada na Figura 2.3(a), tem como objetivo principal a análise de dados e permitir a construção de gráficos complexos sobre esses dados. Contém módulos para a aquisição de dados das redes e, sobre esses dados, permite a um administrador conhecer de forma avançada o estado



(a) *Cacti* (b) *Wireshark*

Figura 2.3: Ecrãs das ferramentas *Cacti* e *Wireshark*

e comportamento da sua rede.

Por fim, uma última atividade, a captura de pacotes, é mais uma técnica para um administrador poder conhecer em detalhe a sua rede. A ferramenta *Wireshark* [50, 238], apresentada na Figura 2.3(b), é, talvez, a mais conhecida para essa atividade e permite a análise dos pacotes de rede. Consegue capturar o tráfego de redes com fios e sem fios de forma simples e permite análises detalhadas sobre todo o tráfego a circular na rede. É uma ferramenta útil por permitir ao administrador compreender a causa de problemas comuns da rede, como perdas de conectividade, dificuldades em contactar o servidor de DNS e/ou baixa velocidade da rede. A ferramenta já inclui muitos filtros para as análises mais frequentes e ainda permite a adição de novos filtros de captura definidos pelos administradores. Para a tarefa de análise da rede ainda tem funções gráficas para a visualização do tráfego capturado e possibilidade de gerar diversos tipos de relatórios.

Uma primeira divisão nas ferramentas de monitorização mais completas surge no seu licenciamento. Os primeiros exemplos são aplicações *open source* – *Nagios*, *Zabbix* e *Ganglia* – e podem ser utilizadas por qualquer administrador sem restrições. O segundo grupo são ferramentas comerciais – BB4 e *Zenoss* – onde é necessário existir um contrato de licenciamento para poderem ser utilizadas. Apesar de existirem diferenças nas funcionalidades, o núcleo principal das funções de monitorização estão presentes em todas as ferramentas. A maior diferença entre ferramentas foca-se no suporte que é prestado, mais ou menos profissional e com tempo de resposta previsível, na utilização da ferramenta e no apoio ao processo de gestão da rede.

A visualização e análise do funcionamento da rede é um componente presente em todas as ferramentas, mas existem proposta mais ricas, como as ferramentas *Cacti* e *Multi Router Traffic Grapher* (MRTG) [195], que oferecem funcionalidades avançadas para essas tarefas. Por fim, o *Wireshark* é uma ferramenta distinta das anteriores porque permite obter novas perspetivas sobre o estado e funcionamento da rede, principalmente nos fluxos existentes e nas configurações detalhadas dos protocolos que estão a ser utilizados.

Uma característica procurada nas ferramentas foram as possibilidades de importação e exportação de dados e configurações. Cada ferramenta tem as suas próprias bases de dados e não permitem a interoperabilidade com outras ferramentas. A única referência de exportação surge

apenas em contexto de atualização de versões, ou seja, a transferência de informações de uma ferramenta para outra de uma versão superior. Caso um administrador opte por mudar de ferramenta de monitorização terá de desenvolver um processo para a conversão de dados, ou a opção é perder todo o histórico de dados e de configurações da rede.

As ferramentas de monitorização permitem recolher informações sobre o funcionamento da rede e atuar de forma a manter o seu funcionamento dentro dos parâmetros definidos pela administração da rede. A informação disponível permite também realizar projeções sobre o futuro funcionamento da rede ou ajudar a compreender o que poderá acontecer se ocorrer alguma alteração (adição de nova zona ou então novas aplicações), mas em ambas as situações este tipo de ferramentas apenas poderá fornecer informações limitadas. Para procurar conhecer a rede de forma mais avançada, principalmente em cenários ainda não existentes, surge o grupo de ferramentas de simulação. De seguida são apresentadas algumas dessas ferramentas.

2.3.3 Simulação

A modelação e a análise de uma simulação sobre um sistema físico envolvem o processo de criação e de experimentação de modelos matemáticos computorizados que representem esse sistema [56]. As redes de dados são um exemplo de um domínio onde a simulação tem sido extensivamente usada para o teste de novos componentes e para a melhoria da eficiência na sua utilização. Assim, o uso de simulação permite: obter um maior conhecimento dos detalhes da operação de um sistema, desenvolver políticas de utilização para melhorar o desempenho, teste de novos conceitos antes da implementação, e, obter informação sem causar interferência no sistema real [208]. Além das características já referidas, as vantagens da utilização de simulação, são, segundo o discutido em [56]: experimentação em tempo reduzido, diminuição dos requisitos analíticos e demonstração dos modelos facilitada.

Os simuladores de rede permitem análises muito ricas e variadas sobre as mais diversas redes, estejam estas em funcionamento ou ainda em desenvolvimento. São um método muito importante para conhecer em detalhe a operação dos componentes de uma rede e são muito utilizadas por várias comunidades relacionadas com as redes, de onde se destaca a comunidade de investigação. Mas, apesar das vantagens enunciadas, surgem ainda algumas dúvidas na sua utilização. O trabalho apresentado em [207] revela as limitações e as incorreções na utilização de gerador de números pseudoaleatórios e na análise dos resultados das simulações, colocando em causa a credibilidade de muitos trabalhos baseados em simulações. Em [146] são indicadas as dificuldades na repetição das simulações devido a existirem poucas informações sobre os simuladores ou mesmo à impossibilidade de os utilizar, aos desvios na inicialização da simulação com prejuízo sobre a identificação da estabilidade do sistema a simular, e, por último, as dúvidas sobre a credibilidade das análises estatísticas que sustentam os resultados. Assim, o seu uso deverá ser sempre bem ponderado. Em [110] os autores analisam as principais limitações de anteriores simuladores e indicam uma nova abordagem no desenvolvimento de um novo simulador, onde se incluem um conjunto de processos simples e transparentes para a documentação das simulações.

O grupo *Simulação* agrega as ferramentas que permitem análises detalhadas ao funcionamento da rede em diferentes domínios, como o balanceamento da carga, a segurança e o tratamento de fluxos específicos. São indicadas apenas as ferramentas que permitem simular uma determinada

configuração de rede ou sequência de eventos. Além da construção detalhada de todo o cenário de simulação, considera-se relevante a possibilidade de poderem ser utilizados os dados adquiridos pelas ferramentas da *Monitorização* ou as configurações definidas pelas ferramentas de *Modelação*.

Assim, pela relevância que estas aplicações têm no domínio das redes e pela flexibilidade que possuem para a análise de diferentes aspetos do funcionamento de uma rede ou de um seu componente, o tipo de ferramentas escolhido para ser detalhado nesta secção foi o simulador de redes, em particular, de eventos discretos.

O número de simuladores desenvolvidos nos últimos anos é demasiado grande e efetuar uma descrição detalhada de todos necessitaria de um espaço diferente do concedido neste documento. Apenas estão presentes as ferramentas cujo objetivo é a simulação de redes de dados. Foram ignoradas muitas ferramentas para redes específicas, como as redes de sensores sem fios e as redes veiculares, ambas como redes emergentes nos últimos anos, optando-se por focar o trabalho em ferramentas indicadas como genéricas para a simulação de um qualquer tipo de rede. Algumas das ferramentas mais usadas para simular rede de sensores sem fios são listadas e comparadas nos trabalhos de [75, 122]. De forma similar, o trabalho de [244] apresenta alguns dos simuladores mais utilizados para avaliar as redes sem fios veiculares. Segue-se então a apresentação dos simuladores.

O *Integrated MUltiprotocol Network Emulator/Simulator* (IMUNES) [288] é um simulador baseado no núcleo do sistema operativo FreeBSD [92], onde são criados múltiplos nós e ligações virtuais podendo formar as mais diversas topologias. Uma característica importante dos nós virtuais é terem as mesmas possibilidades oferecidas pelo núcleo real, permitindo assim simulações próximas do ambiente real, quer a nível da rede quer a nível das aplicações.

O simulador *J-Sim* (inicialmente *JavaSim*) [257] é um ambiente de simulação que utiliza a linguagem Java e é baseado em componentes autónomos. É complementada com linguagens de *scripting* para descrever as simulações, destacando-se o *Tcl* [15] com o *Jacl*. Dispõe de modelos para os objetos mais comuns da arquitetura Internet, algumas arquiteturas de QoS e redes de sensores sem fios [130].

O *JiST/Swans* é um simulador baseado na linguagem Java e está dividido em dois componentes. O *Java in Simulation Time* (JiST) [21] é um simulador de eventos discretos de alto desempenho que funciona sobre uma máquina virtual Java. O *Scalable Wireless Ad hoc Network Simulator* (Swans) [22] está organizado num conjunto de componentes de software independentes e que pode ser composto para formar redes sem fios e redes de sensores sem fios. Os autores destacam a sua eficiência e desempenho na simulação de redes de grande dimensão, comparativamente a outros simuladores semelhantes [20].

O *NCTuns* [274] é um simulador/emulador baseado no núcleo do Linux e que permite a simulação de muitos tipos de redes. A técnica usada pelo simulador é designada de *kernel re-entering* [273] e permite o uso dos recursos do núcleo do Linux, permitindo criar simulações sobre os componentes reais de um sistema operativo como as camadas protocolares de rede, as aplicações e outros utilitários para a configuração, monitorização e análise de resultados e estatísticas.

O *Network Simulator 2* (NS2) [34] é um dos simuladores mais usados para investigação. Uma das suas principais características é a utilização de duas linguagens, C++ [251] e *Object-oriented Tool Command Language* (OTcl) [279], método designado por *split-programming*, para a construção de simulações. A programação das atividades de baixo-nível, onde é necessária rapidez, é feita com o C++, destacando-se o tratamento do encaminhamento de pacotes e o protocolo TCP. As

operações de alto-nível, como obter estatísticas sobre a simulação e a modelação de falhas nas ligações, é feita recorrendo ao OTcl. O grande número de modelos presente no simulador permite a simulação de cenários para muitas das redes atuais. Muitos dos modelos são contribuições dos próprios utilizadores do NS2.

O *Network Simulator 3* (NS3) é uma evolução do NS2, mas numa nova arquitetura de *software*, não existindo compatibilidade entre as descrições de cenários de ambos os simuladores. Os princípios da arquitetura do NS3 são: a escalabilidade, a extensibilidade, a modularidade, a emulação, o desenho claro e a boa documentação [110]. Utiliza apenas uma linguagem, o C++, usado em toda a simulação. Existe a possibilidade, opcional, da utilização de uma segunda linguagem apenas para a descrição de simulações, o *Python* [218]. Uma característica também importante no NS3 é o realismo, ou seja, o desenho do simulador deve aproximar-se das redes e dos objetos de rede reais.

O simulador OMNeT++ é um ambiente de simulação de eventos discretos, utilizando uma arquitetura de componentes para os seus modelos [262, 263]. Utiliza a linguagem C++ para a construção dos modelos e utiliza a linguagem de alto-nível *NEtwork Description* (NED) [261] para definição de componentes maiores e modelos. Destacam-se no OMNet++ as ferramentas gráficas para as tarefas de simulação (baseado no IDE *Eclipse* [74]) e a possibilidade de integração do núcleo do simulador em outras aplicações.

O *Optimized Network Engineering Tools* (*Opnet*) *Modeler* é uma ferramenta comercial para a simulação de redes. Dispõe de ambiente gráfico com muitas funcionalidades, entre as quais, um ambiente para o desenvolvimento de modelos para todos os tipos de redes e tecnologias. Afirma-se como o motor de execução de simulações discretas mais rápido entre outros simuladores semelhantes [156].

O simulador QualNet [184], sucessor do simulador *GloMoSim* [289], é uma ferramenta comercial capaz de prever com um grande nível de precisão o desempenho de redes com fios, redes sem fios e redes híbridas. As suas principais vantagens são [149]: modelos para redes sem fios bons e atualizados, implementação simples de novos protocolos, comparação de protocolos de diferentes camadas, bom ambiente gráfico para modelação rápida, e, escalável. Inclui já modelos para muitos objetos de redes e tecnologias.

O SSFNet [286] é uma coleção de componentes Java para a modelação e simulação de protocolos para a Internet e tem como granularidade o pacote IP. A modelação das camadas de ligação e física é possível em componentes separados. A arquitetura do SSFNet inclui o *Domain Modeling Language* (DML) [68] para a configuração dos modelos, o SSFNet contém os modelos de rede e o *Scalable Simulation Framework* (SSF) é o simulador de eventos discretos desenhado com o objetivo de ser escalável e suportar simulações de grandes redes.

A importância dos simuladores neste trabalho foi grande e, por essa razão, a comparação entre eles é feita de forma mais detalhada. O método escolhido para comparar os simuladores foi a construção de uma tabela com as características mais relevantes de cada um deles e, assim, obter uma visão geral de todas as ferramentas. A dimensão dos dados obrigou a dividir a sua apresentação em duas tabelas, as Tabelas 2.1 e 2.2. Estas resumem muitas das características das ferramentas já apontadas na secção anterior e, de forma a aperfeiçoar a avaliação comparativa entre as várias ferramentas, foram ainda adicionadas outras informações sobre cada simulador. A apresentação dos dados incluídos nas tabelas será feita coluna a coluna.

A coluna 'Licença' indica dois tipos de características: o tipo de liberdade de utilização das

Tabela 2.1: Características dos simuladores avaliados no âmbito deste projeto

Ferramenta	Licença	Estado	Sistemas Operativos	Arquitetura	Linguagens	Técnica	Utilização
IMUNES	*Software* Livre	fevereiro 2012	FreeBSD	*Kernel Re-entering*	comandos shell	Nós Virtuais	Ensino
J-Sim	*Software* Livre	julho 2006	Todos	Componentes Autónomos	Java & Jacl	Eventos Discretos	Ensino Investigação
JiST/SWANS	*Software* Livre, Comercial, Académico	março 2005	Todos	Orientada a Objetos	Java	Eventos Discretos	Investigação
NCTUns	*Software* Livre	Em uso	Linux/Unix	*Kernel Re-entering*	C	Eventos Discretos	Ensino Investigação
Network Simulator 2	*Software* Livre	Em uso	Todos	Orientada a Objetos	OTcl & C++	Eventos Discretos	Ensino Investigação
Network Simulator 3	*Software* Livre	Em uso	Todos	Orientada a Objetos	C++ & Python	Eventos Discretos	Ensino Investigação
OMNeT++	*Software* Livre, Comercial, Académico	Em uso	Todos	Componentes	C++ & NED	Eventos Discretos	Ensino Investigação
OPNET Modeler	Comercial, Académico	Em uso	Todos	Orientada a Objetos	C & C++	Eventos Discretos, Híbrido, Analítico	Ensino Investigação
QualNet	Comercial	Em uso	Todos	Entidades	C & Parsec	Eventos Discretos	Investigação Militar
SSFNet	*Software* Livre, Comercial, Académico	janeiro 2004	Todos	Entidades e Associações	Java & DML	Eventos Discretos	Ensino Investigação

bibliotecas e o seu custo. A informação 'Soft. Livre' (*software* livre) indica que o código fonte está disponível para os utilizadores. Os simuladores sem mais nenhuma informação além desta são simuladores sem custos de aquisição. Os simuladores cuja licença tenham a indicação 'Com.' (comerciais) são aplicações cuja aquisição envolve custos. A informação 'Acad' indica os *softwares* com licenças académicas, ou seja, sem custos, ou custos reduzidos, quando utilizado para fins de ensino/investigação não comercial.

A coluna de 'Estado' indica a data do lançamento da última versão e/ou atualização. Esta informação indica a maturidade e interesse na aplicação. A coluna de 'Sistemas Operativos' indica as plataformas onde a aplicação pode ser executada. Verifica-se que a maioria pode ser executada nas plataformas mais conhecidas.

As colunas 'Arquitetura' e 'Linguagem' estão relacionadas e indicam o método e as ferramentas utilizadas no desenvolvimento dos simuladores e das simulações. Aqui existem três grandes grupos:

- as entidades/componentes, apesar da terminologia distinta, indicam uma forma similar de construir descrição de simulação e seus componentes;
- o grupos dos simuladores orientados-ao-objeto, estão associados a linguagens orientadas ao objeto, *Java* e C++, e;
- os simuladores *kernel re-entering*, característica cujo objetivo é a execução de simulações com uma maior proximidade à realidade.

A coluna 'Técnica' mostra uma similitude entre todos os simuladores. O processo de seleção de ferramentas partiu das ferramentas presentes em outros trabalhos de uma forma mais esparsa. A característica de 'Eventos Discretos' presente em quase todos os simuladores revela uma área da simulação comum aos simuladores de rede e revela a opção da maioria dos trabalhos em manter as comparações dentro deste limite, para simplificar e manter com alguma objetividade o trabalho de comparação.

Tabela 2.2: Características dos simuladores avaliados no âmbito deste projeto (continuação)

Ferramenta	Emulação	Gerador Tráfego	Gerador Topologia	Ambiente Gráfico	Extensabilidade	Redes
IMUNES	Sim	Aplicações reais	Manual	Para a modelação	Desconhecida	Redes com fios
J-Sim	Sim	Aplicações reais e modelos INET	Manual	Para a modelação e depuração	Simples adicionar novos componentes	Redes com/sem fios, sensores, QoS
JiST/SWANS	Não	Modelos mais comuns, mas poucos	Manual	Não tem	Simples adicionar novos componentes	Redes com/sem fios, sensores, QoS
NCTUns	Sim	Aplicações reais e modelos INET	Manual	Para a modelação, análise e depuração	É possível adicionar novos elementos	Redes com/sem fios, sensores, QoS
Network Simulator 2	Sim, limitado	Muitos modelos, tráfego real	Ferramentas externas	NAM (limitado), XGraph, Outros...	Grande	Redes com/sem fios, satélite, sensores, QoS
Network Simulator 3	Sim	Modelos INET	Simples e ferramentas externas	PyViz, NetAnim e Ns3Generator (limitado)	Grande	Redes com/sem fios, sensores
OMNeT++	Sim	Modelos INET	Simples e Ferramentas externas	Eclipse	Grande	Redes com/sem fios, sensores, QoS
OPNET Modeler	Sim	Muitos modelos, tráfego real	Diversas bibliotecas e ferramentas	GUI integrado	Grande	Redes com/sem fios, satélite, sensores, QoS
QualNet	Sim	Muitos modelos, tráfego real	Diversas bibliotecas e ferramentas	GUI integrado	Grande	Redes com/sem fios, satélite, sensores, QoS
SSFNet	Não	Modelos INET, poucos	Manual	GUI simples	É possível adicionar novos elementos	Redes com fios

A coluna 'Utilização' apresenta, para a maioria dos casos, o uso do simulador no âmbito de projetos de ensino e de investigação. Existe ainda uma utilização no âmbito militar, principalmente associado a uma empresa.

A Tabela 2.2 apresenta um conjunto diferente de características, mais próximo das funções e aplicações incluídas em cada simulador.

A primeira coluna de características indica se o simulador também pode ser executado como emulador. Muitos deles contêm esta funcionalidade e, neste caso, a execução da ferramenta também pode interagir com uma rede real, recebendo e enviando pacotes, enriquecendo a simulação com dados reais.

A coluna 'Gerador Tráfego' indica, resumidamente, as aplicações ou geradores de tráfego. As aplicações mais comuns estão presentes em todos os simuladores, variando apenas a quantidade entre eles. Esta é uma característica importante e necessitaria de uma avaliação mais detalhada, mas o propósito alargado desta comparação levou a uma apresentação resumida das capacidades de cada simulador, esperando-se que as diferenças apontadas sejam suficientes para clarificar as potencialidades de cada simulador neste caso.

A coluna 'Gerador Topologia' indica se o simulador contém bibliotecas para a geração de diferentes topologias. A maioria apenas permite a construção de topologias manualmente, mas algumas contêm bibliotecas próprias para a geração de topologia, importante principalmente na construção de topologias com muitos nós. Alguns simuladores têm ferramentas externas para a geração de topologia, mas estas ferramentas adicionam mais uma fase de aprendizagem no processo de simulação.

A coluna 'Ambiente Gráfico' indica para cada simulador a existência, ou não, de ferramentas com ambientes gráfico para a modelação, execução e análise/visualização das simulações. Apesar de existirem simuladores com bons ambientes gráficos, alguns não têm nenhum ou têm ambientes gráficos com muitas limitações.

Tabela 2.3: Número de publicações de cada simulador desde janeiro de 2010 a agosto de 2012

Simulador	Termos Pesquisados	Totais
IMUNES	imunes	7
J-Sim	j-sim	465
JiST/Swans	jist, jist & swans	272
NCTuns	nctuns	96
Network Simulator 2	ns2, ns-2	6327
Network Simulator 3	ns3, ns-3	1086
OMNeT++	omnet	861
Opnet Modeler	opnet	1247
QualNet	qualnet	538
SSFNet	ssfnet	60

A coluna 'Extensibilidade' procurou conhecer para cada simulador as facilidades e a clareza para a adição de novos elementos e tecnologias de rede. Os dados apresentados foram obtidos através da consulta da documentação dos autores das ferramentas.

Por fim, a coluna 'Redes' identifica as redes suportadas por cada simulador. Esta informação foi obtida pelas informações dadas por cada equipa de desenvolvimento e pelos autores de trabalhos que utilizaram o simulador.

Além da análise às características, uma abordagem também escolhida para selecionar os simuladores foi a quantificação da sua utilização por parte da comunidade de investigação. Uma pesquisa feita aos repositórios ACM *Digital Library* (ACM) [3], IEEE *Xplorer* (IEEE) [114] e *SpringerLink* [250] permitiu obter os dados apresentados na Tabela 2.3.

Os valores apresentados na tabela indicam, de forma destacada, o NS2 como o simulador mais referenciado, e, muito provavelmente, o mais utilizado. A análise realizada em [172] foi feita apenas para redes sem fios, mas vai no mesmo sentido e coloca os simuladores da Tabela 2.3 com mais publicações como os simuladores mais utilizados. Excetua-se o NS3, que não foi incluído nesse trabalho. Em termos de possibilidades, os quatro simuladores mais referidos são semelhantes, ou seja, permitem a simulação de muitos tipos de redes e são flexíveis e extensíveis para a adição de novos componentes. O *Opnet Modeler* diferencia-se dos restantes três por ser uma ferramenta comercial. O NS3, apesar do nome, não é uma evolução do NS2, logo não permite a simulação de cenários de rede do NS2. O simulador OMNet++ foi ainda considerado por ser uma ferramenta que inclui uma boa base para a integração de ambientes gráficos e por também suportar muitos tipos de redes.

Os projetos de *software* livre são, normalmente, mais usados em ambiente de investigação. Além do baixo custo, a flexibilidade da sua licença permite que possa ser usado livremente tanto em ambientes académicos como em projetos com as empresas. Apesar de possível a utilização de alguns simuladores comerciais nas universidades por um custo baixo, existem algumas limitações no seu uso que condicionam o que pode ser simulado. Os simuladores NS2, *Opnet Modeler*, NS3 e OMNet++ são dos mais utilizados na investigação na área das redes de telecomunicações e, de entre estes, o *Opnet Modeler* é o único software proprietário/comercial.

Os softwares *Opnet Modeler* e *Qualnet* são aqueles que têm o melhor conjunto de ferramenta e suporte ao desenvolvimento de simulações. As ferramentas para o desenvolvimento de novos modelos, as bibliotecas de modelos, e as ferramentas de análise e visualização estão bem integradas

e suportam o utilizador em todo o processo de criação/análise de um cenário de rede. O OMNet++ é o único exemplo próximo destas ferramentas no grupo das ferramentas *software* livre, tendo um ambiente gráfico para muitas das suas funcionalidades. As restantes ferramentas são, nestes aspetos, inferiores e, dependendo da ferramenta, podem conter uma ou várias ferramentas auxiliares, mas pouco integradas entre elas.

Os *softwares* livres, de onde se destaca o NS2, têm como grande vantagem permitirem aceder a todo o seu código. Um utilizador além de poder adicionar novos modelos, pode ainda modificar todo o núcleo do simulador da forma a ajustar-se ao seu projeto. Principalmente para investigação de novos protocolos e aplicações de rede, e devido a esta característica, os *softwares* livres têm sido preferidos, apesar do esforço necessário para os compreender e utilizar.

Alguns dos simuladores destacam-se no suporte a problemas específicos da simulação de redes, como a simulação de redes de grande dimensão e a fiabilidade dos modelos. Os simuladores *Jist/SWANS* e SSFNet suportam as simulações de grandes redes, mantendo baixos os requisitos de processamento e memória. Esta é uma limitação muito apontada a simuladores mais usados. Os simuladores IMUNES e NCTUNS, que utilizam a técnica *kernel-reentering*, procuram obter simulações cujos resultados sejam mais próximos da realidade, não usando modelos, mas as próprias bibliotecas do sistema operativo para a execução da simulação.

Esta secção procurou apresentar uma visão abrangente das ferramentas de rede mais utilizadas ou representativas de uma determinada tarefa de gestão. Muitas ferramentas não foram consideradas, já que o objetivo deste trabalho não é indicar todas as categorias possíveis. As ferramentas escolhidas visam apresentar as principais funcionalidades utilizadas pelos administradores na gestão de uma rede. Uma dificuldade inicial foi o enquadramento das ferramentas, por vezes difícil pelo número de funções que cada uma continha. Não será difícil encontrar listas que incluem como ferramenta de monitorização uma ferramenta apresentada na modelação, mas tal deve-se a poderem ser utilizadas de forma muito flexível. A separação acabou por ser feita na funcionalidade principal pela qual são conhecidas. Um exemplo desta mistura é o *Nmap* apresentada como ferramenta de modelação, mas muito usada também para monitorização.

Apesar de a gestão de rede, em rigor, não envolver os equipamentos terminais, a maioria das ferramentas apresentadas inclui a gestão dos serviços e outros aspetos do sistema de informação a operar sobre a rede. Assim, e mesmo sendo uma responsabilidade diferente, a gestão dos sistemas informáticos compete, total ou parcialmente, também ao administrador da rede e por isso muitas das ferramentas foram desenhadas para realizar a gestão de ambos os sistemas. As comparações, mais ou menos detalhadas, realizadas nesta secção propuseram-se a mostrar as principais diferenças entre as ferramentas. Por vezes foi nas funcionalidades oferecidas, outras vezes nas razões porque são escolhidas para lá das funcionalidades base. Desde facilidade de utilização, apoio profissional ou mesmo alguma funcionalidade em particular, um administrador em processo de escolha de uma ferramenta, ou de várias, não tem uma tarefa simples. Este deverá procurar um equilíbrio entre o custo imediato ou custo indireto (por necessitar de mais tempo para obter resultados) e o seu conhecimento ou necessidade de apoio externo.

Apesar dos diferentes objetivos e características de todas as ferramentas apresentadas, alguns detalhes merecem algum destaque. O protocolo SNMP é utilizado por muitas das ferramentas e o seu formato de dados é o único formato referido em praticamente todas as ferramentas. A exceção

está apenas no gerador de topologias BRITE que, além seu próprio formato, também importava e exportava dados de outras ferramentas.

A visão da gestão de redes como um ciclo onde várias ferramentas vão sendo utilizadas levaria a pensar que uma linguagem deveria sobressair para transportar a informação da rede de uma ferramenta para outra, mas isso é uma realidade que não existe. Algumas ferramentas são muito abrangentes, mas nenhuma cobre todo o ciclo de vida de uma rede, desde o seu desenho, à sua implementação e operação e, após, às tarefas de manutenção e atualização. Na próxima secção são apresentadas algumas das linguagens utilizadas para descrever redes em diversos ambientes.

2.4 Descrição de Redes de Dados

A maioria das ferramentas de redes utiliza uma linguagem para descrever as redes que suporta e interage. O uso de tal linguagem pode ser transparente para os utilizadores da ferramentas se existirem ferramentas gráficas que facilitem a criação, edição e execução de comandos sobre os objetos de rede. Mas nem sempre esse é o caso e, nessa situação, os utilizadores têm de conhecer a linguagem de forma a poderem descrever toda a rede ou a poderem adicionar ou editar algum dos seus objetos. Nesta secção são apresentadas as linguagens encontradas para a descrição de redes de dados.

2.4.1 Apresentação das Linguagens

Os exemplos de linguagens para a descrição dos mais variados aspetos de uma rede são muitos e diversos. O foco principal do estudo apresentado nesta secção foi colocado nas linguagens que permitem a descrição da topologia e dos objetos da rede. Também importante foi procurar linguagens que permitissem incorporar novos objetos e informações diversas sobre a utilização da rede. Apresentam-se de seguida, as principais linguagens existentes na literatura com essas características.

A linguagem *ANother Modelling Language* (ANML) [139] suporta a descrição de redes hierárquicas, a reutilização de componentes e a sua validação. É utilizada no contexto da simulação de redes e utiliza a linguagem *Domain Modeling Language* (DML) [197] para a definição dos elementos *Schema*, *Database* e *Models*. Os *Schemas* servem para definir a estrutura e as restrições dos componentes de rede que podem ser criados. A extensão de linguagem é simples e é feita através de novos *Schemas*. As *Databases* servem como repositórios de componentes. Por fim, os *Models* definem o cenário de simulação apontando para os componentes e *databases* necessárias.

A *Language for Network Meta-Description based on XML* (LNMet-X) [221] utiliza nós, ligações, agentes e tráfegos para a descrição de topologias de rede. Permite ainda a descrição de eventos que podem ocorrer durante a execução da simulação de rede. A LNMet-X integra a *Framework* NeDaSE [222] e a sua descrição está relacionada com os objetos do simulador ns-2, utilizado nessa *framework*.

A *Network Description Language* (NDL) [259] é uma linguagem para a descrição de redes óticas híbridas e visa a partilha de informação de topologia entre domínios administrativos. O NDL utiliza três classes para identificar os recursos de rede: *Location*, *Device* e *Interface*; e seis propriedades para definir as relações entre as classes, entre outras informações: *locatedAt*, *hasInterface*, *connectedTo*, *description*, *name* e *switchTo*. A notação utilizada para realizar as descrições é o *Resource*

Description Format/XML (RDF/XML) [271]. A linguagem foi alargada pelos seus autores em 2010 com novas classes e propriedades [260].

O trabalho *Network Design Markup Language* (NDML+) [159] descreve muitos dos aspetos relacionados com uma rede de dados. Não apenas descreve os seus objetos e topologias, mas também contém informação para a gestão, para o teste e para os cálculos dos custos de implementação. Focado principalmente para redes com/sem fios locais e de banda larga. Faz parte da *Framework Candy* [158] (secção 2.4.3).

A *NEtwork Description* (NED) [261] é uma linguagem para a descrição de topologias para simuladores de eventos discretos. É muito flexível, na medida que permite a construção dos mais diversos tipos de objetos de rede. A estrutura do NED é baseada em módulos arranjados hierarquicamente e que trocam mensagens entre si. O módulo de topo, designado por *System Module*, contém sub-módulos, sem limites em termos de sub-módulos aninhados. O elemento base é o *Simple Module* e é onde se encontram os algoritmos dos modelos que se pretendem representar. Um módulo pode ter parâmetros para definir o seu comportamento e para configurar a topologia. Através de *Gates* são definidos os pontos de entrada e saída de mensagens e, por fim, as ligações definem a interligação entre os vários módulos. É a linguagem utilizada pelo simulador OMNeT++.

A *Network Modeling Language* (Netml) [6] é uma linguagem para fomentar a partilha de descrições de redes entre diferentes utilizadores. Os objetos de rede incluem os nós, as ligações e os fluxos de tráfego. Os algoritmos associados à linguagem permitem a análise de disponibilidade, de perdas, de encaminhamento, de fluxos de tráfego e dimensionamento de ligações. Associado à linguagem existe uma ferramenta *on-line* para a análise e desenho de redes entre diferentes utilizadores [5].

A *RElational Network Description Language* (RENDL) [79], implementa a descrição de sistemas de rede através de entidades e das suas relações. Segue o formalismo de *Backus-Naur* (BNF) [141] para a definição das entidades e relações. As entidades são os mais variados componentes de rede, organizados seguindo a arquitetura de rede TCP/IP. As relações são interações entre os componentes de rede, divididas em relações horizontais - para os componentes em diferentes nós, mas no mesmo nível de rede, e.g., duas aplicações; e em relações verticais - para as ligações entre componentes de camadas adjacentes no mesmo nó. O RENDL integra a *Framework* NSDF e tem como propósito a monitorização e análise de ambientes de segurança de redes.

A linguagem proposta em [44] (XMLbNSDL) visa a descrição de cenários de simulação, suportada pela linguagem XML [268]. A estrutura da linguagem divide-se na topologia da rede, na descrição do tráfego e nos comandos para a simulação. Os objetos mais importantes são o *node*, o *link* e o *Autonomous System* (AS). Este último serve para representar redes de grande dimensão. Faz parte de uma *framework* para a simulação de redes sobre a Internet e onde se inclui o simulador NS2.

Antes de proceder a um estudo comparativo das linguagens apresentadas, é importante citar as linguagens que foram encontradas mas que não foram incluídas na comparação. São também indicadas as razões da sua não inclusão.

No domínio da gestão das redes surgem algumas normas para a descrição de objetos de redes. O mais conhecido e utilizado é o *Simple Network Management Protocol* (SNMP) [105] é um modelo de informação para a gestão de redes normalizado pelo IETF, onde os dados da rede estão distribuídos pelos vários objetos. Apesar de terem sido previstas muitas operações para o SNMP,

apenas a monitorização tem sido largamente usada, não sendo utilizada a componente de configuração devido a implementações distintas dos fabricantes de equipamentos, levando a uma falta de interoperabilidade.

A linguagem YANG [27] permite a modelação de dados para as duas camadas superiores do protocolo *Network Configuration* (NETConf) [78]. Este último providencia os mecanismos para a instalar, manipular e retirar configurações de dispositivos de rede. A YANG permite a descrição de dados para a configuração e guarda de informações de estado dos dispositivos de rede.

Existe ainda o *Common Information Model* (CIM) [69], da organização *Distributed Management Task Force* (DMTF). É um modelo de informação para a descrição de informação de gestão de aplicações, equipamentos e redes. Não está associada a nenhuma implementação particular, procurando ser independente de qualquer fabricante. As suas descrições permitem a interoperabilidade da informação de gestão entre objetos de rede.

Apesar de todos estes projetos se terem tornado normas, e serem largamente usadas pela indústria, as suas características apontam para uma utilização apenas no domínio da gestão operacional de uma rede, não sendo simples a adaptação das suas descrições para outros domínios de rede, como, por exemplo, a simulação.

Ainda no âmbito de linguagens de descrição no domínio das redes, existem muitos outros exemplos. São indicadas de seguida algumas dessas linguagens tendo cada uma um propósito particular sobre as redes: a *Network Resource Description Language* (NRDL) [42] permite indicar os serviços disponibilizados por cada recurso de rede e suporta a descrição de redes orientadas ao serviço (de *service oriented networking*); o *Simulation Experiment Description Markup Language* (SED-ML) [137] permite descrever experimentos de simulação e procura simplificar a sua portabilidade entre diferentes ambientes de simulação; o *NETwork Protocol Description Language* (NetPDL) [232] visa a descrição dos protocolos da camada 2 à camada 7 OSI; e, o projeto *Language for Embedded Networked Sensing* (LENS) [133] define um ontologia semântica para a descrição de recursos em redes de sensores sem fios.

Poderiam ainda ser indicadas muitas outras linguagens, demonstrando o grande interesse neste método para a exploração de áreas e problemas das redes. Os exemplos visaram mostrar que existem linguagens com um foco específico num aspeto das redes. O NRDL para a descrição de serviços, o SED-ML para a descrição de experimentos de simulação, o NetPDL para a descrição de protocolos e, por fim, o LENS para a descrição de redes de sensores com fios. Por apontarem para um domínio muito específico, estas linguagens também não foram incluídas na comparação apresentada nesta secção.

Um último grupo de linguagens não apresentadas são as linguagens genéricas de programação. Por exemplo, para os simuladores NS2 e NS3, são utilizadas as linguagens OTcl e C++, respetivamente. O foco do trabalho foi em linguagens de descrição de redes, onde os objetos e as suas propriedades são definidos, em abstrações de alto nível. Este tipo de linguagem não inclui a informação sobre a implementação dos objetos para uma determinada ferramenta. Ao contrário, as linguagens genéricas de programação são linguagens que especificam todos os pormenores da sua execução. Sendo a interoperabilidade de ferramentas um princípio deste trabalho, o caminho das linguagens genéricas de programação não foi escolhido por não ser simples, ou possível, para prosseguir a interoperabilidade entre ferramentas de rede.

Tabela 2.4: Características das linguagens para a descrição de redes de dados

Linguagem	Data início	Atividade atual	*Framework*/ Ferramenta	Forma de descrição	Objetos/ elementos base	Área	Requisitos
ANML	2001	?	Simulador IP *Traffic* and *Network* (IP-TN)	DML (e pouco XML)	Componentes, *Schemas*, *Models* e *Database*	Simulação	hierárquica, reutilizável, extensível, redes grandes e complexas
LMNeT-X	2005	?	*Framework* NeDaSE	XML	Nós, ligações e tráfegos	Simulação	independente, expressiva, extensível e tradução simples
NDL	2006	ativo	Redes Óticas Híbridas	RDF/XML	*Location*, *device* e *interface* (2006) + novas classes (2010)	Análise	concisa, interoperável, distribuída, portável, extensível, compreensível por humano
NDML+	2006	ativo	*Framework Candy*	XML	*device*, *nic*, *medium* e *agent*	Planeamento & Análise	extensível, flexível
NED	1998	ativo	Simulador OMNeT++	NED (ou própria)	Módulos (vários tipos)	Simulação	flexível, simples, clareza
Netml	2005	2011	-	XML	Nós, ligações e tráfegos	Planeamento & Análise	independente, partilha
RENDL	2003	?	*Framework* NSDF	BNF	Entidades, relações, estados e ações	Segurança	granular, flexível
XMLbNSDL	2003	?	*Framework* simulação	XML	Nós, ligações e Sistemas Autónomos	Simulação	independente, tradução simples, flexível, extensível

2.4.2 Estudo Comparativo das linguagens

O método para realizar a avaliação às diversas linguagens é similar ao escolhido para a comparação dos simuladores de redes. Assim, na Tabela 2.4 estão inseridas as características de cada uma das linguagens apresentadas no primeiro grupo, ou seja, dedicadas ao suporte da representação de topologias e objetos de rede genéricos.

De igual forma às ferramentas de simulação, a análise das linguagens é realizada seguindo a sequência das colunas. Assim, pelas primeiras duas colunas, verifica-se que apenas as linguagens NDL, NDML+ e NED se mantém ativas. As primeiras duas são projetos recentes, comparativamente aos restantes. A NED demonstra uma longevidade que se destaca das restantes. Tal deve-se ao sucesso do simulador onde é usada, o OMNet++. A forma de obter os dados apresentados nesta coluna para os restantes projetos foi pesquisando sítios da Internet, publicações ou documentos que mostrassem a sua utilização, mas nada foi encontrado e por isso foram marcadas como inativas ou, pelo menos, podemos concluir que o seu desenvolvimento está estagnado. Algumas estão associadas a ferramentas cujo uso já é pequeno ou inexistente e, assim, a linguagem seguiu um fim similar e é pouco utilizada.

A ferramenta e/ou *framework* associada a cada linguagem é, na quase totalidade dos casos, distinta. A exceção está nas linguagens *Netml*, XMLbNSDL e NDML+ que têm em comum a utilização do simulador NS2 nas respetivas *frameworks*. Também aqui o simulador NS2 destaca-se como ferramenta presente no maior número de projetos.

O XML destaca-se como linguagem mais utilizada para suportar as descrições das redes, mas outras, como o DML, RDF/XML, BNF também são utilizadas. A linguagem NED é a única que tem uma sintaxe própria para a definição das descrições. A escolha maioritária do XML deve-se a este ter como vantagens e características [231, 232]: a existência de muitas ferramentas para editar e validar descrições em XML, a tradução simples para sistemas de base de dados, englobar linguagens para transformação e pesquisa de estruturas XML, extensível e, muito importante, a sua estrutura é compreensível por humanos. Tem como desvantagem principal a grande dimensão dos ficheiros XML, comparativamente a outras linguagens [231].

Os objetos de rede presentes em cada linguagem são bastante distintos. Os elementos mais

comuns são o nó e a ligação, sendo por vezes referidos com outros termos – *device* para os nós e *interface* e *nic* para as ligações. Algumas linguagens, como o ANML, o NED, o RENDL, utilizam formas particulares de referir objetos de rede e as relações entre eles. Os objetos de rede referem-se a equipamentos e seus componentes, e surgem referidos como módulos e entidades. As relações entre eles podem sem as ligações físicas entre nós e/ou ligações lógicas entre objetos de determinada camada.

A área de cada uma das linguagens aponta, essencialmente, para os domínios da gestão e da simulação de redes. Estas foram as áreas pesquisadas e, as linguagens apresentadas, que refletem essa restrição. Neste aspeto não há, verdadeiramente, uma separação se considerarmos que os algoritmos de todas as linguagens, ou seja, as ferramentas associadas, realizam uma análise à rede e produzem resultados a serem utilizados pelo utilizador/administrador.

Por fim, na última coluna da Tabela 2.4, surgem os requisitos mais comuns na definição de uma linguagem e estes são a simplicidade e a extensibilidade, ou seja, a facilidade em utilizar e compreender a linguagem e permitir a adição de novos elementos e componentes, respetivamente. A atribuição do termo simplicidade foi feita às características 'flexibilidade', 'expressiva', 'compreensível por humano', 'simples' e 'clareza'. Estas não têm, certamente, significados idênticos, mas o sentido da sua apresentação foi compreendido como apontando para uma linguagem fácil de utilizar por utilizadores inexperientes e cujo conteúdo fosse simples de compreender. No caso da extensibilidade foi feita a associação direta a 'extensível'.

Outra característica relevante e destacada por diferentes autores foi a independência à ferramenta ('independente'), cujo significado alude à possibilidade da descrição contida na linguagem poder ser lida e utilizada por diferentes ferramentas. Os termos 'reutilizável', 'interoperável', 'portável' e 'partilha' têm significados que apontam no mesmo sentido, reforçando este requisito. Outro aspeto identificado nesta lista de requisitos refere-se ao nível de abstração, ou seja, a linguagem ter a capacidade de representar os objetos de rede muito simples e também as redes grandes e/ou complexas ('expressiva', 'granular' e 'redes grandes e complexas').

As restantes características apenas surgem uma vez e são as seguintes: 'distribuída' indica a possibilidade de manter em diferente locais a descrição da rede; 'hierárquica', indicando a estrutura para a descrição da rede; e, 'tradução simples' que refere o nível de esforço para a implementação da conversão entre formatos.

Através deste estudo foi possível verificar que as linguagens ANML, NED e RENDL são as únicas que definem objetos abstratos básicos, permitindo a definição de qualquer objeto a partir deles. Os elementos simples que formam as estruturas são: no caso do ANML os *Models*; no caso do NED os 'módulos' e 'sub-módulos'; e, no caso do RENDL as entidades e as relações.

Contrariamente, em diferentes graus, estão a maioria das linguagens, contendo definições para os objetos das redes de algumas das redes atuais. A NDL para as redes óticas e as XMLbNSDL, *Netml* e LNMet-X para a simulação são todas elas exemplos de linguagens utilizadas em projetos específicos e a sua extensão ou utilização em outros domínios de rede não é simples e direta. A linguagem NDML+ é a única utilizada em muitos domínios das redes, como a gestão e a simulação, mas apenas para algumas tecnologias como a *Ethernet* e o Wi-Fi .

De todas as linguagens, o NDL é a única a integrar um esforço de normalização junto da organização *Open Grid Forum* (OGF) [196]. O grupo de trabalho designa-se por *Network Mark-Up Language Working Group* (NML-WG) [191] e tem como propósito normalizar uma ontologia e

uma estrutura para a descrição de redes. A linguagem NDL ainda é usada na *Infrastructure and Network Description Language* (INDL) [95] para a descrição de infraestruturas de computação. O foco mantém-se igualmente no desenvolvimento de uma estrutura distribuída de descrição de rede óticas híbridas, com informação sobre os recursos físicos e sobre as redes que os interconectam.

O estudo das linguagens apresentadas permitiu identificar as características mais relevantes para qualquer linguagem de descrição das redes. De entre essas características estão a simplicidade das descrições e a possibilidade de adicionar novos elementos. Também relevante, é obter uma linguagem independente de uma qualquer ferramenta, ou seja, que possa ser utilizada por uma qualquer ferramenta de rede.

Considerando os requisitos identificados, apesar de visarem uma utilização alargada, nenhuma das linguagens estudadas foi integrada em mais nenhum projeto além do projeto que a criou. Uma das razões da não adoção dessas linguagens de uma forma mais abrangente deve-se ao facto de elas serem orientadas ao domínio de aplicação, dificultando a sua portabilidade para problemas diferentes.

Por exemplo, algumas linguagens descrevem a informação de domínio, como a simulação, e a informação dos objetos de rede de forma integrada. Considerando que seja necessário exportar essa descrição para outro domínio (ou ferramenta), toda a informação de domínio (simulação) e de objetos deverão também ser transportadas por fazer parte da estrutura, o que poderá não ser requerido e levantar alguma confusão na descrição quando adicionada a informação do novo domínio.

Os trabalhos apresentados em [138, 241, 272, 287] são relacionados a linguagens de descrição propostas nos últimos anos para diferentes áreas das redes. A não inclusão destes trabalhos mais recentes na Tabela 2.4 deve-se a, essencialmente, proporem novamente linguagens orientadas a um determinado domínio de rede e/ou estão relacionadas com uma determinada ferramenta, impedindo uma utilização mais alargada da informação que contêm.

Na secção seguinte são apresentadas algumas das *frameworks* da área de redes e que utilizam algumas destas linguagens.

2.4.3 *Frameworks* para a Gestão de Redes

As *frameworks* de rede são estruturas que, além de uma abstração para a descrição de redes, já incluem um conjunto de ferramentas e processos para a análise e gestão dos mais diversos aspetos de uma rede. Algumas das ferramentas apresentadas na secção 2.3 podem ser consideradas também *frameworks*, por agregarem várias ferramentas e métodos no seu funcionamento. Como exemplo, o NS2 agrega o *Network AniMator* (NAM) [80], o *xGraph* [281]; o NS3 já inclui o *NetAnim* [228] e o *Pyviz* [48]; e, o OMNet++ , sendo um ambiente mais integrado, também integra algumas bibliotecas diversas e ferramentas com diferentes propósitos. Nesta secção são introduzidos alguns projetos cujos resultados estão relacionados com a integração de ferramentas de diferentes grupos de desenvolvimento.

Uma primeira contribuição é a *Framework Computer-Aided Network Design Utility* (CANDY) [157, 158]. Baseada na linguagem *Java*, inclui um grupo de ferramentas e a linguagem NDML+ para o desenho e gestão integrada de redes. A Figura 2.4 apresenta a arquitetura e muitos dos seus componentes.

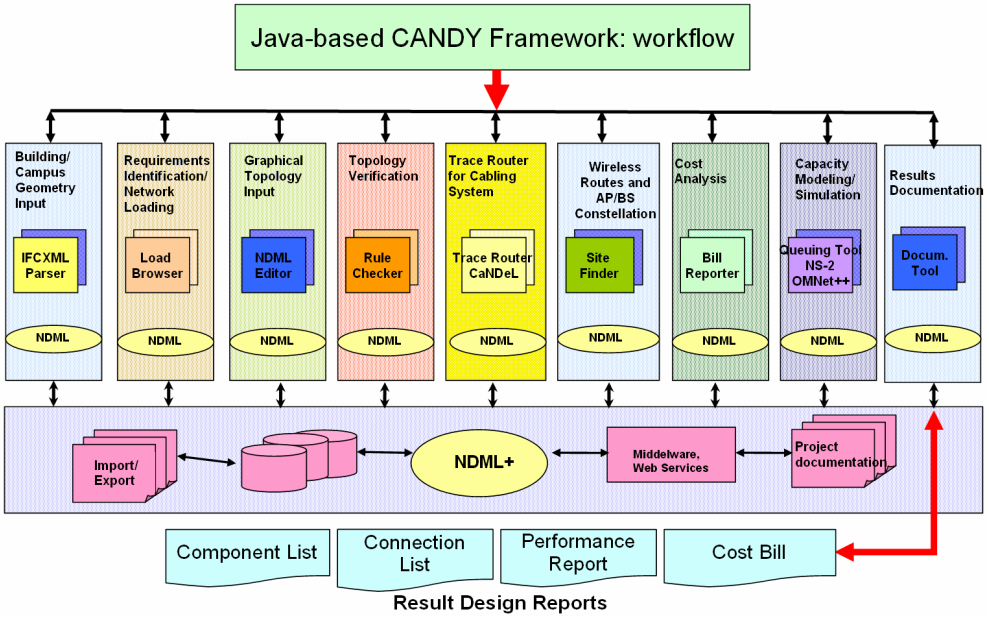


Figura 2.4: Arquitetura da *Framework* CANDY, retirado de [158]

Com a *Framework* CANDY um utilizador pode desenhar uma rede sobre uma planta de um edifício e realizar uma análise ao seu futuro funcionamento. Essa *framework* ainda apoia o administrador a escolher as tecnologias a implementar na rede, a desenhar o sistema de cablagem e constelação de Pontos de Acesso Sem Fios, e a encontrar os equipamentos ativos (de encaminhamento) mais adequados. Permite ainda indicar dados comerciais e financeiros e avaliar o custo aproximado da rede.

Além de algumas bibliotecas próprias para a análise da topologia e dos custos, o simulador NS2 ainda é utilizado para avaliação da capacidade e desempenho da rede. Uma última funcionalidade integrada na *framework* é o apoio à criação da documentação do projeto, permitindo a adição das mais variadas informações sobre a rede. As tecnologias de rede suportadas são as redes IEEE 802.3 Ethernet, IEEE 802.11 WLAN e IEEE 802.16 WiMAX.

Um projeto num domínio diferente é a *Framework Network Design and Simulation Environment* (NeDaSE) [222]. Esta *framework* integra um grupo de ferramentas para o suporte a um ambiente de simulação de redes. A Figura 2.5 apresenta as ferramentas/componentes principais da *Framework* NeDaSE.

Além das diferentes ferramentas de rede, a *framework* contém no seu centro a linguagem LMNeT-X. Na sua versão dois, as ferramentas são as seguintes: *DEsign Tool Lite* (Delite) [35] que permite, graficamente, desenhar e visualizar redes simples, o *Boston University Representative Internet Topology gEnerator* (BRITE) [171] para a geração de topologias de rede, e os simuladores NS2 e *Glomosim* já apresentados na secção 2.3. A linguagem LMNeT-X serve como ponto intermédio para converter as descrições de uma ferramenta no formato de outra ferramenta.

Ainda, as linguagens RENDL e XMLbNSDL estão integradas em diferentes *frameworks*, mas com propósitos mais específicos e em ambiente mais simples e com menos ferramentas. A linguagem RENDL faz parte da *Framework* NSDF [79] e visa a descrição completa de redes no contexto da avaliação da segurança de uma rede. Após a completa descrição de uma rede, pode ser feita uma

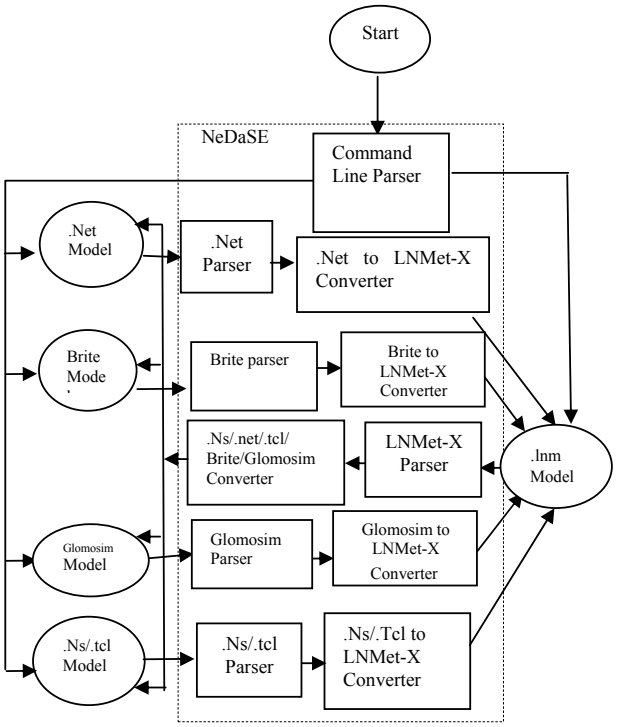


Figura 2.5: Ferramentas da *Framework* NeDaSE, retirado de [222]

avaliação das suas falhas e omissões em termos de segurança. À linguagem XMLbNSDL junta-se ao simulador NS2 para a simplificação na construção de simulações de redes.

O elemento comum em todas as *frameworks* é a presença de uma linguagem central para a guarda e transporte das descrições da rede entre ferramentas. Em todas identificou-se o requisito de extensibilidade no sentido de acomodar novas ferramentas e possibilidade de descrição de novas redes. No entanto, praticamente todas as linguagens estão muito ligadas a uma ferramenta em particular. As linguagens ANML, LMNeT-X, *Netml* e XMLbNSDL, apesar de se declararem linguagens interoperáveis, refletem, em muito, os objetos e as características do NS2. A NDL e a NED foram definidas para aplicações específicas e apenas procuram descrever o contexto dos domínios dessas aplicações. Por fim, apenas o NDML+, da *Framework* CANDY, separa as diversas descrições e poderá permitir acomodar informações para diferentes domínios, mas não é muito flexível para acomodar novos objetos de redes. A consulta ao seu modelo Entidade-Relação mostra que as definições de topologia são limitadas e apenas suportam algumas tecnologias de rede e não é simples a sua extensão.

Baseado nas linguagens estudadas, é possível constatar que na proposta de uma nova linguagem não é simples determinar qual o ponto de equilíbrio entre uma linguagem muito genérica e uma linguagem específica para uma determinada área. Os objetivos funcionais da linguagem determinarão sempre as diretivas a seguir, no entanto, para obter uma linguagem que seja utilizada por muitas ferramentas, será necessário encontrar uma estrutura genérica, sem deixar de ser expressiva.

No entanto, mesmo que algumas linguagens tenham muitas semelhanças, a opção dos seus autores foi definir uma nova estrutura, em vez de reutilizar uma já existente. A provável razão para isso, deve ser dada a especificidade de cada linguagem e a sua aplicação. As contribuições

existentes na literatura apresentam, apenas em poucos casos, comparações com outras linguagens e a fundamentação para a sua criação não é explícita.

As secções anteriores tiveram como objetivo apresentar as tecnologias que servem de contexto, ou que integraram, o trabalho descrito no resto deste documento. Para terminar, são apresentadas as conclusões sobre o estudo realizado e descrito neste capítulo.

2.5 Conclusões

O número de ferramentas de rede existentes é bastante grande e durante o levantamento bibliográfico foi necessário selecionar um grupo que incluísse as mais utilizadas para realizar a análise e gestão das redes. A pesquisa das linguagens para descrever as redes e para descreverem alguns tipos de análise sobre essas redes foi importante para entender os esforços já feitos para suportar a interoperabilidade de informações das redes entre diferentes ferramentas e domínios de utilização.

A comparação das ferramentas não é uma tarefa simples, devido à sua diversidade e às suas características. Sem uma clara definição de requisitos iniciais todo o processo será sempre incerto e limitado. No domínio dos simuladores os trabalhos apresentados em [57, 276] comparam diversas ferramentas e em ambos a primeira conclusão é o reconhecimento da mesma dificuldade. Outra conclusão presente, também importante e complementar, é que a adequabilidade de cada ferramenta está muito relacionada com o ambiente de aplicação e, consequentemente, é difícil atribuir a uma a condição de melhor ferramenta de uma forma global. A comparação das ferramentas dos outros grupos revelou-se igualmente complexa e com conclusões semelhantes.

A dificuldade em utilizar algumas ferramentas é também um aspeto que sobressai. As ferramentas apresentadas obrigam a utilizadores menos experimentados longas curvas de aprendizagem, seja para conhecer todas as possibilidades de configuração gráfica, seja a prática com uma linguagem de programação. Ambas as dificuldades foram apontadas em especial no ambiente dos simuladores, visto que a presença de ambiente gráficos que suportem o processo de descrição de uma rede e execução de uma simulação não é uma realidade em muitos deles. Esta barreira é assim ainda um entrave a uma maior utilização deste método para a análise e avaliação das redes de dados.

Sobre as linguagens para a descrição de ambientes de rede, um primeiro fato foi o grande número de exemplos encontrados. Os trabalhos apresentados na Tabela 2.4, e muitos outros que foram encontrados mas que se optou por não incluir por razões já expostas, integraram projetos de redes onde a regra foi cada um criar a sua linguagem de descrição (proprietária). À parte do domínio de gestão, não foram encontrados exemplos de reutilização total ou parcial de uma linguagem. Os autores não detalham muito sobre esse aspeto, mas, na opinião do autor desta tese, tal deve-se à variedade de objetivos e de pontos de análise de cada projeto, tendo a linguagem, normalmente, um papel menos relevante. Assim, as linguagens propostas tendem a ser simples e adequadas para um determinado projeto, mas difíceis de reutilizar ou ajustar a um projeto muito distinto. Devido à similaridade entre muitas ferramentas de rede, a compatibilidade entre as descrições seria muito útil para ajudar os utilizadores a aprofundar o entendimento sobre o funcionamento da sua rede.

Foram identificadas muitas características comuns a muitas linguagens. A construção da topologia é uma tarefa que necessita de objetos muito precisos e, neste caso, não seria complicado usar uma qualquer linguagem para esse fim. A grande diferença está na variedade de objetos, além do nó e da ligação, que algumas linguagens têm, tornando a sua descrição mais rica, mas também

menos interoperável. A forma como os diferentes objetos de rede se relacionam, também distinta entre linguagens, dificulta ainda mais a reutilização de descrições.

As características que mais se destacaram para uma qualquer linguagem de descrição das redes foram: a simplicidade, a extensibilidade e a independência. No entanto, as linguagens apresentadas não apresentam verdadeiramente todas essas características. Sobre a simplicidade não há comentários, mas a extensibilidade apenas surge em dois projetos – NDL e NDML+ – e apenas dentro da equipa original de desenvolvimento. A independência poderá ser aferida pelo número de projetos distintos a utilizar uma mesma linguagem, e verificou-se que todas as linguagens apenas são utilizadas em um projeto. Este fato mostra, de verdade, a pouca independência das linguagens, apesar de muitas terem esse requisito como princípio de desenho.

Os capítulos 3 e 4 propõem uma nova *framework* de ferramentas de redes e uma nova linguagem, respetivamente. A *framework* pretende agregar ferramentas distintas que se complementem no objetivo de fornecer um conjunto de funcionalidades completas para a análise das redes de dados em diferentes domínios. Será necessário encontrar ferramentas com um determinado conjunto de funcionalidades e desenvolver novas ferramentas que respondam às limitações encontradas. A linguagem servirá como elemento central de suporte à interoperabilidade entre as várias ferramentas, procurando implementar os princípios de simplicidade, extensibilidade e independência. O conhecimento adquirido na revisão das linguagens existentes deverá ser utilizado de forma a ser proposta uma linguagem cuja adoção seja possível em diferentes domínios e projetos.

Capítulo 3

Framework NSDL

3.1 Introdução

O desenvolvimento das aplicações de rede segue, na maioria dos casos, os métodos usados para a criação de aplicações simples e isoladas. Apesar de estes serem adequados para pequenos projetos, normalmente com um tempo de utilização curto e limitado [90], as aplicações desenvolvidas têm como principais desvantagens a sua difícil manutenção e fraca extensibilidade [91].

Nesse sentido, a existência de um método que facilite a criação e extensão de aplicações de rede poderá facilitar no futuro a utilização integrada de várias ferramentas num projeto de rede. Além das vantagens para o desenvolvimento, permitirá ainda uma gestão mais otimizada e eficiente dessa rede por via da extensão do número de ferramentas que poderão ser usadas. Assim, essa estrutura deve proporcionar a interoperabilidade entre as diferentes ferramentas de gestão existentes permitindo a troca de dados entre elas e facilitando o desenvolvimento e a utilização de aplicações e ferramentas de rede.

Este capítulo descreve a proposta de uma nova estrutura com essas características, a *Framework* NSDL, cujo principal objetivo é proporcionar uma infra-estrutura de integração para facilitar e otimizar a gestão e análise de redes através de um conjunto de mecanismos que suportem a interoperabilidade entre diferentes ferramentas de rede, existentes ou a desenvolver. O resto do capítulo está organizado da seguinte forma: na secção 3.2, os conceitos principais, as características, as funções, as vantagens e as desvantagens de *frameworks* de uma forma genérica; na secção 3.3 são discutidos os objetivos, características e vantagens da *Framework* NSDL, bem com cada uma das suas camadas; na secção 3.4 é apresentada a arquitetura de componentes da *Framework* NSDL. Na secção 3.5 serão apresentadas algumas conclusões deste capítulo.

3.2 *Frameworks*

O conceito de *framework*[1], partindo de uma definição genérica presente no dicionário, surge como uma 'estrutura de suporte ou de enquadramento' ou, de outra forma, 'um esqueleto que serve como base para a construção de algo' [83]. No contexto do desenvolvimento de aplicações, é 'um conjunto de aplicativos (como classes, objetos e componentes) que colaboram para suportar uma

[1]Os termos em português encontrados como tradução para *framework* foram 'moldura' e 'arcabouço'. Nenhum foi considerado adequado, e nem o seu uso é corrente, por isso e assim, optou-se por manter o termo original em inglês no título e ao longo deste trabalho.

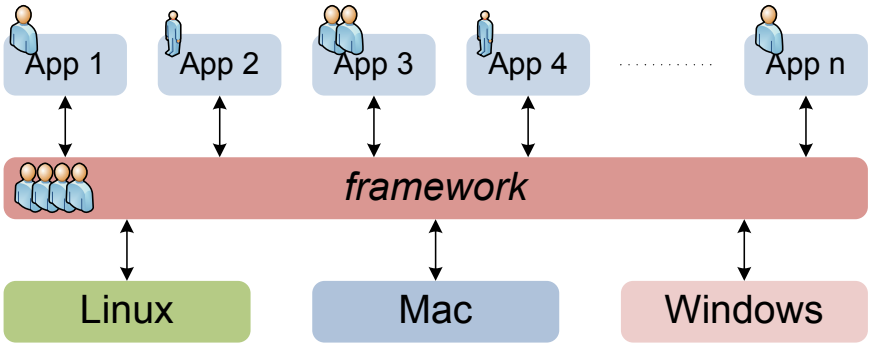


Figura 3.1: Exemplo de uma *framework* multiplataforma

arquitetura reutilizável para uma família de aplicações relacionadas' [132], ou, ainda de uma outra forma, também se pode designar como uma aplicação genérica que suporta a criação de outras aplicações dentro uma mesma família.

O conceito de famílias de aplicações dentro de uma *framework* implica a existência de um conjunto de características comuns entre todas as aplicações. A existência de outras características, próprias de uma dada aplicação ou de grupo de aplicações, também está contemplado [243]. Portanto, o programador de uma aplicação pode avançar na implementação dos requisitos específicos da sua aplicação, confiando no desenvolvimento realizado pelos restantes programadores da *framework* para obter uma aplicação completa.

Um exemplo de um caso vantajoso da utilização deste método é o desenvolvimento de aplicações multiplataforma, ou seja, podem ser executadas em diferentes sistemas operativos, designadas por aplicações portáteis ou portáveis. Neste caso, um grupo desenvolve uma aplicação sobre uma *framework* enquanto outro ou outros grupos têm a responsabilidade de implementar a *framework* em diferentes sistemas operativos. Assim, a aplicação pode ser executada em todos os diferentes sistemas operativos onde existe uma implementação da *framework*, de forma similar. A Figura 3.1 ilustra a estrutura descrita.

Os criadores das aplicações (*App 1.. App n*) não necessitariam conhecer os detalhes dos sistemas operativos, apenas necessitariam conhecer os serviços fornecidos pela *framework*. A equipa responsável pela *framework* define os serviços para as aplicações e implementam as bibliotecas necessárias em cada um dos sistemas operativos. Cada novo serviço adicionado à *framework*, permite a criação de novas funções nas aplicações. A complexidade do trabalho da equipa da *framework* fica em grande parte escondida dos programadores de aplicações. Como exemplos de tecnologias enquadradas neste exemplo são as linguagens Java e Corba , ambas padrões abertos (*open standards*) que suportam, respetivamente, a portabilidade e a comunicação de e entre aplicações.

3.2.1 Vantagens e Desvantagens das *frameworks*

Assim, uma primeira vantagem no uso de uma *framework* para a criação de aplicações é permitir o desenvolvimento com foco nas características específicas dessa aplicação, evitando o tempo necessário para a construção da infraestrutura, também necessária, para o funcionamento dessa mesma aplicação [16].

No entanto, o programador terá de inteirar-se das características da *framework* antes de iniciar o

desenvolvimento da sua aplicação. Um dos requisitos principais para uma *framework* de qualidade é a existência de uma documentação completa e atualizada, isto para conduzir o programador a uma rápida compreensão da sua arquitetura e dos passos necessários para iniciar o desenvolvimento da aplicação da forma mais correta e eficaz [243].

Além da vantagem já referida (ganho de tempo no desenvolvimento de novas aplicações) também outras podem ser referidas [132, 242, 243]:

- Uma maior fiabilidade à nova aplicação devido à reutilização do código da *framework* que já foi desenvolvido e testado, e a redução do esforço no teste e do tempo de desenvolvimento;
- Melhoria nas técnicas de programação, visto ser necessário seguir as regras definidas pela *framework*, normalmente baseada em boas práticas de programação, como a utilização de padrões de desenho [90] e de novas ferramentas de programação, e;
- A atualização da *framework* pode adicionar novas funcionalidades, melhorias de desempenho e de qualidade à aplicação sem requerer a intervenção do utilizador da *framework*.

Apesar das vantagens enunciadas, a utilização das *frameworks* também é condicionada por algumas limitações e poderá, para o desenvolvimento de aplicações, representar algumas desvantagens, como por exemplo [132, 242, 243]:

- A criação de uma nova *framework*, em comparação com o desenvolvimento de uma aplicação isolada, é mais complexa, demorada e, consequentemente, mais onerosa;
- A utilização de uma *framework* implica a sua aprendizagem e esta é, normalmente, longa. A sua utilização deve ser, por isso, avaliada em termos de tempo e custo, comparativamente a criar uma aplicação isolada para o projeto, e;
- A complexidade da *framework* tende a aumentar com o tempo, tornando-se mais exigente a sua utilização, principalmente para novos utilizadores.

Para concluir, uma *framework* visa proporcionar uma estrutura lógica para agregar um conjunto de componentes de *software* fortemente relacionados, e, nesse contexto, desenvolveu-se o tópico de *frameworks* de *software* como estruturas que permitem a criação de aplicações sobre uma base comum. No entanto, este conceito difere da proposta de uma plataforma, que designa a implementação de um componente de *software* para o suporte à criação e execução de aplicações, envolvendo uma integração mais importante entre as aplicações desenvolvidas em diferentes níveis (e.g.: base de dados, *interface*, *proxy* ou todos os componentes)[235].

Os trabalhos apresentados na secção 2.3 são, na sua maioria, aplicações específicas com o propósito de executar um conjunto de tarefas relacionadas com um domínio da área de redes. Por exemplo, os simuladores *Network Simulator 2* (NS2) [34] e *Network Simulator 3* (NS3) [110] servem para a simulação, avaliação e validação de redes. Ainda no domínio das redes, as ferramentas *Network Mapper* (nmap) [160] e *Multi Router Traffic Grapher* (MRTG) [195] monitorizam diferentes parâmetros da rede e apoiam o administrador a configurar adequadamente a rede. Além destas, outras ferramentas, de menor dimensão, ajudam na compreensão de aspetos mais específicos

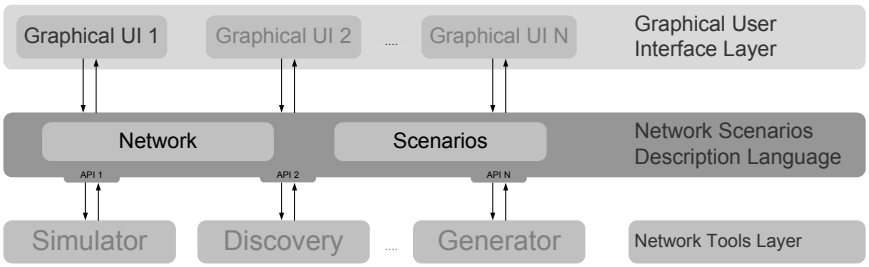


Figura 3.2: Estrutura em três camadas – de topo, central e de base - da *Framework* NSDL

de uma rede e/ou tecnologia, e.g., os trabalhos *HEterogeneous Grooming Optical Network Simulator* (Hegons) de [177] e *Plataforma de Configuração e Monitorização de QoS numa Rede Diffserv* [176]. Sobre estes últimos, o primeiro trabalho realiza simplesmente a avaliação de algoritmos de encaminhamento e atribuição de comprimentos de onda no domínio das redes óticas. O segundo trabalho foca-se na simulação de um cenário de rede com suporte à arquitetura Serviços Diferenciados, sobre o simulador NS2.

Dada a heterogeneidade de ferramentas de rede existentes, e, consequente, a grande variedade de formatos de dados utilizados por essas ferramentas, a questão base abordada nesse trabalho de tese é a proposta de uma solução que permita a sua interoperabilidade. Portanto, a proposta neste projeto é a integração de um conjunto de aplicações (algumas já existentes e outras desenvolvidas de raiz) e de processos de validação e transformação, para dar suporte aos grupos de trabalho na área das redes de dados a interligar funcionalidades e dados entre as várias ferramentas existentes e/ou a desenvolver, designando o projeto como *Framework* NSDL.

Na secção seguinte é apresentada a *Framework* NSDL, uma proposta para o suporte à interoperabilidade de aplicações no domínio das redes de dados.

3.3 *Framework* NSDL

A estrutura proposta neste trabalho designa-se por *Framework* NSDL, adiante designada por *framework* apenas, e engloba um conjunto de aplicações e de bibliotecas com o propósito de apoiar diferentes tarefas no domínio das redes de dados. Mais concretamente, a *framework* é uma base para a integração e para o desenvolvimento de diferentes ferramentas de rede. As aplicações servem para realizar as tarefas de modelação, monitorização, execução, visualização, análise das redes, entre outras. Nas bibliotecas encontram-se os componentes necessários para definir, validar e transformar os dados de rede de forma a serem utilizados pelas diferentes ferramentas da *framework*.

O objetivo principal da *Framework* NSDL é propor uma estrutura onde diferentes aplicações de rede possam ser integradas e onde existam mecanismos que suportem a interoperabilidade entre essas mesmas aplicações. A *framework* deverá não apenas acomodar as aplicações atuais mas também outras aplicações a desenvolver ou integrar futuramente.

A *Framework* NSDL foi definida em três camadas, como apresentado na Figura 3.2, e a cada uma delas foi imputado um conjunto específico de funcionalidades e, consequentemente, aplicações. Observe que a definição da *framework* em três camadas torna a estrutura simples de compreender e associa a cada uma das camadas um grande grupo de aplicações e/ou funcionalidades. A escolha

das três camadas surge do estudo realizado no capítulo 2 e após a avaliação de muitas ferramentas de rede. A diversidade de domínios e tecnologias de rede torna a categorização complexa, mas o grupo de tarefas a desenvolver em cada um destes grupos não é muito diferente. Os objetos e parâmetros envolvidos em cada grupo serão diferentes, mas a rede é a mesma e envolve em grande parte os mesmos objetos nos vários domínios. Por exemplo, numa simulação, entre muitos parâmetros, serão importantes as taxas de transmissão e os atrasos, enquanto na gestão da segurança serão importantes os serviços ativos e as regras de filtragens. Em ambos será descrita a mesma topologia e aplicações ativas e, portanto, deverá ser simples reutilizar o esforço, pelo menos parcial, de construção de uma rede e da informação sobre esta de um domínio para o outro.

Considerou-se que para o primeiro grupo de ferramentas se deveria agregar as tarefas do utilizador. As tarefas comuns são a utilização de uma ferramenta gráfica para a edição, configuração e visualização de uma rede, seus objetos e estados. Um segundo grupo engloba as tarefas feitas de forma automática por diversas ferramentas, onde, normalmente, não é necessária a intervenção do utilizador. A rede é avaliada e, sobre essa tarefa, são produzidas informações que serão usadas por outras ferramentas e também pelo utilizador. Por fim, considerou-se adequado a separação destes grupos através de uma camada intermédia onde toda a informação pudesse ser mantida e esta permitisse a interoperabilidade entre as diversas ferramentas de uma ou de ambas as camadas.

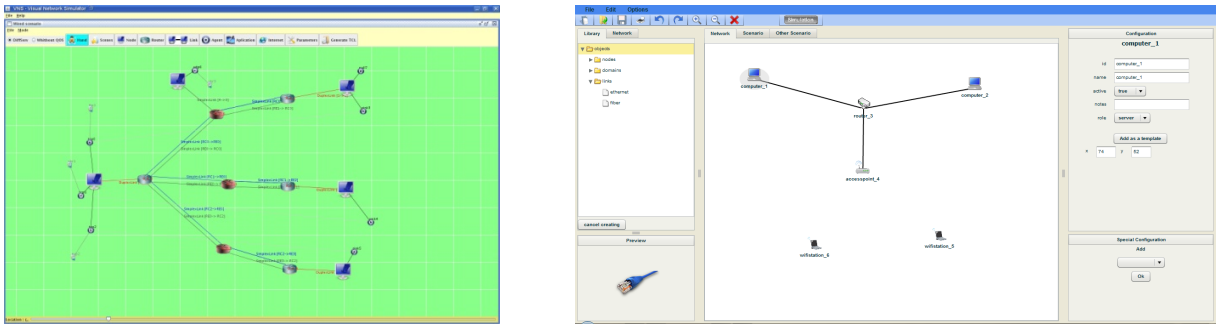
As próximas secções apresentam cada uma dessas camadas e, após, segue-se a descrição de todos os restantes elementos necessários para a implementação da *framework* e que constituem a arquitetura dos componentes.

3.3.1 Camada de Topo – *Interfaces* Gráficos

A camada de topo inclui as aplicações que permitem a interação entre o utilizador e os cenários de rede. As tarefas englobadas na interação são (1) definir todas as características da rede e, quando possível, invocar as funcionalidades disponibilizadas por outras ferramentas, normalmente, das camadas inferiores e (2) visualizar os cenários de rede e analisar os resultados da execução realizada por uma ferramenta da camada de base. Alguns exemplos de funções desta camada para a primeira forma de interação incluem: a caraterização dos objetos de rede, a edição e a construção da topologia e o agendamento de eventos. Na segunda forma de interação encontra-se a visualização das redes e dos seus dados, e.g., para as tarefas de monitorização de uma rede e a análise dos resultados da execução de uma simulação.

As aplicações apresentadas na secção 2.3 têm, em alguns casos, as suas próprias *interfaces* gráficas, como o *Opnet Modeler* ou o *Cacti*, no entanto um grande número de ferramentas apenas permite ao utilizador a definição de redes e suas características através de editores de texto simples, como os casos do JiST/SWANS e do *GloMoSim*, ou através de *interfaces* gráficos limitados, como o NS2. A Tabela 2.2 apresenta algumas das ferramentas existentes incluindo aquelas com ou sem *interface* gráficas.

As ferramentas de *Modelação* apresentadas na secção 2.3.1 serão aquelas que poderão pertencer a esta camada da *framework*. As ferramentas de *Monitorização* apresentadas na secção 2.3.2 também poderão se encaixar nesta camada da *framework*, mas apenas nas áreas das funções de descoberta e visualização de redes. As restantes funcionalidades ficaram associadas à camada de base da *framework*, a apresentar na secção 3.3.3.



(a) *Visual Network Simulator* (VNS) (b) *Visual Network Descriptor* (VND)

Figura 3.3: Ferramentas para a visualização e edição de cenários de rede

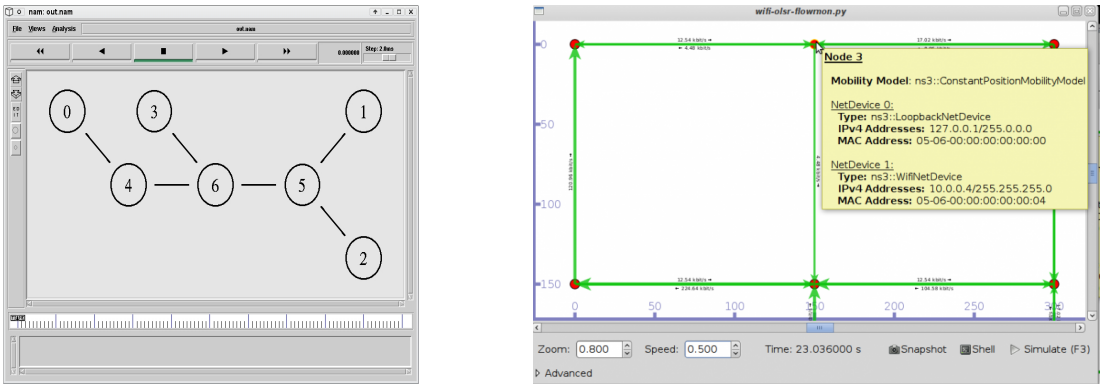
De forma a ilustrar as ferramentas desta camada, discutiremos a seguir as características de algumas contribuições realizadas e também adotadas para a camada de topo, no âmbito da *Framework* NSDL.

Uma primeira contribuição para este trabalho foi a ferramenta *Visual Network Simulator* (VNS) [166], realizada no âmbito do trabalho de Mestrado em Engenharia de Telecomunicações e Redes por Ricardo Plácido [215], realizado na Universidade da Madeira com coorientação do autor desta tese, e cujo objetivo é permitir a construção gráfica de redes com fios, com suporte a arquitetura de Serviços Diferenciados (DiffServ) [28] e realizar a sua simulação no simulador NS2. A Figura 3.3(a) ilustra um exemplo de uma rede definida com a ferramenta.

Com o VNS, o utilizador pode adicionar os nós pretendidos para a sua rede e adicionar a cada um deles os agentes e as aplicações disponíveis no NS2. Para os agentes estão disponíveis os protocolos TCP e UDP, bem como algumas das variantes do TCP, o UDP *sink* e NULL. As aplicações disponíveis são FTP, CBR e *Exponential*. A implementação da arquitetura DiffServ contempla a criação de dois encaminhadores, de borda e de núcleo, além dos mecanismos reguladores de tráfego no DiffServ: o policiador e o marcador para os encaminhadores de borda, e o agendamento para os encaminhadores de núcleo para implementar os *per hop behaviours'* (*phb's*). Devido a definições do NS2, as configurações referidas estão associadas às ligações, ou seja, numa ligação simples entre um encaminhador de borda e um de núcleo são implementados os mecanismos associados aos elementos de borda, e numa ligação entre um encaminhador de núcleo e um encaminhador de borda ou de núcleo são implementados os mecanismos associados aos elementos do núcleo.

Apesar de já existirem outros ambientes gráficos para o NS2, como o *Network AniMator* (NAM) [80], o *NS2 Scenarios Generator* (NSG2) [280] e o *Network Simulation By Mouse* (NSBM) [278], apenas dois permitiam a descrição de cenários com QoS [176] e [291], mas, no caso do primeiro, apenas para uma rede pré-definida e no caso do segundo, a sua utilização não era gratuita. O VNS ultrapassa as limitações detetadas, permitindo configurar os encaminhadores de borda e os encaminhadores de núcleo para uma qualquer rede, estando apenas restrito às limitações do simulador.

O VNS, apesar de prévio à *Framework* NSDL, é uma ferramenta enquadrada na camada de topo e pode ser utilizada para a definição de cenários de simulação de redes com fios a serem usados principalmente com o NS2. Poderá ser usado, no caso de redes simples, com outras ferramentas de simulação e/ou de gestão através da exportação de cenários para essas ferramentas, mas não foi validado para esse fim.



(a) *Network AniMator* (NAM) (b) *Live Simulation visualizer* (PyViz)

Figura 3.4: Ferramentas para a animação de simulações de rede

Uma segunda ferramenta desenvolvida para esta camada e com o fim de suportar de raiz a *Framework* NSDL é o *Visual Network Descriptor* (VND), realizada no âmbito do trabalho de Mestrado em Engenharia Informática do aluno Jesuíno Azevedo, realizado também na Universidade da Madeira [13]. A ferramenta foi desenvolvida para o ambiente da Internet, facilitando o seu uso por muitos utilizadores. Uma das principais características é a possibilidade de inserir, de uma forma muito dinâmica, os mais variados objetos de rede, atuais e futuros. Esta é uma funcionalidade importante para garantir a extensibilidade da ferramenta. A Figura 3.3(b) ilustra um exemplo de uma rede definida com a ferramenta VND.

A ferramenta ainda permite, antes da criação de cenários de rede, ao seu utilizador carregar o conjunto de objetos de rede do seu domínio de interesse, e.g., objetos das redes com fios, sem fios, de uma arquitetura de rede particular, ou, ainda, um qualquer grupo de objetos presentes na sua rede. Desta forma, o utilizador não terá de percorrer longas listas com todos os objetos de rede disponíveis e já definidos, tornando mais simples a construção de cenários de redes. Após esse passo, a rede pode ser integralmente construída e configurada, bem como a informação sobre aspetos da configuração para um determinado cenário. No caso dos cenários, as configurações para a execução de simulações e o com posicionamento dos objetos no espaço para a visualização da rede são as duas configurações disponíveis.

A ferramenta VND, além de levar para o ambiente da Internet a construção de cenários de rede, permitiu a construção de cenários de rede sem fios, característica não presente na ferramenta VNS. As redes sem fios testadas no VND foram rede Wi-Fi, WiMAX e LTE e este trabalho, bem como a utilização do simulador de redes NS3, será descrito em detalhe no capítulo 6.

Ainda na visualização de redes, mas com a característica específica de permitir visualizações dinâmicas, estão as ferramentas de animação. Não foram criadas novas ferramentas mas usadas aquelas incluídas em projetos já existentes. No caso da simulação com a aplicação NS2 foi utilizado o NAM, apresentado na Figura 3.4(a).

O NAM é usado após a execução de simulações e usa um formato próprio de dados que lhe permite reproduzir muitos dos eventos da simulação. Permite ainda visualizar, de forma simples, gráficos de alguns dos parâmetros associados aos nós e às ligações da rede, e, identificar o posicionamento e estado de um ou vários dos pacotes gerados durante a simulação. O NAM também

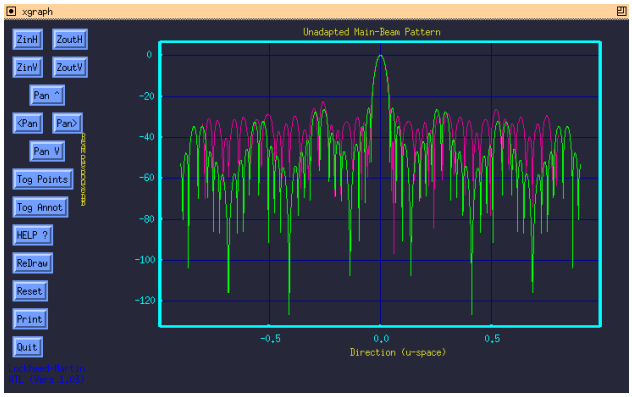


Figura 3.5: Exemplo de um gráfico criado utilizando o *xGraph*

pode ainda ser usado para criar a topologia de rede, mas de forma muito limitada. Um dos limites é disponibilizar apenas os parâmetros mais usados e comuns que estão previstos nas opções da aplicação. Outro limite é a frequência de erros na aplicação.

O apoio ao projeto sobre o simulador de rede NS3, na parte da visualização da simulação, foi feito com a ferramenta *PyViz* [47] (Figura 3.4(b)). Esta ferramenta permite a animação de simulações em tempo real e pode também ser usada para ajudar na depuração do código de uma simulação. A ferramenta permite visualizar a topologia da rede, a perda de pacotes e muitas outras informações de estado dos objetos presentes na rede.

A última categoria de ferramentas definidas para esta camada são as ferramentas para realizar a análise da rede, ou seja, as ferramentas com os dados resultantes de uma execução de uma ou várias tarefas sobre a rede. O exemplo mais claro são os dados obtidos após uma simulação, normalmente no formato de uma matriz, com a informação de todos os eventos ocorridos durante a execução ou um subconjunto de eventos determinado pelo utilizador, neste caso, com o intento de avaliar de uma forma mais restrita o funcionamento da sua rede. Não foram construídas nenhumas ferramentas novas para estas tarefas, mas integraram-se algumas ferramentas existentes. No caso das simulações para a aplicação NS2, foram usadas as linguagens AWK [7] e *Tcl* [15] para retirar todos os resultados pretendidos sobre a matriz de dados, como médias, contagens e tempos sobre os mais variados parâmetros, e, a ferramenta *xGraph* [281], apresentado um exemplo na Figura 3.5, para a visualização de diferentes parâmetros no formato gráfico com duas dimensões.

No caso da aplicação NS3 foi utilizada a ferramenta *Wireshark* [50] para a análise do tráfego gerado durante a simulação (apresentada na Figura 2.3(b)). Este é um analisador de protocolos para redes reais e permite a captura e análise do tráfego de vários tipos de redes. As redes *Ethernet* são suportadas pela versão do *Wireshark* de todos os sistemas operativos, mas, para *Linux* [256] apenas, também são suportados muitos outros tipos de rede, entre eles, *Asynchronous Transfer Mode* (ATM) [103] e *Frame Relay* [100]. O NS3 inclui a funcionalidade de exportação do resultado da simulação no formato de dados do *Wireshark* (formato *pcap* (*Packet Capture*)) e, com base nesses dados podem ser feitas todas as análises à simulação usando as funcionalidades de algumas ferramenta utilizadas para a análise de redes reais.

3.3.2 Camada Central – Linguagem de descrição de redes

A camada central é onde se encontram os componentes nucleares para a integração das ferramentas e é nesta camada que estão (1) definidas as representações dos dados a serem utilizados pelas várias ferramentas da *framework* e (2) onde se encontram os componentes para realizar a validação dos ficheiros com as descrições das redes, a transformação para o formato das diferentes ferramentas, caso necessário, e outras funções para execuções sobre a rede.

O componente basilar da *Framework* NSDL encontra-se nesta camada e é a linguagem NSDL, acrónimo que tem origem em *Network Scenario Description Language*. Esta linguagem foi desenhada para suportar a descrição de redes de dados e a informação dos cenários onde estas possam operar. As características principais desta linguagem são:

- Simplicidade: obtida por meio de uma estrutura intuitiva e com informação clara que permita aos utilizadores humanos, e não apenas 'máquinas', um rápido entendimento e capacidade de edição;
- Flexibilidade: para a descrição de redes simples e de redes mais complexas, prevendo ainda o uso de descrições com diferentes níveis de abstração para a mesma rede, e:
- Extensibilidade: permitindo de forma clara e simples a inclusão de novos objetos de rede e novos cenários de utilização.

Num ficheiro com o formato NSDL estão presentes a descrição da rede e a descrição dos cenários onde a rede opera e é avaliada. A descrição da rede de dados inclui os detalhes dos seus objetos, como por exemplo, os nós, as ligações entre os nós, as aplicações, e outros objetos de rede. A descrição do (ou dos) cenário(s) da rede detalha a informação sobre o seu funcionamento nesse contexto, como por exemplo, no caso de uma simulação deverá incluir a duração e outros parâmetros globais, os eventos pretendidos pelo utilizador para a simulação, e, a definição dos objetos sobre os quais se pretendem obter dados e, consequentemente, os resultados para consultar.

Como foi possível concluir a partir do desenvolvimento do capítulo 2, a falta de interoperabilidade entre as ferramentas de redes é devida à existência de um formato proprietário utilizado por cada uma dessas ferramentas, que em geral não é partilhado pelos demais projetos, não apenas dentro de um domínio de rede, mas também entre diferentes domínios de rede. De outra forma, nenhuma linguagem para representar a rede e as suas características é usada de forma alargada em diferentes domínios, havendo apenas algumas a serem usadas de forma restrita num domínio, destacando apenas o protocolo SNMP [49] para a gestão como formato de dados utilizado de forma alargada, mas com muitas limitações. A separação das informações de domínio é uma das propostas da linguagem, mantidas sobre uma mesma estrutura de rede.

A linguagem NSDL tem como objetivo principal suportar a interoperabilidade das diferentes ferramentas e/ou funcionalidades existentes nas camadas de topo e de base. A característica principal da linguagem que permite a interoperabilidade é a sua estrutura. A descrição da rede, ou seja dos objetos de rede e das suas relações, está separada da descrição dos seus cenários onde essa rede pode operar. Assim, vários cenários podem ser associados a uma mesma rede, permitindo que várias ferramentas possam executar sob a mesma rede as mais variadas funções, mantendo-se toda a informação englobada no mesmo ficheiro, mas hierarquicamente organizada e a informação de

cada cenário separada. O elemento da estrutura onde é feita a descrição da rede será utilizado por todas as ferramentas e todas deverão respeitar e compreender o seu significado. A especificação da informação de cada ferramenta ou domínio ficará num elemento específico da descrição. E, sendo a informação de cada ferramenta distinta, na linguagem NSDL é também proposta uma estrutura comum simples para todas as ferramentas de forma a manter alguma proximidade entre descrições e, consequentemente, a facilitar a compreensão da informação de diferentes domínios. O detalhe e a fundamentação completa da estrutura da linguagem NSDL são apresentados em detalhe no capítulo 4.

Os restantes componentes desta camada são as *Application programming interface's* (API's) que implementam as funções de suporte e de ligação entre componentes da *framework*. O primeiro componente é a 'Validação' e, como o próprio nome indica, permite verificar a correção de uma descrição de um cenário de rede perante a estrutura e regras do formato NSDL. A linguagem utilizada para definir a estrutura e as regras de construção é o XML *Schema* (XSD) [269]. Um segundo componente é a 'Transformação' e este converte um cenário NSDL para o formato de uma ferramenta específica. A linguagem utilizada para conversão é a *Extensible Stylesheet Language Family* (XSL), em particular a XSL *Transformations* (XSLT) [270]. No caso das ferramentas que utilizam nativamente o formato NSDL, destes dois componentes, apenas a 'Validação' será utilizado.

3.3.3 Camada de Base – Execução de tarefas sobre a rede

A base da *framework* inclui as aplicações que executam alguma tarefa sobre as redes e permitem obter dados e informações diversas.

Sobre a execução, uma ou várias funções são invocadas sobre as ferramentas desta camada e decorre então um processamento que, sem intervenção do utilizador, nos permite obter uma avaliação e análise de um cenário de utilização da rede. A característica diferenciadora desta camada em relação, principalmente, à camada de topo é a não interação com o utilizador. Como exemplo, em diferentes domínios, temos os simuladores de rede e as funções de coleção de dados para monitorização da operação de redes reais, ambos com funcionamento autónomo dos utilizadores, exceto, certamente, na sua definição e configuração.

Sobre os dados e as informações obtidas, uma primeira forma comum são as matrizes de resultados que incluem a informação de todos, ou parte, dos objetos presentes na rede durante o tempo de simulação e que servem para calcular os mais variados parâmetros que caraterizam o comportamento da rede, como por exemplo, taxas de erros, número de pacotes perdidos, entre muitos outros cálculos possíveis. Também é comum obter como resultado uma matriz com informação que possibilite rever a simulação, podendo avaliar graficamente o comportamento da rede e dos seus objetos sob a forma de uma animação.

Um facto determinante para a existência desta camada é a existência de um grande número de ferramentas que não possuem qualquer *interface* gráfica, mas que desempenham funções relevantes para uma rede. No trabalho [223] são apresentadas algumas ferramentas de rede sem qualquer *interface* gráfica, ou com *interfaces* gráficas muito limitadas, e estas são exemplos de ferramentas que poderiam beneficiar com as funcionalidades disponibilizadas por outras ferramentas, caso fosse simples a sua integração.

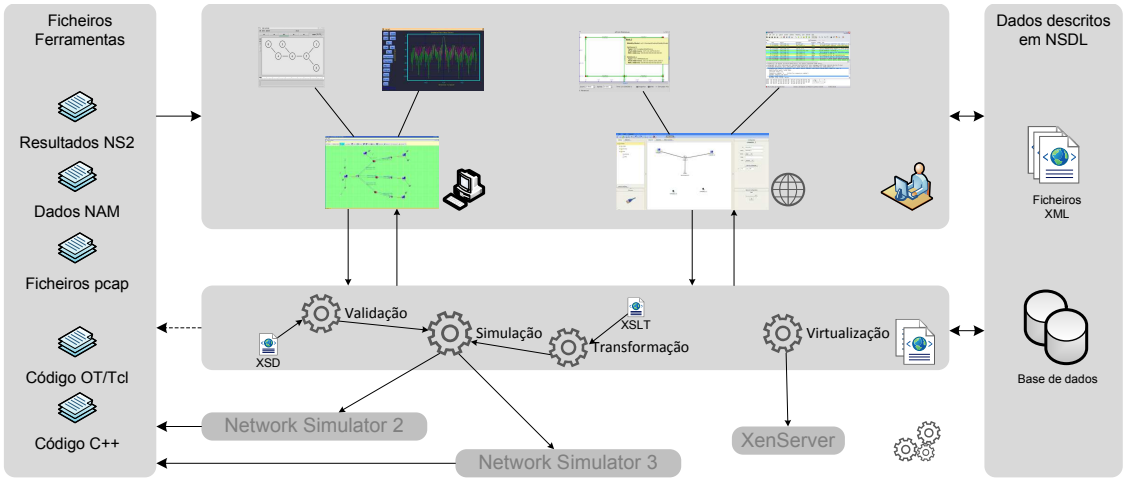


Figura 3.6: Arquitetura dos componentes NSDL já implementados e definidos

3.4 Arquitetura da *framework*

Os elementos referidos nas secções anteriores, desde ferramentas a bibliotecas de funções, existentes em cada camada da *Framework* NSDL são apresentados na Figura 3.6. As interações entre camadas são variadas e, as principais, estão representadas nessa figura. A camada de topo poderá ler e editar ficheiros XML e as descrições guardadas em bases de dados. A camada de topo poderá também invocar as funcionalidades da camada central, solicitando uma validação, uma transformação ou mesmo uma simulação. Neste último caso poderá aceder, após a execução da simulação, a resultados em diferentes formatos, produzindo gráficos e/ou animações.

A camada central recebe os pedidos da camada de topo e realiza uma determinada função. Na *framework*, ao receber um pedido de simulação, primeiro realiza a validação e transformação, e, por último, depois envia o cenário com a descrição da simulação para a ferramenta escolhida. Já se encontra incluída na *framework* a função de Virtualização para a realização de tarefas sobre ambientes virtuais. Apesar de não existir ligação, a Virtualização também envolve a validação e transformação. Uma última interação nesta camada surge com os dados. Futuras funções poderão incluir a avaliação de diferentes descrições de rede com o objetivo de realizar estudos comparativos entre diferentes cenários para melhor compreender algum(s) aspeto particular da(s) rede(s).

A camada de base recebe os pedidos para a execução de uma determinada tarefa e para uma determinada ferramenta. Como já referido, desta execução resultará um ou vários ficheiros com os dados da simulação para consulta e/ou animação.

As camadas de topo e de base incluem as ferramentas de rede atuais e, eventualmente, algumas das aplicações a desenvolver no futuro. As categorias de ferramentas encontradas são muito diversas, desde por tipo de rede, específicas à tecnologia, ou, por grupo de funcionalidades. A forma escolhida para enquadrar cada ferramenta numa camada foi pelo grau de interação do utilizador em cada uma das ferramentas. Assim, as ferramentas com *interface* gráfica e onde o utilizador intervém constantemente ao desempenhar as suas tarefas são colocadas na camada de topo. As ferramentas sem *interface* gráfica e onde o utilizador espera pela execução de uma certa função e onde a sua capacidade de intervenção é baixa ou nula, pertencem à camada de base.

O caso de estudo seguido na Figura 3.6, a simulação, está dividido em definição do cenário,

execução da simulação e visualização dos resultados. A primeira e a última tarefa envolvem ferramentas e o utilizador; a tarefa intermédia envolve somente as definições já construídas e o simulador. Existem algumas ferramentas que permitem interação durante a execução, como o *PyViz*, mas são pouco comuns e, por isso, não são consideradas como fundamentais para a definição da *framework*. O ambiente de simulação acabou por validar a escolha das funcionalidades para as camadas da *framework*, primeiro, pelas funcionalidades identificadas nesse cenário, segundo, pelas ferramentas estudadas e, posteriormente, integradas na *framework* proposta.

Um segundo ambiente, explorado em menor grau, foi o da gestão de redes, mas rapidamente se encontraram ferramentas que, seguindo esta classificação, se encaixavam na *framework*. Por exemplo, a ferramenta *tcpdump* [131], sem *interface* gráfica, permite capturar pacotes numa rede e o resultado dessa captura pode ser analisado na ferramenta *Wireshark*, já com *interface* gráfica. Ainda com o *tcpdump*, podem ser executadas capturas de pacotes que servirão como entrada de tráfego em simuladores de rede.

Apesar desta divisão, também foi possível encontrar aplicações com funcionalidades das duas camadas. Os simuladores de rede OMNET++ [262] e NS2 são exemplos desse tipo de aplicações e ambos têm componentes gráficos para a edição e visualização de algumas características da rede, ou seja, uma funcionalidade da camada de topo, e executam simulações, que após arranque, não têm intervenção do utilizador, uma funcionalidade da camada de base. Nesta fase, a *framework* já integra um conjunto de ferramentas do ambiente de simulação e outras para a edição e configuração de redes, mas é intenção da equipa de desenvolvimento obter uma aplicação que agregue todas as funcionalidades suportadas pelas *framework*. A aplicação VND já é um primeiro resultado e onde se espera que venha a ser adicionado o suporte à análise e visualização de resultados, oferecendo assim uma opção ao *xGraph* e ao *Wireshark*.

A interoperabilidade conseguida através da *Framework* NSDL é muito diversa devido à presença de variadas ferramentas com múltiplas funções. Com o propósito de clarificar essa interoperabilidade foi categorizada a sua separação em dois tipos, ou formas, de interoperabilidade: horizontal e vertical.

A *interoperabilidade horizontal* surge entre as ferramentas e as funções da mesma camada. A vantagem conseguida nesta interoperabilidade é a realização de uma mesma operação, mas numa ferramenta diferente, permitindo, por exemplo, a repetição da edição ou da avaliação de um determinado cenário de rede. Na camada de topo temos várias ferramentas para a visualização de uma rede e dos seus cenários, e, nesse caso, podemos encontrar em cada ferramenta uma funcionalidade diferente e/ou única em relação às restantes para a especificação e visualização do cenário de rede. Na camada de base, seguindo o caso de estudo Simulação, o mesmo cenário de rede poderá ser avaliado em diferentes simuladores conseguindo uma confirmação dos resultados ou um conjunto de dados novos para aprofundar a análise à rede.

A *interoperabilidade vertical* envolve a utilização de ferramentas com diferentes objetivos, permitindo uma análise da rede num novo cenário. Havendo uma descrição de um cenário de rede, podemos facilmente avaliar a rede para um cenário diferente. Um exemplo é existir uma descrição de uma rede num cenário para a sua gestão, com todos os seus objetos e parâmetros de funcionamento. Sobre essa rede, podemos acrescentar a informação necessária para uma simulação e conhecer o seu comportamento nesse cenário.

As duas formas de interoperabilidade podem ser associadas permitindo obter para uma de-

terminada rede uma maior e melhor informação sobre o seu funcionamento e apresentando novas possibilidades de utilização. Esperam-se diferentes vantagens com cada tipo de interoperabilidade: o aprofundamento do conhecimento sobre o funcionamento da rede obtém-se principalmente com a interoperabilidade horizontal; e, o alargamento das opções e possibilidades de utilização da rede, com a interoperabilidade vertical.

A última secção deste capítulo apresenta algumas conclusões sobre a definição da *Framework* NSDL.

3.5 Conclusões

A área das telecomunicações, e em particular a área das redes de dados, foi identificada pelos programadores de aplicações como um domínio complexo onde, tradicionalmente, não existem *frameworks* de qualidade [243]. A prática mostra que estes programadores iniciam o ciclo de desenvolvimento de novas aplicações com bibliotecas novas e não reutilizam algum do código já existente.

A grande diversidade de ferramentas para as mais diversas tarefas na área das redes de dados requer estruturas abrangentes e integradoras. Após o estudo das ferramentas de redes e suas características, foi proposta a *Framework* NSDL que visa ultrapassar um problema apontado recorrentemente pela comunidade de utilizadores: a interoperabilidade entre ferramentas. A estrutura da *framework* em três camadas pretende que seja simples a integração de ferramentas com características muito diferentes. A sua camada central será responsável pelas tarefas de validar e converter os dados entre os vários formatos das várias ferramentas e direcionar novas ferramentas para a adoção de um formato comum, a linguagem NSDL.

As *frameworks* apresentadas na secção 2.4.3 permitem aos seus utilizadores a integração de diversas funcionalidades na área da gestão de redes, obtendo alguma interoperabilidade. A *framework* CANDY ainda apresenta um conjunto de funcionalidades alargada, mas sobre um conjunto restrito de tipos de rede. A *framework NeDaSe* é um projeto menos abrangente e apenas se foca na simulação de redes. A *Framework* NSDL procura não ter esses limites e foi definida para suportar qualquer tipo de domínio de rede e de ferramentas. A sua estrutura simples em três camadas permite uma divisão clara das funcionalidades básicas necessárias no suporte às redes e é também possível a integração de funcionalidades para qualquer domínio de rede, seja por intermédio de novas ferramentas desse domínio ou mesmo de objetos de rede.

Como referido neste capítulo, o componente central da *framework* proposta é a linguagem NSDL, fundamental no suporte de todas as interações entre ferramentas e entre o utilizador. A sua simplicidade deve facilitar a sua adoção, mas não deverá deixar de ser flexível para suportar descrições mais complexas e futuras evoluções das redes de dados. O capítulo 4 descreve em detalhe a linguagem NSDL, ou seja, a sua estrutura e as regras para a descrição de redes de dados.

Capítulo 4

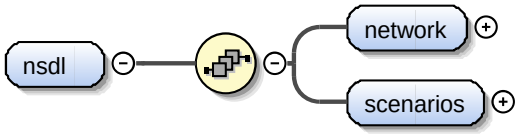
Network Scenarios Description Language (NSDL)

4.1 Introdução

A descrição, ou definição, de redes é um tópico abordado nos mais diferentes domínios e com os mais variados objetivos. Na matemática e nas ciências da computação existe uma forma comum e simples de representar uma rede e é recorrendo aos grafos [29]. Estes são estruturas constituídas, basicamente, por dois tipos de entidades: os vértices, por vezes referidos como nós ou ainda pontos, e as arestas, por vezes referidas como linhas. A sua aplicação é extensa, e, no domínio das ciências da computação, os grafos são usados para descrever "redes de comunicação, a organização da informação, os dispositivos computacionais"entre outros [225]. Um exemplo claro da sua utilização é na descrição de estruturas de informação como os sítios da Internet. Nestes, as páginas são representadas como vértices e as ligações entre elas são representadas como arestas. A teoria de grafos desenvolveu algoritmos para os mais variados fins, sendo o problema mais referido aquele que procura encontrar o caminho mais curto entre dois nós.

A linguagem a apresentar neste capítulo visa a representação de um tipo de redes, as redes de comunicação. Os objetos existentes nestas redes, pela sua especificidade, necessitam de estruturas complexas para a sua representação. O uso dos grafos, ou seja, os nós e as arestas, devido à sua simplicidade não foi considerada a estrutura mais adequada para aglomerar todas as características das entidades que se pretendem representar porém, todos os seus princípios e muitos dos seus algoritmos podem e são utilizados no domínio das redes.

A proposta de uma nova linguagem para o domínio das redes, além da representação da própria rede, visa também a representação de outras informações relacionadas com a rede, nomeadamente, o ambiente em que ela opera ou é analisada, adiante referida como os cenários de rede. Para clarificar, a representação da rede inclui a informação dos objetos existentes na rede, as relações entre eles e as suas características. A representação dos cenários de rede adicionará uma ou várias dimensões de informação à descrição da rede, acrescentando detalhe sobre a utilização ou a operação da rede num determinado contexto, com fins de gestão, avaliação e/ou teste. Como exemplos de informações a serem adicionadas à rede temos, por exemplo, os dados necessários para a execução de uma simulação de rede, ou ainda, os parâmetros necessários para instalar a rede num ambiente virtual para posterior operação ou teste.



(a) Estrutura base do NSDL

```
1 <xs:element name="nsdl" >
2         <xs:complexType>
3                 <xs:sequence>
4                         <xs:element ref="network" use="required"/>
5                         <xs:element ref="scenarios"/>
6                 </xs:sequence>
7         </xs:complexType>
8 </xs:element>
```

(b) XML *Schema* da estrutura base do NSDL

Figura 4.1: Estrutura base de um ficheiro NSDL

O resto deste capítulo detalhará as características da linguagem proposta e dos dados que pretendem conter na sua descrição. A secção 4.2 apresenta a estrutura base de uma descrição e os elementos comuns a todos os objetos. A secção 4.3 descreve os elementos que farão parte da descrição da topologia da rede. A secção 4.4 apresenta os cenários que conterão informação extra sobre a rede. Na secção 4.5 é explicada a estrutura para a organização das especificações NSDL. Por fim, em 4.6 são apresentadas algumas conclusões sobre a linguagem NSDL.

4.2 Estrutura da Linguagem NSDL

A nova linguagem proposta neste capítulo designa-se por *Network Scenarios Description Language* (NSDL) e, como referido na introdução, visa suportar a descrição de uma qualquer rede de comunicação, seja esta simples com apenas alguns objetos de redes, como redes de grande dimensão com muitos e variados objetos. O estudo das linguagens realizado no capítulo Estado da Arte procurou destacar as características mais importantes numa linguagem para a descrição de redes de comunicação, tendo sido concluído que sobressaíam a simplicidade, extensibilidade e independência. A linguagem NSDL procurou seguir estes princípios ao longo da sua definição de forma a poder se considerar uma solução para a interoperabilidade entre ferramentas de rede.

Uma primeira característica da estrutura da NSDL é a divisão dos dados da rede em dois grandes grupos: <network> e <scenarios>. O elemento <network> inclui os objetos de rede comuns, como o <node>, o <link>, a <application>, o <protocol> e o <interface>. Um objeto menos comum, mas que também está presente na NSDL é o <domain>. No elemento <scenarios> estarão os componentes dos cenários de rede, como por exemplo, os cenários de simulação e/ou os cenários de visualização. Os primeiros na descrição de experiências de simulação sobre a rede definida, os segundos para a configuração da visualização da rede em ambientes gráficos. A Figura 4.1 ilustra o elemento raiz, <nsdl>, e os elementos base, <network> e <scenarios>, existentes em todas as descrições feitas com a linguagem NSDL, como uma imagem - Figura 4.1(a) - e como um código no formato XML *Schema* - Figura 4.1(b). Como forma de estruturar a apresentação da linguagem, a

fundamentação e definição de todos e cada um destes elementos é deixada para as secções seguintes.

A apresentação dos restantes objetos da NSDL será feita de uma forma similar à utilizada para descrever a sua estrutura base, através da apresentação da figura da estrutura do objeto e a da sua definição em XML *Schema*. A descrição de alguns elementos, pela sua dimensão, será complementada com uma tabela com o detalhe dos seus atributos e elementos.

Apesar de haver uma tradução exata para cada um dos objetos da linguagem referidos neste capítulo, os termos que definem os objetos serão apresentados em inglês por serem utilizados como palavras reservadas na construção da linguagem.

4.2.1 Objeto NSDL

Em termos de implementação, a linguagem NSDL tem como elemento base o objeto NSDL, adiante tratado como *object*, e, assim, todos os objetos da rede, ou um qualquer objeto pertencente a um cenário, são identificados e caraterizados a partir da mesma estrutura base. A linguagem foi definida de forma a ser flexível e permitir acomodar os mais diversos objetos de rede. Ainda, de forma a assegurar a consistência na definição dos objetos existentes num cenário de rede NSDL e para, consequentemente, não limitar a interoperabilidade entre diferentes cenários, foi necessário garantir uma estrutura comum mínima e alguns princípios de desenho para proporcionar a consistência na definição dos objetos existentes num atual ou futuro cenário de rede NSDL.

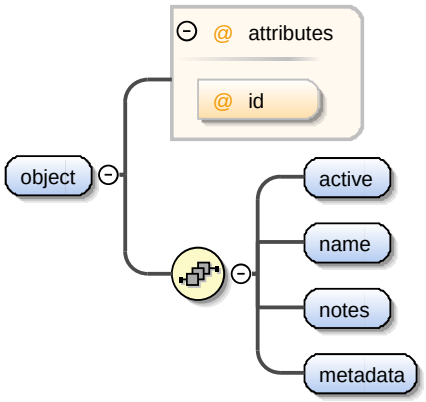
As informações presentes neste objeto providenciam a identificação unívoca de qualquer objeto num cenário e permitem a adição de mais algumas características. Um elemento presente no *object* é o <active> e contém o estado do objeto no âmbito do cenário em termos de estar ativo ou inativo. Os atributos e elementos do objeto NSDL, designado por type_nsdlobject, são apresentados na Figura 4.2.

De entre os atributos e os elementos apresentados, apenas o atributo @id é obrigatório. Os restantes dados apenas serão preenchidos se o utilizador o pretender. Na Figura 4.2(a) é apresentada uma representação do objeto com os seus atributos e elementos. Na Figura 4.2(b) temos a representação desse mesmo objeto em XML *Schema*. As propriedades de cada um dos elementos e atributos são apresentadas na tabela 4.1.

Todos os atributos e os elementos são de um tipo de dados particular mas, no caso do elemento <metadata> este elemento é a base para uma estrutura de informação mais complexa. A *metadata*, ou metadados, ou ainda meta-informação, contém dados sobre os dados, ou seja, à descrição já realizada sobre o cenário de rede NSDL podem ser adicionadas informações sobre essa mesma descrição. Exemplos de dados para os metadados são as informações de autoria, de data de criação e modificação, os direitos de utilização – <copyright> – entre muitos outros dados. A importância

Tabela 4.1: Detalhe dos elementos e atributos do <object>

@Atributo ou Elemento	Obrigatório	Ocorre (0 até n)	Tipo Dados	Único	Observaçoes
@id	Sim	1	ID	Sim	Identificador único para cada objeto
active	Não	0..1	Booleano	Não	Estado inicial do objeto, ativo por omissão
name	Não	0..1	Texto	Não	Designação do objeto
notes	Não	0..1	Texto	Não	Descrição do objeto
metadata	Não	0..1	metadados	Não	Dados complementares sobre o objeto



(a) Estrutura de *object*

```
1 <xs:complexType name="type_nsdlobject">
2       <xs:attribute ref="id" use="required"/>
3       <xs:sequence minOccurs="0">
4               <xs:element ref="active" minOccurs="0" maxOccurs="1"/>
5               <xs:element ref="name" minOccurs="0" maxOccurs="1"/>
6               <xs:element ref="notes" minOccurs="0" maxOccurs="1"/>
7               <xs:element ref="metadata" minOccurs="0" maxOccurs="1"/>
8       </xs:sequence>
9 </xs:complexType>
```

(b) XML *Schema* da estrutura de *object*

Figura 4.2: Objeto base NSDL, comum a todos os objetos NSDL

da informação contida nos metadados surge nas facilidades que proporciona na criação, na gestão e na partilha dos recursos que descreve [246]. No âmbito deste trabalho estas facilidades não foram implementadas, optando-se apenas pela definição de uma estrutura simples com as seguintes informações: título, descrição, autor, data de criação, e, direitos de utilização.

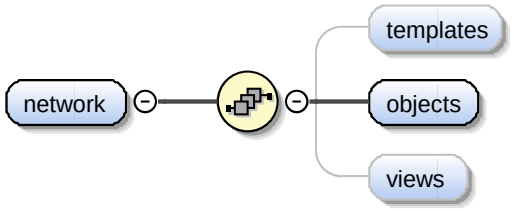
A apresentação dos restantes objetos da linguagem NSDL será feita de uma forma similar à utilizada para descrever o *object*.

4.3 Network

O elemento <network>, como o nome indica, contém a descrição da rede, mais especificamente a descrição e as características dos objetos que formam a rede, como os nós e as ligações entre eles, e muitos outros objetos presente numa rede. Além da descrição da rede, em <network> existem dois elementos, <templates> e <views>, que ajudam e simplificam a tarefa de descrever uma rede.

O elemento <network> é obrigatório, logo, a sua ausência numa descrição NSDL não é possível. Os cenários são sempre definidos genericamente, mas implicam a existência de referências para objetos da rede. Assim, não faria sentido, por exemplo, permitir a existência de cenários sem obrigar a presença de uma rede. A Figura 4.3 apresenta os elementos que fazem parte do elemento <network>.

O elemento <objects> é o componente principal da descrição de rede visto ser neste ponto da estrutura onde estarão todos os objetos de rede. É um elemento obrigatório conjuntamente



(a) Estrutura de <network>

```
1 <xs:element name="network">
2       <xs:complexType>
3             <xs:sequence>
4                   <xs:element name="templates" minOccurs="0"/>
5                   <xs:element name="objects"/>
6                   <xs:element name="views" minOccurs="0"/>
7             </xs:sequence>
8       </xs:complexType>
9 </xs:element>
```

(b) XML *Schema* da estrutura de <network>

Figura 4.3: Elemento <network>, contém os objetos da rede e a sua descrição

com <nsdl> e <network> numa descrição NSDL. O elemento <templates> contém objetos de rede genéricos que podem ser invocados pelos objetos de <objects>, através do atributo @template (detalhado posteriormente). O elemento <views> permite a definição de grupos de objetos de rede para organizar, por exemplo, a rede por domínios e para a referência de múltiplos objetos pelos cenários. Devido à sua importância, faremos a descrição dos <objects> primeiro, e, após, serão apresentadas as características e a importância dos <templates> e das <views>.

4.3.1 Elemento <objects>

O elemento <objects> contempla os objetos presentes na rede a descrever. Os objetos incluídos neste elemento são objetos concretos e estão relacionados com os equipamentos ou as configurações que constituem a rede. A referência a esta relação entre objetos da descrição e objetos concretos é feito com o propósito de clarificar e fazer a distinção com os objetos incluídos nos <templates> que, neste caso, são objetos genéricos para apoio à descrição do cenário, ou seja, não têm qualquer ligação a um objeto real.

Os objetos de rede definidos inicialmente para a NSDL foram seis: <node>, <link>, <domain>, <application>, <protocol> e <interface>. A escolha deste conjunto de objetos em particular surge, essencialmente, de duas influências:

- A primeira influência aponta para o estudo das várias linguagens de descrição existentes para as redes e, consequentemente, para o conjunto de objetos que foi encontrado nas diferentes linguagens. O <node> e o <link> são os elementos comuns em todas as linguagens. Os outros objetos surgem nas diversas linguagens de uma forma mais pontual. Por vezes existem como objetos, outras vezes as suas propriedades incluem-se nas propriedades de outro objeto, normalmente o <node>. Como exemplo, é comum encontrar dados sobre <protocol> e <application> dentro de um <node>, sem qualquer separação, como a linguagem XMLbNSDL

[44], e;

- A segunda influência surge dos equipamentos de rede e terminais reais. A linguagem representa de uma forma análoga à realidade os equipamentos e os seus componentes internos. Um exemplo de algo em falta em muitas das linguagens é o *interface*, mas este existe em todos os equipamentos de rede reais.

A NSDL procura ser uma linguagem simples de compreender, aproximando-se o mais possível da realidade, e sem deixar de possibilitar a descrição detalhada de um cenário de rede. O conjunto de objetos proposto engloba a maioria dos elementos presentes numa qualquer rede e a sua inclusão numa descrição de rede é simples e feita numa estrutura clara. Se for necessário suportar um maior detalhe na caraterização da rede, os objetos podem ser estendidos com novos objetos e com outras características.

Nas secções seguintes será feita uma descrição mais detalhada de cada um dos objetos. Segue-se uma breve apresentação de algumas das suas características:

- O *node* e o *link* são os objetos basilares; estão presentes em todas as linguagens de descrição de redes, seja no domínio das redes de comunicação, seja em outros tipos de redes, e descrevem, respetivamente, os equipamentos ativos de rede e as ligações entre eles, e;

- O *domain* permite incluir num cenário de rede um objeto que representa uma rede especifica e/ou, se necessário, complexa, como por exemplo a Internet. Podem, assim, coexistir numa descrição objetos como diferentes níveis de abstração.

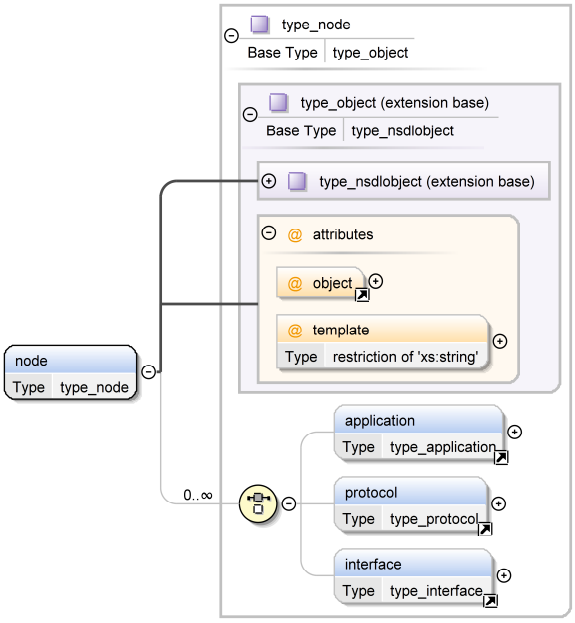
Os restantes três objetos têm a particularidade de existirem apenas dentro do <node>, ou de uma qualquer extensão sua. Entre estes três objetos existem também algumas relações, nomeadamente: entre a <application> e <protocol>, e; entre <protocol> e <interface>. Na primeira relação implica que uma <application> pode estar ligada a um <protocol>, na segunda temos um <protocol> que pode estar ligado a um <interface>. Na descrição detalhada dos objetos serão indicados exemplos da utilização e da importância destas relações. Os objetos são apresentados sucintamente de seguida:

- O elemento <application> permite incluir na descrição da rede uma aplicação que pode gerar, receber ou encaminhar tráfego, e.g., o FTP e o *Telnet*, nas suas componentes cliente e servidor;

- O <protocol>, como o nome indica, permite adicionar à descrição da rede as definições dos protocolos presentes na rede, e.g., o protocolo IP ou o *Routing Information Protocol* (RIP), e;

- O objeto <interface> serve para a descrição das placas de rede que interligam um node a uma rede. Podem, em alguns casos, serem idênticos ao <link>. Por exemplo: dois *node's* ligados por um *link* ou dois *nodes* onde cada um tem um *interface* terá o mesmo significado. No entanto, e no caso das redes sem fios, os *interfaces* são necessários para incluir na rede um nó sem fios, visto que o *link* não foi definido para esses cenários.

As características de cada objeto, além das comuns a todos eles e já descritas na secção 4.2.1, são a seleção do conjunto de informações que constituem um denominador comum entre os objetos de



(a) Estrutura de um <node>

```
1  <xs:element name="node" type="type_node"/>
2  ...
3  <xs:complexType name="type_node">
4         <xs:complexContent>
5                <xs:extension base="type_object">
6                       <xs:choice maxOccurs="unbounded" minOccurs="0">
7                              <xs:element ref="application" minOccurs="0"/>
8                              <xs:element ref="protocol" minOccurs="0"/>
9                              <xs:element ref="interface" minOccurs="0"/>
10                      </xs:choice>
11               </xs:extension>
12        </xs:complexContent>
13 </xs:complexType>
```

(b) XML *Schema* da estrutura de um <node>

Figura 4.4: Elemento <node> com todos os elementos e atributos incluindo os objetos *application*, *protocol* e *interface*

um dado tipo. Por exemplo, veremos no objeto <link> que uma das suas propriedades é a largura de banda (*bandwith*), visto ser uma propriedade que existe em todos os tipos de ligações. Outras propriedades, mais específicas num objeto de rede, não devem ser incluídas no objeto base mas sim numa extensão (especialização) particular desse objeto base. Um exemplo são as propriedades de Qualidade de Serviço (QoS) que existem em nós definidos para suportar essas soluções, mas que não são genéricas, logo não devem estar no objeto <node>. Para este exemplo em concreto pode ser indicado o objeto <dscorenode> – nó de núcleo para a arquitetura de Serviços Diferenciados, a ser apresentado na secção 5.5 – que é um <node>, mas foi estendido para conter, entre outras, as propriedades para a definição de políticas de agendamento (*scheduling*) do tráfego da rede.

Os objetos de rede, além de serem um objeto NSDL, recebem dois novos atributos: @template e @object. O primeiro, como o nome indica irá associar um objeto específico a um objeto genérico que deverá existir no elemento <templates>. O segundo atributo define o tipo de objeto de rede

que cada objeto constitui. Ainda sobre @object, esta informação é desnecessária para os objetos base por ser o mesmo valor que a designação do elemento (elemento <node> e @object *node*), mas é importante existir nas extensões para simplificar a implementação das transformações entre linguagens e apoiar na validação de cenários. Também, por vezes, este atributo também clarifica num ficheiro NSDL qual o tipo de objeto que está a ser descrito. Na Figura 4.4(a) são apresentados estes atributos. A utilização de um qualquer destes atributos é opcional.

Nas subseções seguintes serão apresentados em maior detalhe cada um dos objetos de base definidos para a linguagem NSDL.

Elemento <node>

O elemento <node> no contexto das redes de comunicação representa os equipamentos ativos. Além das características de qualquer objeto, apresentadas na Figura 4.4 nos elementos type_object e type_nsdlobject, o <node> pode conter as <application>, <protocol> e <interface> que pretenderem. O <node>, apesar de fundamental na construção de uma topologia de rede, não tem muita informação própria. Algumas extensões (ou especializações) do *node* já definidas, como *router*, *switch* e *access point*, já adicionam mais algumas informações na descrição de uma rede, no entanto mantendo a simplicidade do *node*.

Além da importância para a topologia, o *node* contém os objetos *application*, *protocol* e *interface* e são estes que introduzem e interagem com o tráfego existente na rede. Um *node* pode incluir qualquer número destes objetos.

Na secção 4.5 são apresentados todas as extensões já definidas para o *node*. Algumas das extensões mais relevantes do *node* serão também demonstradas na secção 5.5 e, neste caso, apresentarão diversos componentes para a implementação de cenários com Qualidade de Serviço.

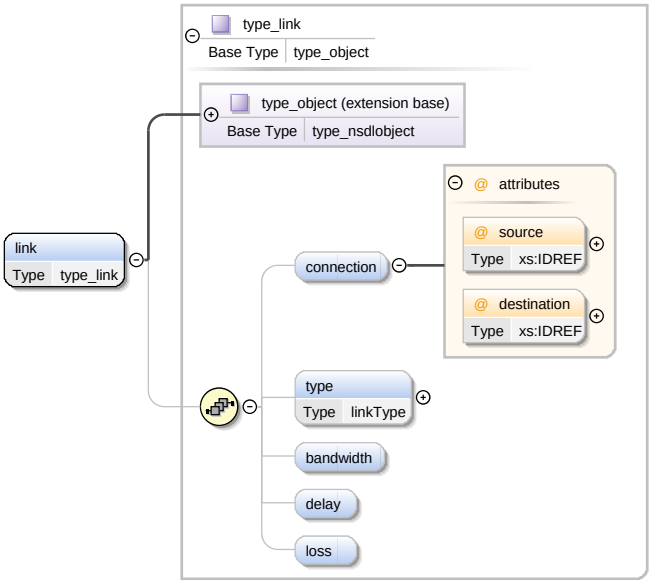
Elemento <link>

O elemento <link> implementa a conexão entre dois *nodes* de uma rede. Para que o tráfego possa ser conduzido diretamente de um *node* para outro, deverá existir, pelo menos, um *link* entre eles. A Figura 4.5 apresenta a estrutura de um *link* em NSDL.

Além dos dados base de um objeto, o <link> tem como elemento principal o elemento <connection>. Este contém os atributos @source e @destination, e aqui são guardadas as informações que relacionam dois *node's* numa rede. Além deste elemento, e atributos, o <link> ainda tem o <type>, <bandwidth>, <delay> e <loss>. A tabela 4.2 contém toda a descrição destes componentes do *link*.

O elemento <type> especifica o tipo de ligação em termos de sentidos do tráfego. No caso de '*simplex*' refere-se apenas a tráfego a circular do nó de origem para o nó de destino (ambos definidos em <connection>) e no caso de '*duplex*' em ambos os sentidos (nesse caso @source e @destination apenas identificam os *node's* sem indicar um sentido para o tráfego). Os três elementos restantes, <bandwidth>, <delay> e <loss>, permitem indicar os valores de características básicas de rede: a velocidade, o atraso e a taxa de perdas na ligação.

Na secção 4.5 são apresentados as extensões já definidas para o *link* e que incluem alguns tipos de ligações mais comuns nas redes atuais.



(a) Estrutura de um <link>

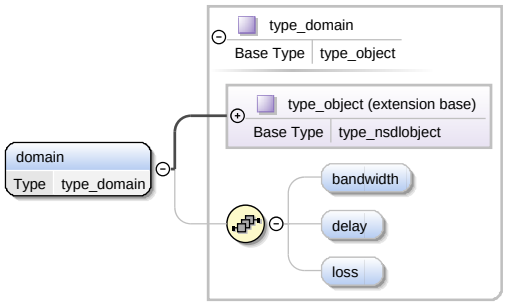
```
1  <xs:element name="node" type="type_node"/>
2  ...
3  <xs:complexType name="type_link">
4         <xs:complexContent>
5                <xs:extension base="type_object">
6                       <xs:sequence minOccurs="0">
7                              <xs:element name="connection" minOccurs="0">
8                                     <xs:complexType>
9                                            <xs:attribute name="source"/>
10                                           <xs:attribute name="destination"/>
11                                    </xs:complexType>
12                             </xs:element>
13                             <xs:element name="type" minOccurs="0"/>
14                             <xs:element name="bandwidth" minOccurs="0"/>
15                             <xs:element name="delay" minOccurs="0"/>
16                             <xs:element name="loss" minOccurs="0"/>
17                      </xs:sequence>
18               </xs:extension>
19        </xs:complexContent>
20 </xs:complexType>
```

(b) XML *Schema* da estrutura de um <link>

Figura 4.5: Elemento <link>, objeto para interligar os nós de uma rede

Tabela 4.2: Detalhe dos elementos e atributos do <link>

@Atributo ou Elemento	Obrigatório	Ocorre (0 até n)	Tipo Dados	Único	Observaçoes
connection	Sim	1	Estrutura	Sim	Estrutura para guardar a origem e destino da ligação
@source	Sim	1	ID	Sim	Identificador do nó origem
@destination	Sim	1	ID	Sim	Identificador do nó de destino
type	Não	0..1	linkType	Sim	Tipo de ligação, simplex ou duplex
bandwidth	Não	0..1	Texto	Sim	Largura de banda em bits por segundo
delay	Não	0..1	Texto	Sim	Atraso em segundos
loss	Não	0..1	Texto	Sim	Perda em percentagem



(a) Estrutura de um <domain>

```
1  <xs:element name="domain" type="type_domain"/>
2  ...
3  <xs:complexType name="type_domain">
4         <xs:complexContent>
5                <xs:extension base="type_object">
6                       <xs:sequence minOccurs="0">
7                              <xs:element name="bandwidth" minOccurs="0"/>
8                              <xs:element name="delay" minOccurs="0"/>
9                              <xs:element name="loss" minOccurs="0"/>
10                      </xs:sequence>
11               </xs:extension>
12        </xs:complexContent>
13 </xs:complexType>
```

(b) XML *Schema* da estrutura de um <domain>

Figura 4.6: Elemento <domain>, objeto híbrido que permite integrar na rede uma abstração de uma rede

Elemento <domain>

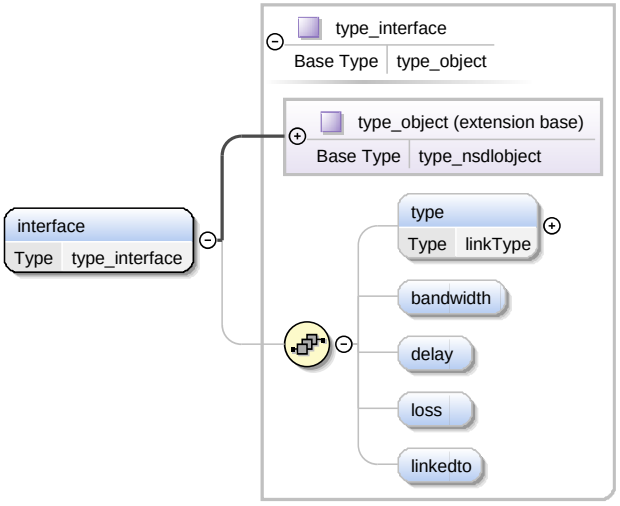
O *domain* é um objeto híbrido e que não tem correspondência direta com um objeto concreto. O seu objetivo é representar uma rede, ou uma tecnologia, de uma forma simples. A referência a ser um objeto híbrido deve-se a poder ser referenciado, ou ligado, por *link's*, tal e qual um *node*, mas também contém propriedades de um *link*, como a largura de banda e o atraso. Ou seja, um *domain* é uma mistura de propriedades de um *node* e de um *link*. A Figura 4.6 apresenta o elemento <domain>, e a sua estrutura em XML *Schema*.

Na linguagem NSDL, o *domain* é um dos principais responsáveis pela capacidade de abstração. Uma descrição de uma qualquer rede pode ser realizada especificando todos os objetos de uma forma precisa, mas, se necessitarmos de uma representação simplificada dessa rede o objeto *domain* pode ser usado para obter um modelo de toda essa rede.

Os elementos do objeto base *domain*, sendo iguais ao *link*, não serão apresentados em tabela. O <domain> pode ainda ser estendido como <qos> e <diffserv>, podendo então incluir um conjunto muito variado de propriedades. Na secção 4.5 são apresentados as extensões já definidas para o *domain*.

Elemento <interface>

O elemento <interface> serve para adicionar placas de rede aos cenários de rede e é responsável pela transferência de mensagens entre os *nodes*. Este pode ser das mais variadas tecnologias, sejam



(a) Estrutura de um <interface>

```
1  <xs:element name="interface" type="type_interface"/>
2  ...
3  <xs:complexType name="type_interface">
4         <xs:complexContent>
5                <xs:extension base="type_object">
6                       <xs:sequence minOccurs="0">
7                              <xs:element name="type" minOccurs="0"/>
8                              <xs:element name="bandwidth" minOccurs="0"/>
9                              <xs:element name="delay" minOccurs="0"/>
10                             <xs:element minOccurs="0" name="loss"/>
11                             <xs:element minOccurs="0" name="linkedto"/>
12                      </xs:sequence>
13               </xs:extension>
14        </xs:complexContent>
15 </xs:complexType>
```

(b) XML *Schema* da estrutura de um <interface>

Figura 4.7: Elemento <interface>, representa as placas de rede existentes nos nós de uma rede

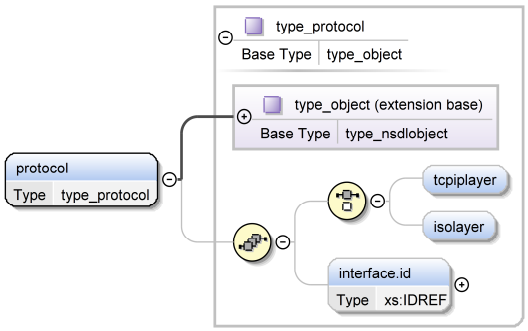
elas com fios e/ou sem fios. Como já referido na secção 4.3.1, o *interface* pode, se forem utilizadas algumas configurações de rede com fios, ter um significado igual ao *link*. No caso das redes sem fios, a representação prevista na NSDL é conseguida apenas com o *interface*. A Figura 4.7 apresenta a estrutura de um *interface*.

O *interface* base apresentado na Figura 4.7 tem as mesmas propriedades do *link*, exceto no elemento <linkedto>, este presente em *interface*, e no elemento <connection>, apenas presente no *link*. O elemento <linkedto> permite a referência – ligação – a outro *interface*. O *interface* apenas pode ser adicionado dentro de um *node*, ou dentro de uma qualquer extensão do *node*.

Na secção 4.5 são apresentados as extensões já definidas para o *interface*, de destacar os *interfaces* para a representação de redes sem fios.

Elemento <protocol>

O elemento <protocol> tem como objetivo adicionar à rede a informação dos protocolos de rede presentes no cenário a construir. O <protocol>, à semelhança do <interface>, apenas pode ser



(a) Estrutura de um <protocol>

```
1  <xs:element name="protocol" type="type_protocol"/>
2  ...
3  <xs:complexType name="type_protocol">
4         <xs:complexContent>
5                <xs:extension base="type_object">
6                       <xs:sequence minOccurs="0">
7                              <xs:choice minOccurs="0">
8                                     <xs:element name="tcpiplayer"/>
9                                     <xs:element name="isolayer"/>
10                             </xs:choice>
11                             <xs:element name="interface.id"/>
12                      </xs:sequence>
13               </xs:extension>
14        </xs:complexContent>
15 </xs:complexType>
```

(b) XML *Schema* da estrutura de um <protocol>

Figura 4.8: Elemento <protocol>, permite adicionar dados de um protocolo a uma rede

inserido num node. Os objetos *interface* e *application* da NSDL surgem para definir as características de rede das camadas de topo e de base de uma arquitetura de rede. O *protocol* pretende suportar todas as restantes características, ou seja, objetos, entre estas duas camadas, independentemente da arquitetura de rede. Um fator para esta definição menos aprofundada do *protocol* está na preocupação da NSDL em manter nos objetos base uma grande simplicidade para permitir a sua utilização por utilizadores inexperientes. Um utilizador comum tem sobre uma rede a perceção da existência de aplicações. Tem também, em menor grau, a perceção da existência de placas e ligações. Já no caso do protocolo, e ainda nesta perspetiva, este é um componente, normalmente, transparente ou desconhecido para o utilizador.

Se nos restantes objetos conseguiu-se, em variados graus, apontar um conjunto de propriedades comuns para definir cada objeto base, no caso do *protocol* tal definição não foi possível devido ao

Tabela 4.3: Detalhe dos elementos e atributos do <protocol>

@Atributo ou Elemento	Obrigatório	Ocorre (0 até n)	Tipo Dados	Único	Observaçoes
tcpiplayer	Não	0..1	Texto	Sim	Camada de rede na arquitetura TCP/IP
isolayer	Não	0..1	Texto	Sim	Camada de rede na arquitetura OSI
Interface.id	Não	0..1	ID	Não	Ligação para um interface

Tabela 4.4: Detalhe dos elementos e atributos do <application>

@Atributo ou Elemento	Obrigatório	Ocorre (0 até n)	Tipo Dados	Único	Observaçoes
role	Não	0..1	Texto	Sim	Define o comportamento, server, client ou peer
rate	Não	0..1	Texto	Sim	Taxa de transmissão em bits por segundo
packetsize	Não	0..1	Texto	Sim	Dimensão do pacote em bytes
src.app	Não	0..1	ID	Não	Identificação da aplicação de origem, se cliente
dst.app	Não	0..1	ID	Não	Identificação da aplicação de destino, se servidor
protocol.id	Não	0..1	ID	Sim	Identificação do protocolo que suporta a aplicação

número elevado de protocolos, e à sua variedade em cada uma das diversas camadas protocolares. A estrutura do objeto base *protocol*, apresentado na Figura 4.8, ficou assim reduzida a um conjunto pequeno de elementos, deixando para cada uma das suas extensões os elementos e atributos associados aos protocolos. Na tabela 4.3 são apresentados os elementos e atributos do <protocol> em detalhe.

Os elementos <tcpiplayer> e <isolayer> podem ser adicionados à descrição do protocolo e definem qual a camada onde está o protocolo, dentro da arquitetura escolhida. Ambos são opcionais e permitem, principalmente no caso de novos protocolos ou protocolos menos comuns, dar a conhecer a utilizadores menos experientes a camada em que o protocolo opera. O elemento <interface.id> permite a ligação entre *protocol's* e *interface's*. Como exemplo, as configurações do endereço IP de um *node*, no caso de existir apenas um *interface* ou *link*, não obrigam a nenhuma associação com um *interface* mas, no caso de existirem dois ou mais *interface's*, a configuração de um endereço IP, sem referência ao *interface*, é ambígua, sendo portanto necessária a associação a um *interface*.

Na secção 4.5 são apresentados as extensões já definidas para o <protocol>.

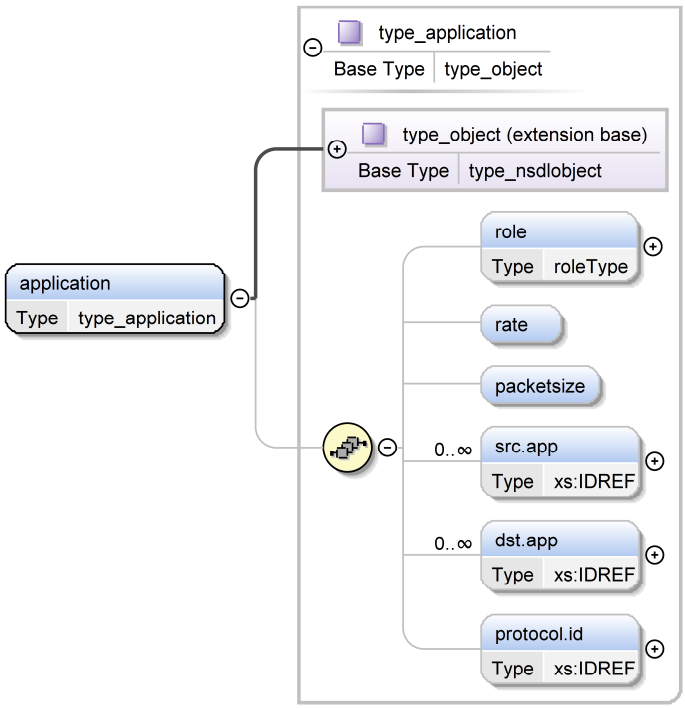
Elemento <application>

A <application> é o objeto NSDL mais próximo do utilizador e é o objeto gerador da maioria do tráfego a circular na rede. Os pacotes de rede que não são perdidos e/ou descartados pela rede serão recebidos por *application's* também. O objeto base *application* mantém apenas os elementos necessários para caraterizar uma aplicação genérica. Este objeto tem já definido na linguagem NSDL muitas extensões para permitir a descrição de redes com servidores *web*, servidores de ficheiros, entre outras aplicações. A Figura 4.9 apresenta a estrutura do objeto *aplication*. Os elementos do objeto *application* são apresentados em detalhe na tabela 4.4.

O elemento <role> carateriza o tipo de comportamento da <application>. Os elementos <rate> e <packetsize> definem a taxa e o tamanho do pacote, respetivamente. O <src.app> e <dst.app> identificam os objetos *application* a que estão ligados, normalmente apenas um, mas é possível, no caso de uma *application* mais complexa, este poder ser servidor e cliente ao mesmo tempo. O elemento <protocol.id> permite indicar qual o protocolo que suporta a aplicação. Por omissão as *application* têm um determinado *protocol*, mas pode ser alterado ou necessário configurar de forma particular o *protocol*. Um exemplo possível é alterar o *protocol* por omissão da aplicação FTP, o TCP, pelo *protocol* UDP.

Na secção 4.5 são apresentados as extensões já definidas para a <application>.

Apesar de poderem ser usados os objetos base, na maioria dos cenários desenvolvidos a utilização



(a) Estrutura de uma <application>

```
<xs:element name="application" type="type_application"/>
...
<xs:complexType name="type_application">
        <xs:complexContent>
                <xs:extension base="type_object">
                        <xs:sequence minOccurs="0">
                                <xs:element name="role" minOccurs="0"/>
                                <xs:element name="rate" minOccurs="0"/>
                                <xs:element name="packetsize" minOccurs="0"/>
                                <xs:element name="src.app" minOccurs="0"/>
                                <xs:element name="dst.app" minOccurs="0"/>
                                <xs:element name="protocol.id" minOccurs="0" />
                        </xs:sequence>
                </xs:extension>
        </xs:complexContent>
</xs:complexType>
```

(b) XML *Schema* da estrutura de uma <application>

Figura 4.9: Elemento <application>, permite configurar uma aplicação que opera na rede

```
...
<objects>
   <computer id="computer1" >
      <http id="c_1HTTP">
         <role>client</role>
      </http>
      <ipv4 id="c_1IP">
         <address>DHCP</address>
      </ipv4>
   </computer>
   <computer id="computer2">
      <http id="c_2HTTP">
         <role>client</role>
      </http>
      <ipv4 id="c_2IP">
         <address>192.168.1.1</address>
         <mask>255.255.255.0</mask>
      </ipv4>
   </computer>
...
```

(a) Sem *templates*

```
...
<templates>
    <computer id="templPC">
        <http id="tempHTTP">
            <role>client</role>
        </http>
        <ipv4 id="templIP">
            <address>DHCP</address>
        </ipv4>
    </computer>
</templates>
<objects>
    <computer id="computer1" template="templPC"/>
    <computer id="computer2" template="templPC">
        <ipv4 id="c_2IP">
            <address>192.168.1.1</address>
            <mask>255.255.255.0</mask>
        </ipv4>
    </computer>
...
```

(b) Com *templates*

Figura 4.10: Código de exemplo de utilização dos *templates*

de extensões é muito mais usual por permitir representações mais fidedignas. Mas a compreensão da base de construção dos objetos considerou-se fundamental para a compreensão da linguagem NSDL. Nos capítulos 5, 6 e 7 são apresentadas implementações da linguagem para diversos cenários de rede e onde surgem novos objetos, extensões destes objetos base.

De seguida, dá-se sequência à apresentação de outros componentes da linguagem NSDL, os <templates>.

4.3.2 Elemento <templates>

A construção de cenários de rede envolve frequentemente a definição de um número elevado de objetos de rede. A definição individualizada de cada um deles é sempre possível, mas é comum acontecer em diversos cenários a utilização de grandes quantidades de objetos, todos com propriedades comuns. Como exemplo, nas redes de sensores sem fios é normal existirem dezenas ou centenas de *nodes* com as mesmas características, alterando-se apenas entre eles a sua posição e a sua identificação. Outro exemplo é nas redes empresariais com um número elevado de computadores cliente, todos com as mesmas características (com o mesmo sistema operativo, navegadores, entre outras aplicações).

Os *templates*, no âmbito da NSDL, permitem definir objetos genéricos com configurações específicas que podem ser reutilizados sem limites. Um utilizador, recorrendo a um dos exemplos do parágrafo anterior, pode definir um *computer* (extensão de *node*) no elemento <templates>, e configurá-lo de forma completa (*interfaces* e *applications*). A cada objeto *computer* adicionado na rede, dentro do elemento <objects>, pode ser definido o atributo @template apontando para o objeto criado nos *templates*, e, assim, o objeto herda todas as configurações do objeto genérico. As Figuras 4.10(a) e 4.10(b) apresentam duas descrições idênticas, respetivamente sem e com *templates*.

A utilização dos *templates* não invalida a especificação particular de um objeto, mesmo que ligado a um objeto *template*. Todas as características redefinidas no objeto em <objects> preva-

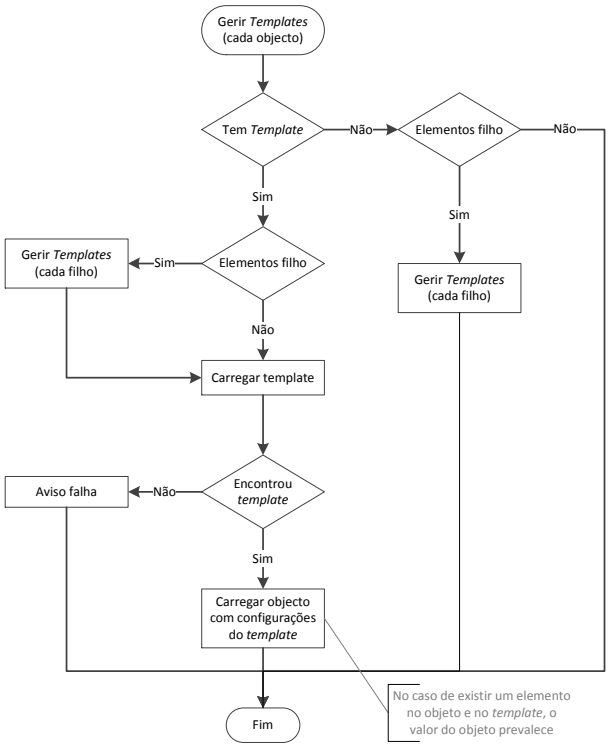


Figura 4.11: Processo para inserir as descrições completas no objeto que tem ligações com um *template*

lecem sobre as definições feita nos <templates>. Na Figura 4.10(b) o objeto <computer> com a identificação computer2 tem uma *application http/client*, não apresentada, mas incluída por estar este objeto associado a um *template*, e, contrariamente ao *template*, tem um endereço IP versão 4 estático, não recebendo por isso a configuração automática via o protocolo de configuração dinâmica de *hosts* (DHCP) associada ao *template*.

O processo de tradução do código com *templates* em descrições completas é feito recorrendo a uma implementação recursiva. Isto é possível devido à NSDL ser, essencialmente, uma descrição estruturada em árvore. Na Figura 4.11 está o fluxograma simplificado do processo aplicado a cada objeto para obter a descrição completa.

O processo aplica-se a todos os objetos e ao encontrar objetos com referência para os *templates* copia as características. Ao encontrar um objeto dentro de outro objeto, invoca para esse objeto novamente a função atual. O uso deste processo não é obrigatório e deverá ser usado apenas quando necessário. Uma descrição é sempre válida, quer seja na forma mais curta recorrendo a *templates*, quer seja na forma mais completa sem os *templates*. O processo apresentado serve para os objetos base apresentados neste capítulo e para uma qualquer extensão desses objetos.

A importância e a utilidade deste passo no processo de transformação entre uma descrição NSDL e qualquer outra descrição de cenários de rede está na simplificação obtida. Assim, um processo de transformação apenas necessita definir a transformação de cada objeto completo e não a transformação do objeto completo e do *template* do objeto. Como ambas as descrições, com e sem *templates*, são idênticas e existe um processo de tradução entre elas, a transformação apenas necessita ser implementada para um dos casos, normalmente a versão sem *templates*.

A desvantagem na utilização deste processo de tradução de *template* para o objeto está no tempo longo de processamento da descrição. O processo de tradução dos *templates* cobre todos os elementos de todos os objetos e, para cenários grandes, pode tornar o processo de transformação mais demorado.

A secção seguinte apresenta outra funcionalidade para ajudar a simplificar e reforçar as descrições NSDL, oo elemento <views>.

4.3.3 Elemento <views>

Um cenário de rede pode conter um número pequeno ou um número bastante elevado de objetos. Uma funcionalidade relevante na descrição de recursos de rede é a possibilidade de relacionar os objetos de uma forma independente às relações criadas na descrição da rede definida em *objects*. Como exemplo, podemos apontar numa rede os equipamentos e as ligações de uma certa área. Outro exemplo, é reunir os objetos existentes ao longo de um determinado caminho de rede. Existindo a possibilidade de criar um conjunto de objetos, as ações que se pretendam fazer sobre os objetos do conjunto podem ser declaradas apenas uma única vez, simplificando a descrição das ações. A atualização de um parâmetro, como por exemplo o atraso de uma ligação, pode ser declarado objeto a objeto ou, existindo um grupo com essas ligações, a alteração poderá ser aplicada sobre o grupo, reduzindo a declaração da ação para apenas uma linha, em vez de tantas linhas quantas ligações. Assim, o agrupamento de objetos de uma rede é uma funcionalidade importante e é conseguida através do elemento <views>.

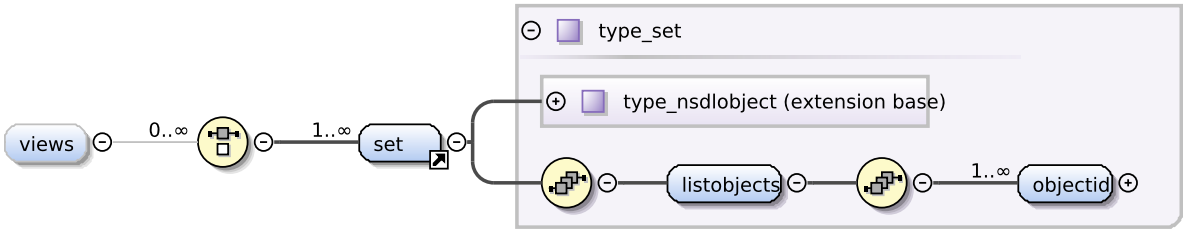
De notar que os exemplos apresentados referem-se a ações a incluir nos cenários (secção 4.4). A descrição da rede realizada no elemento <objects> apenas permite um valor para cada parâmetro, ou seja, no global apresenta o estado da rede num dado momento, normalmente, o estado inicial da rede. A Figura 4.12 apresenta a estrutura de suporte à criação de grupos numa descrição NSDL.

O elemento <views> permite a inclusão de qualquer número de conjuntos numa descrição. O elemento <set> é onde são indicados os objetos do conjunto e este elemento tem como base o objeto NSDL. Para identificar os objetos do grupo, o elemento <listobjects> serve como raiz para os elementos <objectid>, que contêm a referência para os objetos selecionados de <objects>.

O elemento <views> acrescenta informação para a organização da rede, no entanto, considerando apenas o contexto de <network>, não adiciona mais nenhuma informação. A sua principal utilidade surge na associação com os cenários, ou seja, com a utilização da rede num contexto particular. O exemplo descrito anteriormente mostra a simplificação da reconfiguração de parte de uma rede – os *links* – para, por exemplo, um cenário de gestão. Mas muitos outros cenários fazem uso desta facilidade, por exemplo:

- na simulação, além de reconfiguração de múltiplos objetos, podem-se usar, durante a execução de uma simulação, os grupos de objetos para a posterior análise e visualização dos resultados de forma agregada (apresentado no capítulo 5 e 6), e;
- na visualização, podemos construir apresentações gráficas das redes organizadas a partir dos grupos criados, por exemplo, colorir de forma idêntica objetos do mesmo grupo.

O elemento <views> apenas permite a identificação de grupos de objetos de rede, mas a sua utilização nos cenários pode ser muito variada, visando a simplificação de muitas ações a realizar



(a) Estrutura de <views>

```
1  ...
2  <xs:element name="views" minOccurs="0">
3     <xs:complexType>
4        <xs:choice maxOccurs="unbounded" minOccurs="0">
5           <xs:element ref="set" maxOccurs="unbounded"/>
6        </xs:choice>
7     </xs:complexType>
8  </xs:element>
9  ...
10 <xs:element name="set" type="type_set"/>
11 <xs:complexType name="type_set">
12    <xs:complexContent>
13       <xs:extension base="type_nsdlobject">
14          <xs:sequence>
15             <xs:element minOccurs="1" name="listobjects">
16                <xs:complexType>
17                   <xs:sequence>
18                      <xs:element name="objectid" maxOccurs="unbounded"/>
19                   </xs:sequence>
20                </xs:complexType>
21             </xs:element>
22          </xs:sequence>
23       </xs:extension>
24    </xs:complexContent>
25 </xs:complexType>
26 ...
```

(b) XML *Schema* da estrutura de <views>

Figura 4.12: Elemento <views> e sua definição de forma a permitir construir grupos de objetos

sobre a rede.

Com o elemento <views> concluiu-se a apresentação de <network>, onde é definida a rede com todos os seus objetos e propriedades. Resumidamente, em <network> podem ser incluídos objetos genéricos de rede usando <templates>, onde é descrita de forma completa a rede objeto a objeto, e, caso necessário, para definir grupos de objetos pode-se dispor do elemento <views>.

Na próxima secção é apresentado o componente complementar a <network> numa descrição NSDL, o elemento <scenarios>. Este elemento é usado para adicionar informação de contexto de utilização ou operação da rede.

4.4 Cenários

Os cenários de uma descrição NSDL são uma das principais contribuições deste trabalho. A adição de informação de contexto, e por contexto deve-se entender um ambiente de operação e/ou utilização da rede, é, na quase totalidade das linguagens apresentadas na secção 2.4, intercalada com a informação da rede. Nas linguagens onde a informação de rede é independente de outras informações, o âmbito de utilização é muito limitado.

As ferramentas incluídas nas categorias de *Modelação*, *Monitorização* e *Simulação*, e apresentadas na secção 2.3, são muito variadas em termos de funcionalidades e de objetivos, e, mais importante para este projeto, os formatos de dados de todas estas ferramentas, que realizam a descrição das redes e dos seus domínios, são muito diversos e, pelo menos de uma forma direta, incompatíveis. Não foi encontrada nenhuma ferramenta que partilhasse total ou parcialmente as suas informações de rede e existem apenas algumas ferramentas que contêm módulos para a exportação e/ou importação de parte das descrições de outras ferramentas, via conversão de dados.

As razões desta realidade podem dever-se a (1) foco de cada trabalho em problemas/domínios de rede específicos e desinteresse no trabalho realizado em outros domínios, e (2) pouco interesse em interligar ferramentas. Em geral, (1) a maioria dos trabalhos são feitos no âmbito de projetos académicos de mestrado e/ou doutoramento, levando na maioria dos casos a ciclos de vida do projeto bastantes curtos ou de utilização muito específica, como o J-Sim [134] ou o *TOolbox for Traffic Engineering Methods* (TOTEM) [151] para a Engenharia de Tráfego. Da mesma forma, também (2) existem as ferramentas de grande abrangência em termos de objetos e domínios de rede que, pela sua dimensão e complexidade, não tornam simples a definição de uma base de objetos e definições que seja reutilizável por outras ferramentas, como o *Network Simulator 2* [34] para a simulação e o *Network Mapper* [160] para a gestão de redes. Uma última razão, não explorada no âmbito deste projeto, pode ser o interesse em proteger uma certa ferramenta/formato, seja por fatores comerciais, seja pela não necessidade em alterar os métodos e formatos usados.

A atual dificuldade em reutilizar, ou interligar, as diferentes ferramentas de redes, devido à falta de um formato aceite de forma abrangente, também é identificada por alguns grupos de investigação [188, 211]. Os cenários em NSDL permitem agregar, sem limite, a informação de múltiplos ambientes de operação da rede, e, consequentemente, de ferramentas de rede, concedendo a todos eles uma clara e simples forma de referência sobre os objetos da rede. Nas secções seguintes é detalhada cada um dos seus elementos e explicada a sua estrutura.

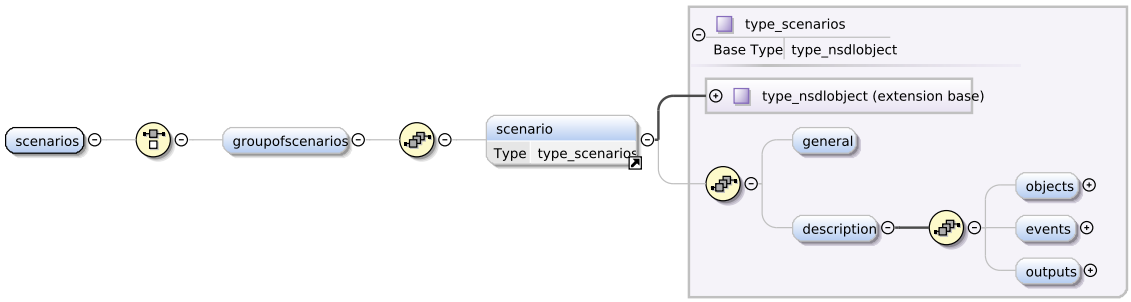


Figura 4.13: Estrutura base para a definição de cenários em NSDL

Tabela 4.5: Detalhe dos elementos que pertencem a uma estrutura genérica de <scenarios>

@Atributo ou Elemento	Obrigatório	Ocorre (0 até n)	Tipo Dados	Único	Observaçoes
groupofscenari	Não	0..n	Estrutura	Não	Nome do grupo de cenários, *simulations* ou *visualizations*
scenarios	Sim	0..n	Estrutura	Não	Nome do *scenario*
general	Sim	0..1	Estrutura	Sim	Configurações gerais do cenário
description	Sim	0..1	Estrutura	Sim	Configurações diversas sobre objetos da rede do cenário
objects	Não	0..1	Estrutura	Sim	Configurações particulares sobre os objetos
events	Não	0..1	Estrutura	Sim	Configuração de eventos para o cenário
outputs	Não	0..1	Estrutura	Sim	Configuração de resultados para o cenário

4.4.1 Estrutura base dos cenários NSDL

À semelhança de muitos outros componentes da linguagem NSDL, os cenários seguem uma estrutura semelhante de forma a apoiar a interoperabilidade da linguagem. A Figura 4.13 apresenta os elementos que constituem essa estrutura base genérica para a definição de todos os cenários específicos em NSDL.

Um qualquer cenário é um objeto e, portanto, herda as propriedades do objeto NSDL. Além do conjunto de elementos herdados, a restante estrutura de um cenário será, normalmente, muito distinta de um outro cenário. Por exemplo, os dados de configuração para a gestão de uma rede serão sempre muitos diferentes dos dados de configuração de uma simulação. Os cenários definidos no decorrer do projeto NSDL – visualização e simulação – não têm entre eles muitos elementos em comum e, assim, foi apenas definida uma estrutura base e alguns elementos adicionais que devem, sempre que possível, ser usados para garantir estruturas o mais semelhantes possível.

A tabela 4.5 apresenta os detalhes de todos os elementos presentes na estrutura apresentada na Figura 4.13. Alguns dos elementos, nomeadamente <objects>, <events> e <outputs>, têm uma estrutura própria, mas serão apresentados apenas nos cenários particulares em que participam.

Os termos *groupofscenarios* e *scenarios* não são os termos exatos. Devem ser substituídos pela designação escolhida para o grupo de cenários e para o cenário. Como exemplo, para um cenário de simulações foram escolhidos os termos *simulations*, para guardar as informações de todas as simulações de um cenário de rede; e, *simulation*, para guardar as informações de uma simulação em particular. Apesar de ser possível criar diversos grupos de cenários, um grupo particular apenas pode existir uma vez. Dentro do grupo não há número limite para o elemento <simulation>.

O elemento <general> contém as configurações genéricas do cenário. Excetuando as informações

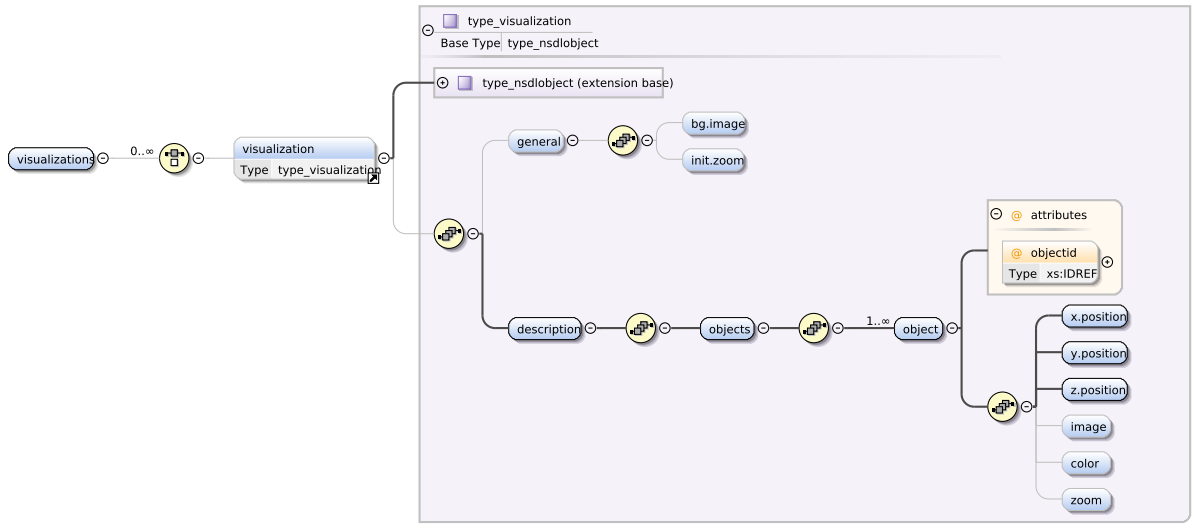


Figura 4.14: Estrutura de base para a descrição da visualização dos objetos e da rede

dos objetos da rede, que têm um elemento próprio nos cenários, o elemento <general> pode conter todo o tipo de configurações globais para a descrição do cenário. Um exemplo de um parâmetro existente neste elemento é o <duration>, pertencente ao cenário *simulation*, e que significa a duração de uma simulação. Refere-se a algo global ao cenário e nunca particular a um objeto ou grupo de objetos.

O elemento <description> recebe as configurações do cenário relacionadas com os objetos de rede. Os grupos definidos foram os <objects>, <events> e <outputs>. O <objects> visa permitir adicionar propriedades a um objeto qualquer da rede. Um exemplo de utilização é apresentado no cenário 4.4.2 e adiciona informações para a apresentação gráfica dos objetos e da rede. O elemento <events> permite incluir ações sobre a rede em tempos determinados. Como exemplo, pode ser a alteração de uma propriedade, do estado de um objeto ou espoletar uma ação. Já o elemento <outputs> permite definir filtros e formatos para os resultados de uma simulação ou execução sobre a rede. Os elementos <events> e <outputs> são detalhados no cenário 4.4.3.

Os elementos <groupofscenarios>, <scenarios>, <general> e <description> existem na descrição de qualquer cenário. Os elementos <objects>, <events> e <outputs>, bem como outros que possam vir a ser definidos, existem apenas nos cenários onde se aplicam. De forma a ilustrar a utilização do elemento <scenarios>, as secções 4.4.2 e 4.4.3 descrevem a implementação de dois grupos de cenários para a linguagem NSDL.

4.4.2 Visualização

O cenário *visualizations* foi criado com o objetivo de permitir a apresentação das redes e dos seus objetos no contexto de um ambiente gráfico. A informação guardada neste cenário permite posicionar no espaço e associar objetos gráficos a cada um dos objetos da rede. A Figura 4.14 apresenta a estrutura do cenário *visualization*.

A estrutura do cenário da Figura 4.14 segue a estrutura dos cenários e, assim, no elemento general estão os elementos gerais para a apresentação gráfica, como uma imagem de fundo (<bg.image>), e a distância da visualização (<init.zoom>). Em <description> existe a possibilidade de caraterizar

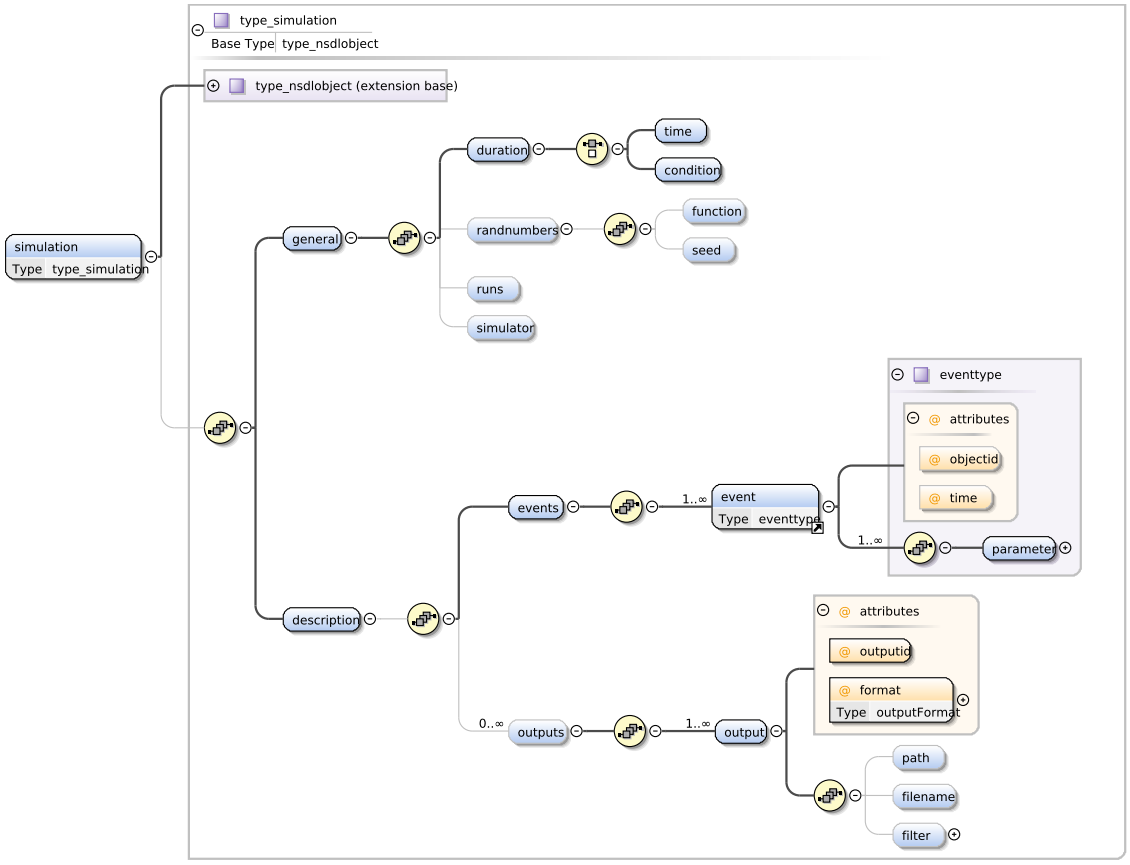


Figura 4.15: Estrutura base para a descrição de simulações de rede

graficamente cada objeto. Os elementos obrigatórios são as coordenadas em três dimensões presentes nos elementos <x.position>, <y.position> e <z.position>. Outras características permitidas para os objetos de rede são <image>, <color> e <zoom>, para associar a cada objeto uma imagem, uma cor e uma dimensão, respetivamente.

A estrutura definida pode ainda ser melhorada com a inclusão de outros elementos que caraterizem mais detalhadamente os objetos e que permitam *interfaces* amigáveis para a apresentação e compreensão da rede.

Na secção seguinte é apresentado um cenário de rede com um propósito diferente, a simulação de redes.

4.4.3 Simulação

O cenário *simulations*, como o nome indica, pretende conter as definições para suportar a descrição de simulações de redes. A descrição da simulação de uma rede deve conter os objetos da rede e permitir ao utilizador definir a forma como a rede irá operar. A operação da rede define-se através de um estado inicial e por alterações provocadas em momentos determinados de forma a obter a informação pretendida sobre o funcionamento da rede. O estado inicial é, essencialmente, definido no elemento <network>, descrito na secção 4.3, mas no cenário de simulação esse estado pode ser atualizado.

O elemento <simulation> segue a estrutura de base dos cenários em NSDL e, assim, contém

o elemento <general> com os dados gerais da simulação e outros elementos específicos para uma simulação, como os eventos e os resultados. A Figura 4.15 apresenta a estrutura base para a descrição de uma simulação em NSDL.

O elemento <general> contém informações gerais do cenário que, neste caso, são os dados para a configuração geral da simulação. Os elementos incluídos neste elemento foram escolhidos pela sua importância numa descrição qualquer de uma simulação. Assim, o primeiro elemento descrito é o <duration> que define o tempo de duração da simulação recorrendo, de forma simples, ao elemento <time> ou por intermédio da definição de uma condição de paragem incluída no elemento <condition>. O elemento <randnumbers> permite indicar uma função para geração de números aleatórios. Contém o elemento <function> e o valor da sua variável de base no elemento <seed>. O elemento <runs> permite alterar o número de execuções consecutivas da simulação. O último elemento de <general>, <simulator>, contém a designação do simulador a usar. Com exceção do elemento <duration>, que é obrigatório, todos os elementos são opcionais. Se esses valores não forem explicitados, os valores a serem considerados serão aqueles indicados por defeito em cada ferramenta. Diferentes extensões para múltiplos ambientes e ferramentas de simulação poderão obrigar à inclusão de um maior número de parâmetros gerais para uma simulação.

O elemento <events>, permite criar uma lista de eventos da simulação definida pelo utilizador. Os eventos do utilizador passam pela ativação ou desativação de objetos e pela alteração de parâmetros dos objetos. Como base, cada elemento <event> contém um apontador para um objeto, já definido em <network> (no atributo @objectid) e o momento da simulação em que o evento deverá ocorrer (no atributo @time). O evento pode tanto ser sobre um objeto único ou sobre um grupo de objetos, neste último caso recorrendo às <views>. A definição das características do evento é feita no elemento <parameter> e, mais precisamente, nos seus atributos @name e @value. Em @name é indicado o elemento que se pretende alterar, por exemplo o atraso de uma ligação ou mesmo a sua largura de banda. O atributo @value serve para conter o valor do parâmetro.

Além de objetos, @name também pode indicar uma ação genérica sobre o objeto e, para esse fim, deverá conter o termo '*action*'. Neste caso, @value pode apenas tomar os valores '*start*', '*stop*' e '*pause*'. Uma última possibilidade de utilização do elemento <parameter> é indicar '*script*' em @name e em @value descrever um conjunto de instruções a ser incluído na simulação. Este código permite a inclusão de ações muito variadas a ocorrer num dado momento da simulação. Como nota, os termos indicados entre ' ' pertencem a um conjunto de palavras reservadas, e estas não podem ser usadas na criação de objetos, seja nos elementos seja nos atributos.

O elemento <outputs> permite escolher quais os resultados a obter após a simulação e quais os seus respetivos formatos. São possíveis múltiplos resultados, cada um definido num elemento output, e na sua definição ocorre, primeiro, os atributos @outputid e @format a identificar, respetivamente, o grupo de dados e o seu formato. Os formatos possíveis são dependentes da ferramenta que será utilizada. Na sequência, estão os elementos <path> e <filename> indicando o local e o nome do ficheiro que irá conter os dados. Por último, o elemento <filter> permite descrever os filtros a serem aplicados aos dados extraídos da execução da simulação. Desta forma, permite-se a guarda dos dados referentes ao objeto, ou objetos por intermédio das <views>, que se pretende avaliar. O filtro permite apontar para um objeto e para parâmetros de um objeto. Como exemplo de ambas as situações, podemos obter todo o tráfego que passou num determinado *link* – objeto – ou podemos conhecer apenas o valor da taxa de transferência (*throughput*) ocorrido nesse mesmo

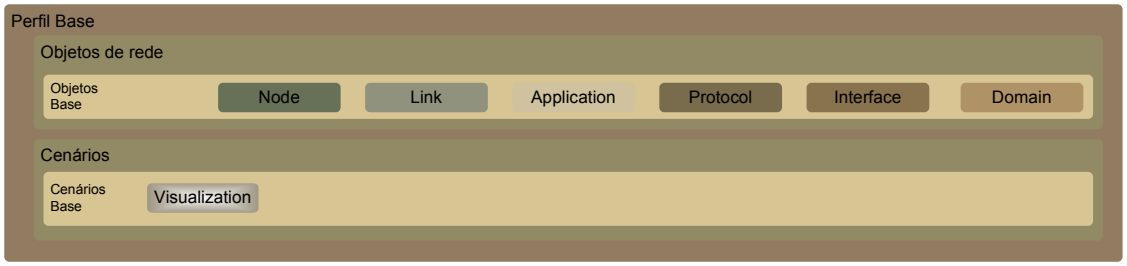


Figura 4.16: Perfil NSDL base com os objetos iniciais

link indicando para isso, além do objeto, o elemento <bandwitdh>.

A proposta da linguagem NSDL para descrever redes é feita através dos elementos <network> e <scenarios>. Sendo possível a extensão dos objetos existentes em cada um dos elementos, o número total de elementos a gerir pode-se tornar muito grande e variado. A gestão e a organização desse número crescente de elementos são conseguidas utilizando um último componente da linguagem e que é designado por perfil. A sua apresentação é feita na secção seguinte.

4.5 Perfis da linguagem NSDL

A descrição de redes usando o grupo de objetos definidos na base da linguagem NSDL é simples devido ao pequeno número de objetos e cenários. Através da experiência obtida na utilização da linguagem e da *framework* foi inicialmente constatado que a gestão do repositório de objetos definido no âmbito da linguagem era conseguida sem grande dificuldade. O alargamento da linguagem NSDL, possível através da criação de extensões aos objetos base e pela definição de novos cenários, tornou a gestão dos seus objetos mais complexa. Assim, para apoiar a organização dos objetos NSDL foi definido um novo componente que agrega um conjunto de objetos de rede e/ou de cenários relacionados, designado por 'perfil'.

Os perfis podem ser definidos com base em duas categorias: perfil de objetos e perfil de cenários. Um perfil de objetos, como o nome indica, serve apenas para conter os objetos de rede. O perfil de cenários é um perfil que contém um grupo de objetos de rede, já associado a um contexto de execução, ou seja, a pelo menos um cenário. Uma característica também presente nos perfis é a possibilidade de um perfil poder complementar outro perfil. Isto é, integrar outro perfil e, assim, simplificar a estruturação de um grande número de objetos.

Um exemplo da integração de perfis surge na criação de uma ferramenta e no desenvolvimento de novas versões. A primeira versão da ferramenta terá um perfil NSDL particular e com o aparecimento de novas versões são criados novos perfis, sempre integrando os anteriores o que simplifica a tarefa de definição de novos perfis. Havendo a reutilização dos perfis, existe também uma maior garantia de compatibilidade entre as diferentes versões da ferramenta. A Figura 4.16 apresenta o perfil base, e este é constituído pelos objetos base definidos no âmbito da linguagem NSDL e pelo cenário de visualização. Outros perfis poderão ser criados a partir do perfil base, mas poderão também ser criados perfis mais simples, por exemplo, com o node e o link apenas, que são os objetos mínimos para a descrição de cenários de rede mais simples.

Uma vantagem importante obtida com os perfis é o reforço dos princípios da linguagem, princi-

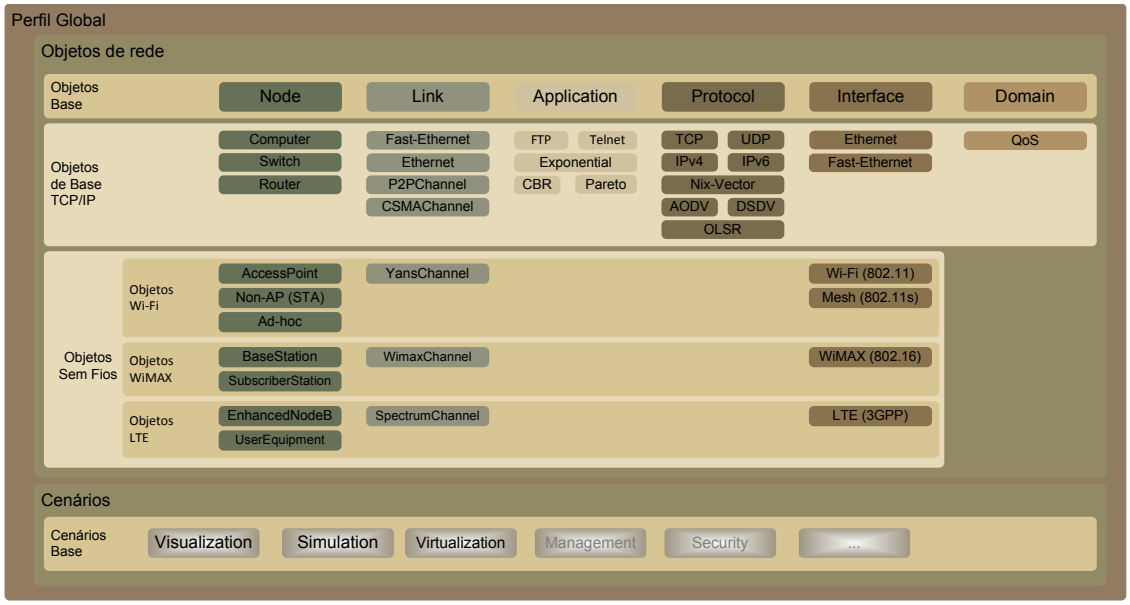


Figura 4.17: Perfil com grande parte dos objetos já definidos para a linguagem NSDL

palmente a simplicidade e a interoperabilidade. Um utilizador da *framework* ao adicionar uma nova ferramenta pode criar todos os objetos e cenários necessários de raiz, ou seja, poderá definir novos objetos de rede e seus respetivos cenários. Caso encontre um perfil já definido e adequado (parcialmente ou integralmente) às suas necessidades, poderá utilizá-lo completamente ou então integrá-lo no seu novo perfil. Em ambas as situações, as descrições de cenários de rede que construir com base nesse perfil terão de forma completa, ou pelo menos de forma parcial, a possibilidade de serem utilizados em diversas ferramentas, conseguindo, deste modo, um elevado grau de compatibilidade entre as ferramentas. O código completo das definições dos objetos que foram apresentados neste capítulo, e que constituem o perfil base, pode ser consultado no Apêndice A.

A Figura 4.17 apresenta muitas extensões de objetos já criadas no âmbito da linguagem NSDL e, além dos cenários já referidos neste capítulo, *simulation* e *visualization*, também identifica outros cenários previstos no âmbito da *Framework* NSDL.

Os perfis revelaram-se essenciais para a *Framework* NSDL pelo suporte providenciado na definição de uma organização clara dos seus objetos e por facilitarem a integração de novos cenários. A gestão das extensões da *framework* seria muito trabalhosa sem um mecanismo como o perfil. Outra consequência da sua inexistência seria o aparecimento de múltiplos objetos similares que sem a possibilidade de os relacionar, comprometeria a interoperabilidade entre ferramentas.

Com os perfis finda a parte do capítulo reservado à apresentação da linguagem NSDL e dos seus componentes. Seguindo a própria estrutura da linguagem foram identificados: primeiro, os objetos de rede; depois, os cenários de rede criados; terminando a apresentação com os perfis da linguagem. Na próxima secção serão apresentadas as conclusões sobre a linguagem NSDL.

4.6 Conclusões

A principal limitação encontrada nas linguagens de descrição existentes foi a dificuldade em estas serem reutilizadas entre diferentes ferramentas e/ou domínio de rede. A proposta de uma nova

linguagem, a NSDL, foi a opção considerada adequada para facilitar a interoperabilidade entre ferramentas de rede. Um princípio seguido na linguagem NSDL foi a existência de dois elementos para organizar a informação de um cenário de rede. A topologia de rede e as características dos objetos de rede são colocados sob <network>. As informações sobre uma ou várias determinadas perspetivas ou contextos de gestão e operação da rede são colocadas em <scenarios>.

A separação criada entre os elementos <network> e <scenarios>, mais do que uma forma simples de organização, visa garantir uma melhor interoperabilidade entre ferramentas de rede ao nível da inclusão e construção de cenários de rede. Os objetos de rede definidos procuram ser simples e relacionados com os componentes de rede reais e, assim, deverão ser facilmente interpretados por uma qualquer ferramenta de rede ou utilizador. Portanto, numa primeira abordagem, procurou-se atingir uma interoperabilidade o mais extensa e alargada possível. A descrição detalhada de um qualquer objeto ou domínio rede é também possível através da definição de novos objetos e/ou da construção de extensões dos objetos já existentes. Esta deverá ser feita com algum cuidado para não comprometer e dificultar a interoperabilidade.

Os cenários serão responsáveis pela descrição de diferentes características, como o ambiente envolvente da rede, das interações com os utilizadores e de outros requisitos ou informações que se pretendam sobre essa rede. A consulta e edição de cenários de rede específicos apenas será possível por ferramentas desses domínios e/ou ferramentas com propósitos semelhantes. A separação das estruturas tem também como vantagem a flexibilidade, isto porque permite a criação e a manutenção de diferentes descrições de cenários para a mesma rede.

A linguagem NSDL, além da estrutura com todos os objetos e cenários apresentados, inclui ainda um último componente para a organização de grupos de elementos NSDL: o perfil. Surge como mais um componente para reforçar a interoperabilidade. O perfil simplifica a tarefa inicial de utilização da NSDL e permite guiar o crescimento da linguagem de uma forma mais controlada, orientando os utilizadores para a reutilização de objetos e cenários.

A linguagem NSDL integra muitas das características já presentes em outras linguagens. Os principais objetos e características não são elementos inovadores, mas a divisão proposta para o cenário de rede e alguns dos objetos, como o domínio e, com algumas exceções, o *interface*, não existem nas outras linguagens. O foco deste trabalho foi a interoperabilidade entre ferramentas de rede e, para atingir esse fim, as descrições dos objetos de rede são independentes de uma qualquer ferramenta, algo também não observado nas linguagens estudadas. Outros componentes da linguagem, como os *templates*, *views* e perfis, facilitam a utilização da linguagem na construção de cenários complexos e variados. Espera-se assim conseguir uma linguagem simples, flexível e abrangente na descrição de redes de comunicação.

O resto desta tese apresenta algumas das implementações e testes feitos à linguagem NSDL. Os capítulos 5 e 6 apresentam a extensão da linguagem, e da *Framework NSDL*, para os simuladores *Network Simulator 2* e *Network Simulator 3*, respetivamente. O capítulo 7 estende a linguagem para os ambientes virtuais, permitindo a realização automática de simulações de rede recorrendo a máquinas virtuais.

Capítulo 5

Implementação NSDL de Redes com Qualidade de Serviço com NS2

5.1 Introdução

A primeira aplicação integrada da *Framework* NSDL foi o simulador de redes *Network Simulator 2* (NS2) e a sua escolha deveu-se, principalmente, a este ser um dos simuladores de rede mais usados para o teste e para a validação de novos componentes para as redes de dados pelas comunidades de investigação e desenvolvimento da área das redes. Além da sua grande utilização, uma segunda razão para a sua escolha foi também procurar utilizar a *framework* para ultrapassar algumas das suas limitações. Uma das limitações detetada foi a dificuldade em ser utilizado por utilizadores pouco experientes, sobretudo para a especificação de cenários de redes com Qualidade de Serviço (QoS), mais especificadamente para a arquitetura de Serviços Diferenciados (DiffServ). Assim, o trabalho apresentado neste capítulo detalha o processo de integração do NS2 na *Framework* NSDL, dando uma atenção particular aos objetos da arquitetura DiffServ.

A organização do resto do capítulo é detalhada de seguida. A secção 5.2 apresenta as principais características e aspetos da realização de simulações de redes no NS2. Na secção 5.3 é apresentada uma visão geral da arquitetura DiffServ e os seus componentes principais. É feita ainda uma breve comparação com outra arquitetura, os Serviços Integrados, e são apresentadas as principais vantagens e desvantagens da sua utilização. Segue-se, na secção 5.4, a apresentação do detalhe da implementação dos *routers* de borda e de núcleo para a arquitetura DiffServ no simulador NS2. A definição em NSDL dos demais objetos da arquitetura de DiffServ é então detalhada na secção 5.5, apresentando a relação entre a especificação dos objetos DiffServ em NS2 e os objetos suportados pela *Framework* NSDL. A transformação dos cenários definidos em NSDL para a linguagem utilizada pelo NS2 é apresentada na secção 5.6. Nesta secção é aprofundada a estrutura de cada uma das linguagens e mostrado como estas se relacionam. Os testes feitos à implementação e à utilização do NS2 com a *Framework* NSDL por um grupo de utilizadores são descritos na secção 5.7 e são apresentados os principais resultados dessa utilização. Na secção 5.8 são apresentadas as conclusões deste capítulo, destacando os principais resultados obtidos do desenvolvimento do perfil NS2.

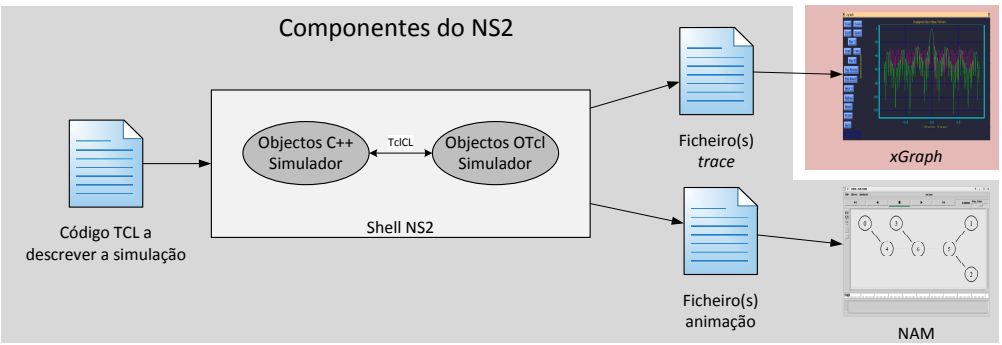


Figura 5.1: Arquitetura do *Network Simulator 2*

5.2 *Network Simulator 2*

O *Network Simulator 2* (NS2) é um simulador de redes orientados ao objeto e de eventos discretos. Surgem em 1989 as primeiras versões do NS2 como variante do simulador REAL [136] e, em 1995, sob o projeto *Virtual InterNetwork Testbed* (VINT) da *Defense Advanced Research Projects Agency* (DARPA) são feitos os maiores desenvolvimentos do seu núcleo e os principais modelos de rede [34]. A DARPA mantém a gestão do desenvolvimento e atualização do simulador, mas é graças a uma comunidade grande de utilizadores e contribuidores que tornam o NS2 num dos simuladores mais utilizados para a investigação e desenvolvimento de tecnologias de rede [34, 146]. Além de ser usado principalmente nas áreas referidas, o NS2 é também usado no ensino de redes de computadores [45, 203, 220].

O NS2 foi construído de forma a permitir a simulação dos mais variados tipos de redes. Cobre um grande número de aplicações, de tipos de redes, de objetos de rede e de modelos de tráfego. De entre muitas possibilidades, as simulações no NS2 podem incluir, por exemplo, os seguintes elementos [15]:

- Os protocolos de transporte TCP, UDP e RTP;
- A geração de comportamentos de tráfego FTP, *Telnet*, Web, CBR e VBR;
- Os mecanismos de gestão de fila *DropTail*, RED e CBQ;
- O algoritmo de encaminhamento *Dijkstra*;
- Os algoritmos de agendamento *Fair Queuing*, *Deficit Round Robin* e *First-in First-out*;
- As redes com fios e as redes sem fios (locais e satélite);
- O *multicast* e protocolos da camada MAC para Local Area Networks, e;
- Diversas tecnologias como o GRPS, o IPv6 móvel, o RSRV, o MPLS e as redes *Ad-hoc*.

A lista não está completa e muitos outros exemplos de objetos e tecnologias de rede podem ser simulados. A Figura 5.1 apresenta a arquitetura do NS2 com os seus principais componentes. O código *OTcl* contém todos os objetos de rede e as suas propriedades, as relações entre eles e a configuração dos eventos da simulação. O núcleo do NS2 (*Shell* NS2) interpreta o código *OTcl*

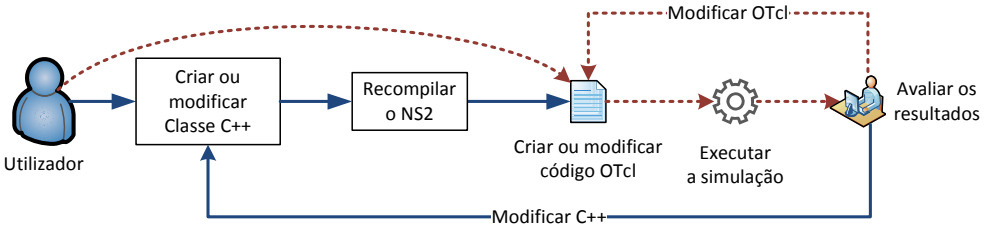


Figura 5.2: Processo de utilização do NS2, adaptado de [220]

e executa a simulação produzindo resultados em diversos formatos. Os formatos implementados permitem a validação e avaliação da simulação através de dois formatos:

- O ficheiro *trace*, contendo o registo de todos os eventos ocorridos durante a simulação. A análise deste ficheiro permite obter estatísticas variadas e a construção de gráficos para a visualização dos mais variados aspetos da simulação. Este último é exemplificado com o componente xGraph [281] na Figura 5.1, utilitário Unix/Linux externo ao NS2 mas utilizado no âmbito deste, e;

- O ficheiro animação, contém a informação dos eventos que permitem a apresentação da simulação sob a forma de uma animação na ferramenta *Network ANimator* (NAM) [80].

O NS2 utiliza duas linguagens para a construção de simulações: o C++ e o *Object-oriented Tool Command Language* (OTcl) [279]. Ambas são orientadas ao objeto, mas têm no NS2 papéis distintos. O C++ é uma linguagem que necessita ser compilada (lento), mas a sua execução é rápida. O *OTcl* é uma linguagem interpretada (flexível), mas mais lenta que o C++. O uso equilibrado das duas linguagens permite a construção eficiente de simulações no NS2. A execução de simulações com muitos objetos obriga a ter em consideração a linguagem C++ por questões de desempenho. A execução de simulações mais simples pode ser descrita apenas recorrendo ao *OTcl*, permitindo muitas alterações para cada repetição da simulação [127].

O C++ é usado no núcleo do NS2 para a escrita do conjunto de primitivas de simulação que necessitam de grande desempenho, como o agendamento dos eventos e os componentes de rede base, de forma a reduzir os tempos de processamento dos pacotes e dos eventos. Os objetos compilados a partir do código C++ são ligados ao interpretador *OTcl* por intermédio da criação de objetos *OTcl* idênticos, tendo as mesmas funções e variáveis oferecidas pelo objeto C++. Assim, o controlo dos objetos C++ é feito pelos objetos *OTcl*. A Figura 5.1 apresenta a ligação entre os objetos C++ e *OTcl*, concretizada na prática pela linguagem *TCL with CLasses* (TclCL). Existe ainda a possibilidade de existirem objetos C++ e *OTcl* sem qualquer ligação. Os primeiros devido a não ser necessário o seu controlo na simulação. Os segundos por iniciativa do utilizador para criarem um componente não existente no núcleo do simulador.

A Figura 5.2 apresenta o processo de utilização do NS2 para o desenvolvimento de simulações. O processo mais curto, indicado a linha tracejada na Figura 5.2, é o processo utilizado para a realização de simulações com os objetos já disponíveis no simulador. Os utilizadores criam um *script OTcl*, executam a simulação e avaliam os resultados. Poderão, se necessário, retificar o *script OTcl* e repetir o processo até cumprirem os objetivos da simulação.

O processo mais longo apresentado na Figura 5.2 adiciona a criação dos novos objetos C++ (ou novos objetos de rede) e a recompilação do NS2. Após a realização do resto do processo até à avaliação dos resultados, o utilizador poderá apenas retificar o *OTcl* ou reiniciará o processo modificando a classe C++.

A vantagem principal do NS2 é sua grande base de utilizadores e que, em muitos casos, contribuíram com novos objetos para o simulador. Este fato deu ao NS2 a possibilidade de simular muitos tipos de redes e, consequentemente, um maior interesse por parte de outros utilizadores. A opção de duas linguagens, apesar de tornar mais complexa a sua utilização para novos utilizadores, não impediu a sua utilização na maioria dos trabalhos de investigação publicados, por exemplo, nas atas da conferência *ACM International Symposium on Mobile Ad Hoc Networking and Computing* (MobiHoc) [146].

O NS2 tem ainda capacidades de emulação, ou seja, o simulador pode ser integrado e interagir com uma rede real [144, 163] e existem também alguns projetos para o desenvolvimento de componentes para a implementação de simulações paralelas e distribuídas com o NS2 [226, 229, 282].

5.3 Arquitetura de Serviços Diferenciados

As redes de dados construídas nas últimas décadas proporcionaram, fundamentalmente, aos seus utilizadores um serviço padrão, cujo característica principal era a preocupação na entrega correta dos dados, usando para isso uma ou várias rotas de entre as disponíveis entre o emissor e receptor dos dados e reenvio de informação em caso de perdas ou erros. O protocolo de transporte TCP (*Transport Control Protocol*) [217] é um dos principais exemplos da implementação desse conceito. O serviço disponibilizado para os utilizadores destas redes ficou assim conhecido como 'melhor-esforço', do inglês *best-effort*. Dessa forma, os dados a transmitir são entregues de uma forma fiável e o mais rapidamente possível, mas, em termos práticos, sem qualquer garantia concreta em termos de largura de banda, latência, taxa de erros e/ou de perdas.

A entrega fiável dos dados é suficiente para as aplicações tradicionais, como o e-mail, a transferência de ficheiros e/ou o *Telnet*, mas, para aplicações com requisitos temporais, como as emissões e as conferências de áudio e vídeo, tal não é suficiente. Estas aplicações são mais exigentes em termos de largura de banda e mais sensíveis em termos latência e taxa de perdas. Assim, emergem na década de 90 duas arquiteturas para a Qualidade de Serviço (QoS) em rede de comunicação, com princípios distintos, que, sobre as redes de dados tradicionais, implementam mecanismos para a melhoria do serviço dado aos utilizadores, principalmente na garantia de um tratamento mais previsível para os seus fluxos de dados.

A primeira arquitetura de QoS designou-se por Serviços Integrados (IntServ) [32] e utiliza o princípio da reserva dos recursos, ou seja, anterior ao envio dos dados, existe um processo de reserva de recursos em todos os equipamentos existentes na rota, tendo por base as características do fluxo a enviar e os requisitos do serviço e/ou utilizador. O processo de reserva utiliza um protocolo de sinalização para o efeito e, associado aos IntServ, surge o protocolo *Resource Reservation Protocol* (RSVP) [33] para realizar essa tarefa. No caso do processo de reserva terminar em sucesso, os dados são então enviados pela rota reservada, esperando obter da rede o tratamento pedido em termos de largura de banda e atraso. No caso de, pelo menos, um equipamento da rota não confirmar a reserva, então o processo falha e os dados apenas podem ser enviados numa base de *best-effort*.

A arquitetura IntServ adicionou ao tráfego a circular nas redes de dados uma característica nova, além da entrega fiável: a possibilidade de este circular na rede com garantias em termos de largura de banda, de atraso e de perdas. Outras vantagens desta arquitetura são [165]:

- A sua flexibilidade, permitindo a diferentes fluxos com diferentes requisitos poderem ter tratamentos distintos ao atravessar a rede;
- Darem aos operadores das redes um mecanismo que permitia oferecer aos clientes vários níveis de serviço e, com isso, aumentarem as suas receitas, e;
- Suportarem um modelo lado-a-lado (*end-to-end*), onde as aplicações podem requisitar um serviço à rede para funcionarem de forma correta.

A grande limitação dos IntServ é a sua escalabilidade, ou seja, a reserva de alguns fluxos é facilmente conseguida mas, num ambiente como a Internet, onde existe um grande número de fluxos, a gestão das reservas torna-se muito complexa. A tarefa de manter a informação de estado de cada uma das reservas é crescente com o aumento de fluxos, exigindo ao nó uma maior quantidade de recursos, em termos de desempenho e de quantidade de memória. Em redes locais o uso dos IntServ é considerado viável, mas na Internet, pela sua dimensão e heterogeneidade, tal não é possível.

A segunda arquitetura para a QoS são os Serviços Diferenciados (DiffServ) [28]. A sua característica principal é a utilização de mecanismos de provisionamento nas redes para a gestão de um conjunto limitado de fluxos, designados por classes de tráfego. Ao contrário de IntServ, que é baseada na reserva dos recursos de rede para cada fluxo de dados, DiffServ prepara os recursos da rede para o suporte a apenas algumas classes de tráfego e garantem a marcação dos fluxos individuais de dados na sua origem ou na entrada do domínio de rede DiffServ de forma a pertencerem a uma das classes suportadas. Ou seja, em DiffServ o tratamento é feito por grupo de fluxos, ou classe, e não por fluxo individual, como em IntServ.

Uma razão forte para o aparecimento de DiffServ foi de forma a colmatar a limitação de escalabilidade presente em IntServ em redes de grande dimensão ou com um grande número de fluxos de dados. Os recursos necessários para gerir um número pequeno de classes de tráfego são muito menores que no caso de IntServ e tornam DiffServ uma solução adequada para redes de grande dimensão, como a Internet. Outras tarefas, mais complexas e exigentes, como classificação e policiamento da utilização da rede, estão implementadas, mas apenas nos limites (borda) das redes. A maior simplicidade em cada nó do interior da rede visa um encaminhamento rápido, obtendo-se, previsivelmente, um melhor desempenho da rede, comparativamente a IntServ.

O tratamento dado ao tráfego de cada classe em cada nó é designado por *Per-Hop Behaviour* (*phb*). A utilização dos *phb*'s permite a inexistência de estados para cada fluxo de dados individual. Cada pacote que chega ao nós é identificado por um código de 6 bits colocado no campo *Type of Service* do cabeçalho *Internet Protocol* (IP) que se designa por *DiffServ Codepoint* (DSCP) [186]. Por consulta ao DSCP no pacote, o *phb*'s é identificado é o pacote é direcionado para a fila respetiva onde será tratado da forma que estiver definida para a sua classe. As propriedades que constituem um dado tratamento são a prioridade (relativa a outros *phb*'s) e/ou atraso e probabilidade de descarte em caso de congestão da rede. Os mecanismos utilizados são as filas, ou dito de outra forma, a gestão das filas e uso de políticas diversas para o agendamento de pacotes.

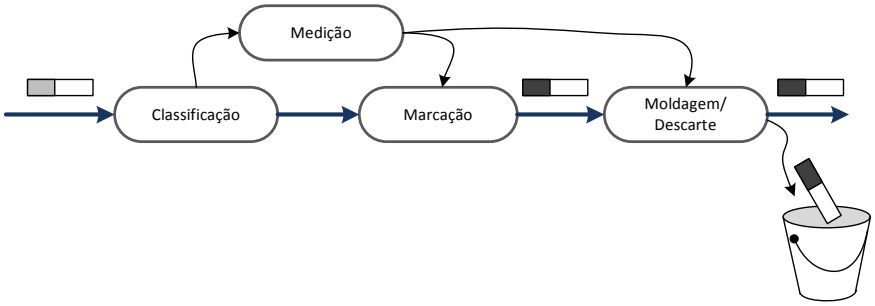


Figura 5.3: Componentes do condicionamento de tráfego à entrada de um domínio DiffServ

Os componentes, ou mecanismos, pertencentes à arquitetura DiffServ podem ser agrupados em dois grupos, estando cada grupo relacionado com um conjunto de nós. Assim, as funcionalidades de identificação/classificação e condicionamento dos fluxos ocorrem nos limites da rede e são da responsabilidade dos nós de entrada e saída da rede, designados por nós de borda. As funções de encaminhamento de cada pacote baseado no tratamento definido para a sua classe a são implementadas nos nós interiores da rede ou nós de núcleo. Nas secções seguintes são apresentadas as funções de cada um dos nós.

5.3.1 Funções dos nós de borda

Os nós de borda estão nos limites da rede e referem-se aos nós onde os fluxos de dados dão entrada e/ou saída da rede. Estes conceitos de entrada e saída dependem da existência de um sentido corrente para o fluxo de dados, mas se tal não acontecer, então os significados são dinâmicos, ou seja, um nó pode ser entrada na rede para um fluxo e ser saída de rede para outro fluxo. As funções principais e que serão apresentadas no âmbito de DiffServ são funções na entrada da rede, e como a distinção entre o papel de entrada e saída não é fundamental, por razões de clareza, serão identificados, simplesmente, como nós de borda.

É responsabilidade dos nós de borda realizar o condicionamento do tráfego que entra na rede. O condicionamento passa por aceitar e marcar (ou não) o tráfego que entra na rede e por ajustá-lo aos contratos de serviço definidos previamente. A concretização do condicionamento tem como primeira função a classificação de cada pacote num determinado agregado. Esta classificação baseia-se num conjunto de cinco componentes presentes nos cabeçalhos dos protocolos TCP e IP: os endereços de origem e destino, as portas de origem e destino e o protocolo; este conjunto é referido como 5-*tuples*. De seguida é invocada a função de medida que mantém informação de tráfego para cada fluxo e determina se o pacote atual está dentro dos parâmetros definidos no *Service Level Agreement* (SLA)[1], considerando-o então '*in-profile*', ou, caso contrário, '*out-of-profile*'. A marcação é então realizada atribuindo o DSCP adequado ao pacote. No caso do pacote já estar marcado pode saltar a marcação ou, devido a estar '*out-of-profile*', pode ser remarcado. Por último, o pacote poderá ser moldado, através de tampões – *buffers* – de forma a alcançar a taxa definida no SLA, ou mesmo ser descartado. Estes dois casos ocorrem com diferentes probabilidades e apenas nos momentos de congestão na rede [25]. A Figura 5.3 apresenta os componentes descritos.

[1] O SLA é um contrato entre um cliente e um fornecedor de serviços e, no contexto dos DiffServ, especifica o tipo de tratamento recebido pelos fluxos de tráfego do cliente ao passarem pela rede do fornecedor de serviços.

Neste momento, os pacotes entram no núcleo da rede onde receberá o tratamento adequado ao seu agregado.

5.3.2 Funções dos nós de núcleo

O núcleo da rede deverá ser eficiente no encaminhamento dos pacotes que lá transitam e, assim, cada salto – ou *hop* – de um nó para o seguinte deverá ser feito no menor tempo possível. A avaliação de cada pacote ocorre pela consulta ao DSCP e, por esse facto, considera-se muito mais expedito que a consulta aos 5-*tuples* realizada no caso de IntServ.

A cada DSCP corresponde um *phb*, significando uma fila e uma prioridade específica em relação aos restantes agregados de fluxos. Esta diferença nos recursos de cada nó de núcleo concretiza diferentes tratamentos para cada pacote a circular na rede, em termos de largura de banda, atraso, variação do atraso e perdas. O número possível de *phb*'s é grande e, de forma a coordenar a sua implementação em muitos nós de núcleo, duas classes de serviços foram normalizadas: os Serviços Expeditos (EF, de *Expedited Forwarding*) [65] e os Serviços Assegurados (AF, de *Assured Forwarding*) [107].

Os AF pretendem fornecer uma gama variada de serviços melhores que o comum *best-effort*, ou seja, vários níveis de atraso, variação do atraso e perdas, adequados a diferentes necessidades e, normalmente, surge associado a tráfego mais elástico, ou seja, a um tráfego que tolera algum atraso e/ou perdas.

Os EF pretendem suportar os fluxos que requerem serviços com um nível baixo de perdas, atrasos mínimos e com pequenas variações no atraso [14]. Um exemplo de aplicação dos EF é no transporte de dados em tempo real, como uma conferência áudio. A gama de valores possíveis para os AF é bastante maior que para os EF.

O tráfego *best-effort* não terá qualquer tratamento específico e ser-lhe-á aplicado um tratamento mais simples e designado como *Default Forwarding* (DF). Poderá ser-lhe garantida alguma largura de banda, mas os pacotes desta classe poderão ser perdidos, reordenados, duplicados ou atrasados aleatoriamente.

A configuração dos *phb*'s de um domínio poderá ser feita de duas formas: estática e dinâmica. No caso da configuração estática, a rede é planeada com base em padrões de tráfego previstos e, em cada nó, são definidos os *phb*'s adequados. É um método simples, mas pouco flexível, sendo pouco adequado para redes com padrões de tráfego variáveis e imprevistos. A configuração dinâmica baseia-se na utilização de técnicas de gestão da rede que monitorizam e atualizam os *phb*'s de forma a manter a rede num nível de uso próximo dos seus objetivos. O trabalho apresentado em [187] é um exemplo de uma implementação de uma arquitetura de gestão de um domínio DiffServ recorrendo a uma entidade central com funções de administração, designada por *Bandwitdh Broker* (BB). O BB contém, sobre a rede, as políticas de funcionamento e o registo dos níveis de utilização de cada um dos fluxos e, na recepção de novos pedidos de tráfego, reorganiza a rede, respeitando as políticas e os fluxos existentes.

5.3.3 Vantagens e desvantagens

A arquitetura DiffServ funciona no princípio de provisionamento da rede com base nos dados estatísticos previstos ou atuais da rede e na implementação de tratamentos diferenciados em termos

relativos, ou seja, um fluxo será mais prioritário e receberá um melhor tratamento que outro fluxo, no caso de congestão da rede. As principais vantagens deste mecanismo de gestão de uma rede são:

- **Escalabilidade**: no facto de ser apenas necessário em qualquer nó da rede manter um conjunto limitado de informações, ou seja, o estado de cada uma das classes e não o estado de cada fluxo. Esta característica permite que DiffServ possa ser implementada em redes de grande dimensão, como a Internet;
- **Otimização da rede**: a agregação de tráfego permite uma utilização mais eficiente da rede, permitindo mais fluxos e uma maior utilização dos recursos, e;
- **Rapidez**: o envio de dados pode-se iniciar de imediato, não requerendo um processo de admissão. Após a configuração do policiamento no nó de borda, para que este reconheça o fluxo, todas as sessões de dados desse fluxo podem ocorrer sem atrasos.

As vantagens de DiffServ são, em grande parte, as desvantagens de IntServ e é normal que assim seja visto que foram desenhados, essencialmente, para oferecer um serviço para os contextos onde IntServ não é uma solução adequada. No entanto, DiffServ também tem algumas desvantagens que impedem uma utilização mais alargada desta arquitetura e que são:

- **Sem garantias rígidas**: pode oferecer garantias estatísticas, mas difícil de garantir um certo valor para o atraso ou perda no caso de uma grande congestão na rede a um qualquer fluxo;
- **Pouco flexível**: a diferenciação de tráfego está sempre limitada a um determinado número de classes, e esse facto poderá ser relevante em alguns contextos, e;
- **Sem suporte lado-a-lado (*end-to-end*)**: é uma arquitetura adequada para as redes de núcleo, onde se realiza a gestão de fluxos agregados, mas que não suporta o serviço entre a origem e o destino do fluxo, fundamental para algumas aplicações exigentes em termos um ou vários parâmetros de rede (largura de banda e/ou atraso, por exemplo).

DiffServ, foco principal desta secção, surge então como uma opção mais adequada em termos de escalabilidade e otimização da rede, mas menos apropriado no tratamento de fluxos individuais, em ambos os casos relativamente à IntServ. De forma complementar a esta secção, serão apresentadas, brevemente, mais alguns tópicos relevantes na área de QoS.

Uma tecnologia muito importante, e posterior às duas arquiteturas já referidas, para a QoS é o *Multi Protocol Label Switch* (MPLS) [234] e tem como objetivo o encaminhamento rápido dos pacotes surgindo como uma convergência entre o ambiente sem ligação do IP e os circuitos virtuais da tecnologia *Asynchronous Transfer Mode* (ATM) [275]. O MPLS oferece aos ambientes IP um paradigma mais simples de encaminhamento, por intermédio da adição de um cabeçalho MPLS aos pacotes IP, e, mecanismos para o estabelecimento de caminhos explícitos na rede, com o propósito de suportar serviços mais complexos e com requisitos variados. O MPLS surge como uma tecnologia fundamental para a aplicação de conceitos avançados para a gestão de redes, designados como Engenharia de Tráfego (TE, de *Traffic Engeneering*). A TE tem como propósito a aplicação de princípios científicos e tecnológicos para a medição, caraterização, modelação e controlo de tráfego [12]. O balanceamento do tráfego por diferentes caminhos como forma de utilização otimizada de uma rede é um exemplo do uso das tecnologias MPLS e TE.

A existência de cenários de rede com requisitos mais complexos levou à necessidade, e ao aparecimento, de diferentes abordagens para o problema da QoS. Os trabalhos [26, 85, 86], através de abordagens híbridas, juntam algumas das arquiteturas e tecnologias para soluções mais flexíveis e ajustadas a diferentes cenários, mas também com um grau crescente de complexidade. Em [165] as diversas opções são apresentadas e é realizada uma análise comparativa. Além das redes com fios, também são apresentadas contribuições para a QoS no domínio das redes de sensores sem-fios.

Como citado anteriormente, uma vez que o NS2 já vem sendo utilizado amplamente tanto na academia como na indústria [31, 182, 282], sobretudo em relação à implementação de cenários de rede DiffServ, a escolha desta ferramenta veio ao encontro da necessidade de validação da proposta da *Framework* NSDL dada a maturidade de NS2 e dos cenários existentes para os testes realizados com essa *framework*.

5.4 Serviços Diferenciados no *Network Simulator 2*

Nesta secção é realizada uma breve apresentação dos componentes de NS2 que implementam a arquitetura DiffServ.

A implementação dos componentes principais de DiffServ no NS2 foi realizada pela companhia *Nortel Networks* (NN) [164]. O objeto escolhido para a execução das funcionalidades da arquitetura foi a fila (*Queue*), associado às ligações (*link*'s) entre os nós. As políticas de fila disponíveis são o *DropTail* [113], *Class Based Queueing* (CBQ) [162] e *Random Early Detection* (RED) [87], sendo a última a base escolhida para construção de um mecanismo que suportasse múltiplas filas físicas, filas virtuais e as diversas configurações necessárias para a diferenciação de tráfego.

Assim, nesta secção será introduzido primeiro o RED para a gestão das filas. De seguida serão discutidas as funcionalidades associadas aos nós de borda e, de seguida, as funções dos nós de núcleo.

5.4.1 Gestão das Filas

Os algoritmos para a gestão de uma fila de pacotes são variados e, no NS2 existem diversas opções. No caso da implementação dos DiffServ, a NN optou por utilizar o *Random Early Detection* (RED) [87] por o considerar ajustado para a implementações dos mecanismos de prioridade e descarte dos AF.

O RED é um algoritmo para a minimização da congestão em nós da rede, mais precisamente em *routers*. É um substituto ao algoritmo *Drop Tail*, cujo funcionamento se baseia em aceitar pacotes enquanto tiver recursos e a proceder ao seu descarte mal esses recursos se esgotem. O *Drop Tail* é muito usado devido à sua simplicidade e robustez, mas não assegura um tratamento justo a alguns fluxos e pode provocar a sincronização global dos fluxos [113].

A atuação do RED passa pela deteção da congestão antes de realmente surgir, capturando alguns dos seus sinais prévios. Tendo por base o princípio de que uma fila grande ao longo do tempo é sinal de congestão na rede, o descarte e/ou marcação dos pacotes pode ocorrer antes de a fila estar realmente cheia, sinalizando assim as fontes de tráfego e, previsivelmente, minimizar a congestão e o descarte de pacotes. O valor usado para a deteção da congestão é o tamanho médio da fila. Assim, a operação do RED toma três comportamentos distintos, mediante o grau

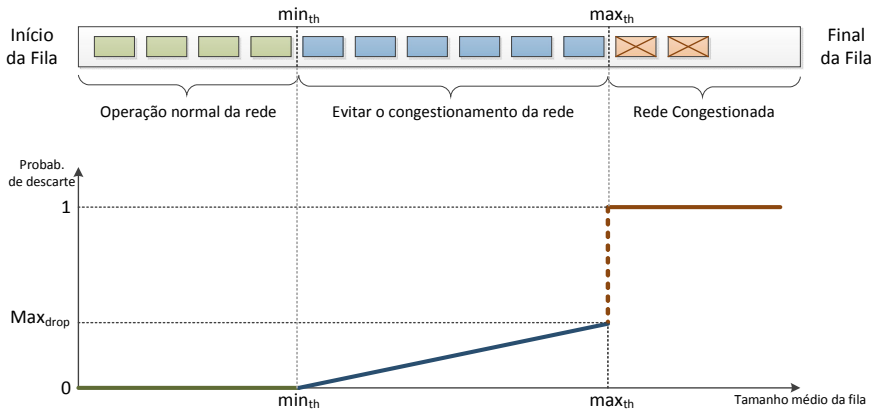


Figura 5.4: Comportamento do *Random Early Detection* (RED)

de congestão na rede, ou seja, mediante a alteração do tamanho médio da fila (TMF). De seguida é descrito cada um dos comportamentos aplicados a um pacote:

- **Operação normal da rede**: o TMF é inferior ao parâmetro *Minimum Threshold* ($\min_{th}$), implica o não descarte e/ou a não marcação do pacote;
- **Evitar o congestionamento da rede**: o TMF é superior ao $\min_{th}$, mas inferior ao *Maximum Threshold* ($\max_{th}$), implica o descarte e/ou marcação do pacote com probabilidade variável entre zero e um determinado valor de probabilidade máximo de descarte, Max_{drop}, e;
- **Controlo da congestão**: o TMF é superior ao $\max_{th}$, implica o descarte e /ou marcação do pacote.

A Figura 5.4 apresenta o gráfico de operação do RED descrita. A implementação do RED nos DiffServ/NS2 acontece associando uma fila física de pacotes a uma classe AF e usando diversas filas virtuais para cada uma das prioridades dentro da classe AF [164]. A configuração dos parâmetros RED com valores distintos para cada fila virtual causa um número de descarte de pacotes diferente em cada uma das filas. Como exemplo, o tráfego pertencente a uma classe AF e o tráfego *best-effort* terão dois códigos diferentes. O primeiro terá valores de $\min_{th}$ e $\max_{th}$ superiores ao segundo, levando à entrada do tráfego AF mais tardiamente na zona de congestionamento e de provável descarte.

O objeto NS2 utilizado para implementar as políticas de gestão das filas na arquitetura DiffServ é o dsREDQueue e é uma extensão do objeto REDQueue, por sua vez, já uma extensão de *Queue*. Este objeto tem todos os parâmetros para configurar as funcionalidades dos *routers* de borda e de núcleo. A característica principal do RED para a diferenciação de tráfego está presente na forma de estruturar as filas. No caso do NS2, foi criado dois tipos de filas para o RED: a fila física e a fila virtual. A fila física recebe várias classes de tráfego e está relacionada com as políticas de agendamento (a apresentar na secção 5.5.1). As filas virtuais são responsáveis pela implementação de diferentes tratamentos a um determinado grupo de pacotes. A estrutura de uma fila completa é apresentada na Figura 5.5.

O limite ao número de filas físicas e virtuais no NS2 é de quatro e três, respetivamente. A estrutura de fila apresentada na Figura 5.5 é uma configuração possível. Neste exemplo existe uma

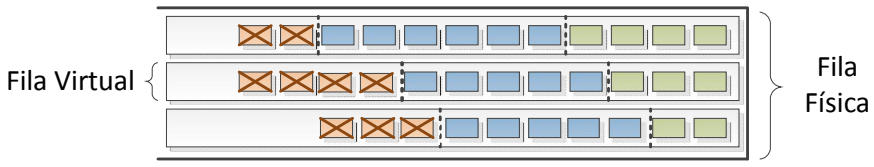


Figura 5.5: Estrutura de uma fila física, contendo três filas virtuais com diferentes parâmetros

fila física com três filas virtuais. O número de filas virtuais é, por vezes, referido como o número de precedências da fila física. A entrada numa fila virtual particular acontece quando surge um pacote marcado com o código associado a essa fila virtual. Os comportamentos de cada fila virtual são configurados recorrendo aos parâmetros $\min_{th}$, $\max_{th}$ e Max_{drop}.

5.4.2 Funções dos *routers* de borda

As responsabilidades do router de borda são a classificação, a marcação e policiamento de cada pacote. No caso do NS2, o objeto definido para realizar essas funções é o *edgeQueue* e é uma extensão do *dsREDQueue*. Além da responsabilidade de gestão das filas, herdada do objeto *dsREDQueue*, o *edgeQueue* tem a responsabilidade de marcar e policiar os fluxos de tráfego. O NS2 realiza estas funções através da classe *Policy*.

A classificação de cada fluxo é feita por intermédio da identificação dos seus nós origem e destino, pertencendo assim todo o tráfego entre esses dois pontos ao mesmo agregado. O agregado é então policiado recorrendo a um policiador e tendo por base uma taxa de transferência máxima, e outros parâmetros dependentes do policiador.

O funcionamento genérico dos policiadores baseia-se no uso de dois códigos: o primeiro, designado de código inicial, marca o tráfego dentro do perfil previamente definido para o agregado, e o segundo código para marcar o tráfego fora do perfil (*In* e *Out profile*). Alguns policiadores podem operar com 3 códigos, existindo então mais um nível para diferenciar o tráfego. Associado a cada policiador existe um medidor, responsável pela atualização das variáveis de estado usadas pelo policiador para verificar se o pacote recebido está dentro ou fora do perfil.

Os policiadores disponíveis no NS2, com os seus parâmetros, são os seguintes [164]:

- *Time Sliding Window Two Color Marker* (TSW2CM), usa uma taxa combinada de dados – *Committed information rate* (CIR) – e duas precedências, sendo a menos prioritária usada quando o CIR é ultrapassado [59];
- *Time Sliding Window Three Color Marker* (TSW3CM), usa um CIR, uma taxa de pico de dados – *Peak information rate* (PIR) – e três precedências. A precedência intermédia é usada quando o CIR é ultrapassado e a precedência menor é usada quando o PIR é ultrapassado [82];
- *Token Bucket*, usa o CIR e uma rajada combinada de dados – *Committed burst size* (CBS) – e duas precedências. A precedência menor é usada num pacote se, aquando da sua recepção, o balde está vazio [253];
- *Single Rate Three Color Marker* (srTCM), usa o CIR, o CBS para colocar o pacote numa de três precedências, detalhado em [108], e;

Tabela 5.1: Lista de parâmetros do NS2 para policiamento de tráfego

Parâmetro	Observações	Policiador(es)
Source node ID	Identificador do nó de origem	Todos
Destination node ID	Identificador do nó de destino	Todos
Policer type	Designação do policiador de tráfego	Todos
Meter type	Medidor de tráfego	Todos
Initial code point	Código inicial	Todos
CIR	*Committed information rate*	Todos
CBS	*Committed burst size*	*TokenBucket*, srTCM e trTCM
EBS	*Excess burst size*	srTCM
PIR	*Peak information rate*	TSW3CM e trTCM
PBS	*Peak burst size*	trTCM
C *bucket*	*Committed bucket*	
E *bucket*	*Excess bucket*	
P *bucket*	*Peak bucket*	
Arrival time of last packet	Tempo de chegada do último pacote	
Average sending rate	Taxa média de envio	
TSW *window length*	Janela para os policiadores TSW	TSW2CM e TSW3CM

- *Two Rate Three Color Marker* (trTCM), usa o CIR, o CBS, o PIR, e a dimensão da rajada de pico – *Peak burst size* (PBS) – para escolher entre duas precedências, como descrito em [109].

A Tabela 5.1 apresenta os parâmetros disponíveis no NS2 para os policiadores. Um policiador específico poderá apenas usar alguns desses parâmetros. A gestão das filas numa rede real é realizada no *node*. Ou seja, um pacote entra no nó é encaminhado e, posteriormente, tratado mediante a carga e parâmetros de uma certa fila. No NS2 o objeto fila (*queue*) está implementado no *link* e, no caso particular dos DiffServ, necessita da definição de ligações unidirecionais, ou seja, *link's simplex*. As funções de borda devem ser associadas aos *link*'s que partem de um *router edge* para um *router core*. A Figura 5.6 contém um extrato de um código *OTcl* de uma simulação de rede com DiffServ e, neste código, é apresentada a implementação dos comandos que realizam as funções de borda num *router edge*.

Apesar da limitação na classificação do tráfego, os comandos da implementação *Nortel Networks* permitem caracterizar bastante bem todas as funções de borda de uma rede DiffServ. Existem muitos outros componentes que podem ainda ser implementados, como os *Bandwith Brokers*, mas os componentes já descritos permitem construir simulações com diferenciação de tráfego. A explicação detalhada das funcionalidades presentes na Figura 5.6 podem ser consultadas no Anexo B.1.

5.4.3 Funções dos *routers* de núcleo

As funções no núcleo de uma rede DiffServ servem para encaminhar de forma expedita os pacotes, mas também de forma diferenciada. O objeto dsREDQueue contém os mecanismos já descritos na secção 5.4.1 para a criação e gestão das filas. O mecanismo ainda não descrito é o agendamento de pacotes (*scheduling*), ou seja, as regras para a seleção da fila de onde sairá o próximo pacote.

Os algoritmos de agendamento disponíveis no contexto do NS2/DiffServ são os seguintes [192]:

- *Round Robin* (RR), é o algoritmo por omissão e retira pacotes de todas as filas físicas de forma igual, independente do tamanho da fila e/ou qualquer prioridade de tráfego, passando

```
1  $ns simplex-link $edge $core 10Mb 10ms dsRED/edge
2  ...
3  set fEdCo [[$ns link $edge $core] queue]
4  ...
5  $fEdCo set numQueues_ 2
6  $fEdCo setNumPrec 1
7  $fEdCo addPolicyEntry [$srvA id] [$cltA id] TSW2CM 10 100000
8  $fEdCo addPolicyEntry [$srvB id] [$cltB id] TSW2CM 0 500000
9  $fEdCo addPolicerEntry TSW2CM 10 0;
10 $fEdCo addPolicerEntry TSW2CM 0 0
11 $fEdCo configQ 0 0 5 15 0.3
12 $fEdCo configQ 1 0 15 30 0.7
13 $fEdCo addPHBEntry 10 0 0
14 $fEdCo addPHBEntry 0 1 0
15 ...
16 $fEdCo printPolicyTable
17 $fEdCo printPolicerTable
18 $fEdCo printPHBTable
```

Figura 5.6: Extrato de código Tcl para a configuração das funções de borda

de fila em fila circular e regularmente. Não existe, verdadeiramente, diferenciação de tráfego a este nível;

- *Weighted Round Robin* (WRR), é uma extensão ao RR e permite adicionar diferentes pesos a cada fila, definindo um valor em termos de número de pacotes ou bytes que irá retirar de cada fila num dado momento. As filas de maior peso terão mais pacotes servidos relativamente às restantes. Este algoritmo já permite descriminar o tráfego a este nível e, por passar por todas as filas, permite que os pacotes das filas de menor peso sejam também atendidos;

- *Weighted Interleaved Round Robin* (WIRR), é uma extensão ao WRR e onde se mantêm a filosofia de pesos para cada fila, mas a regra de passagem de uma fila para a seguinte é feito de forma distinta. Tem como objetivo servir as filas de forma mais equilibrada, e;

- *Priority Queuing* (PRI), atribui um valor de prioridade a cada fila física. As filas de maior prioridade terão os seus pacotes colocados na rede rapidamente, permitindo a estas manter baixos valores de atraso e variação do atraso. Pode conduzir a grandes atrasos aos pacotes das filas de menor prioridade, visto que apenas serão servidos após as filas de maior prioridade estarem vazias.

O RR, o WRR e o WIRR são, normalmente, associados a cenários com classes AF. O PRI é, normalmente, utilizado para cenário onde existe uma classe EF. Além do algoritmo de agendamento, o número de filas, os parâmetros do RED e a forma como é classificado o tráfego, são todos responsáveis pelo nível de diferenciação obtido. É pouco provável obter um determinado tratamento a partir da configuração e/ou utilização de apenas um componente. O trabalho de [153] apresenta uma comparação dos diversos algoritmos para diferentes contextos de rede.

O trabalho apresentado em [145] apresenta um grupo de outros algoritmos de agendamento, nomeadamente WFQ, WF2Q+, SCFQ, SFQ, e LLQ; que podem ser adicionados e utilizados nas simulações realizadas com o NS2.

Além dos comandos para o preenchimento das tabelas de policiamento, a implementação da *Nortel Networks* ainda incluiu funções para conhecer o conteúdo da tabela de políticas, o estado

```
1  $ns simplex-link $core $edge 10Mb 10ms dsRED/core
2  ...
3  set fCoEd [[$ns link $core $edge] queue]
4  ...
5  $fCoEd set numQueues 2
6  $fCoEd setNumPrec 1
7  $fCoEd setSchedularMode WRR
8  $fCoEd addQueueWeights 0 2.5
9  $fCoEd addQueueWeights 1 7.5
10 $fCoEd addPHBEntry 10 0 0
11 $fCoEd addPHBEntry 0 1 0
12 $fCoEd configQ 0 0 10 20 0.25
13 $fCoEd configQ 1 0 20 40 0.75
```

Figura 5.7: Extracto de código Tcl para a configuração das funções de núcleo

dos parâmetros do *Token Bucket* e para obter as estatísticas de cada classe em cada momento de uma simulação.

A Figura 5.7 apresenta um extrato de código para o tráfego num nó de núcleo. Como já referido antes, estas configurações são implementadas num link simplex, mas, neste caso, do nó de core para outro nó (*edge* ou *core*). No exemplo é apresentado de um nó *core* para um nó *edge*. A explicação detalhada das funcionalidades presentes Figura 5.7 podem ser consultadas no Anexo B.2.

A descrição feita refere alguma similaridade entre a implementação das funções de borda e de núcleo. Os comandos para as configurações das filas utilizam os mesmos comandos e podem até ter os mesmos valores, mas, no caso dos valores, estes não têm de ser iguais. De nó para nó podem existir diferentes fluxos e, consequentemente, diferente número de filas e configurações. Nos cenários mais simples, a opção é replicar as configurações das filas por todas as ligações, mas tal não é obrigatório.

Na próxima secção é realizada a apresentação da estrutura dos objetos DiffServ suportados na linguagem NSDL.

5.5 Serviços Diferenciados em NSDL

A representação de qualquer objeto de rede na linguagem NSDL segue o princípio de ser o mais abstrato possível e, principalmente, não estar relacionado com nenhuma ferramenta particular. A definição proposta de seguida foi orientada, essencialmente, na descrição realizada em [28]. Os objetos definidos foram o nó de borda e o nó de núcleo, <dsegdenode> e <dscorenode> respetivamente.

O nó de borda fará a classificação, marcação e policiamento. O nó de núcleo será responsável pelo encaminhamento diferenciado. Ambos os objetos são extensões do elemento <router>.

5.5.1 Nó de borda DiffServ da NSDL

A definição do nó de borda DiffServ na NSDL permite a inclusão de várias entradas, uma para cada fluxo a integrar na rede como tráfego diferenciado. A entrada define o tráfego para uma dada classe e faz a sua gestão na entrada da rede. A Figura 5.8 apresenta a estrutura deste objeto na NSDL.

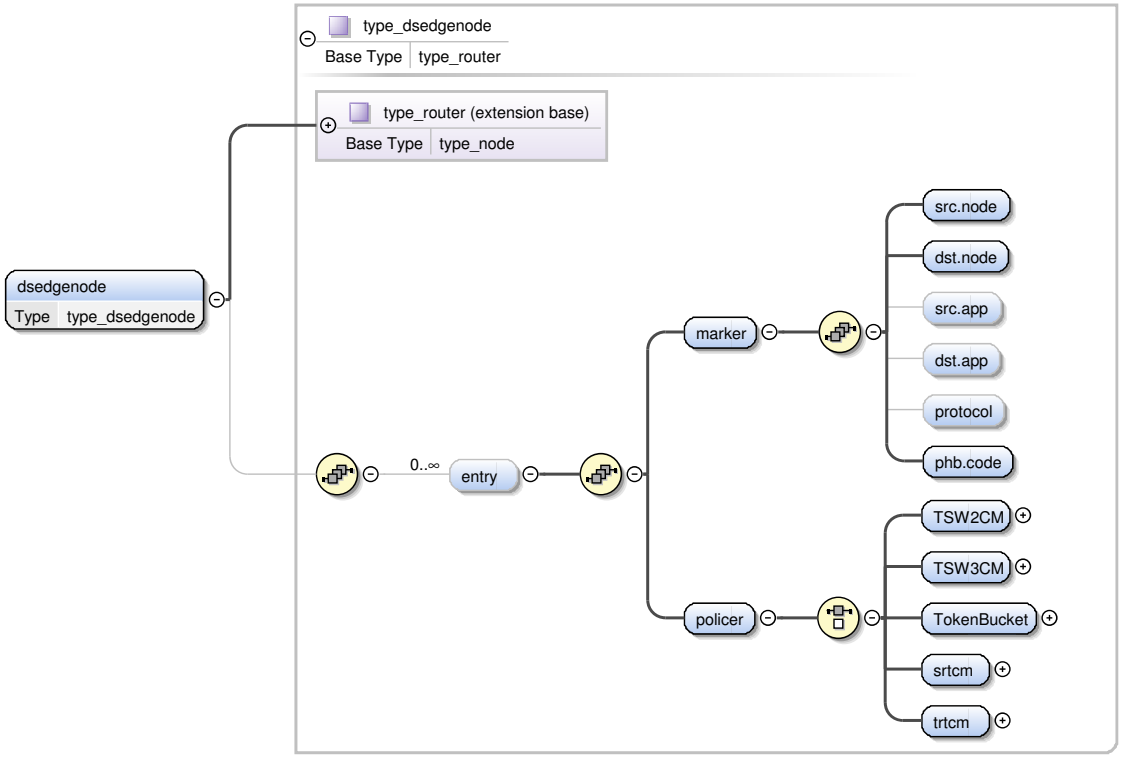


Figura 5.8: Estrutura de um nó de borda DiffServ da NSDL

Um primeiro elemento, <marker>, realiza a marcação dos pacotes usando os 5-*tuples*, que foram adaptados para a NSDL: nós e aplicações de origem e destino e o protocolo. Cada pacote com estes parâmetros, todos ou parte deles, a atravessar o nó de borda é marcado com um determinado DSCP – <phb.code> no código. O elemento <policer> concretiza a gestão de cada fluxo optando pelo policiador adequado para a implementação do policiamento do fluxo.

No caso particular da utilização do NS2, a marcação apenas é feita recorrendo aos nós de origem e destino. No caso dos policiadores, na Figura 5.8 estão apresentados aqueles que são suportados pelo NS2.

5.5.2 Nó de núcleo DiffServ da NSDL

A estrutura do nó de núcleo DiffServ na NSDL contém dois elementos na sua raiz: a configuração dos *per-hop behaviours*, <phbs>, e a política de agendamento, especificada no elemento <scheduler>. A Figura 5.9 apresenta a estrutura do nó de núcleo.

A configuração de cada fila é feita a partir dos valores colocados em cada entrada do elemento <phbs>. Os elementos <max> e <min> e <precedence> são os parâmetros $\min_{th}$, $\max_{th}$ e Max_{drop} do RED, respetivamente.

A implementação das filas no NS2, dividida em filas físicas e virtuais, não permite uma ligação direta entre os *phb*'s e uma fila física/virtual particular. A opção de adicionar um elemento extra para conter essa informação, que é específica de uma ferramenta, vai contra o princípio de independência da NSDL. Assim escolheu-se, para solucionar esta questão, seguir os códigos normalizados para os AF, EF e DF, ou seja, o utilizador ao configurar um cenário deverá escolher apenas de entre os códigos destas classes. Nesta tarefa, uma ferramenta gráfica poderá conter já os

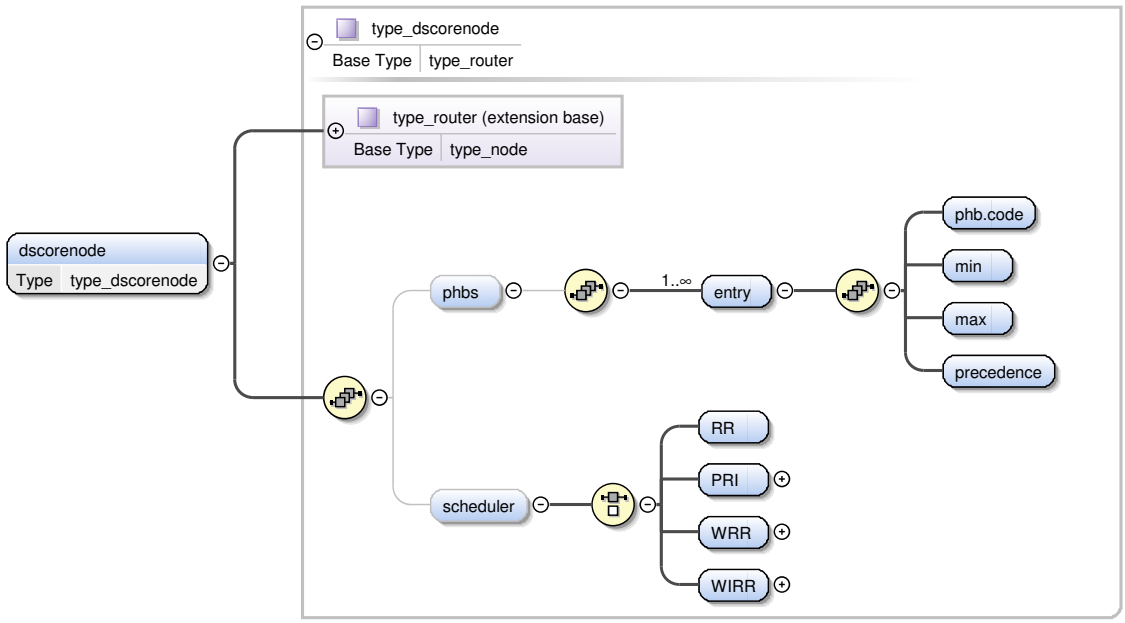


Figura 5.9: Estrutura do nó de núcleo DiffServ da NSDL

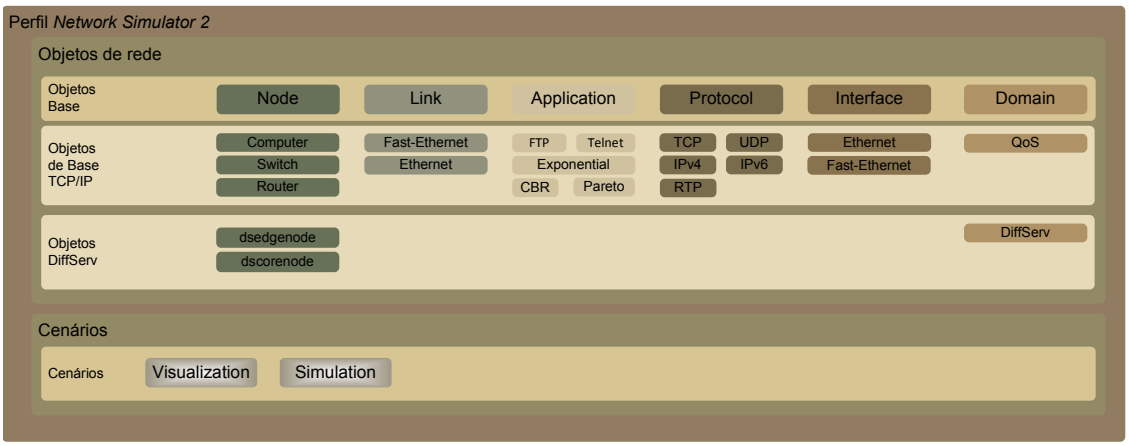


Figura 5.10: Objectos de rede NSDL que integram o perfil NS2

valores corretos e apoiar o utilizador na escolha dos *phb*'s. No caso de surgirem valores diferentes, estes serão enquadrados no serviço/classe mais próximo, até ao limite de filas permitido no NS2. O número de filas físicas e virtuais para o código *OTcl* é calculado mediante o *phb*'s existentes.

5.5.3 Perfil NS2 na NSDL

O perfil permite-nos saber quais os cenários que podem ser definidos para um dado contexto, neste caso indica-nos quais os tipos de redes que podem ser simuladas no NS2. A Figura 5.10 apresenta todos os objetos e perfis que constituem o perfil NS2.

O perfil NS2 integra, primeiro, os cinco objetos base da NSDL (perfil base), os objetos base TCP/IP já definidos e ainda os dois objetos DiffServ apresentados nas secções anteriores. Os cenários são a *Visualização* e a *Simulação*. Este perfil não refere todos os objetos que o NS2 suporta, mas os objetos que o NS2 suporta e que já estão definidos na NSDL.

A importância do perfil surge na contribuição feita para a interoperabilidade entre ferramentas. O grau de interoperabilidade entre diferentes ferramentas pode ser obtido cruzando os seus perfis. Um perfil deve integrar os objetos cuja validação e tradução já foi construída e testada. Assim, o resultado da interseção de dois perfis dará o grau de interoperabilidade entre as ferramentas. Um cenário específico poderá ser completamente compatível entre duas ferramentas, caso os objetos que integre estejam presentes em ambos os perfis das ferramentas. Além de permitir aferir do grau de interoperabilidade, o uso de perfis permite encaminhar os criadores de objetos para a reutilização e/ou extensão de definições, promovendo assim interoperabilidade.

É possível e simples alargar o número de objetos de um perfil, bastando adicionar a especificação do objeto em termos de elementos que o constituem e as regras de tradução para a linguagem da ferramenta, neste caso o *OTcl*. Na próxima secção é descrito este processo.

5.6 Transformação de cenários NSDL para NS2

A geração de simulações para a ferramenta NS2 é feita a partir das estruturas XML definidas para a NSDL. Em alguns casos a conversão é simples e direta, como por exemplo os nós, em outros casos a conversão obriga à análise de múltiplos objetos NSDL para o cálculo dos dados a gerar para a linguagem do NS2.

A estrutura base de uma simulação em NS2 é apresentada na próxima secção para, na secção 5.6.2, descrever a conversão detalhada dos objetos do perfil NS2 da NSDL para o simulador e, na secção 5.6.3, discutir a transformação dos objetos DiffServ.

5.6.1 Estrutura de uma simulação NS2

O NS2 permite duas linguagens para a especificação de objetos e de cenários de simulação. A primeira linguagem, o C++, é utilizada, essencialmente, para a criação de modelos – objetos – para o simulador e está no núcleo de toda a ferramenta. Pode também ser usada para especificar simulações, mas é pouco comum esta utilização pela sua baixa flexibilidade (necessita de compilar toda a aplicação por cada alteração no código). A importância do C++ é suportar a execução eficiente da simulação.

A segunda linguagem é o *OTcl* e é usada para configurar a simulação que o utilizador quer realizar e serve como *frontend* para a parte da ferramenta desenvolvida em C++. A descrição do cenário de simulação é, de forma simples, a definição da topologia de rede e dos eventos que o utilizador pretende para o seu cenário. Sendo o *OTcl* uma linguagem de programação genérica e muito flexível, esta também permite a especificação de objetos complexos, mas, neste contexto, não é usada normalmente para este fim. É a opção mais comum para a descrição de simulações no NS2, ligando-se aos objetos já construídos e compilados em C++. Sendo o código *OTcl* interpretado, a sua alteração não obriga à compilação de toda a aplicação. É menos eficiente, comparativamente ao C++, mas compensa com a flexibilidade permitida ao criador de simulações. A linguagem *OTcl* foi a escolhida no âmbito deste trabalho para o intercâmbio da descrição de cenários de rede entre diferentes ferramentas.

A Figura 5.11 ilustra, à esquerda, uma estrutura NSDL genérica e, à direita, extratos de código *OTcl* para uma simulação a executar sobre a rede apresentada na parte superior da imagem, uma

rede simples com dois nós e uma ligação. A análise da documentação da ferramenta e a consulta de muitos exemplos – reais e de teste – conduziram à identificação de um padrão comum para a codificação.

As ligações entre a estrutura NSDL e o código *OTcl* da Figura 5.11 demonstram quais os elementos necessários para a criação de cada um dos grupos de código comuns. Mantendo presente a Figura 5.11, de seguida é listado e descrito cada um desses grupos:

- **Parâmetros iniciais**: incluem as definições iniciais e os dados globais da simulação, como por exemplo, as variáveis globais e a alteração dos valores por omissão da simulação;

- **Topologia**: cria os nós e as ligações entre eles. Permite a parametrização detalhada de todas as características destes dois objetos, como por exemplo, a largura de banda de uma ligação e a aparência de um nó na ferramenta de animação NAM;

- **Geração de tráfego**: cria as aplicações, responsáveis pela geração do tráfego, e os agentes para o transporte desse mesmo tráfego pela rede. A implementação completa deste grupo de objetos apenas pode acontecer após a criação dos nós devido à ligação dos agentes de transporte com os nós. Como exemplos de aplicações e agentes temos o FTP e o TCP, respetivamente;

- **Eventos**: lista de eventos definidos pelo utilizador para a simulação. A linguagem permite a definição de muitos tipos eventos, sendo os mais comuns o arranque e paragem da geração de tráfego (aplicações), a alteração das propriedades dos objetos (desativar uma ligação, alteração de uma largura de banda, e muitos outras). Deve ser feita apenas após a criação de todos os objetos, já que a maioria dos eventos refere um ou vários objetos;

- **Resultados**: define a forma de construção dos resultados da simulação. No caso do NS2, de raiz, permite dois formatos. O primeiro são matrizes de eventos da simulação, onde todos os eventos que ocorrem são guardados, como por exemplo: a criação de um pacote, a saída do pacote de um nó, a entrada do pacote em outro nó, e, o possível descarte de pacote. O segundo formato permite a criação de um ficheiro de dados para posterior animação da simulação pela ferramenta NAM. O utilizador pode ainda gerar os resultados da simulação da forma que lhe for mais adequada, tendo no entanto que desenvolver o código necessário para esse fim, e;

- **Término**: incluir no cenário o código necessário para definir um ponto de paragem da simulação. A forma mais comum é indicar um momento temporal específico para o término da simulação, mas pode ser criado código que avalie continuamente a simulação e que a termine após ser atingida uma certa condição, como por exemplo: a quantidade de tráfego num nó ou o número de pacotes perdidos numa ligação.

A linguagem *OTcl* é flexível o suficiente para a criação de uma simulação bastante complexa e específica, mas, no caso do projeto NSDL, apenas foram especificadas as transformações para o perfil apresentado. De seguida são referidas funções que implementam a transformação do XML em *OTcl*.

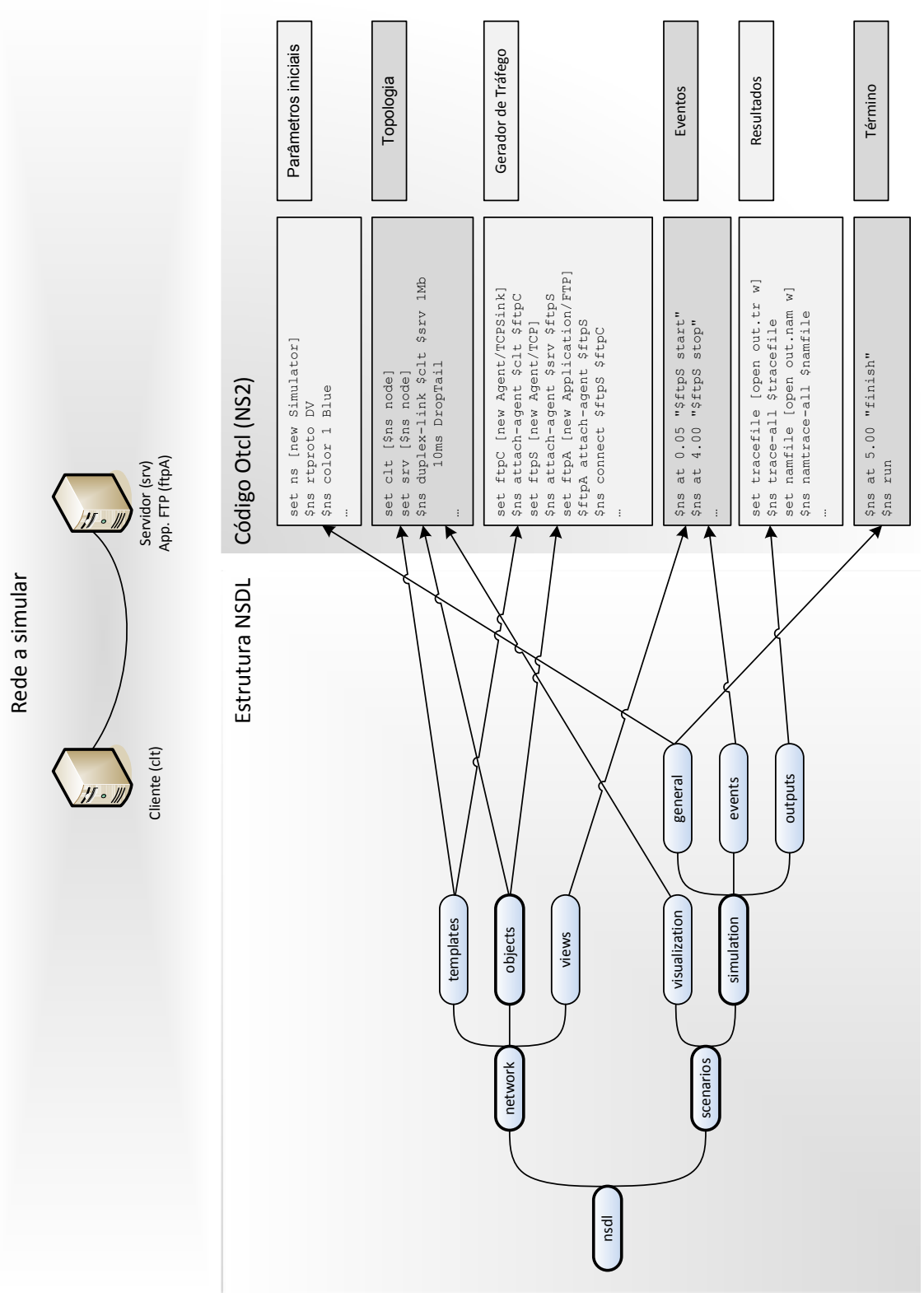


Figura 5.11: Esquema de transformação de um cenário NSDL em código para a ferramenta NS2

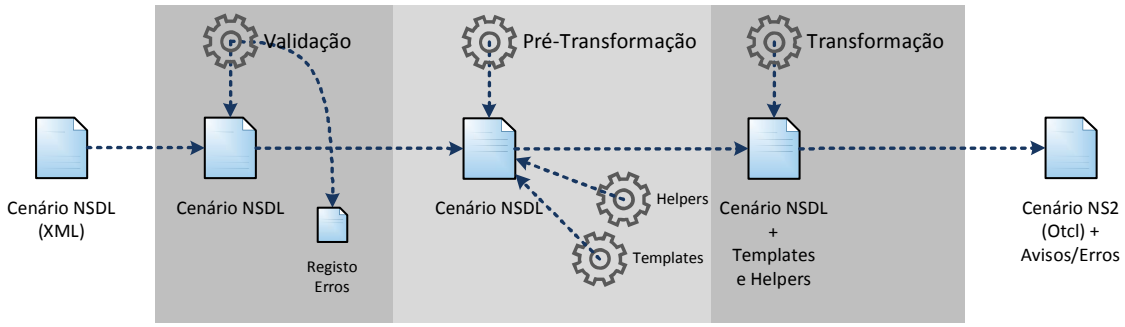


Figura 5.12: Sequência de transformação de um cenário NSDL para uma simulação NS2

5.6.2 Metodologia para a Transformação

A transformação de um cenário NSDL numa simulação para o NS2 foi estruturada em três passos: a validação, a pré-transformação e a transformação. A Figura 5.12 apresenta a sequência e mais alguns componentes do processo.

O passo *Validação* verifica se a estrutura do ficheiro NSDL está correta, baseando-se nos objetos e nas regras definidas para o perfil. A implementação da operação é feita por intermédio de uma aplicação 'validadora' (XSD *validator*) que recebe, primeiro, o ficheiro com as definições do perfil, sob a forma de um ficheiro XSD (XML *Schema*) e, segundo, o ficheiro XML com o cenário de rede no formato NSDL. As operações de validação avaliam os elementos do cenário e as suas propriedades com as definições do perfil e executam outras funções do perfil para garantir que os dados do cenário são coerentes. Ambas são definidas em XML *Schemas*.

Além da verificação da estrutura do cenário, neste passo também é gerada uma lista de entradas com informação de cada erro e/ou aviso detetado para futura depuração do cenário. Os erros distinguem-se dos avisos no efeito que têm sobre a execução da simulação. Um erro é uma situação que não permitirá a correta execução do cenário, como um objeto não existente no perfil ou o valor de uma propriedade fora dos limites. Um aviso é uma situação que poderá permitir a execução correta do cenário, mas que, dependendo de diversos fatores, poderá causar erros ou execuções imprevistas e, como tal, deverá ser verificado pelo criador do cenário.

O passo de *Pré-transformação* tem como propósito facilitar o processo de transformação. A primeira função, *Templates*, verifica se no cenário existem elementos em <templates> e, caso existam, carrega/atualiza os objetos do elemento <network> com os dados dos <templates>. No futuro, poderão ser criadas transformações para objetos definidos nos <templates>, mas tal não foi realizado nesta fase, ou seja, apenas existem transformações para os objetos da rede. As razões desta função e a forma como está implementada estão apresentadas na secção 4.3.2.

A segunda função, *Helpers*, permite contornar algumas das limitações do XSL em termos de facilidade de programação, desempenho e funções não implementadas. A criação e alteração de variáveis, a realização de ciclos e invocação de procedimentos são funcionalidades comuns e presentes em qualquer linguagem de programação genérica. No caso do XSL, estas funcionalidades existem, mas de forma limitada e obrigam à utilização de muitas linhas de código para realizar funções simples, com prejuízo para o desempenho do processo de transformação.

Um exemplo é a execução de expressões dinâmicas *XPath*, ou seja, realizar uma pesquisa medi-

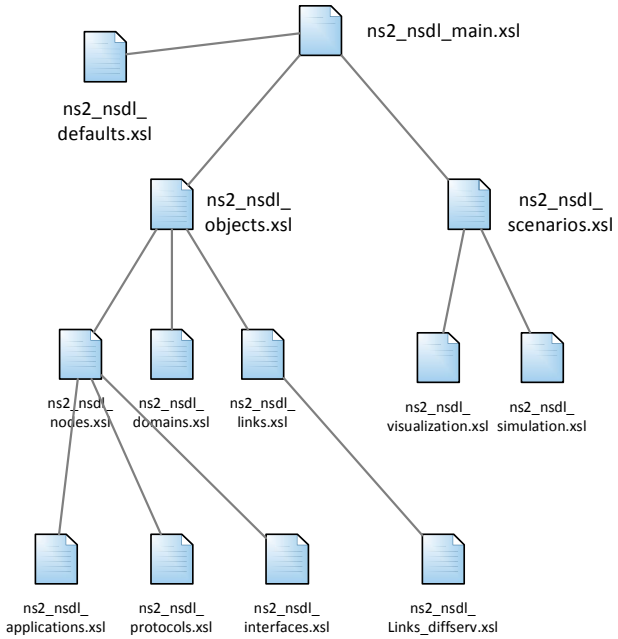


Figura 5.13: Hierarquia de ficheiros utilizados no processo de transformação de um cenário NSDL para um ficheiro *OTcl*

ante um valor de um elemento ou atributo presente no documento XML. Nas implementações base de XSL *Transformations* (XSLT) esse valor tem de existir no código XSL, ou seja, o programador teria de conhecer um valor antes de existir o ficheiro XML que o contém, o que é, naturalmente, impossível. A forma utilizada para contornar a situação é programando toda a pesquisa, mas, em cenários grandes, tal pode tomar muito tempo. Uma outra forma de resolver este problema seria utilizar diferentes processadores da linguagem XSLT, que visam essencialmente, contornar as suas limitações, como por exemplo o *Saxon* [240].

O componente *Helpers* é uma concessão na *Framework* NSDL, visto ser o único componente a não utilizar nenhuma linguagem da família XML. Pode ser escolhida uma linguagem de programação qualquer e, para esta fase do projeto, foi escolhida a linguagem *php* [213] devido a ter sido implementada uma plataforma Web para a validação e tradução de cenário NSDL em *php*.

Por último, o passo mais relevante desta secção, a *Transformação*. A Figura 5.12 apenas refere a passagem de um ficheiro XML para um ficheiro *OTcl*, mas a funções realizadas e a sequência de implementação são dados importantes para perceber todas as características da transformação. A Figura 5.13 apresenta a hierarquia dos ficheiros XSL (*EXtensible Stylesheet Language*) que são utilizados em todo o processo de transformação.

O ficheiro no topo da hierarquia é responsável por conter as instruções iniciais do processo de transformação e por chamar todos os ficheiros necessários. Seguindo a estrutura da própria NSDL, cada ficheiro da transformação NSDL para NS2 definido está relacionado com cada objeto NSDL, como no caso da transformação dos nós, e das suas extensões, que estão definidas no ficheiro ns2_nsdl_nodes.xsl. Todos os restantes objetos e cenários seguem esta regra, exceto dois ficheiros particulares. O primeiro, ns2_nsdl_defaults.xsl, guarda os valores por omissão de cada variável do NS2. O segundo, ns2_nsdl_links_diffserv.xsl, contém o código específico para a implementação dos Diffserv e, como no NS2, são implementados nas ligações, optou-se por manter essa relação

```
1 <node id="cliente">
2         <name>cliente_web</name>
3 </node>
```

(a) Linguagem NSDL

```
1 cliente [$ns2 node]
2 $cliente label cliente_web
3 ...
```

(b) Linguagem OTcl/NS2

Figura 5.14: Objeto Nó descrito em duas linguagens

```
1  <xsl:template match="node">
2         <xsl:call-template name="callBasicNode"/>
3  </xsl:template>
4  (...)
5  <xsl:template name="callBasicNode">
6         <xsl:value-of select="$lb.n"/>
7         <xsl:text>set </xsl:text><xsl:value-of select="@id" />
8                 <xsl:text> [$</xsl:text><xsl:value-of select="$ns2.nsdl" />
9                 <xsl:text> node]</xsl:text><xsl:value-of select="$lb.n"/>
10        <xsl:choose>
11                <xsl:when test="name">
12                        <xsl:text>$</xsl:text><xsl:value-of select="@id" />
13                        <xsl:text> label "</xsl:text><xsl:value-of select="name" />
14                        <xsl:text>"</xsl:text><xsl:value-of select="$lb.n"/>
15                </xsl:when>
16                <xsl:otherwise>
17                        <xsl:text>$</xsl:text><xsl:value-of select="@id" />
18                        <xsl:text> label </xsl:text><xsl:value-of select="@id" />
19                        <xsl:value-of select="$lb.n"/>
20                </xsl:otherwise>
21        </xsl:choose>
22 </xsl:template>
23 (...)
```

Figura 5.15: Código XSL que define a transformação de um nó NSDL para um nó NS2

estendendo os links base com os links DiffServ em NS2.

A maior complexidade no processo de transformação foi relacionar os elementos da estrutura NSDL com os objetos da estrutura *OTcl*. Em muitos dos casos a tradução de um objeto particular é simples e direta, gerando-se o código para a criação do objeto em *OTcl* e recorrendo às propriedade do objeto NSDL para carregar os valores necessários em *OTcl*. A Figura 5.14(a) apresenta a estrutura de um nó NSDL e a Figura 5.14(b) o seu equivalente em *OTcl*. Para complementar, a Figura 5.15 apresenta o código XSL que concretiza a transformação do nó NSDL para NS2.

O valor do atributo @id na descrição NSDL (Figura 5.14(a)) serve para definir a variável cliente em *OTcl* (Figura 5.14(b)) e o valor contido no elemento <nome> é atribuído à propriedade label de um node *OTcl*. O código XSL da transformação, Figura 5.15, foi organizado com um primeiro comando *template* do XSL e este invoca um segundo *template* XSL, callBasicNode, onde está a geração do código para o nó. A razão dos dois passos está na reutilização do código conseguida, importante por existirem muitas características em comum entre o <node> e os objetos que são suas extensões, como o <computer> ou o <router>.

Apesar do exemplo anterior apresentar uma transformação muito simples, muitos outros objetos em *OTcl* não são criados de uma forma tão direta e independente. A geração do código completo para a configuração de uma aplicação servidor e de um seu cliente é um exemplo dos casos menos

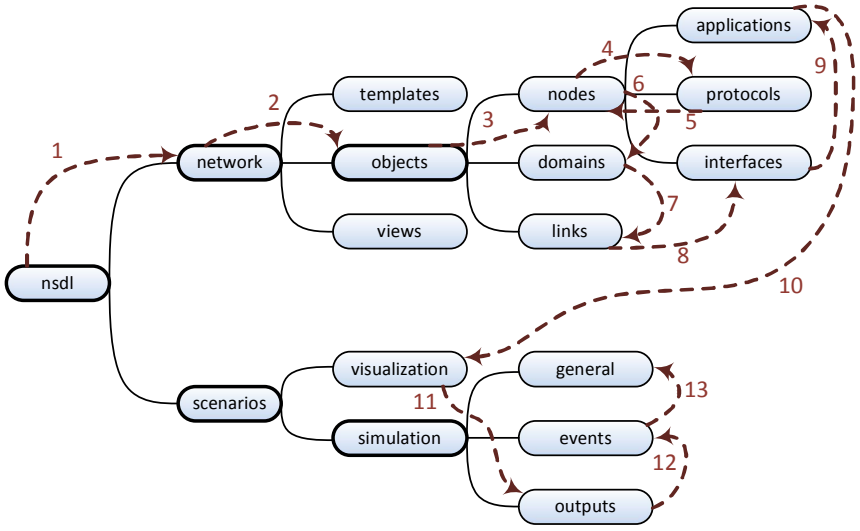


Figura 5.16: Sequência utilizada para a transformação de elementos NSDL numa simulação NS2

simples. No código simulador NS2, a aplicação opera sob um agente e este por sua vez existe quando associado a um nó. A configuração dos nós é independente de outros objetos, mas a configuração dos agentes apenas pode ser feita após a criação dos nós e a configuração das aplicações apenas após as configurações dos agentes e dos nós de origem e destino da aplicação.

A sequência apresentada na secção 5.6.1 já separa a topologia – com a criação dos nós e das ligações – da criação das aplicações e agentes – geração de tráfego. Assim, e para equiparar as estruturas das descrições de cada linguagem foi necessário seguir uma sequência particular e que está apresentada na Figura 5.16.

As fases 1 a 8 implementam a topologia, sendo os primeiros passos usados para inserir os parâmetros iniciais da simulação e, depois, mais precisamente nas fases 3 a 8, acontece a geração do código dos objetos nó, protocolo, domínio e ligação. Na fase 3 são criados os nós e na fase 4 os protocolos (agentes para o NS2). Como a ligação entre agentes do NS2 obriga a ambos estarem criados, o processo volta ao nó, e consequente protocolo, até todos estarem definidos, mas, por opção, ainda não é implementada nesta fase a ligação entre agentes, ou seja, entre origem e destino do tráfego.

Seguem-se os domínios, fase 6, e, para o término da topologia, as ligações e os *interfaces*. Estes objetos apenas podem ser criados após os nós e os domínios o terem sido. Na NSDL os *interfaces* podem ter ligações aos protocolos para, por exemplo, definir o endereço IP associado a um *interface*. Mas, no NS2, o objeto *interface* em redes com fios não existe e o endereçamento disponível baseia-se apenas nos códigos gerados na interpretação do *OTcl*. Poderia ser retirado o objeto *interface* do perfil NS2 e, assim, contornar esta situação, mas optou-se por manter o *interface* no perfil e equiparar a transformação para um *link* NS2. Esta solução garante uma utilização coerente dos *interfaces* nos cenários com fios do NS2 e, como vantagem, evita-se a retirada de um objeto base de um perfil, o que poderia comprometer fortemente a interoperabilidade das ferramentas que usassem esse perfil. Outra razão para manter o *interface* no perfil NS2 deve-se à sua necessidade para os cenários de rede sem fios, um tipo de redes a apresentar Capítulo 6.

A fase 9 gera o código para criar as aplicações, ligando-as aos protocolos/agentes criados na

```
1  <node id="cliente" />
2  ...
3  <visualization id="visual01">
4          <name>Visual 01</name>
5          <description>
6                  ...
7                  <object id="cliente">
8                          <x.position>200</x.position>
9                          <y.position>100</y.position>
10                         <z.position>0</z.position>
11                         <color>Blue</color>
12                 </object>
13                 ...
14 </visualization>
```

(a) Linguagem NSDL

```
1  #Create simulator object
2  set ns [new Simulator]
3  ...
4  #Create nodes
5  set cliente [$ns node]
6  ...
7  #Configs for NAM
8  $cliente label 'cliente'
9  $cliente X_ 200
10 $cliente Y_ 100
11 $cliente Z_ 0
12 $cliente color blue
13 ...
14 #Rest of simulation
```

(b) Linguagem OTcl/NS2

Figura 5.17: Código para a visualização de um objeto em duas linguagens

fase 4, e então conclui a ligação entre os agentes origem e destino. Assim, e seguindo a sequência apresentada, a geração das aplicações é conseguida sem erros de interpretação do NS2.

Os passos finais, fases 10 a 13, têm como objetivo a geração do código para construir os vários cenários especificados para o NS2. A fase 10 é a declaração dos valores para a visualização da topologia na ferramenta NAM. Os valores permitem colocar cada nó numa posição específica, apontada pelas coordenadas XYZ, e ainda referir alguns atributos (cor e forma). As Figuras 5.17(a) e 5.17(b), linhas 7 a 12, apresentam um exemplo dos dados para a configuração da visualização de um nó (cliente) em NSDL e em *OTcl*, respetivamente.

Por fim, as fases 11 a 13 estão relacionadas com o cenário de simulação NSDL e implementam a configuração dos resultados, dos eventos e dos parâmetros finais da simulação, respetivamente. As Figuras 5.18(a) e 5.18(b) apresentam um exemplo de parte um cenário de simulação com a codificação em NSDL e em NS2 para a especificação dos eventos e dos resultados.

Os eventos descrito nas linhas 4 a 11 do exemplo da Figura 5.18(a) configuram o arranque e a paragem de um servidor FTP, nos instantes 0 e 5 da simulação, respetivamente. As linhas 12 e 13 da Figura 5.18(b) são as mesmas ações, mas agora em *OTcl*. Os eventos podem ainda permitir ativar e desativar objetos de rede e alterar os valores dos seus parâmetros.

Os resultados em NSDL são configurados nas linhas 12 a 15 da Figura 5.18(a) e nas linhas 15 a 20 das Figuras 5.18(b) surge o equivalente em *OTcl*. O exemplo apresentado dos resultados tem como consequência a captura ficheiros de texto de todos os eventos, nos formatos de trace e de animação. Existe a possibilidade de acrescentar às configurações de resultados diversos filtros, e estes permitem reduzir os resultados para apenas os eventos de um ou vários objetos da rede. Outro tipo de configuração possível para os resultados é a indicação da execução de uma análise de alguns dados durante a simulação e apresentar os resultados calculados de, por exemplo, pacotes perdidos, largura de banda ocupada e os diferentes atrasos (mínimo, máximo, médio e a variação).

A conclusão da apresentação da transformação dos cenários NSDL para NS2 está ainda incompleta, restando os objetos DiffServ. A próxima subseção detalha os passos necessários para essa tarefa.

```
<simulation id="sim01">
...
        <description>
                <events>
                        <event objectid="ftpS" time="0.0">
                                <parameter name="action" value="start"/>
                        </event>
                        <event objectid=" ftpS" time="5.0">
                                <parameter name="action" value="stop"/>
                        </event>
                </events>
                <outputs>
                        <output outputid="main" format="trace"/>
                        <output outputid="anim" format="namtrace"/>
                </outputs>
        </description>
</simulation>
```

(a) Linguagem NSDL

```
#Create a simulator object
set ns [new Simulator]
...
#Create nodes, links and agents
...
#Setup a FTP over TCP connection
set ftpS [new Application/FTP]
$ftpS attach-agent $tcp
$ftpS set type_ FTP
...
#Configure application start and stop
$ns at 0.00 "$ftpS start"
$ns at 5.00 "$ftpS stop"
...
#Open the trace file
set tracefile [open main.tr w]
$ns trace-all $tracefile
#Open the NAM trace file
set namfile [open anim.nam w]
$ns namtrace-all $namfile
...
#Finish the simulation after 10 seconds
$ns at 10.0 "finish"
#Run the simulation
$ns run
```

(b) Linguagem OTcl/NS2

Figura 5.18: Configuração de eventos e resultados num cenário de simulação em duas linguagens

```
<dsedgenode id="RTedge" object="node">
        <entry>
                <marker>
                        <src.node>server</src.node>
                        <dst.node>client</dst.node>
                        <phb.code>10</phb.code>
                </marker>
                <policer>
                        <TokenBucket>
                                <cir>1000000</cir>
                                <cbs>3000</cbs>
                                <new.phb.code>12</new.phb.code>
                        </TokenBucket>
                </policer>
        </entry>
        ...
</dsedgenode>
```

Figura 5.19: Código NSDL para criar um nó de borda DiffServ

```
$fEdCo set numQueues_ 2
$fEdCo setNumPrec 2
$fEdCo addPolicyEntry [$server id][$client id] TokenBucket 10 1000000 3000
$fEdCo addPolicerEntry TokenBucket 10 12
$fEdCo addPHBEntry 0 0 0
$fEdCo addPHBEntry 10 1 0
$fEdCo addPHBEntry 12 1 1
$fEdCo configQ 0 0 0 10 1.00
$fEdCo configQ 1 0 10 20 0.2
$fEdCo configQ 1 1 20 30 0.5
```

Figura 5.20: Código NS2 para definir as políticas de um *link* DiffServ

5.6.3 Transformação dos objetos NSDL/DiffServ

O processo de transformação para estes objetos envolve algumas especifidades uma vez que, na NSDL, é representado um nó (de borda ou de núcleo), e, no NS2, duas ligações, ou seja, os parâmetros DiffServ do nó serão colocados em duas ligações *simplex* no NS2. O ficheiro que contém a especificação da transformação está na Figura 5.13 e podemos verificar que é uma extensão das ligações (ficheiro ns2_nsdl_links_diffserv.xsl).

Optou-se por configurar em NSDL todos os parâmetros de uma arquitetura DiffServ nos *routers* de borda e núcleo, como descrito em [28] e também por ser a abstração mais próxima da realidade prática. O ajuste do cenário à estrutura do código do NS2 é da responsabilidade do processo de transformação e é, dessa forma, transparente para o criador do cenário NSDL. A transformação de um objeto DiffServ NSDL, neste caso um nó de borda, é apresentada nas Figuras 5.19 e 5.20 onde são apresentados os códigos em ambos os formatos.

A Figura 5.19 tem um nó de borda descrito em NSDL. A sua estrutura já foi apresentada na secção 5.5.1. Na Figura 5.20 encontra-se o código para a marcação e policiamento de um fluxo de tráfego entre o *server* e *client* para a ferramenta NS2. Os detalhes do seu conteúdo são descritos no Apêndice B.1.

A Figura 5.21 apresenta a abordagem necessária para geração do código dos objetos DiffServ e é executada para cada ligação que exista no cenário. A linha tracejada mostra o caminho do processo no caso de um nó de borda, este exemplo, que surge referido como 'Edge to Core' devido

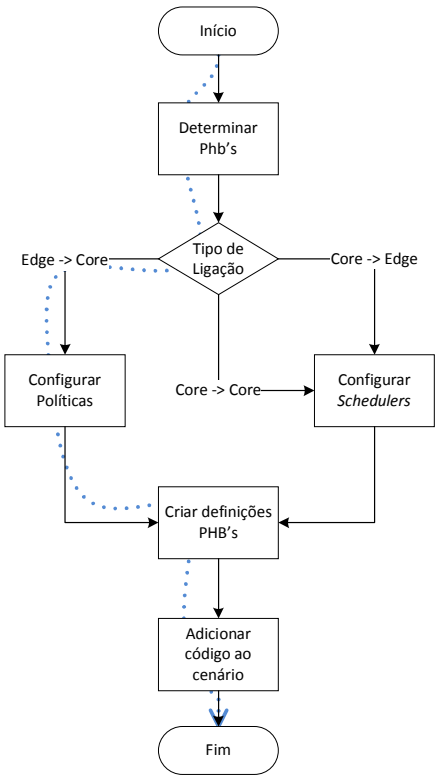


Figura 5.21: Abordagem necessária para a geração do código de um nó DiffServ

ao sentido real que a ligação terá no NS2.

A abordagem apresentada está simplificada e, em cada função, existem diversos cálculos para determinar quais os valores que devem ser colocados em cada parâmetro. De seguida, cada uma dessas funções é apresentada em termos de funcionamento e objetivos:

Determinar *phb*'s: são pesquisados nos objetos DiffServ ligados pelo *link* todos os *phb*'s definidos e é preenchida uma matriz. Esta permitirá calcular o número de filas físicas e filas virtuais e, em termos de código *OTcl*, resultará nas linhas 1 e 2 da Figura 5.20;

Configurar Políticas: é responsável por gerar os valores para as linhas que realizam a marcação e o policiamento dos fluxos, fazendo, para isso, a avaliação de cada elemento <entry>. Resultará nas linhas 3 e 4 da Figura 5.20;

Criar as definições dos *phb*'s: partindo dos códigos normalizados para as classes AF, EF e DF, é associado a cada *phb* uma fila. Após esta associação, são configurados os parâmetros RED de cada uma das filas. O código gerado é o presente nas linhas 5 a 10 da Figura 5.20, e;

Adicionar ao cenário: todas as configurações anteriores são, nesta fase, geradas sob a forma de XML e estas são adicionadas ao cenário NSDL no objeto respetivo. A próxima fase, que é onde se concretiza a transformação, é que verdadeiramente gera o *OTcl*. Mas, após este passo, a transformação é simples e passa apenas por uma correspondência direta entre a estrutura adicionada e o código *OTcl*.

A função presente na Figura 5.21 e não descrita na lista anterior é a configuração do *scheduler*. Esta função apenas se aplica às ligações que partem de um nó de núcleo (*core to edge* e *core to core*)

e é responsável pela configuração dos algoritmos de agendamento. Apesar de esta ser uma função muito diferente da configuração das políticas, o princípio de transformação é muito semelhante ao já apresentado para o nó de borda. Mais detalhes sobre o processo de transformação do *scheduler* pode ser encontrado no Anexo B.2.

A próxima secção descreve os testes realizados para validar os vários componentes da *Framework* NSDL com o NS2.

5.7 Validação do uso de NSDL para NS2

A validação da *Framework* NSDL e do seu perfil NS2 foi definida sob duas vertentes. A primeira procurou verificar o funcionamento correto e a representatividade da *Framework* NSDL e passou pela criação do maior número de redes distintas e pela sua verificação com diferentes ferramentas. O objetivo principal foi a deteção de falhas nas ferramentas criadas, nos cenários construídos e nas transformações geradas. A segunda vertente pediu aos criadores de cenários em NSDL e NS2 a opinião sobre a criação de cenários manualmente nas duas linguagens e sobre a utilização da ferramenta para a geração de código. Procurou-se, a partir do ponto de vista do utilizador, aferir sobre a relevância e a utilidade da *Framework* NSDL e do perfil NS2.

Os testes ocorreram ao longo de dois segundos semestres de dois anos letivos do curso Mestrado em Telecomunicações e Redes, da Universidade da Madeira, e os alunos da cadeira Tecnologias Avançadas de Redes foram os escolhidos devido a terem no seu programa o desenvolvimento de simulações com o NS2.

As próximas duas secções apresentam alguns dados e resultados sobre os dois processos de validação.

5.7.1 Avaliação das bibliotecas de código e das ferramentas

A disciplina *Tecnologias Avançadas de Redes* inclui no seu programa um capítulo sobre a Qualidade de Serviço e uma secção reservada aos Serviços Diferenciados. A componente prática da disciplina, ministrada pelo autor desta tese, é feita sobre o simulador NS2 e, aí, pede-se aos alunos que aprendam durante o semestre a construir, executar e avaliar simulações de redes de dados.

A componente prática envolve a aprendizagem do simulador NS2, mais particularmente a linguagem *OTcl*. São também utilizadas outras ferramentas do NS2, como o NAM para as animações das simulações e a linguagem AWK para a codificação e cálculos de dados para a apresentação de gráficos com o *xGraph*. O contato com o simulador começa com cenários simples, e são realizados testes de ligação e de encaminhamento. No decorrer do semestre são inseridos outros problemas que envolvem a compreensão de mecanismos basilares das redes, como os protocolos de transporte e a gestão das filas, e, no último terço do semestre, apresentam-se os objetos do NS2 que permitem simular a diferenciação de tráfego.

A avaliação da parte prática é feita por intermédio de dois projetos e, em ambos, é pedido ao aluno/grupo que definam um cenário de simulação em *OTcl*, concretizem a sua execução no NS2 e façam a análise crítica dos seus resultados.

A introdução da *Framework* NSDL na disciplina surge apenas na parte prática. Os alunos tomam contato com a linguagem e sua estrutura, e são apresentadas as ferramentas que permitem

Tabela 5.2: Tabela com os indicadores obtidos

Indicador	Valor
Número Total de Submissões	1557
Número de submissões únicas	171
Número de submissões/cenário	9,1
Número de erros/cenário até 10 bytes	53%
Número de erros/cenário até 100 bytes	73%
Número de erros por cenário (10 bytes)	4,8
Relação tamanho NSDL/OTcl	233%
Dimensão do maior ficheiro	112.379 B

editar o código XML e transformar os cenários de rede em código *OTcl*. Nesta fase não foi implementada a função que permite fechar o ciclo, ou seja, executar a simulação, obrigando os alunos a realizar esse último passo manualmente. Os valores que serão referidos nos parágrafos seguintes estão sintetizados na Tabela 5.2.

Sobre a edição dos cenários os resultados foram muito diversos devido à utilização de muitas ferramentas. Inicialmente apenas era possível a edição com diferentes editores de texto ou de XML convencionais e os alunos do primeiro semestre identificaram como principal lacuna a morosidade em criar o código XML. No segundo semestre já foi utilizada a ferramenta VND (apresentada na secção 3.3.1) e este ambiente gráfico trouxe à *framework* uma grande facilidade para a geração de cenários em XML.

Durante a avaliação da NSDL, foram submetidos para validação e transformação 1557 cenários válidos em termos de XML. Houve um número maior de submissões, mas foram retiradas deste valor por serem ficheiros não XML ou apresentarem erros diversos, e, em ambas as situações, consideraram-se não relevantes para a verificação do processo de validação e transformação da NSDL.

A filtragem da lista de submissões por nome do ficheiro, data e tamanho permitiu-nos detetar que das 1557 submissões, 171 podem ser consideradas como submissões únicas, ou seja, são cenários de rede únicos. Os parâmetros da filtragem apresentados foram escolhidos de forma a procurar identificar o número correto de cenários distintos. Inicialmente a filtragem foi feita apenas pelo nome do ficheiro e o resultado foi um número inferior de cenários: 107. Mas, após alguma análise, detetou-se que certos nomes foram usados repetidamente, como por exemplo, "TAR"ou "NSDL", e, em alguns casos, também as datas e os tamanhos eram muito diferentes, mostrando que o número de cenários era diferente e maior. Como exemplo, os nomes referidos foram usados em anos letivos diferentes, e, consultando os ficheiros, confirmou-se que eram redes diferentes.

Um segundo valor retirado dos dois indicadores anteriores é a média de submissões por cenário até este estar terminado e que é de 9-10, aproximadamente. Uma componente importante relacionada com este número era distinguir quantas vezes a submissão foi feita por erro ou por adição de novos elementos. A submissão em consequência de existirem novos elementos nos cenários não é relevante nesta análise mas, o valor de submissões devido a erro indica-nos uma maior, ou menor, dificuldade na criação de cenários NSDL. Não houve o registo de nenhum dado com esse indicador e apenas indiretamente foi possível ser obtido. A forma encontrada passou, primeiro, por retirar as submissões iniciais de cada cenário e, depois, avaliar as diferenças nos tamanhos dos ficheiros

pertencentes ao mesmo cenário. Definiu-se que nenhuma ou uma pequena diferença (zero a alguns *bytes*) indicam um erro; e que grandes diferenças, da ordem das dezenas de *bytes* ou maior, indicam a adição de novos elementos. Este método é pouco preciso e poderá ter uma margem de erro grande, mas foi o único possível nesta fase. O valor encontrado de ficheiros com alterações de 0 a 10 *bytes* foi de 821, ou seja, uma taxa de erro de 53%. Este valor poderá subir até aos 70% se o limite superior for de 100 *bytes* na análise às diferenças entre ficheiros. Daqui é possível concluir que, sendo metade das submissões para corrigir erros, o processo de edição ou a estrutura da linguagem ainda apresentam algumas dificuldades. É também relevante enquadrar este valor no número final de submissões por cenário, ou seja, este valor implica um número baixo de submissões para obter o cenário correto e final (4-5, aproximadamente) devido a erro.

Os erros referidos no parágrafo anterior devem-se, na sua grande maioria, a pequenas falhas na construção dos cenários, mas foram também detetados dois tipos de erros presentes no processo de validação e transformação. O primeiro tipo de erro, o mais simples, implicava alguma lacuna ou falha na estrutura e/ou no processo de transformação. O acompanhamento feito aos utilizadores pela equipa de desenvolvimento da *Framework* NSDL permitiu detetar rapidamente muitos destes erros e, na grande maioria das situações, a sua correção foi simples. O maior desafio estava na deteção e apenas poderia ser ultrapassado se a variedade de cenários fosse tal que testasse todas as opções possíveis para os cenários de rede. Os números indicados anteriormente para as submissões e cenários distintos garantem que a probabilidade do aparecimento de erro será baixa para as últimas versões dos objetos definidos.

Um segundo tipo de erro, mais complexo, envolve a definição da estrutura dos objetos e a sua adequação às ferramentas. O erro não ocorre na validação e na transformação, mas sim na execução do *OTcl*. Nesta situação temos dois casos: a sintaxe do *OTcl* está incorreta e a simulação não se concretiza, e; a sintaxe está correta, mas a simulação não ocorre da forma esperada. A deteção do primeiro caso é simples, visto que o simulador informará o utilizador. A deteção do segundo caso é mais complicada e apenas acontece após avaliação dos resultados da simulação. A verificação da execução da simulação da forma esperada e da coerência dos resultados terá de ser feita pelo utilizador. A maior preocupação após o desenvolvimento das bibliotecas de código NSDL, neste caso para o NS2, foi realizar o maior número de testes com os utilizadores para serem detetados os erros que ainda estivessem no código. O projeto de TAR, envolvendo os componentes dos DiffServ e um número razoável de alunos, permitiu testar grande parte das opções e, após pequenas correções, conseguir confirmar a geração de cenários corretos e coerentes.

Um último dado do processo de transformação está relacionado com a dimensão das descrições. O ficheiro XML é bastante maior que o *OTcl*, mais precisamente, 2.3 vezes maior. É conhecida a característica do XML que conduz a ficheiros maiores para a descrição dos seus dados. Poderá ser uma limitação em termos de escalabilidade, mas para os cenários testados, a dimensão do ficheiro nunca foi um entrave ao bom funcionamento. Os dados de uma simulação simples são da ordem das dezenas ou centenas de *MBytes*. Um ficheiro para a descrição de um cenário são da ordem dos *KBytes* (o maior encontrado tinha pouco mais de $100 Kbytes$). Os cenários de grande dimensão poderão a partir de certo ponto atrasar um pouco algumas funções, mas, durante os testes realizados, não houve qualquer problema em nenhum dos cenários.

Esta secção apresentou alguns dos dados encontrados no teste da *Framework* NSDL pelos seus criadores. A próxima secção apresenta os resultados de um inquérito aos seus utilizadores.

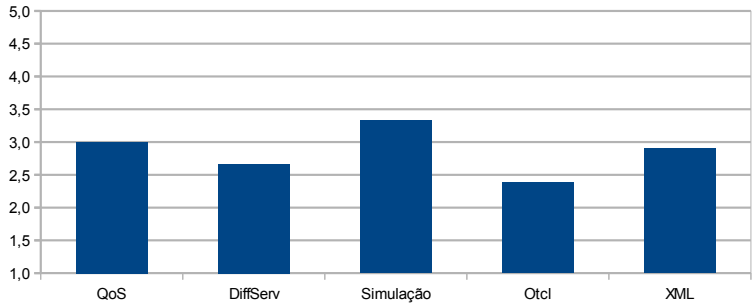


Figura 5.22: Indicadores sobre os conhecimentos dos tópicos envolvidos nos projetos de Tecnologias Avançadas de Redes

5.7.2 Inquérito aos utilizadores da *framework*

Os testes de funcionamento permitiram detetar e corrigir muitos erros do desenvolvimento das bibliotecas da *Framework* NSDL. O retorno de informações por parte dos alunos acelerou a correção das bibliotecas com falhas. O outro momento de participação por parte dos alunos ocorreu no preenchimento de um inquérito e foi igualmente importante para receber informações sobre a relevância, facilidade de utilização e comparação com outras ferramentas.

O objetivo do inquérito foi determinar qual a facilidade/dificuldade na utilização da NSDL por parte dos utilizadores e procurar comparar a criação de cenário na NSDL e no NS2. Através de questões sobre o uso dos vários objetos comuns de cada linguagem, como o nó, a ligação, a aplicação, os eventos, entre outros, procurou-se também confirmar se os objetos mais complexos, seriam efetivamente os objetos mais difíceis de utilizar.

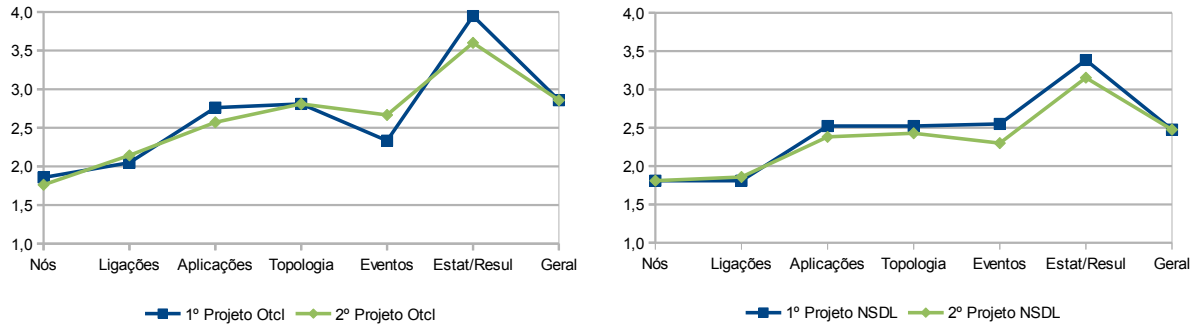
O inquérito foi respondido por 21 alunos, de dois anos letivos. O número de alunos de 09/10 foi 18. O número de alunos de 10/11 foi 3. Os resultados apresentados abaixo resultam dos dados de todos os inquéritos. Houve uma diferença grande nos resultados de ambos os anos, mas como um deles tem apenas 3 alunos, a sua influência nos resultados finais é pequena. Na reflexão aos inquéritos apresentada nos parágrafos seguintes e onde se considera relevante a diferença, tal fato é indicado. O inquérito pode ser consultado no apêndice C.

O inquérito foi desenhado em diferentes secções, tendo cada qual um propósito mais específico. De seguida apresentam-se cada uma das cinco secções do inquérito.

Conhecimentos prévios: os projetos de TAR envolvem o conhecimento de diferentes tecnologias e conceito de base. Procurou-se então saber quais as principais dificuldades dos alunos, principalmente para procurar relacionar com os resultados do resto de inquérito. A ausência de bases mínimas em um ou vários dos tópicos poderia levar a resultados muito fracos e a não se atingirem os mínimos de cada projeto. A Figura 5.22 apresenta os valores para as questões sobre os conhecimentos prévios, onde o valor um significa muito pouco conhecimento e o valor cinco bastante conhecimento.

Como nota adicional, este inquérito foi realizado no final da disciplina de TAR mas esta questão procura saber qual o nível de conhecimento adquirido pelos alunos em alguns tópicos antes do funcionamento da disciplina.

Os resultados mostram o fraco conhecimento dos alunos nos diversos tópicos no início de



(a) Projetos com a linguagem OTcl

(b) Projetos com a linguagem NSDL

Figura 5.23: Níveis de dificuldade na utilização dos objetos em ambos os projetos

TAR. Os tópicos de QoS e DiffServ já tinham sido apresentados anteriormente, mas de forma superficial. As linguagens *OTcl* e XML eram pouco conhecidas, e menos conhecida a primeira. O tópico Simulação (de redes) obteve o resultado mais elevado. Tal era esperado devido a existirem nos programas de disciplinas anteriores a TAR diferentes ferramentas para a construção e simulação de redes, *e.g.*, o *Packet Tracer*.

Apesar de os resultados não serem positivos, ambos os projetos foram concluídos por todos os alunos. A apresentação dos tópicos durante a disciplina e o esforço dos alunos foram suficientes, ultrapassando a limitação inicial nos conhecimentos.

Utilização do NS2/*OTcl*: Neste grupo de questões abordou-se a construção de cenários de simulação com a linguagem *OTcl*. O primeiro projeto, mais simples, utilizava objetos comuns de rede. O segundo projeto, além dos objetos do primeiro projeto, utilizava objetos DiffServ e mais detalhe na análise dos resultados. A Figura 5.23 apresenta os valores apontados pelos alunos na utilização de diferentes componentes da linguagem em ambos os projetos. A gama de valores das respostas permitida aos alunos variou de um a cinco. O valor mínimo indica muita facilidade na utilização de um determinado objeto e o valor máximo considera bastante complexa a configuração desse objeto.

Uma primeira nota é a proximidade dos valores entre o primeiro e o segundo projeto, sendo o grau de complexidade e de quantidade de trabalho de ambos muito diferentes. Deduz-se que, apesar das dificuldades iniciais aquando do primeiro projeto, as dificuldades com os cenários mais complexos durante o segundo projeto não foram, em termos relativos, maiores, diminuindo mesmo em alguns casos (aplicações e estatísticas/resultados). O valor 'Geral' refere-se a uma questão sobre a generalidade dos objetos e, neste caso, obtém o mesmo resultado em ambos os projetos. Como esperado, a programação das estatísticas/resultados da simulação é o componente mais complexo de criar. A criação dos nós e ligações são os objetos mais simples de configurar.

Utilização da NSDL: as questões sobre a NSDL versaram sobre os mesmos componentes do grupo anterior, para efeitos de comparação, e ainda sobre outros objetos apenas existentes na NSDL (*templates*, *views* e *scenarios*). A Figura 5.23(b) apresenta os resultados da NSDL, mas apenas para os elementos comuns a ambas as linguagens e onde os valores um a cinco

têm o mesmo significado descrito para os resultados do *OTcl* (Figura 5.23(a)).

Os resultados foram muito semelhantes aos obtidos para o *OTcl*. Ambas as curvas, primeiro e segundo projeto, estão próximas, crê-se, pelas mesmas razões verificadas no *OTcl*. A única diferença é uma tendência de maior facilidade no segundo projeto de NSDL (quatro objetos mais simples de definir no segundo projeto de NSDL versus três objetos no *OTcl*). Os componentes mais simples e complexos repetem-se com os do *OTcl*, e este era o resultado esperado. Ou seja, no geral não se encontram diferenças grandes entre ambas as linguagens exceto na média global dos seus valores, 2.6 para o *OTcl* e 2.3 para a NSDL, o que indica uma maior facilidade na utilização da NSDL, mas pouco pronunciada, sendo por isso pouco relevante. Em ambos os casos, a grande maioria dos alunos usou editores de texto simples e pode estar nesse facto a proximidade de resultados. Enquanto para o *OTcl* não há ambientes gráficos, para a NSDL é esperado que a introdução de ambientes gráficos permita uma mais simples construção de cenários.

Neste ponto existe uma diferença de algum relevo nos resultados dos inquéritos de ambos os anos, mas, se forem retirados os inquéritos do ano letivo 10/11 (3 inquéritos), as diferenças mantém-se iguais, apenas apresenta os valores globais mais baixos, 2.4 para o *OTcl* e 2.0 para a NSDL. Ou seja, apesar de resultados um pouco diferentes, as conclusões são as mesmas. Os valores dos elementos NSDL não existentes no *OTcl* – *templates*, *views* e *scenarios* – tiveram valores próximos da média global já apresentada, indicando portanto, uma facilidade/complexidade similar com a configuração dos restantes objetos. Estes componentes têm ainda a característica de terem sido os menos usados, ou seja, enquanto para os restantes objetos houve a indicação da sua utilização por todos os alunos, estes foram usados apenas por parte dos alunos.

Uma opção presente em todas as questões destes últimos dois grupos é 'não utilizou'. Serviu para identificar os componentes mais e menos utilizados. No *OTcl* essa opção foi selecionada apenas 0,8% das vezes, ou seja, quase nunca e por isso sem qualquer efeito na avaliação do *OTcl*. No caso da NSDL a opção 'não utilizou' foi selecionada 11% das vezes e a sua escolha concentrou-se muito em apenas alguns componentes (*templates*, *views* e estatísticas/resultados). Os elementos <templates> e <views> são opcionais e, por isso, a sua não utilização por alguns alunos não é inesperada (25% dos alunos não usaram estes componentes). O componente estatísticas/resultados permitia poucas opções e por isso acabou também por ser pouco utilizado (38% dos alunos não definiram as estatísticas). Pela sua importância no ciclo de uma simulação, este componente deverá ser revisto e aprofundado em versões futuras.

Ainda no grupo de questões da NSDL, os alunos foram questionados sobre a utilidade dos componentes da NSDL não existentes no *OTcl*. Apesar de no parágrafo anterior se referir que parte dos alunos não usou alguns dos objetos NSDL, a opinião dos alunos foi muito favorável à sua existência para a descrição de redes de dados. A Figura 5.24 apresenta os valores para cada um dos componentes. Destaca-se ainda o valor muito positivo dado a 'Múltiplos Cenários' por este elemento ser uma das características principais da *Framework* NSDL.

Comparação das linguagens: esta secção pediu aos alunos para compararem diretamente

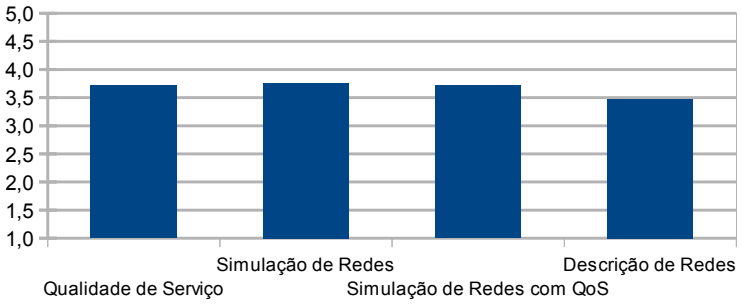


Figura 5.24: Utilidade e relevância de componentes existentes apenas na NSDL

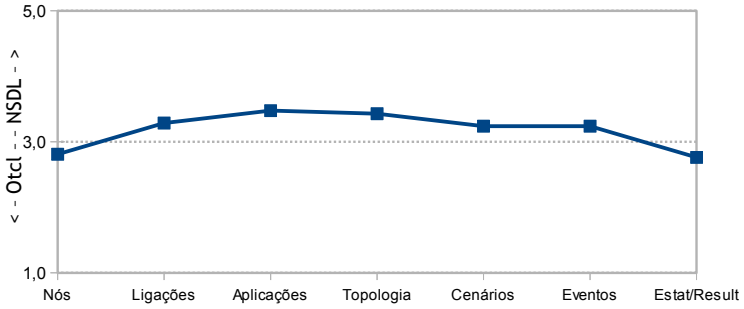


Figura 5.25: Facilidade de utilização de cada uma das linguagens e para cada objeto

ambas as linguagens, objeto a objeto. O objetivo deste grupo de questões era confirmar os resultados obtidos anteriormente. A Figura 5.25 apresenta os valores que indicam para cada objeto qual foi a linguagem considerada mais simples de usar.

Sendo a gama de valores entre um e cinco, onde um indica mais facilidade do *OTcl* e cinco mais facilidade da NSDL, pode-se considerar o valor três como indicador de igual dificuldade/facilidade para as duas linguagens. Os nós e as estatísticas/resultados estão abaixo de três, indicando uma maior facilidade do *OTcl*. No caso dos nós deve-se a ser o objeto mais simples do *OTcl*, onde a sua definição é simples e muito curta. O resultado das estatísticas/resultados deve-se a este componente na NSDL não permitir muitas opções, sendo por isso pouco escolhida.

Os restantes objetos tiveram uma certa tendência para a NSDL, mas sem grande intensidade. Não é um resultado inesperado e vai ao encontro das conclusões retiradas anteriormente no que se refere à utilização dos objetos.

Uso de duas linguagens: O último grupo de questões procurou saber qual o resultado da utilização de duas linguagens na aprendizagem dos conceitos e tópicos de TAR.

O resultado foi no geral positivo, em particular para a simulação de redes, o tópico explorado durante as aulas práticas com as duas linguagens. O tempo despendido a compreender as duas linguagens talvez seja a razão por estes resultados não serem melhores.

A exploração do mesmo cenário por duas ferramentas leva a um menor aprofundamento visto ser necessário repetir certas tarefas, diminuindo a possibilidade de explorar opções mais complexas se fosse apenas utilizada uma linguagem. Mas, por outro lado, reforça a compreensão

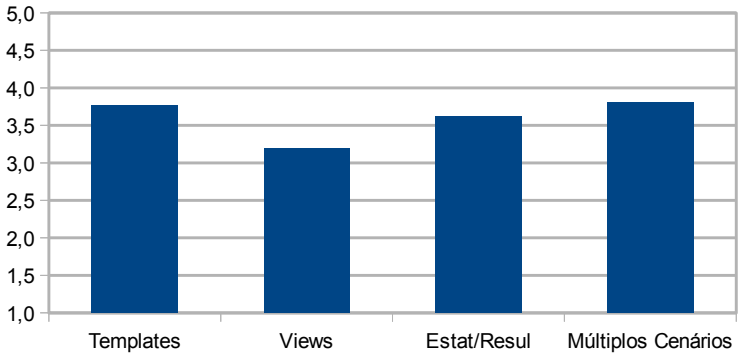


Figura 5.26: Opinião sobre a aprendizagem dos tópicos de TAR com uso de duas linguagens para a simulação de redes

de alguns conceitos, visto as duas linguagens serem muito diferentes e, assim, apresentarem o mesmo conceito sobre diferentes perspectivas, o que foi um fato com os objetos da arquitetura DiffServ.

No final do inquérito, pediu-se aos alunos, numa caixa de texto aberto, que adicionassem qualquer opinião sobre a NSDL e o seu uso em TAR. Apenas 8 alunos (aproximadamente 40%) escreveram um comentário neste espaço. Os comentários positivos foram 4, e destacam a simplicidade e a abstração da NSDL como principais vantagens. Os comentários negativos, também 4, apontam algumas limitações nas ferramentas disponibilizadas que conduzem a muito trabalho, a pouca documentação com exemplos de cenários feitos em NSDL, e, os erros ainda encontrados na linguagem, que se esperava um pouco mais madura. Alguns comentários são esperados, outros dão indicações importantes sobre quais os próximos passos para a ferramenta, de onde se destaca uma melhor documentação para ajudar à adoção da NSDL.

A possibilidade de ter um grande número de utilizadores a experimentar e a testar a linguagem NSDL foi muito vantajoso para este projeto. Permitiu detetar erros e lacunas nas definições dos objetos que, de outra forma, seria pouco provável detetar. E as opiniões dadas no inquérito permitiram obter dados importantes sobre a utilização e relevância da NSDL para a simulação de redes, com e sem QoS, e sobre a construção de descrições de redes.

Na secção seguinte este capítulo é terminado com algumas conclusões do trabalho realizado na aplicação da *Framework* NSDL ao simulador de redes NS2.

5.8 Conclusões

A implementação do perfil NS2 em termos de validação e transformação foi um projeto importante para a *Framework* NSDL, visto ter sido a sua primeira utilização de uma forma mais abrangente e ter ocorrido num contexto prático relevante. A abrangência do desenvolvimento em termos de objetos e componentes para a simulação na ferramenta NS2 cumpriu com os seus objetivos iniciais, conseguindo-se no final uma ferramenta completa para a edição, execução e visualização de simulações no simulador NS2. A sua validação e teste por muitos utilizadores permitiu detetar e

corrigir muitos erros e limitações presentes nas bibliotecas para a validação e para a transformação, e obter uma versão funcional da *framework*.

A escolha dos DiffServ revelou-se adequada visto ter obrigado à definição de cenários bastante mais complexos que as redes com fios comuns e, assim, testar os princípios de conceção da *Framework* NSDL de uma forma mais alargada e aprofundada. A definição em NSDL dos objetos DiffServ foi um desafio grande e conseguir implementar a sua transformação para o NS2 foi uma tarefa complexa. As ferramentas relacionadas com suporte ao DiffServ apresentam limitações, que a *framework* ultrapassa, e de onde se destaca a possibilidade de descrever cenários DiffServ mais próximos da realidade. Algumas das ferramentas encontradas permitem simular cenários DiffServ, mas apenas para cenários pré-definidos [66, 176] ou limitadas em termos de parametrização.

A estrutura para a validação e para a transformação segue uma organização hierárquica. Sendo ambas as estruturas implementadas em linguagens da família do XML, a proximidade em termos de manipulação e compreensão permitiu simplificar todo o processo. Trouxe algumas limitações e surgiram alguns problemas, que seriam resolvidos de forma simples por linguagens de programação genéricas, mas, na grande maioria dos casos, permitiu definir para a linguagem NSDL as estruturas de validação completas e as regras necessárias para a transformação de um cenário. A validação está implementada apenas em XML *Schemas* (XSD) e a linguagem suporta todas as características e regras necessárias para a descrição das mais variadas redes. A transformação, apesar de maioritariamente implementada em XSL *Transformations*, já foi um pouco mais problemática devido a limitações da linguagem e também a diferenças nos interpretadores (parsers) que produziam, por vezes, resultados distintos para os mesmos cenários. Obrigou a algumas adaptações e exceções, como por exemplo, o módulo *helpers*, mas, em futuras versões, estas limitações poderão ser ultrapassadas.

A geração automática de código implementada não cobre todos os cenários de rede possíveis no simulador NS2, no entanto, avaliando a documentação e os exemplos que serviram de base para o trabalho, o número de cenários possível de transformação para o perfil NS2 é bastante abrangente. Os cenários com fios podem ser construídos em NSDL com um número praticamente ilimitados de objetos. No caso dos DiffServ a descrição permitida pela NSDL ultrapassa as capacidades do NS2, aceitando a descrição de cenário muito mais complexos, de entre vários casos, por exemplo, na dimensão da tabela de policiamento e no número de filas suportadas.

Os alunos de TAR, ao terem conhecimento do simulador NS2, permitiram aprofundar o teste da *Framework* NSDL criando e convertendo cenários para o NS2. Muitos dos erros e das faltas foram corrigidos devido ao seu empenho na utilização da *framework*. Após a utilização, as opiniões dos alunos deixadas num inquérito permitiram detetar os pontos fortes e fracos da *framework* e assim melhor significativamente a sua qualidade. Destaca-se o apontar das dificuldades em criar cenários em ambientes não gráficos e ficar a conhecer os objetos mais simples e mais difíceis de configurar.

Capítulo 6

Redes Sem Fios com o *Network Simulator 3*

6.1 Introdução

Os avanços técnicos atingidos nos últimos anos sobre as redes sem fios levam estas a ocupar cada vez mais um espaço nas opções de conectividade entre os diversos equipamentos existentes. A funcionalidade de uma operação sem fios e a flexibilidade em conseguir rapidamente providenciar ligação em locais particulares são duas das razões para a adoção cada vez maior das tecnologias de conectividade sem fios.

O simulador de redes *Network Simulator 3* (NS3) permite a simulação de redes com e sem fios e foi escolhido como ferramenta para ilustrar a integração proporcionada pela NSDL para o domínio das redes sem fios. Foram também implementadas as bibliotecas para a simulação de redes com fios simples, mas, neste capítulo, é dado destaque à implementação de simulações de redes sem fios, nomeadamente Wi-Fi, WiMAX e LTE.

De seguida descreve-se a organização do resto do capítulo. A secção 6.2 introduz o simulador de redes *Network Simulator 3*. A secção 6.3 apresenta as tecnologias de redes sem fios implementadas na *Framework* NSDL. A secção 6.4 introduz os objetos definidos na NSDL para representar as redes sem fios. A secção 6.5 apresenta alguns casos de estudo utilizando os simuladores *Network Simulator 2* e *Network Simulator 3*. Por fim, na secção 6.6 são apresentadas as conclusões da implementação deste projeto.

6.2 *Network Simulator 3*

O *Network Simulator 2* (NS2) deixou de ter, a partir de 2004, projetos dedicados para o seu desenvolvimento e atualização. No entanto, um grupo de investigação envolvendo a Universidade de Washington, o *Georgia Institute of Technology*, o ICSI *Center for Internet Research*, com a colaboração do grupo de investigação *Planete* do INRIA *Sophia-Antipolis*, submeteram a financiamento de um projeto para uma evolução ao NS2 designada por *Network Simulator 3* (NS3), um novo simulador de eventos discretos para a investigação e o ensino de redes de computadores. No âmbito deste projeto ainda foi prevista a continuidade do suporte ao NS2.

Os objetivos gerais do NS3 mantêm algumas características do NS2, como ser *software* livre

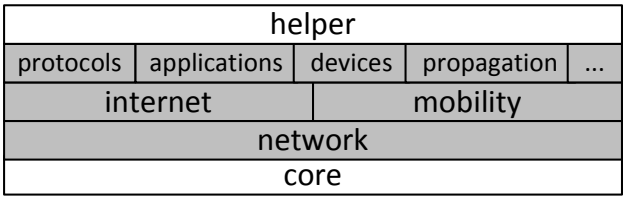


Figura 6.1: Módulos do simulador NS3

e de distribuição gratuita, com ênfase no desenvolvimento pela sua comunidade de utilizadores. Importante também no NS3 foi não repetir as limitações do NS2, principalmente em termos de escalabilidade, extensibilidade, modularidade e emulação. A organização e a existência de documentação completa e atualizada é outro dos objetivos para o projeto [110].

O NS3 utiliza a linguagem C++ para a implementação do núcleo e dos modelos, e também para a descrição da simulação. A utilização de uma linguagem apenas visa simplificar a utilização do NS3 e, mais importante, facilitar a depuração das simulações. A simulação é assim um programa C++ que é compilado e executado. De forma opcional, é possível utilizar a linguagem *Python* [218]. O uso das linguagens de programação genéricas permite incorporar todas as suas vantagens no desenvolvimento de simulações. O sistema Waf [179], uma *framework* baseada em *Python*, é utilizado para a configuração, compilação e instalação das aplicações.

Além do NS2, o NS3 foi buscar conceitos, e código, aos simuladores *Georgia Tech Network Simulator* (GTNets) [227] e ao *Yet Another Network Simulator* (YANS) [148]. Ao primeiro, GTNets, deve-se a melhor escalabilidade para a simulação de grandes redes [276]. Ao YANS devem-se muitos modelos para as redes sem fios. O sistema utilizado para a documentação completa do simulador é o Doxygen [71]. O NS3 está organizado em módulos da forma apresentada na Figura 6.1.

O módulo '*core*' contém as bibliotecas C++ para a simplificação da programação, para a gestão dos eventos, o agendamento e o cálculo do tempo da simulação. Os módulos cinza contêm os vários modelos para os diversos objetos de redes. O módulo '*network*' contém as classes base, como o *Node*, *NetDevice*, endereçamento (IPv4, MAC, entre outras), filas e *sockets*. Contém ainda a manipulação dos pacotes (*tags* e cabeçalhos) e dos seus formatos para exportação (*pcap* e ASCII). Os módulos '*core*' e '*network*' formam uma base para a simulação de um qualquer tipo de rede, não apenas Internet [110].

O módulo '*mobility*' contém os modelos de mobilidade estático, aleatório, entre outros. Os restantes módulos cinza contêm muitos e diversos modelos de protocolos, aplicações, dispositivos, propagação, entre outros. Estes últimos são independentes dos módulos '*network*', ou seja, podem ser usados em vários tipos de redes.

O módulo '*helper*' permite invocar todos os objetos dos restantes módulos de forma mais simples, encapsulando muitos dos detalhes, e programar em C++, ou *Python*, a simulação pretendida. No caso de a simulação ser descrita em *Python*, esta é convertida automaticamente para C++ através do projeto *Python Bindings Generator* [46].

O NS3 tem como objetos fundamentais para a descrição de topologias de rede os seguintes elementos:

- *Node*, para a representação de um computador ou um dispositivo com uma placa mãe,

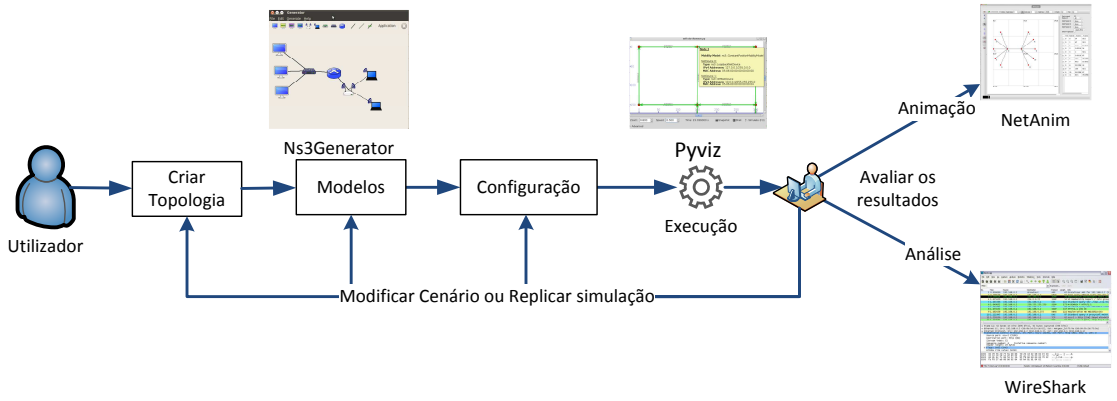


Figura 6.2: Estrutura de uma simulação com o *Network Simulator 3*

memória, processador e *interfaces* de entrada e saída;

- *Application*, responsável pela geração e consumo de pacotes e instalado num Node e pode interagir com diversas camadas de rede;
- *NetDevice*, é uma placa de rede que pode ser ligada a um *interface* de entrada e saída de um *Node*, e;
- *Channel*, uma ligação física entre múltiplos *NetDevice's*.

Na escolha dos objetos fundamentais houve a preocupação de representar os objetos comuns utilizados pelos utilizadores num sistema de comunicação de dados reais.

O projeto base do NS3 não inclui o desenvolvimento de *interfaces* gráficos para construir, depurar, executar e visualizar simulações, deixando para terceiros o desenvolvimento desses componentes [110]. O sítio da Internet do NS3 apresenta várias ferramentas gráficas para a realização de diversas tarefas do ciclo de simulação: a ferramenta *NS3 Generator* [277] permite a geração de código C++ a partir de um ambiente de *drag-and-drop*; a ferramenta *NetAnim* [228] realiza a animação da simulação a partir de ficheiros XML construídos durante a execução da simulação; e, o *PyViz* [48] apresenta a topologia e a dinâmica da rede durante a simulação, ou seja, em tempo real. Apesar de já permitirem a realização de muitas tarefas, as ferramentas ainda têm poucas funcionalidades, principalmente para a configuração de cenários de simulação.

A Figura 6.2 apresenta a estrutura de uma simulação com o NS3, e algumas das ferramentas gráficas que permitem já realizar algumas funções.

O processo de simulação no NS3 não é muito distinto do processo necessário em outros simuladores. O primeiro passo, descrição da simulação, pode ser dividido em três fases: Criar a *Topologia*, escolher os *Modelos* e fazer as *Configurações*. A ferramenta *NS3 Generator* permite realizar a descrição das redes e a configuração de alguns objetos graficamente e gera automaticamente o código C++; A escolha dos modelos em cada objeto enriquece a rede com os componentes que irão operar sobre a rede; Por fim, é realizada a configuração complementar da simulação como os valores dos atributos, modificação dos valores por defeito e os variáveis globais/ambiente.

O passo 'Execução' compila e executa o código C++ com a simulação. O uso de apenas uma linguagem simplifica, se necessário, a depuração do código para a deteção e correção de falhas na

simulação. Nesta fase pode ainda ser usada a emulação, integrando dados de redes reais e em funcionamento no ambiente do simulador. A aplicação *PyViz* permite visualizar graficamente o decorrer da simulação.

Os últimos passos, Animação e Análise, permitem, pós-simulação, visualizar sob a forma de uma animação todos os eventos da simulação com a ferramenta *NetAnim*, e analisar os dados da simulação de uma forma semelhante à análise de uma rede real, recorrendo a um analisador de pacotes, o *Wireshark* [50], ou a formatos e ferramentas do utilizador. Se existir a necessidade de reformular algum aspeto da simulação, o processo retorna ao um ponto anterior e é modificado o código. Pode ser ainda repetida a simulação, mas de forma independente, para reforçar os resultados, através da alteração de alguma das variáveis.

As principais vantagens do NS3 relativamente a outros simuladores e, principalmente, ao NS2 é a sua escalabilidade [276] e a sua arquitetura de desenvolvimento que permite uma melhor qualidade do código existente [89]. Surge assim como um simulador em crescente utilização, em consequência de ter conseguido agregar as melhores características de outros trabalhos e ter procurado manter a comunidade do NS2 com a utilização de alguns dos seus modelos. A falta de ferramentas gráficas flexíveis e completas para a descrição de simulações é a principal lacuna apontada ao NS3.

No âmbito do trabalho desta tese foi implementado um processo para a construção de topologias para as redes sem fios e para a geração do código C++ para o NS3. A próxima secção apresenta algumas das redes sem fios implementadas.

6.3 Redes Sem Fios

Como o próprio nome indica, as redes sem fios caraterizam-se pela inexistência de qualquer cabo para a comunicação entre os seus nós. Em substituição, as tecnologias utilizadas para a transmissão das mensagens entre os equipamentos são as ondas de radio, as ondas de infravermelhos e/ou os raios laser. A classificação mais utilizada para diferenciar as redes sem fios é baseada na distância entre os seus nós e, a cada grupo, é, normalmente, associado um conjunto de tecnologias ajustadas em termos de distância e de velocidade de transmissão requeridas por cada cenário. As classificações criadas são as seguintes:

- Redes Sem Fios Pessoais (*Wireless Personal Area Network* – WPAN): são redes onde os seus equipamentos operam próximos uns dos outros, na ordem dos poucos metros (máximo 10 metros), e pretende-se que a comunicação não seja detetada a partir dessas distâncias. Alguns exemplos das tecnologias que suportam as redes WPAN estão na família da norma 802.15 do *Institute of Electrical and Electronics Engineers* (IEEE) e são o *Bluetooth* [115] e o *ZigBee* [116];

- Redes Sem Fios Locais (*Wireless Local Area Networks* - WLAN): São das redes sem fios mais comuns e permitem a comunicação entre os equipamentos a grande velocidade (11-54+ Mbps) e a distâncias até algumas centenas de metros. A norma IEEE 802.11/Wi-Fi [117] descreve todas as características necessárias para a implementação destas redes;

- Redes Sem Fios Metropolitanas (*Wireless Metropolitan Area Network* - WMAN): São redes com áreas superiores às WLAN e podem abranger uma cidade e/ou um grande campus de

uma universidade. São também usadas para a interligação de WLAN's ou LAN's e designadas como redes de Banda Larga Sem Fios (*Wireless Broadband*). As suas características estão definidas na norma IEEE 802.16 e o nome comum que tem sido utilizado para a referir é "WiMAX", a partir do termo *Worldwide Interoperability for Microwave Access* do grupo WiMAX *Forum Industry Alliance* [77, 205], e;

- Redes Sem Fios de Longa Distância (*Wireless Wide Area Network* - WWAN): são as comuns redes celulares utilizadas para a comunicação com telemóveis e, mais recentemente, com os telemóveis de última geração (vulgo smartphones). Inclui as tecnologias *Global System for Mobile Communications* (GSM) [101], *Universal Mobile Telecommunication Systems* (UMTS) [2], *Long-Term Evolution* (LTE) [1, 239] e WiMAX. Esta última já surge nas WMAN, mas algumas melhorias permitiram a sua operação a distâncias superiores, colocando o WiMAX também neste grupo.

As redes com fios são ainda a principal opção para muitos ambientes de redes, principalmente se for necessário ter comunicações de alta velocidade e grande fiabilidade. No entanto, as atuais redes sem fios, apresentam já muitas vantagens, de onde se destacam [94]:

- **Mobilidade**: o utilizador pode operar sobre a rede em diferentes pontos da organização, não estando fixo aos pontos onde pode ligar o seu equipamento à rede com fios;
- **Facilidade e rapidez de instalação**: a instalação de redes com fios é justificada, por exemplo, em situações de difícil instalação, como em edifícios históricos, onde colocar cabos em paredes/chãos sem calhas técnicas é dispendioso e inestético. As redes sem fios permitem levar as redes de dados a esses espaços facilmente;
- **Flexibilidade**: além da mobilidade já referida, existe a mobilidade para outras organizações, e a facilidade em expandir a rede (escalabilidade), e;
- **Custo**: não sempre, mas em alguns casos os custos de uma solução sem fios são inferiores.

Apesar das vantagens indicadas, as redes sem fios apresentam ainda as seguintes limitações comparativamente às redes com fios:

- **Velocidade**: a velocidade de uma rede com fios comum (*fast-ethernet*) é bastante superior e com menos interferências do meio ambiente;
- **Segurança**: os ataques possíveis a uma infraestrutura sem fios são em maior número e mais simples (escuta, bloqueio e desvio), e;
- **Configuração**: normalmente mais complexa de configurar e de manter que uma rede com fios.

A extensão da *Framework* NSDL para as redes sem fios teve nesta primeira fase um aluno de mestrado, Jorge Sousa, envolvido durante o período de 10 meses [248]. A ferramenta escolhida para testar as descrições NSDL de redes sem fios foi o simulador *Network Simulator 3* (NS3) devido à sua crescente utilização no domínio das redes sem fios e por já conter muitos modelos para estas redes. Assim, foi realizada neste projeto a implementação da extensão da *Framework* NSDL com as tecnologias Wi-Fi, WiMAX e LTE. De seguida é feita uma introdução a cada uma dessas tecnologias, destacando os objetos mais comuns em cada uma delas.

Tabela 6.1: Características de algumas normas da família 802.11

Norma Wi-Fi	Frequência (GHz)	Velocidade de Transmissão (Mbps)	Alcance de Transmissão (m)
802.11a (1999)	5	até 54	120
802.11b (1999)	2.4	até 11	140
802.11g (2003)	2.4	até 54	140
802.11n (2009)	2.4/5	150 a 300	250
802.11ac (2014)	5	433 a 6930	?

6.3.1 Redes IEEE 802.11/Wi-Fi

A norma 802.11 do IEEE, também conhecida pelo termo Wi-Fi (de *Wireless Fidelity*), surge em 1997 e já teve desde esse ano incrementos diversos para acomodar novos requisitos e melhoramento técnicos. As redes Wi-Fi são hoje uma das opções mais utilizadas, quer no ambiente doméstico quer no ambiente empresarial. É simples a ligação de um computador, fixo ou portátil, a redes sem fios domésticas, empresariais e mesmo públicas, para a partilha de ficheiros, acesso a serviço e/ou à Internet.

A primeira norma previa velocidades de transmissão de 1 e 2 Mbps. Entre muitas melhorias ao longo dos anos, uma importante atualização da norma surge em 2009 e esta adicionou novas técnicas para conseguir uma maior velocidade, até aos 300 Mbps, e de alcance de transmissão, até aos 250 metros. Em 2012 surgiu o documento mais recente da norma, sem grandes alterações técnicas e mais focado na junção das informações presentes nas diversas atualizações, obtendo assim um documento com toda a informação atualizada [119]. A Tabela 6.1 apresenta as características mais relevantes de algumas das normas da família 802.11 [94, 120].

A última linha da Tabela 6.1 apresenta a norma 802.11ac, ainda em versão de trabalho (*draft*) e a ratificar em 2014 [121], que pretende especificar redes sem fios, dentro da família das redes 802.11, com velocidades de transmissão a partir dos 433Mbps [120]. Resumidamente, introduz novas funcionalidades na camada Física e melhora a camada de Controlo do Acesso ao Meio (MAC, de *medium access control*), comparativamente à norma anterior, o 802.11n [199].

Além das normas para a camada física apresentadas na Tabela 6.1, existem ainda outras normas com outros propósitos, e.g., a norma 802.11e define os aspetos relacionados com a Qualidade de Serviço (QoS) nestas redes; a norma 802.11i especifica os mecanismos de segurança e autenticação; e, a norma 802.11s especifica as configurações para a construção de redes *Mesh*. Além destes aspetos, existe ainda a definição das questões do *roaming*, as definições regionais para a Europa e para o Japão, a gestão de diferentes elementos, a comunicação entre veículos, entre outros.

Os componentes principais de uma rede 802.11 estão apresentados na Figura 6.3 e têm as seguintes características:

- **Estação**: as redes são feitas para permitir a comunicação entre estações, logo, as estações são equipamentos com placas de rede sem fios. Estes equipamentos podem ser fixos ou móveis e como exemplos destes equipamentos temos desde o computador pessoal comum, aos computadores portáteis e os telemóveis de última geração (*smartphone*);
- **Pontos de Acesso (*Access Points* – AP)**: em certas redes servem como elemento intermédio de comunicação entre as estações e permitem também a ligação a outras redes,

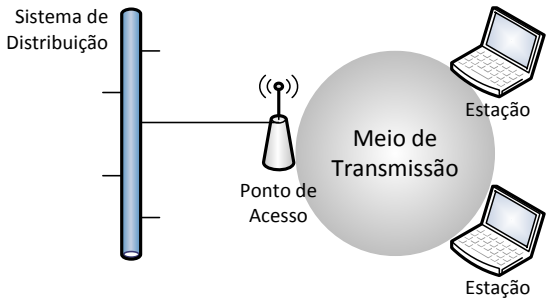


Figura 6.3: Componentes de uma rede 802.11

adaptando a trama da rede 802.11 para o formato adequado à outra rede. Realizam ainda muitas outras funções, como a configuração da estação, o controlo de admissão e a QoS;

- **Meio de Transmissão**: meio físico por onde se movem as tramas de estação para estação. Existem diferentes mecanismos definidos para a camada física que podem ser utilizadas e que suportam o 802.11 MAC, e;
- **Sistema de Distribuição**: são as redes de *backbone* e realizam a interligação de diferentes *Access Points*. A norma não define nenhum sistema particular para realizar esta função, mas a rede *Ethernet* tem sido a opção mais comum para este fim.

Numa rede 802.11 ao grupo de estações que comunicam entre si é designado como o *Basic Service Set* (BSS) e operam numa *Basic Service Area* (BSA). Os BSS podem ser organizados em dois tipos de redes: independentes (Figura 6.4(a)) e Infraestrutura (Figura 6.4(b)).

(a) BSS Independente

(b) BSS Infraestrutura

Figura 6.4: Tipos de organização dos BSS's

O BSS Independente apresentado na Figura 6.4(a) apenas tem estações e qualquer uma pode comunicar com outra, desde que esteja ao seu alcance. Este tipo de redes é tipicamente usado para ligações de curta duração e para comunicações pontuais, como uma partilha de um ficheiro num dado momento apenas entre dois ou mais intervenientes. São também designadas como redes *Ad-Hoc*.

A Figura 6.4(b) apresenta um BSS Infraestrutura e, neste caso, já temos um AP que opera como ponto central para todas as comunicações entre as estações do BSS. A comunicação de uma trama entre duas estações passará sempre pelo AP, tornando a comunicação mais lenta, mas com as seguintes vantagens:

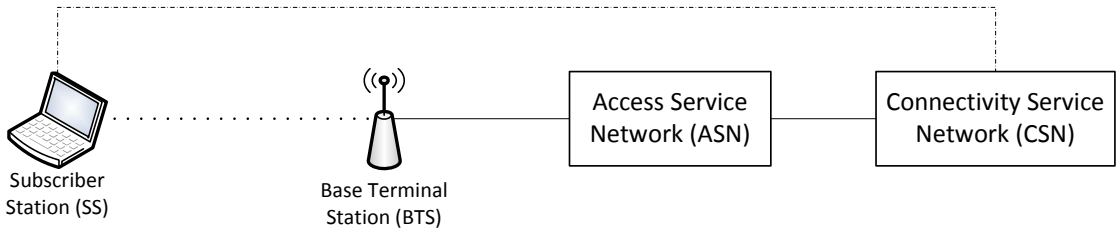


Figura 6.5: Arquitetura simplificada de uma rede WiMAX

- **Distância**: definida não pelo alcance entre as estações, mas entre as estações e o AP, permitindo assim a comunicação entre estações distantes entre si, e;

- **Poupança de energia**: a existência de AP's permite a poupança de energia às estações. Um AP, detetando que uma estação desligou o transmissor, pode guardar as suas tramas e concretizar a sua entrega apenas quando a estação ficar ativa e iniciar a transmissão de tramas para o AP.

Os BSS podem cobrir as áreas de um escritório ou ambiente doméstico, mas são incapazes de alcançar áreas maiores. A cobertura de áreas maiores é feita através da interligação de BSS recorrendo a um Serviço de Distribuição e essa nova estrutura designa-se por *Extended Service Set* (ESS). Os AP's numa ESS operam de forma coordenada de tal forma que um *router* que queira comunicar com uma estação, apenas necessita conhecer o seu endereço MAC, ficando para a ESS determinar a que AP ela está associada.

As redes 802.11 contêm ainda muitos outros pormenores mas, tendo presente o âmbito deste trabalho, apenas é feita a apresentação dos elementos principais.

6.3.2 Redes WiMAX (IEEE 802.16)

As redes WiMAX (de *Worldwide Interoperability for Microwave Access*) oferecem um serviço distinto de outras redes sem fios em termos de velocidade e cobertura. Primeiro, procuram obter as velocidades próximas daquelas atingidas pelas redes Wi-Fi e, segundo, conseguir distâncias de comunicação e os mecanismos de QoS possíveis nas redes móveis 3G/celulares. A norma 802.16-2004 foi definida para a ligação sem fios de estações fixas e permite velocidades de 72Mbps até distâncias de 50Km. A norma 802.16e-2005 define as características para a ligação de estações móveis e são possíveis distâncias de 5 a 15 Km, com velocidades até 40 Mbps [205].

A norma 802.16 utiliza técnicas de rádio avançadas para as camadas PHY e MAC para obter uma grande eficiência na utilização do espectro de frequências e para um maior controlo da QoS. Uma importante característica é a sua arquitetura completamente baseada no protocolo IP, acompanhada também com a possibilidade de operação simples (*plug and play*) e multivendedor. Em consequência, estas características permitem diferentes utilizações e diferentes modelos de serviço [81].

Além dos equipamentos dos utilizadores, designados por *Subscriber Station* (SS), existem ainda outros componentes numa rede WiMAX. A Figura 6.5 apresenta uma arquitetura simplificada da rede WiMAX, e as ligações entre os seus elementos [142].

A *Base Terminal Station* (BTS) pode ser um equipamento fixo ou móvel, instalado dentro ou fora de um edifício e garante a comunicação rádio entre as SS e entre estas e o resto do sistema. Realiza também o encaminhamento das mensagens das SS para os restantes componentes ou para outras redes.

A *Access Service Network* (ASN) inclui as SS e as BTS e a sua principal função é providenciar o acesso rádio aos SS e encaminhar as mensagens de autenticação, autorização e contabilidade (*accounting*) para os níveis superiores, ou seja, para o *Connectivity Service Network* (CSN). É também um ponto de agregação de tráfego das BTS e onde se realizam as funções comuns a todas as BTS. A mobilidade de uma SS entre BTS dentro da mesma ASN, ou ASN's sob o mesmo CSN, é gerida ao nível da ASN e transparente para o CSN. São mantidas as configurações IP e geridas as ligações de *downlink* e *uplink* do SS para a nova BTS.

O CSN é responsável pelas funções de rede que garantem a conectividade IP entre as SS. De entre as diversas funções do CSN temos a tradução de endereços, o registo dos SS locais e externos, a autenticação e o armazenamento dos registos das chamadas. O CSN é também responsável por garantir que o uso da rede por parte dos seus subscritores cumpre com o definido no seu perfil e realizar as funções de controlo de admissão. A gestão das ligações com outros operadores de redes WiMAX, e não só, é gerida também no CSN.

A ligação existente entre a SS e o CSN é uma ligação lógica e está associada às funções de autenticação, autorização, configuração IP e mobilidade. As funções entre o ASN e CSN são a autenticação, a autorização, a contabilidade, a gestão das políticas de utilização e a gestão da mobilidade.

O WiMAX é identificado também como uma tecnologia sem fios 4G por ser um possível sucessor às tecnologias Wi-Fi e 3G, limitadas essencialmente pelas fracas possibilidades de gestão da QoS e de limitada cobertura [285]. Na próxima secção apresenta-se uma tecnologia com características semelhantes, mas com uma origem distinta, o *Long-Term Evolution*.

6.3.3 Redes *Long-Term Evolution*

A tecnologia *Long-Term Evolution* (LTE) pertence à família das tecnologias *Universal Mobile Telecommunications System* (UMTS) e foi uma das primeiras tecnologias de banda larga a baseada em *Orthogonal frequency-division multiplexing* (OFDM). Uma importante característica do LTE é manter a compatibilidade com outras tecnologias de redes móveis, como o *Global System for Mobile Communications* (GSM) e o *High Speed Packet Access* (HSPA), algo não possível no WiMAX. A versão 8 da norma LTE permite taxas de transmissão até aos 300 Mbps para *downlink* e 75 Mbps para *uplink* e permite uma operação flexível da largura de banda até aos 20 MHz. A versão 10, conhecida como LTE-*Advanced* (LTE-A), vem, por um lado, aumentar as taxas e o alcance da transmissão, e, por outro, baixar a latência, conseguindo assim uma melhor experiência na utilização da rede por parte dos seus utilizadores [96].

Um conceito também relevante nas redes LTE é o portador *Evolved Packet System* (EPS *bearer*) que suporta o tráfego IP da *gateway* de uma rede IP até ao equipamento do utilizador (*User Equipment* – UE). O *bearer* é um fluxo de pacotes IP com uma determinada QoS entre estes dois equipamentos. Um fluxo de Voz Sobre IP (VoIP) terá um *bearer* com parâmetros que permitam uma largura de banda mínima e um atraso máximo ajustado aos requisitos deste tipo de tráfego.

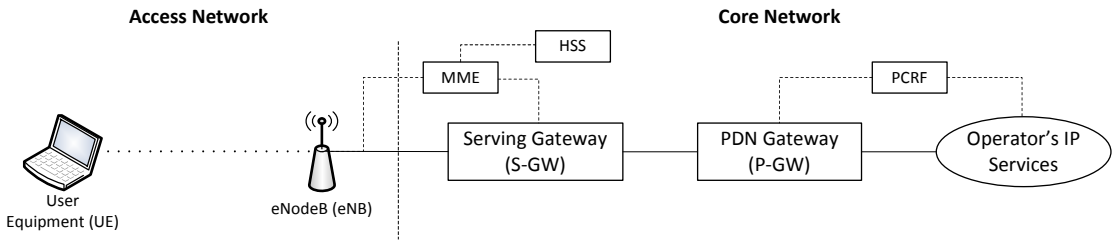


Figura 6.6: Arquitetura simplificada de uma rede LTE

Um fluxo FTP poderá ter um *bearer* sem parâmetros de QoS, recebendo assim um serviço *best-effort*. A Figura 6.6 apresenta a arquitetura simplificada de uma rede LTE, com os componentes relevantes no âmbito deste projeto.

O equipamento do utilizador é designado como UE e a estação de ligação ao resto da rede é o *eNodeB* (eNB). Ambos estes equipamentos fazem parte da rede de acesso. Os componentes principais da rede de núcleo apresentados na Figura 6.6 são o *Serving Gateway* (S-GW), PDN *Gateway* (P-GW) e Serviços IP. De seguida são apresentadas, resumidamente, as funções principais destes componentes, referindo ainda alguns dos outros componentes lógicos também presentes na rede de núcleo [202].

O S-GW é o ponto por onde são transferidos todos os pacotes IP e serve de âncora para a mobilidade local dos *bearers* quando os UE's se movem entre eNB's. O S-GW guarda a informação dos *bearers* quando a estação está *idle* e os pacotes para *downlink* e realiza, ainda, algumas funções administrativas para contabilidade (volume de dados enviados e recebidos).

O P-GW é responsável pela alocação de endereços IP para as UE's e pela implementação da QoS. Executa a filtragem do tráfego *downlink* dos pacotes IP para os diferentes *bearers*, baseado em modelos de tráfego.

O *Mobility Management Entity* (MME) é responsável pela sinalização entre o UE e a rede de núcleo. Envolve as funções relacionadas com a gestão dos *bearers* (estabelecimento, manutenção e término) e funções relacionadas com a gestão da ligação (estabelecimento e segurança).

O *Home Subscriber Server* (HSS) contém as informações do utilizador em termos de perfil de QoS, os PDN's a que se pode ligar e as restrições de roaming. Mantém também, dinamicamente, outras informações do utilizador, como o MME ao qual está no momento ligado. Pode ainda conter outros dados para a autenticação dos utilizadores, como as chaves de segurança.

O *Policy Control and Charging Rules Funtion* (PCRF) coopera com o P-GW em termos de QoS, atribuindo as autorizações sob a forma de identificadores de classe QoS e taxas de transmissão para um dado fluxo, de acordo com o perfil do subscritor.

Esta secção pretendeu apresentar as características e componentes principais de cada uma das tecnologias utilizadas no projeto NSDL. Na secção seguinte será apresentada a representação NSDL dos objetos das redes sem fios já presentes no simulador NS3.

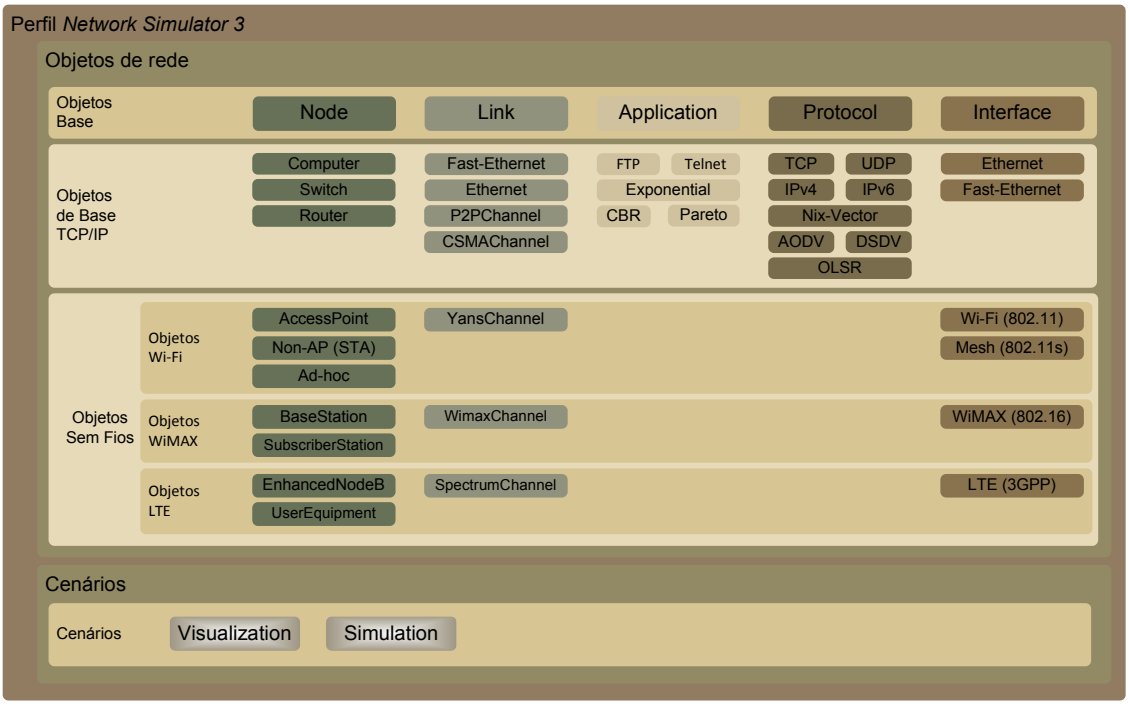


Figura 6.7: Extensão do perfil base com os objetos das redes sem fios

6.4 Representação de Redes Sem Fios NS3 em NSDL

O trabalho descrito nesta secção apresenta a definição de objetos para as redes sem fios na linguagem NSDL. Os objetos definidos procuraram capturar as características principais dos objetos reais e fazer o seu enquadramento nos objetos disponibilizados pela linguagem NSDL. Procurou-se manter a sua caracterização próxima das especificações gerais, mas houve também a necessidade de haver uma aproximação à ferramenta onde estes objetos seriam usados, o simulador NS3. Seria desejável manter as definições independentes da ferramenta, mas, a partir de determinado nível de especificação, o esforço é difícil, deixando ao responsável da especificação o dever de procurar um equilíbrio entre manter a especificação geral ou muito próxima à ferramenta. No caso do trabalho apresentado de seguida, houve alguma proximidade com o simulador e, assim, a especificação dos objetos em NSDL segue as suas potencialidades, e, claro, também as suas limitações.

6.4.1 Extensão ao perfil base NSDL

O perfil base foi estendido de forma a conter os objetos identificados nas redes sem fios apresentadas na secção anterior. A Figura 6.7 apresenta os objetos adicionados ao perfil base e ao perfil TCP/IP no âmbito deste projeto. A contribuição está representada no perfil 'Objetos Sem Fios' e em algumas adições aos 'Objetos de Base TCP/IP', e que são referidas nesta secção.

Os nós adicionados, e presentes na coluna mais à esquerda da Figura 6.7, são os equipamentos principais de cada uma das rede sem fios. Não contêm funcionalidades, mas é onde serão inseridos os restantes objetos que definem cada equipamento, nomeadamente os *interfaces*. Em cada tipo de rede sem fios existe uma estação e um equipamento central responsável pela comunicação entre as estações, e.g., *BaseStation* e *SubscriberStation* para as redes WiMAX.

Na coluna seguinte surgem as ligações com as extensões definidas para cada uma das redes. Nas redes Wi-Fi surge o *YansChannel*, de *Yet Another Network Simulator* e adaptado para o NS3 por [148]. Este modelo permite-nos simular o efeito de propagação de um canal de comunicação sem fios numa rede 802.11, com um determinado modelo de perdas (*friis*, *cost231*, *jakes*, *random*, *rayground*, *logdistance*, *logdistance3*, *nakagami*, *fixedrss*, *matrix* ou ainda *range*) e outro de atraso (*constant* ou *random*) a definir pelo utilizador. De forma semelhante, o trabalho de [84] oferece ao NS3 um modelo para um canal de transmissão virtual para as redes WiMAX e designa-se de *WimaxChannel*. Neste objeto o utilizador pode escolher entre os diversos modelos de perdas definidos pelo autor (*friis*, *cost231*, *random* ou *logdistance*). Por fim, para os *link*'s surge o *SpectrumChannel* para as rede LTE e tem os mesmos parâmetros de configuração que as redes WiMAX, atraso (*random* ou *constant*) e perdas (*lte*).

A nível das aplicações, coluna seguinte, não foi adicionado nenhum objeto novo. É normal que assim seja, visto que a abordagem às redes sem fios se foca nas camadas física e de ligação.

A coluna dos protocolos não tem nenhum objeto na zona das redes sem fios, mas foram adicionados alguns protocolos para o encaminhamento de pacotes para redes *Ad-Hoc* móveis (MANET's). Os protocolos foram os seguintes:

- *Ad hoc On-Demand Distance Vector* (AODV): permite o encaminhamento de mensagens entre nós móveis determinando os caminhos quando necessário e mantendo a informação apenas do próximo salto (*hop*) para cada destino e não da rota completa. Periodicamente refresca a lista dos seus vizinhos [210];

- *Destination-Sequenced Distance Vector* (DSDV): mantém ativamente em tabela a informação do estado de toda a rede a nível dos nós e das ligações e usa como métrica a contagem dos saltos entre origem e destino. Em [180] é feita a sua implementação para o simulador NS3;

- *Dynamic Nix-Vector Routing* (*Nix-Vector*): baseado no protocolo de encaminhamento para redes com fios NIx-Vector (NV) [230] que cria índices de vizinhos (*neighour index*) como forma de simplificação das tabelas de encaminhamentos, em termos de tamanho, poupando espaço de armazenamento. O *Nix-Vector* realiza eficientemente a validação das rotas antes de estas serem usadas, mantém um reduzido número de rotas por destino e o cabeçalho de encaminhamento é menor. Assim, os seus autores apontam-no como um protocolo estável, bastante escalável em termos de dimensão da rede, mobilidade e volume de tráfego [152], e;

- *Optimized Link State Routing Protocol* (OLSR): semelhante ao DSDV, é um dos protocolos mais usados com o NS3. O conceito chave neste protocolo é a utilização de *multipoint relay* (MPR). Estes nós são selecionados de entre todos os nós para realizarem a difusão (*broadcast*) no processo de *flooding*, reduzindo a carga na rede causada por estas mensagens, comparativamente a outros protocolos onde todos os nós participam neste processo [61].

Por fim, a extensão da NSDL para as redes sem fios concluiu com os *interfaces*. Estes objetos são os mais relevantes por incorporarem as funções das camadas física e de ligação destas redes, ponto fulcral na maioria das arquiteturas de redes sem fios. De seguida, e à semelhança de outros capítulos, são apresentados os *interfaces* de uma forma mais detalhada e para cada uma das redes.

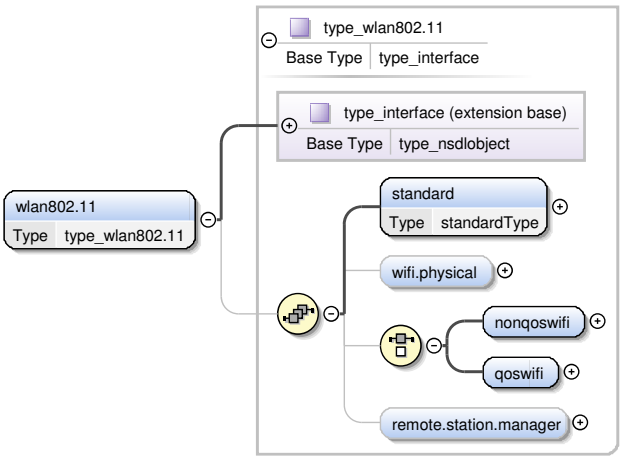


Figura 6.8: Especificação principal de um *interface* para redes Wi-Fi

6.4.2 Objetos das Redes Wi-Fi

Os modelos implementados no simulador NS3 para as redes Wi-Fi, ou 802.11, permitem simular os seguintes aspetos destas redes [194]: redes 802.11 nos modos *ad-hoc* e infraestrutura; as camadas físicas para as normas 802.11a, 802.11b e 802.11g; elementos de QoS (802.11e); diversos modelos de perdas e atraso na propagação do sinal; diversos algoritmos para o controlo da taxa de transmissão; e, redes *Mesh* (802.11s).

As redes Wi-Fi contêm dois tipos de nós: a estação e o AP. O *interface* que deverá ser adicionado a cada nó de um determinado tipo é o mesmo, havendo diferenciação num parâmetro para o tipo. A Figura 6.8 apresenta os elementos principais.

De seguida detalham-se os diversos elementos do *interface* <wlan802.11>:

<**standard**> Define qual a norma a que pertence o *interface*. Os valores possíveis são: *80211a*, *80211b*, *80211g*, *80211_10Mhz*, *80211_5Mhz* ou *holland*;

<**wifi.physical**> Contém a configuração dos parâmetros da camada física (transmissão e receção do sinal) e a ligação para o canal de comunicação *Yans* apresentado na secção anterior;

<**nonqoswifi**> **ou** <**qoswifi**> Os dois parâmetros referidos configuram a componente de gestão dos fluxos de dados e são, em grande parte, iguais. Apenas um pode ser selecionado em cada *interface* e estes parâmetros, primeiro, definem o tipo de nó, ou responsabilidade do nó, e podem se definir como *Access Point* (AP), *Non Access Point* (STA) e *Ad-Hoc Point* (Adhoc). A distinção entre STA e Adhoc está no modo de cada um, sendo Infraestrutura no caso do STA e Independente para o Adhoc. E em segundo, contêm os elementos para a configuração de acesso ao meio (MAC). No caso de o *interface* ter ativa a característica de QoS, deve ser usado o elemento <qoswifi>, e neste existem ainda os seguintes elementos para classificar o tráfego em quatro classes: voz, vídeo, *best-effort* e *background*, e;

<**remote.station.manager**> Permite gerir e manter informação de estado sobre a estação e apenas para o modo infraestrutura. Esta informação permite selecionar os parâmetros de transmissão de cada pacote. Contém também a possibilidade de selecionar e configurar um algoritmo para

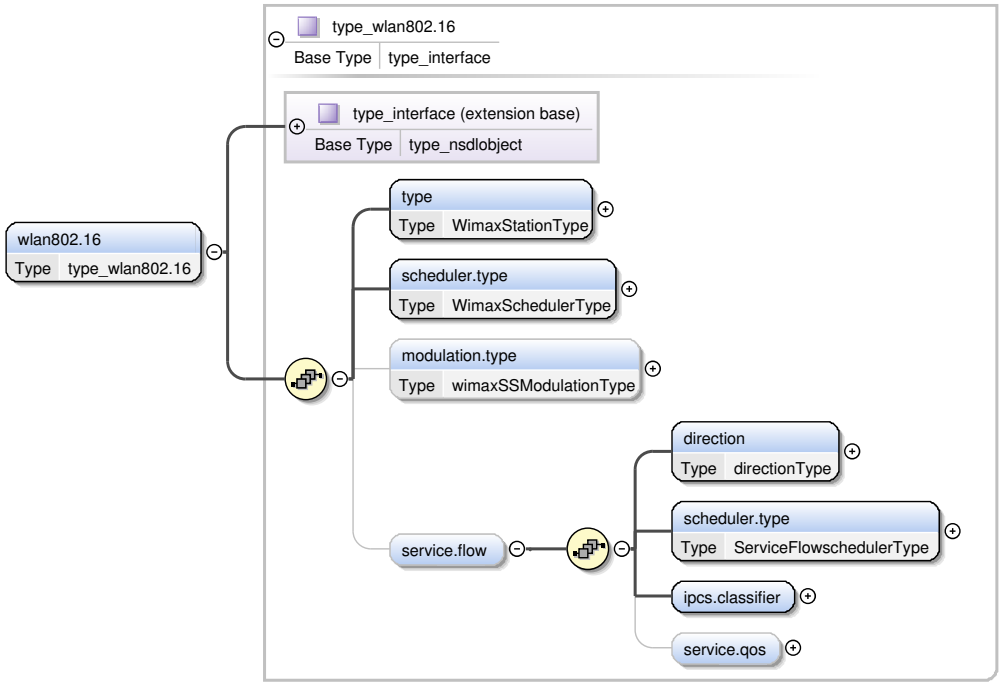


Figura 6.9: Especificação principal de um *interface* para redes WiMAX

o controlo da taxa de transmissão, de entre os seguintes: *Automatic Rate FallBack* (ARF), *Adaptative Automatic Rate Fallback Algorithm* (AARF), *Efficient Collision Detection for Auto Rate Fallback Algorithm* (AARF-CD), *Adaptive Multi Rate Retry* (AMRR), *Collision-Aware Rate Adaptation* (CARA), *ConstantRate*, *Ideal*, *Rate control algorithm developed by Atsushi Onoe* (ONOE), e, *Robust Rate Adaptation Algorithm* (RRAA).

6.4.3 Objetos das Redes WiMAX

Os modelos implementados no simulador NS3 para as redes WiMAX, ou 802.16, têm as seguintes funcionalidades [84, 123]: uma realística e escalável modelação da camada física e do canal; um classificador de pacotes para a camada de convergência IP; agendamento eficiente para *uplink* e *downlink*; suporte a serviços de *multicast* e *broadcast*; e, rastreamento dos pacotes para análise e avaliação.

As redes WiMAX contêm dois tipos de nós: *Base Station* (BTS) e *Subscriber Station* (SS). O *interface* a ser adicionado a cada nó desta rede é o mesmo, permitindo a diferenciação num elemento próprio para o tipo (<type>). A Figura 6.9 apresenta todos os restantes elementos principais.

Além do <type>, temos ainda os seguintes elementos na definição de um *interface* <wlan802.16> ou WiMAX:

<**scheduler.type**> a implementação do WiMAX no NS3 tem as seguintes políticas de agendamento para *uplink* e *downlink*: simples e baseada em prioridades (SIMPLE), *real-time polling service* (RTPS) e, por último e apenas para *uplink*, *migration-based* (MBQOS);

<**modulation.type**> o tipo de modulação para as ondas rádio é apenas aplicável às SS e pode ser de entre um dos seguintes valores: *Binary Phase Shift Keying* (bpsk_12), *Quadrature Phase*

Shift Keying (qpsk_34), e *Quadrature Amplitude Modulation* (qam16_12, qam16_34, qam64_23 e qam64_34), e;

<**service.flow**> é o fluxo, unidirecional, de pacotes gerido com configurações de QoS, como por exemplo a prioridade, a taxa e agendamento. É composto pelos elementos <direction>, que define se o fluxo é *uplink* ou *downlink*; o elemento <scheduler.type>, que se aplica aos fluxos; o elemento <ipcs.classifier>, um classificador baseado nos 5-*Tuples*, e; o elemento <service.qos>, que define os parâmetros de QoS para o fluxo, como a taxa de transmissão, o atraso e a variação do atraso.

6.4.4 Objetos das Redes LTE

Os modelos implementados no simulador NS3 para as redes LTE permitem avaliar os seguintes aspetos destas redes [193]: gestão dos recursos rádio, agendamento de pacotes com QoS, coordenação das interferências *Inter-cell*, e, acesso ao espectro dinâmico. Os requisitos considerados na definição dos modelos LTE no NS3 para cumprir a avaliação apontada têm em conta:

- a nível rádio, a granularidade do modelo vai até ao nível do *Resource Block* para modelar com precisão o agendamento de pacotes e a interferência Inter-Cell;
- o simulador deve permitir dezenas de eNB's e centenas de EU's, não permitindo a modelação até ao nível do símbolo, exigente em termos computacionais e que permitira apenas um eNB e alguns EU's;
- possibilidade de ter diferentes células, com diferentes frequências portadoras e larguras de banda, e;
- ter em conta alguns aspetos da implementação real das políticas de agendamento e do tratamento dos pacotes IP em pacotes da camada *Radio Link Control*, aproximando assim os modelos das implementações reais.

As redes LTE contêm dois tipos de nós: *User Equipment* (UE) e *enhanced NodeB* (eNB). O *interface* que deverá ser adicionado a cada nó de um determinado tipo é o mesmo, havendo diferenciação num parâmetro para o tipo. A Figura 6.10 apresenta os elementos principais contemplados.

Além do <type>, temos os seguintes elementos na definição de um *interface* <lte>:

<**lte.physical**> apenas permite a associação para os canais de comunicação já apresentados nas ligações com o *SpectrumChannel*, e;

<**enb.id**> aponta para o identificador do *enhanced NodeB*;

<**radio.bearer**> é composto pelos elementos <direction>, que define se o fluxo é uplink ou downlink; o <bearer.type>, que permite indicar o tipo de portadora para o sinal; o <ipcs.classifier>, como classificador do pacote e baseado nos 5-Tuples; e, o elemento <bearer.qos>, que permite ao utilizador configurar alguns parâmetros de QoS.

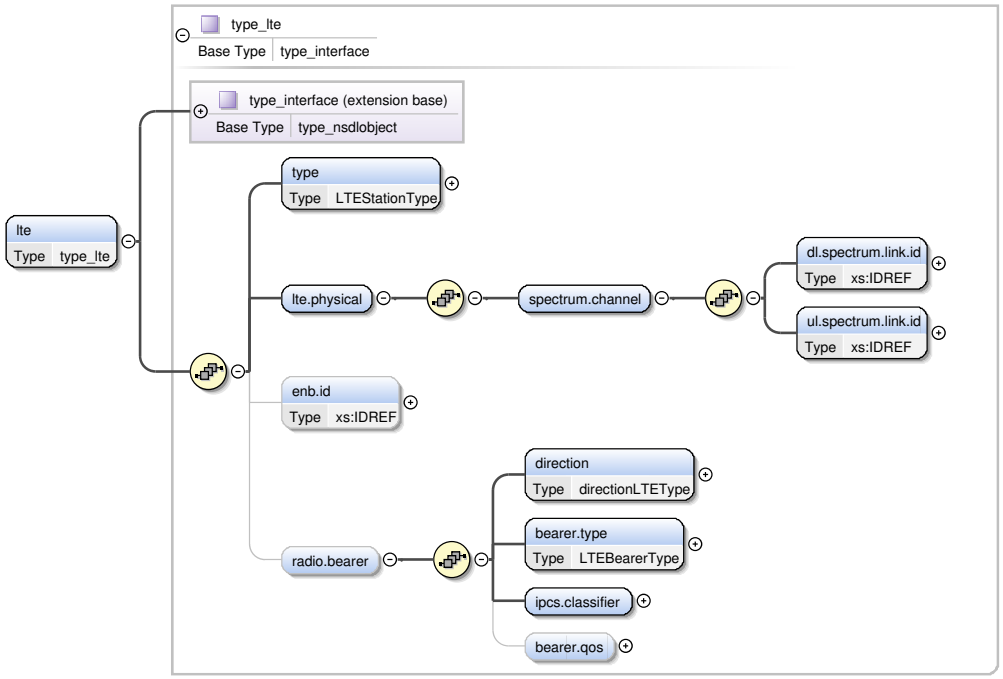


Figura 6.10: Especificação principal de um *interface* para redes LTE

```
1  <!-- Complex Types definition -->
2  <xs:complexType name="type_computer">
3         <xs:complexContent>
4                <xs:extension base="type_node">
5                       <xs:sequence minOccurs="0">
6                              <xs:element name="role" minOccurs="0"/>
7                       </xs:sequence>
8                </xs:extension>
9         </xs:complexContent>
10 </xs:complexType>
```

Figura 6.11: XML *Schema* com as regras de validação de um nó

6.4.5 Validação e Transformação NSDL/NS3

A validação de cada um dos cenários de redes sem fios é feita por intermédio de um conjunto de ficheiros XML *Schemas* (XSD). Cruzando ambas as estruturas, cenário e validação, é verificado, para o primeiro, a sua correção detetando eventuais lacunas e/ou erros. O autor utilizou os ficheiros XSD já criados para a *Framework* NSDL e para o projeto com o NS2 e adicionou os objetos do perfil NS3. A figura 6.11 apresenta um extrato da validação de um objeto NSDL, neste caso um computador.

As Figuras 6.8, 6.9 e 6.10 representam graficamente as estruturas de validação para cada objeto e podem ser representadas num código semelhante ao da Figura 6.11. Devido à dimensão da estrutura de qualquer um dos objetos referidos, optou-se por colocar um exemplo de um objeto mais simples, um computador.

A transformação é realizada com ficheiros XSL *Transformation*, e, apesar de ser novamente usada a estrutura de transformação do NS2, neste caso foi necessário adaptar todo o código para realizar a transformação para a linguagem utilizada pelo NS3, o C++. A linguagem de cada

```
1  <!-- Tratamento dos No's -->
2  <xsl:template match="node">
3          <xsl:value-of select="$lb.n"/>  <!-- line break -->
4          <xsl:text>/* *** Node </xsl:text> <!-- Comenta'rio -->
5          <xsl:value-of select="@id" />
6          <xsl:text> *** */</xsl:text>
7          <xsl:value-of select="$lb.n"/>
8          <xsl:text>Ptr%Node$ </xsl:text>
9          <xsl:value-of select="@id" /> <!-- varia'vel/no -->
10         <xsl:text> = CreateObject%Node$ (); </xsl:text>
11         <xsl:value-of select="$lb.n"/>
12         <xsl:text>/**** Getting node Ipv4 ***/ </xsl:text>
13         <xsl:value-of select="$lb.n"/>
14         <xsl:text>Ptr%Ipv4$ Ipv4</xsl:text> <!-- enderecar -->
15         <xsl:value-of select="@id" />
16         <xsl:text> = </xsl:text>
17         <xsl:value-of select="@id" />
18         <xsl:text>-$GetObject%Ipv4$ (); </xsl:text>
19         <xsl:value-of select="$lb.n"/>  <!-- line break -->
20 </xsl:template>
```

Figura 6.12: XSL *Transformation* com as regras de transformação de um nó

simulador é distinta e não é fácil conseguir algum tipo de reutilização de código por estes terem sintaxes e semânticas muito diferentes. Portanto, o processo seguido para a construção das regras de transformação foi o mesmo do projeto NS2, mas o código criado é novo e específico para o NS3. A Figura 6.12 apresenta um extrato de código para a transformação de uma descrição NSDL para NS3.

O código da Figura 6.12 recebe um elemento XML <node> e, com o seu atributo @id, constrói as linhas de código NS3 que definem um nó e o seu endereço IP versão 4. O resultado de um processo de transformação pode ser visualizado na Figura 6.16, linhas 1 e 34.

Deste modo, a estrutura de validação e de transformação seguida na extensão da NSDL para o NS3 é semelhante à já utilizada na *Framework* NSDL. Os componentes completos são apresentados na Figura 6.13 aos quais foram adotados os princípios da arquitetura *Model-View-Controller* (MVC) [224] para organizar todas as funções e todos os ficheiros.

A Figura 6.13 apresenta no Modelo (*Model*) todos os ficheiros necessários para a validação (*.xsd) e os ficheiros envolvidos na transformação (*.xsl). As funções presentes nas restantes componentes (*Controller* e *View*) permitem ao utilizador aceder às páginas para a submissão do cenário e invocar a validação e transformação desse mesmo cenário.

A transformação de um elemento XML (ou objeto de rede NSDL) para uma linguagem particular, como já referido na secção 5.6 não é uma tarefa complexa. Varia muito de objeto para objeto e pode ser simples e direta ou então necessitar se relacionar com outros objetos e proceder a alguns cálculos. A tarefa onde é necessária mais atenção é na estruturação da sequência de criação de código, tendo presente a globalidade dos elementos necessários e não trocando a ordem em que estes devem/podem surgir. Assim, para realizar a transformação de um cenário NSDL para um script NS3 foram definidos os seguintes passos:

1. Inclusão das bibliotecas C++ necessárias para o cenário a construir;
2. Instanciação de *Helpers* necessários para o cenário;
3. Definição das portas típicas usadas pelas aplicações no NS3;

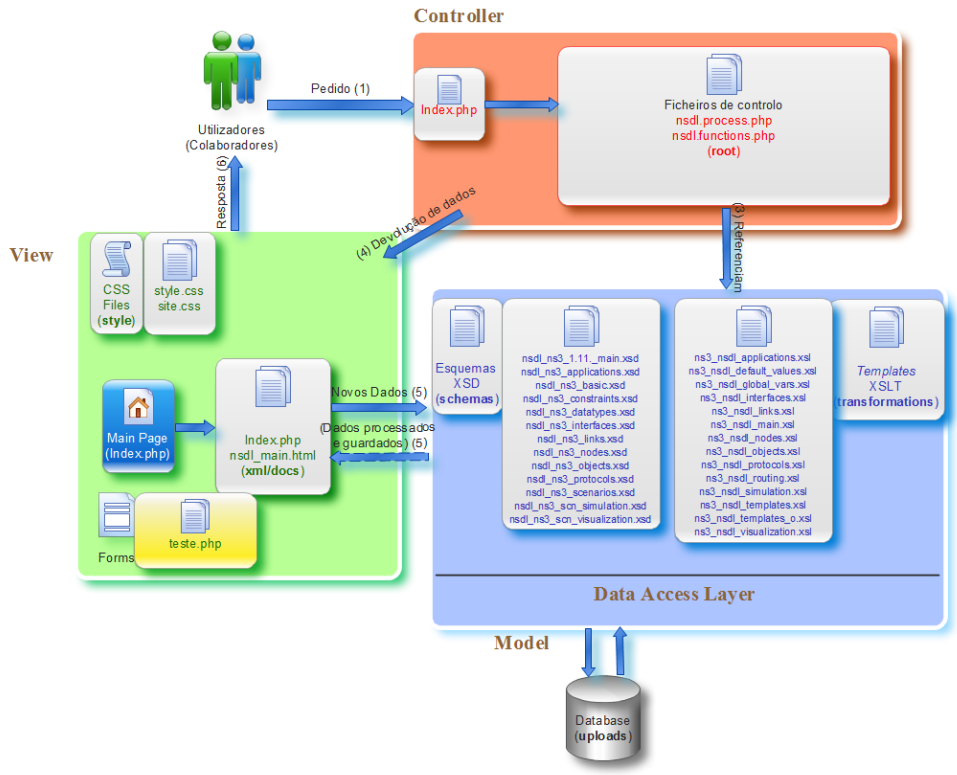


Figura 6.13: Arquitetura de componentes para a validação e transformação NSDL/NS3 [248]

4. Instanciação dos nós e suas características;
5. Associação do módulo Internet e encaminhamento globais;
6. Criação das ligações;
7. Definição dos *interfaces*;
8. Endereçamento IP (versão 4 e 6) dos nós;
9. Definição e agendamento das aplicações (servidor e cliente);
10. Configuração global da simulação;
11. Configuração global do cenário de visualização e dos modelos de mobilidade, e;
12. Configuração dos ficheiros de resultados.

O procedimento normal foi o apresentado, mas para alguns cenários específicos, podem existir mais alguns passos intermédios. Por exemplo, os *Service Flows* e os *Radio Bearers* das redes WiMAX e LTE, respetivamente, são adicionados após o passo 11.

Após a correta validação é executada a transformação que produz o ficheiro C++. Este ficheiro é depois utilizado pelo NS3 para desenvolver a simulação descrita nele e produzirá outros ficheiros com os resultados. No Apêndice D pode ser consultada uma descrição NSDL de uma rede e o resultado da sua transformação para uma descrição C++ para o NS3 seguindo os passos descritos.

Na secção seguinte são apresentados alguns exemplos da utilização dos diversos componentes da *Framework* NSDL/NS3.

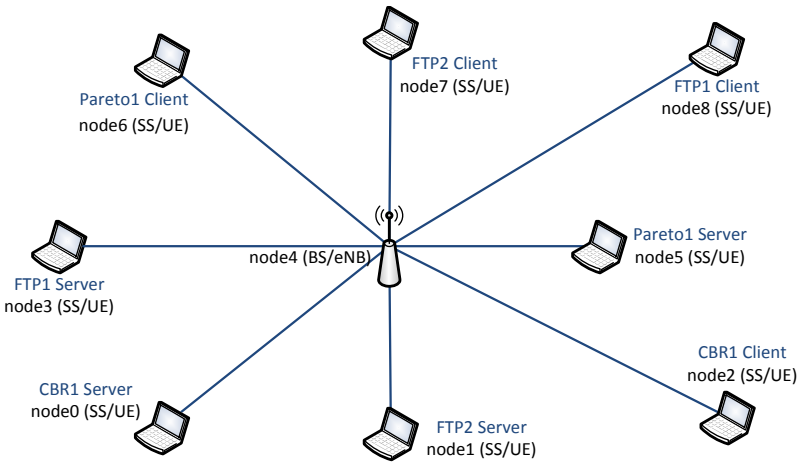


Figura 6.14: Topologia de rede dos cenários WiMAX e LTE [167]

6.5 Casos de estudo

O perfil apresentado na Figura 6.7 mostra todas as redes e objetos possíveis de integrar em cenários NSDL. Os cenários escolhidos para exemplificar o uso da *framework* foram uma rede WiMAX, uma rede LTE e cenários de redes com fios. A escolha procurou ser suficientemente abrangente para o espaço deste capítulo. No caso do WiMAX e LTE estes são apresentados conjuntamente e é realizada uma comparação entre si. No caso do cenário de redes com fios, são usados dois simuladores, o NS2 e o NS3.

6.5.1 Exemplo Redes WiMAX e LTE

O caso de estudo apresentado nesta secção procurou, a partir de uma estrutura semelhante no formato NSDL, criar duas simulações, uma com cada uma das redes e com os parâmetros os mais próximos possíveis para, assim, comparar os modelos de cada tecnologia no simulador de redes NS3. A topologia de rede escolhida pode ser visualizada na Figura 6.14.

A rede é composta por nove nós, tendo o nó central a responsabilidade de encaminhar todo o tráfego entre as restantes estações. O nó central assume o papel de *Base Station* ou *eNodeB* e os restantes nós representam as *Subscriber Station* ou os *User Equipment* para as redes WiMAX e LTE, respetivamente.

Os *link*'s da rede WiMAX foram configurados com OFDM (classe *OfdmWimaxChannel*) e os link's da rede LTE foram configurados com *Single Spectrum Model* (classe *SingleModelSpectrum-Channel*). Em ambas as configurações foram aplicados os parâmetros por omissão. O exemplo de código presente na Figura 6.15 apresenta um extrato de uma descrição NSDL de uma rede WiMAX e o resultado da sua transformação para código C++ é apresentado na Figura 6.16.

Começando pela Figura 6.15, os objetos escolhidos são o nó (@id's *node4* e *node5*), o protocolo IP (@id *ipv4node3*) e o *interface* (@id's *wimaxN4* e *wimaxN5*). Em NS3 basta indicar o endereço da rede do protocolo IP uma vez e é possível configurar os endereços de rede de todos os nós automaticamente com base nessa informação. Os *interfaces*, neste contexto, acabam por ser os elementos mais importantes e contêm a informação do tipo de equipamento, do agendamento e da

```
<node id="node4">
        <ipv4 id="ipv4node3">
                <interface.id>wimaxN4</interface.id>
                <mask>255.255.255.0</mask>
                <net.address>10.1.4.0</net.address>
        </ipv4>
        <wlan802.16 id="wimaxN4">
                <type>bs</type>
                <scheduler.type>simple</scheduler.type>
        </wlan802.16>
</node>

<node id="node5">
        <wlan802.16 id="wimaxN5">
                <type>ss</type>
                <scheduler.type>simple</scheduler.type>
                <modulation.type>qam16_12</modulation.type>
                (...)
        </wlan802.16>
        (...)
</node>
```

Figura 6.15: Descrição NSDL parcial de um cenário de rede WiMAX

modulação.

Na Figura 6.16 está o código C++ para a especificação destes objetos. As linhas 1 e 2 apenas criam os nós, depois surge a configuração dos *interfaces*, linhas 5 a 24) e podemos verificar que o esquema de modelação por omissão é o OFDM (linhas 10 e 21: SIMPLE_PHY_TYPE_OFDM), por não ter sido referido na descrição NSDL. No final do exemplo, linhas 32 a 35, surge a configuração do endereço da rede e, automaticamente, depois a atribuição dos endereços para cada nó. A descrição completa deste cenário de rede WiMAX em NSDL e a sua transformação para o código C++ para uma simulação no NS3 pode ser vista no Apêndice D.

No momento da criação destes testes haviam algumas limitações nos modelos do WiMAX [84] e LTE [214] no NS3, nomeadamente:

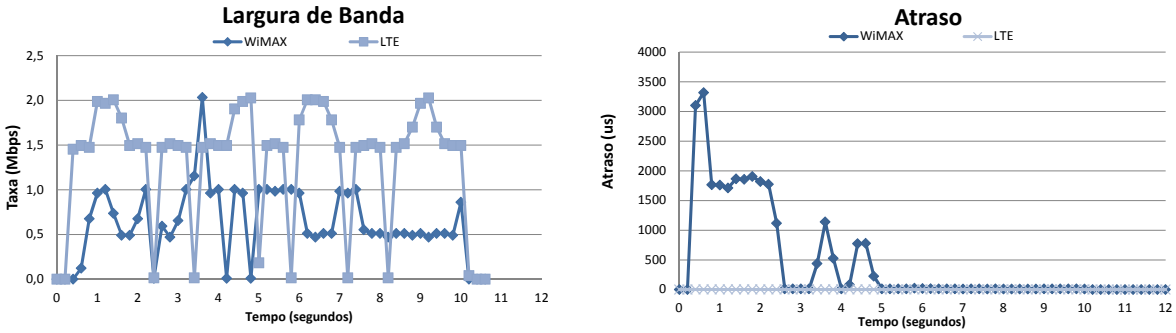
- Nas redes WiMAX existe apenas um *Service Flow* por SS, obrigando a apenas uma aplicação em cada SS, e;
- Nas redes LTE, apenas o agendamento de *downlink* está implementado, logo não é possível testar fluxos do UE para o eNB, permitindo apenas a existência de aplicações no eNB. Esta limitação obriga a uma análise cuidada dos resultados por criar uma distinção grande entre os cenários. O cenário WiMAX terá dois saltos entre os servidores e clientes e o cenário LTE terá apenas um salto entre servidor e clientes.

O cenário tem três servidores e três clientes e definiu-se o início da geração de tráfego TCP aos 0,2 segundos e a geração de tráfego UDP aos 0,6 segundos. Aos 10 segundos termina a geração de todo o tráfego e aos 12 termina a simulação. A Figura 6.17 contém os gráficos para a largura de banda e atraso de ambos os cenários.

Além destes resultados, foi ainda feita a contagem dos pacotes perdidos durante a simulação. O LTE não apresentou perdas porém o WiMAX teve 54% dos pacotes perdidos. Apesar de os resultados do LTE apresentarem um melhor desempenho, o que está de acordo com os outros

```
1  Ptr<Node> node4 = CreateObject<Node> (); /* *** Node node4 *** */
2  Ptr<Node> node5 = CreateObject<Node> (); /* *** Node node5 *** */
3  (...)
4  /***** Interfaces Setting *****/
5  Ipv4InterfaceContainer iwimaxN4;
6  Ipv6InterfaceContainer iwimaxN4v6;
7  WimaxHelper::SchedulerType wimaxN4scheduler = WimaxHelper::SCHED_TYPE_SIMPLE;
8  NetDeviceContainer dnode4d = wimax.Install ( NodeContainer (node4) ,
9                       WimaxHelper::DEVICE_TYPE_BASE_STATION ,
10                      WimaxHelper::SIMPLE_PHY_TYPE_OFDM , wimaxN4scheduler);
11
12 Ptr<BaseStationNetDevice> node4BS
13 node4BS = dnode4d.Get(0) ->GetObject<BaseStationNetDevice> ();
14
15 /***** Interfaces Setting *****/
16 Ipv4InterfaceContainer iwimaxN5;
17 Ipv6InterfaceContainer iwimaxN5v6;
18 WimaxHelper::SchedulerType wimaxN5scheduler = WimaxHelper::SCHED_TYPE_SIMPLE;
19 NetDeviceContainer dnode5d = wimax.Install ( NodeContainer (node5) ,
20                      WimaxHelper::DEVICE_TYPE_SUBSCRIBER_STATION ,
21                      WimaxHelper::SIMPLE_PHY_TYPE_OFDM , wimaxN5scheduler);
22
23 Ptr<SubscriberStationNetDevice> node5SS
24 node5SS = dnode5d.Get(0) ->GetObject<SubscriberStationNetDevice> ();
25
26 /***** Modulation Type Setting *****/
27 node5SS->SetModulationType (WimaxPhy::MODULATION_TYPE_QAM16_12 )
28
29 (...)
30 /*** Protocol creation ***/
31 /*** WiMAX Ipv4 Addressing ***/
32 Ipv4AddressHelper ipv4node3;
33 ipv4node3.SetBase("10.1.4.0", "255.255.255.0");
34 iwimaxN4 = ipv4node3.Assign(dnode4d);
35 iwimaxN5 = ipv4node3.Assign(dnode5d);
36 (...)
```

Figura 6.16: Especificação parcial em C++ de um cenário de rede WiMAX



(a) Largura de Banda

(b) Atraso

Figura 6.17: Gráficos comparativos entre o cenário WiMAX e LTE

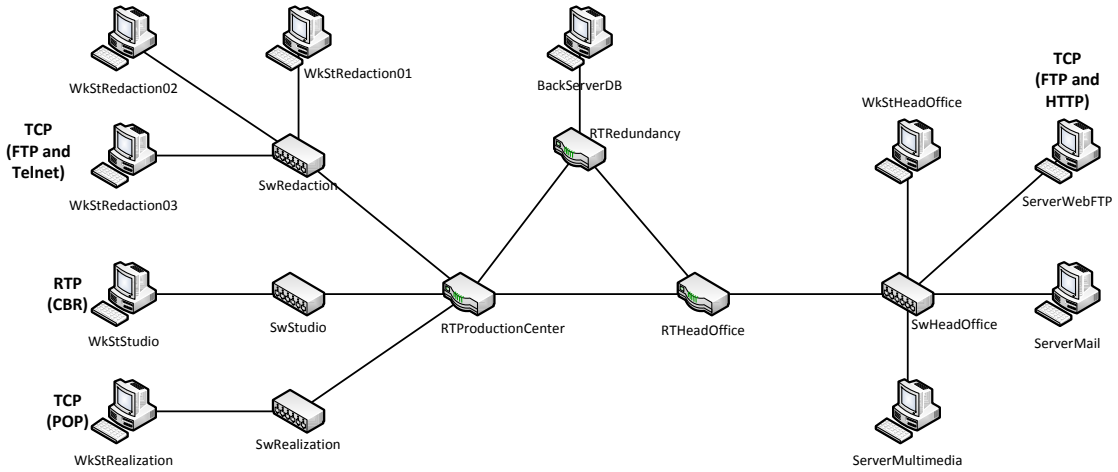


Figura 6.18: Cenário de redes com fios simulado nos simuladores NS2 e NS3

estudos comparativos [17], as limitações dos modelos NS3 não permitem no momento do estudo aceitar como concludentes estes resultados entre as tecnologias. Os trabalhos [173, 189] propõem um modelo mais completo para a simulação das redes LTE no NS3.

Foram feitas diversas variações ao teste descrito para conhecer um pouco mais dos modelos WiMAX e LTE do NS3 e esses testes podem ser consultados em [248]. Uma diferente perspetiva deste trabalho, também muito importante para a *Framework* NSDL, era confirmar a correta geração de código C++ para simulações com redes sem fios e esse aspeto foi conseguido para as dezenas de cenários testados.

6.5.2 Rede com Fios em NS2 e NS3

O objetivo deste capítulo foi abordar as redes sem fios no NS3, mas, para conseguir fazê-lo, foi necessário implementar muitos componentes de base das redes e entre eles, os objetos das redes com fios. Sendo um dos objetivos principais da *framework* a interoperabilidade, aproveitou-se a oportunidade para utilizar duas ferramentas similares para simularem o mesmo cenário.

Assim, o cenário de rede com fios serviu para um teste à interoperabilidade da *Framework* NSDL. Foram construídos diferentes cenários em NSDL e transformados em códigos NS2 e NS3. Ou seja, a mesma descrição de rede foi transformada em dois códigos distintos, cada qual para uma ferramenta particular.

Um dos cenários criados e simulado nas duas ferramentas de simulação é o apresentado na Figura 6.18. A rede apresentada na Figura 6.18 procura mostrar o grau de complexidade testada na geração de cenários de rede. Existem múltiplas aplicações, protocolos e nós, todos ligados de forma variada e com parâmetros distintos. Também, como princípio para a comparação, optou-se por adicionar o mínimo de parâmetros na criação dos objetos, conduzindo assim os simuladores a utilizarem os seus valores internos por omissão para os mais variados parâmetros.

Os resultados obtidos acabaram por conduzir a valores muito diferentes de largura de banda, atraso e perdas, o que, de início, se pensaria que não poderia acontecer por estas serem simulações sobre a mesma rede. Mas, sendo os núcleos dos simuladores e seus modelos de rede diferentes, os resultados distintos não são inesperados. Seria necessário avaliar parâmetro a parâmetro, modelo

a modelo disponível e encontrar a maior proximidade possível entre cenários para avaliar corretamente, mas esse não era o objetivo do trabalho.

Mais uma vez, o fundamental era confirmar que a geração de código OTcl e C++ era correta e permitia executar simulações nas ferramentas respetivas. Todos os cenários avaliados e os resultados obtidos podem ser conhecidos em [248].

Na próxima são apresentadas algumas conclusões deste capítulo.

6.6 Conclusões

A utilização da *Framework* NSDL para propor uma abordagem à criação de cenários de simulação no NS3 foi bem conseguida neste projeto. Demonstrou-se a possibilidade de criar diversos cenários de redes em ambientes gráficos integrados na *framework* e, em sequência, realizar a sua execução no simulador NS3.

Os novos objetos NSDL criados permitem caraterizar corretamente os objetos do NS3 e, com eles, criar as mais variadas simulações. Para o tempo de trabalho disponível para este projeto, o número de redes coberta foi bastante maior que o esperado. O planeamento foi feito para integrar as redes Wi-Fi e *Mesh*, mas no final conseguiu-se incluir também as redes WiMAX e LTE.

A construção das estruturas de validação e transformação para o NS3 foi realizada seguindo os mesmos processos já utilizados para construir as outras estruturas da *Framework* NSDL. A validação é feita tendo por base o estudo das redes e conclui-se com a especificação dos objetos e das suas características em XML *Schemas*. A transformação é definida a partir do código da ferramenta e, nesta implementação, a estrutura de XSL *Transformations* usada para o NS2 foi facilmente adaptada para o NS3. Foram necessárias diversas semanas para se obter o código gerado automaticamente. Contudo, a quantidade de objetos e redes cobertas mostram que a estrutura reutilizada foi adequada e conseguiu-se uma rápida extensão da *framework* para algumas redes sem fios e para uma nova ferramenta, o simulador NS3.

Uma validação importante para as novas bibliotecas da *framework* era a sua correção na transformação para a linguagem do simulador. A variedade de tecnologias e topologias testadas com sucesso garantem a sua boa implementação, não apenas para as redes sem fios, mas também para redes com fios. Os casos de estudo procuraram demonstrar com alguns exemplos a variedade e complexidade de cenários que podem ser descritos em NSDL e traduzidos para os simuladores NS2 e NS3.

A única opção ainda a ser melhor refletida foi a de alargar a quantidade de redes disponíveis para o NS3 o que levou a ser escolhido o caminho da simplificação na definição dos novos objetos. Observou-se que estes foram construídos, principalmente, com base nas características do NS3 e menos nas características genéricas das tecnologias. Em conclusão, a maioria dos parâmetros dos objetos das redes sem fios são os parâmetros disponíveis nos modelos destes objetos no NS3. Portanto, teria sido preferível ter definições menos dependentes das ferramentas, mesmo que isso implicasse a criação de menos objetos de rede. Por esse fato, a reutilização dos novos objetos em outras ferramentas poderá colocar em causa a interoperabilidade da *framework*.

Capítulo 7

Simulação de Redes em Ambientes Virtuais

7.1 Introdução

A análise e validação completa de novos protocolos ou elementos de uma rede de dados são conseguidas na sua integralidade implementando esses componentes numa rede real (contrariamente às aproximações obtidas através de métodos como a modelagem analítica ou a simulação). As limitações em utilizar tal opção, como o custo e a complexidade na replicação de problemas em redes existentes, são suficientemente importantes para que este método seja preterido, pelo menos numa fase inicial do projeto de redes. A utilização de ferramentas de simulação permite descrever os objetos existentes nas redes e todos os eventos que nelas ocorrem, além de proporcionar resultados o mais aproximado o possível do real, por essas razões a simulação tem sido a opção mais utilizada na otimização e implementação de projetos de redes [147].

O trabalho realizado neste projeto visa aproximar os ambientes de simulação com os ambientes reais de uma rede, e assim conseguir resultados mais próximos dos ambientes de rede reais. O método usado consiste em criar uma rede real, com todos os eventos pretendidos para testar um protocolo e/ou tecnologia de rede, através de um ambiente virtual. As plataformas de virtualização atuais permitem muita flexibilidade na criação de nós e ligações e podem ser adicionados a maioria dos sistemas operativos e as arquiteturas de computação utilizadas nas redes comuns.

Este capítulo apresenta, no âmbito da *Framework* NSDL, o desenvolvimento de bibliotecas para a geração de código que permitem configurar e implementar um cenário de redes (através da criação dos nós, das suas ligações, aplicações e os eventos necessários), tirando proveito da interoperabilidade proporcionada por NSDL, de forma a realizar testes reais de redes, mas em ambientes virtuais. Os resultados apresentados neste capítulo foram realizados no âmbito do trabalho de Mestrado em Engenharia Informática da aluna Joana Araújo [11], que foi co-orientada pelo autor desta tese.

O capítulo está organizado da seguinte forma. A secção 7.2 introduz o tópico de ambientes virtualizados e apresenta a plataforma utilizada neste projeto. De seguida, na secção 7.3, apresenta os objetos usados e os componentes adicionados à *Framework* NSDL. A secção 7.4 apresenta um caso de estudo que permitiu testar a geração do código. Por fim, na secção 7.5, são apresentadas algumas conclusões.

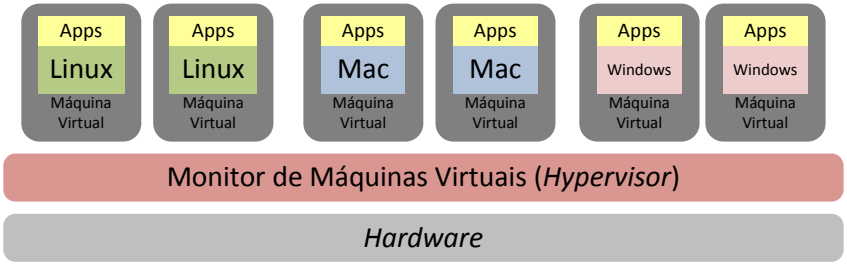


Figura 7.1: Elementos principais de um sistema de máquinas virtuais

7.2 Ambientes Virtualizados

A virtualização é uma técnica que existe há algumas décadas [98], no entanto nos últimos anos começou a ser utilizada de forma alargada para a substituição de servidores e computadores reais e, por isso, ainda apresenta muitos desafios.

Os componentes que podem ser virtualizados são muitos e variados e, nesta secção, será apresentada uma introdução à virtualização de computadores e, em particular, à virtualização de redes.

7.2.1 Virtualização de computadores

Os computadores modernos têm capacidade suficiente para suportar a execução de vários sistemas operativos simultaneamente. A forma mais comum de o realizar é através da virtualização, ou seja, da criação de mecanismos que permitam que o hardware de um computador possa ser partilhado em tempo real por vários sistemas operativos. Na virtualização é criada uma abstração de hardware para cada sistema operativo, designada por 'máquina virtual' (VM), onde existe a ilusão de acesso aos recursos de hardware de forma exclusiva [63].

O responsável pela criação da abstração de hardware é um software designado como Monitor de Máquinas Virtuais, ou simplesmente *hypervisor*. Este software realiza todas as funções de gestão das estruturas do hardware, como a gestão da memória, a gestão dos dispositivos de entrada e saída, e gestão dos controladores para o acesso direto à memória. A Figura 7.1 apresenta a estrutura de um sistemas de VMs.

O *hypervisor* fornece então, a cada VM, um ambiente completo para a sua execução. Existem algumas diferenças entre *hypervisors* [63]:

- Virtualização Emulada: nesta forma de virtualização, os recursos físicos são completamente emulados em software, particularmente as instruções do processador. O *hypervisor* recebe todas as chamadas que o sistema operativo realiza sobre o hardware virtualizado e, após algumas validações, repete o comando sobre o hardware real. Uma grande desvantagem deste método é a penalização que ocorre em termos de desempenho, e;

- Para-Virtualização: a virtualização aqui é conseguida construindo uma abstração similar, mas não idêntica, ao hardware real. Cada instância de um sistema operativo requer diferentes recursos e estes podem ser obtidos junto do *hypervisor*. Esta colaboração entre a VM e o *hypervisor* permite uma melhor utilização do hardware e um melhor desempenho das VM's. Este método, conjuntamente com as inovações conseguidas nas arquiteturas dos processadores

para suporte da virtualização, permite grandes ganhos de desempenho na virtualização.

Ainda sobre a Figura 7.1, os sistemas operativos e as aplicações são os elementos tradicionais de software presentes num sistema computacional. A utilização de virtualização, principalmente em ambientes empresariais, procura atingir duas vantagens importantes [143]:

- Redução de custos: conseguida devido à colocação do mesmo conjunto de serviços mas num menor número de servidores, comparativamente a um ambiente não virtualizado, e também por conseguir uma diminuição no consumo energético, e;
- Melhorias de gestão: é possível realizar algumas tarefas, como migrações e atualizações, de forma transparente para os utilizadores. Não é necessário desligar as máquinas para usar novos equipamentos, bastando agora apenas a sua migração para esses equipamentos sem interromper o serviço. Em servidores não virtualizados é complexo realizar algo semelhante.

Ainda existem algumas dificuldades e barreiras para a utilização da virtualização de forma mais alargada e que são [143]:

- Desempenho: comparativamente a um sistema não virtualizado, um sistema virtualizado com as mesmas características tem um desempenho inferior. Uma razão para isto deve-se ao facto de que neste momento todo o *software* e *hardware* são ainda desenhados para um contexto de funcionamento não virtual. Um exemplo é o acesso aos discos rígidos, rápido num sistema normal, mas, potencialmente, mais demorado e/ou complexo num sistema virtual;
- Dificuldades na gestão: a gestão de um número de servidores normais é igual à gestão do mesmo número de servidores virtualizados. A estes últimos ainda acresce a gestão do *hypervisor* que os suporta. Outro aspeto são os *interfaces* gráficos (GUIs) para a gestão. Estes são suficientes para algumas dezenas de máquinas, mas desadequados para realidades com centenas ou milhares de máquinas. Apesar de alguns ganhos em algumas tarefas, globalmente a gestão ainda é mais trabalhosa e complexa;
- Escala: os *softwares* de virtualização têm limitações de escala, sendo pouco adequados para ambientes onde é necessário ter um número grande de sistemas. Sendo uma das vantagens da virtualização a gestão centralizada dos servidores, esta limitação obriga a construir diferentes sistemas e diferentes ambientes de gestão e, em consequência, aumentando novamente a dificuldade na gestão;
- Interoperabilidade: a realidade mostra a coexistência de sistemas virtualizados (por vezes distintos) com sistemas não virtualizados. Cada um tem a sua forma de configuração, descrição, API's, bem como o seu conjunto de ferramentas de gestão, assim, criando 'ilhas' de gestão. A integração dos diversos sistemas não é possível, aumentando a complexidade na operação do sistema global da empresa, e;
- Resolução de problemas: A estrutura tenológica comum nas empresas contém os servidores, o armazenamento e os equipamentos de rede. Um problema neste ambiente é, normalmente, mais simples de detetar, de identificar a sua fonte e de encontrar uma possível resolução. Em ambientes virtualizados esta estrutura está completamente integrada em apenas uma máquina

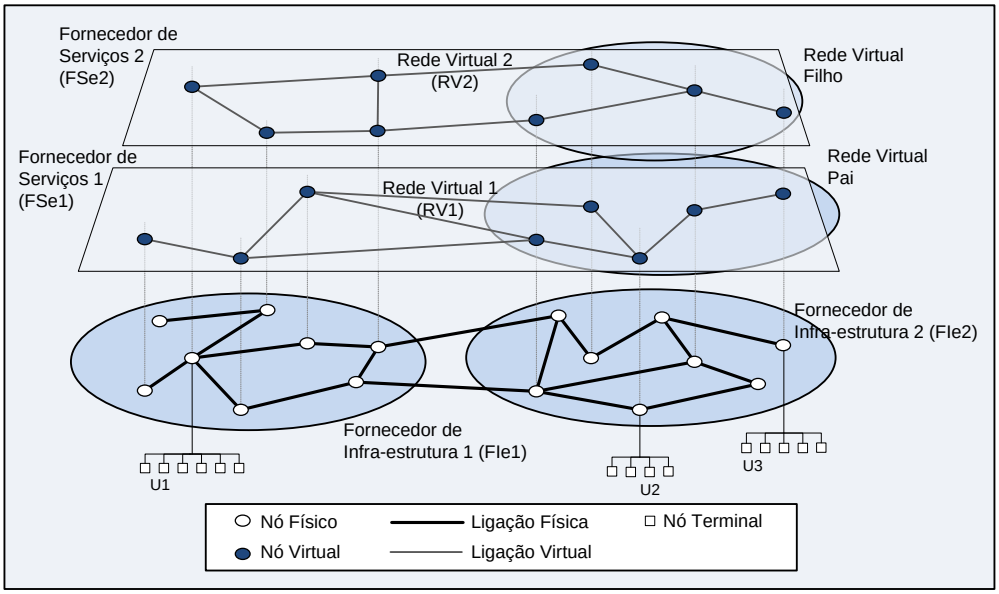


Figura 7.2: Ambiente de redes virtuais, adaptado de [55]

física e, na ocorrência de problemas, a sua deteção e correção é muito mais complexa por obrigar à consulta das informações dos diferentes elementos.

A virtualização apresenta ainda muitas limitações, mas na maioria estas surgem devido à virtualização ser uma tecnologia com poucos anos de utilização alargada. O desenvolvimento de ferramentas e de ambientes de virtualização necessitam ultrapassar um conjunto de problemas particulares a estes sistemas [72]. No mais a virtualização obriga a um esforço muito grande para a sua administração e quando surgirem as ferramentas que facilitem a gestão e promovam a integração e interoperabilidade entre sistemas distintos, muitas das desvantagens poderão ser minimizadas.

7.2.2 Virtualização de redes

Além da possibilidade de virtualizar máquinas individuais, é também possível a virtualização de redes. Esta forma de virtualização é conseguida colocando sobre uma infraestrutura de rede física real um conjunto de redes virtuais. Esta forma de organização separa os serviços associados a uma entidade, o fornecedor de serviços de Internet (*Internet Service Providers* – ISP), em dois componentes e, normalmente, relacionados com duas entidades: o fornecedor da infraestrutura (*Infraestructure Providers* – InPs), responsável pela gestão da infraestrutura física, e o fornecedor de serviços (*Service Providers* – SPs), criador de serviços agregando os recursos de múltiplos InPs e fornecendo serviços fim-a-fim. De forma análoga às VMs, múltiplas redes virtuais podem coexistir sobre a mesma infraestrutura física, isoladas umas das outras. O propósito principal desta organização é procurar criar serviços que ultrapassem algumas das limitações encontradas na Internet tradicional [54]. A Figura 7.2 apresenta um conjunto de redes virtuais sobre uma rede física.

A Figura 7.2 apresenta na sua parte inferior dois fornecedores de infraestrutura (FIe1 e FIe2). Sobre essas duas infraestruturas surgem duas redes virtuais (RV1 e RV2), cada qual com um conjunto de nós e ligações virtuais e que são um subconjunto de nós e ligações da rede física. Um nó virtual tem uma relação direta com um nó físico. Uma ligação virtual pode ligar dois nós

virtuais por intermédio de várias ligações e nós físicos. Os fornecedores de serviços (FSe1 e FSe2) são responsáveis pela criação das redes virtuais de forma a fornecerem serviços fim-a-fim entre os utilizadores finais (U1, U2 e U3).

Os projetos para a virtualização de redes são baseados num subconjunto de quatro tecnologias já existentes nas redes atuais e que permitem a coexistência de múltiplas redes no mesmo meio físico, e são as seguintes:

- *Virtual Local Area Network* (VLAN): são entidades lógicas relacionadas e que pertencem ao mesmo domínio de difusão. Normalmente são da camada 2 e uma trama irá conter um indicador, VLAN ID, que define a que rede a trama pertence [118];

- *Virtual Private Network* (VPN): é uma rede privada a operar sob uma rede pública, como a Internet, e onde o tráfego privado é separado e protegido do restante tráfego. Uma VPN pode ser implementada em diversas camadas de rede pois existem tecnologias em cada camada de rede para esse fim. O tipo de rede, o nível de desempenho e a segurança são os parâmetros analisados e que conduzem à escolha do tipo de VPN a implementar [54];

- Redes ativas e programáveis: não são redes virtualizadas, mas a maioria dos projetos de virtualização de redes baseia-se no princípio de programação para a coexistência de diferentes redes. A programação é responsável por manter ambientes isolados para evitar conflitos e instabilidade nas redes [41], e;

- *Overlay Networks*: é uma rede virtual com uma topologia virtual própria construída sobre uma rede física. São redes criadas na camada de aplicação e não têm restrições geográficas. As suas vantagens são a flexibilidade e a capacidade de adaptação a mudanças na rede física. Têm sido usadas na abordagem de problemas relacionados com o encaminhamento de dados na Internet, o fornecimento de serviços de *multicast*, a garantia de QoS, a segurança, a distribuição de conteúdos, a partilha de ficheiros e o armazenamento [54, 254].

A implementação apresentada no resto deste capítulo procurou utilizar as possibilidades existentes para a virtualização de computadores, com as suas aplicações, e das ligações para as comunicações entre VMs. A virtualização dos computadores é mais direta, sendo a criação de VMs a base do funcionamento das plataformas de virtualização. A virtualização das ligações é também possível na plataforma, criando ligações entre máquinas de um mesmo domínio e isolando as comunicações de máquinas de outros domínios. Para interligar os domínios foi necessário adicionar um componente (descrito na secção 7.4) para realizar as funções de encaminhamento e de segurança.

Existem ainda outras formas de virtualização de recursos computacionais [237], como o armazenamento e as aplicações, mas por não terem sido abordados neste projeto não são referidos nesta secção.

De seguida é apresentado o *hypervisor* utilizado no âmbito deste projeto.

7.2.3 *XenServer®*

O *XenServer* é uma plataforma para a virtualização de servidores, desenvolvido pela empresa *Citrix Systems, Inc.* [58]. O *XenServer* é baseado no *Xen hypervisor* desenvolvido pela *Computer Laboratory* da Universidade de *Cambridge*. Sendo um *hypervisor* do tipo para-virtualização destacou-se

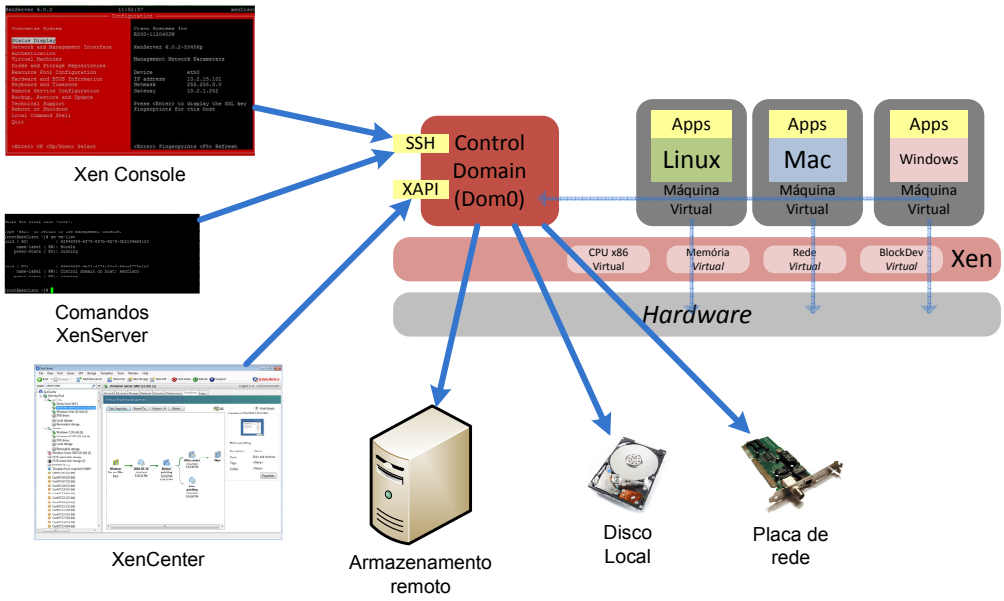


Figura 7.3: Arquitetura do *XenServer*

pelo desempenho das suas VMs, próximo de uma máquina nativa [19]. A Figura 7.3 apresenta os componentes de um servidor *XenServer*.

As VMs (Linux®, Apple®[1] e Microsoft®) acedem ao hardware através do hypervisor. Os *interfaces Xen Console*, *XenCenter* e comandos *XenServer* permitem ao administrador realizar todas as tarefas necessárias para gerir o servidor. A concretização da gestão do *XenServer* é feita no componente *Control Domain* (Dom0), um domínio privilegiado usado para gerir outros domínios e as políticas de atribuição de recursos [102].

Além da virtualização de máquinas com a arquitetura x86, o *XenServer* tem ainda suporte para muitas outras funcionalidades, das quais se destacam: a recuperação rápida em caso de falha, o armazenamento integrado (para a replicação dos dados, de-duplicação, imagem e clonagem), a associação de recursos vídeo às VMs para aumento do desempenho gráfico, as ferramentas para a conversão de VMs, e, o *XenMotion* (consegue a migrações de VMs entre servidores mantendo ativa a VM). As funções referidas fazem parte da versão gratuita e de utilização livre, utilizada neste projeto. As restantes funcionalidades presentes no *XenServer* fazem parte de versões mais avançadas do servidor e envolvem já algum custo.

Além do *Xen hypervisor*, existem outras ferramentas semelhantes para a virtualização. O concorrente mais próximo em termos de funcionalidades e âmbito é o *VMware vSphere* [265]. Existem ainda o *Hyper-V* [112], o KVM [140], o *Parallels* [204], mas são menos usados e sendo focada a sua utilização num determinado sistema operativo, respetivamente, o *Windows*®, o *Linux* e o Mac OS X®.

Determinar a melhor plataforma pode ser complexo, já que alguns elementos e funcionalidades são melhores numa plataforma e outros elementos melhores em outra plataforma. Comparando apenas o *Xen* com o *vSphere*, a característica principal do *vSphere* é ser do tipo virtualização emulada,

[1] A arquitetura de processadores implementada é a da família x86. Apesar do sistema operativo Mac OS X se basear nessa arquitetura, existem algumas restrições no seu licenciamento e que permitem a criação de VMs Mac OS X apenas sobre hardware da Apple.

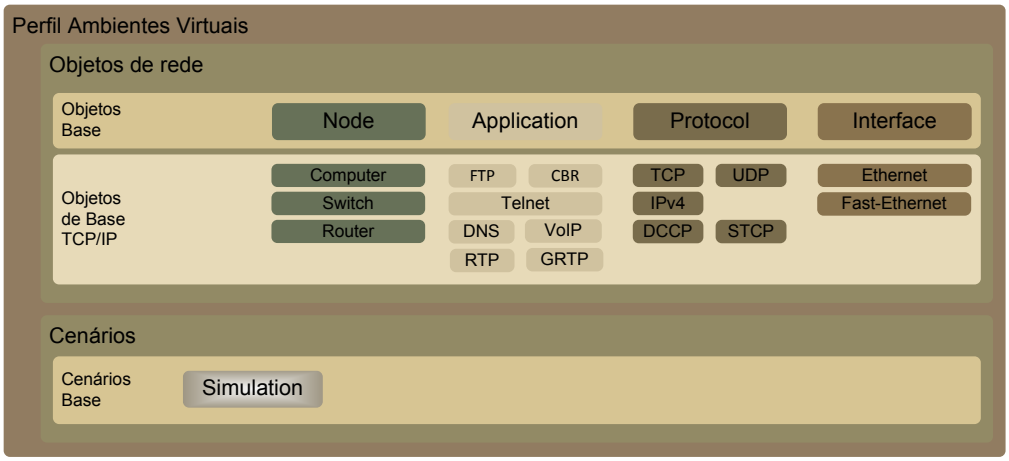


Figura 7.4: Objetos e cenário do perfil de virtualização da *Framework* NSDL

ou *full virtualization*, ou seja, neste caso é utilizada uma técnica que combina a execução direta e a tradução dos binários [4]. O trabalho apresentado em [209] realiza um estudo de comparativo onde se comprova o melhor desempenho do *vSphere* nesse contexto. Da mesma forma, esse estudo vem de encontro dos resultados apresentados em [19] onde se mostra que o *Xen* tem, em contexto de para-virtualização, um desempenho muito próximo da máquina nativa. Os trabalhos introduzidos em [51, 212, 283] apresentam outras análises e comparações entre estes, e outros, *hypervisors*.

A secção seguinte apresenta a forma encontrada de integrar os ambientes virtuais na *Framework* NSDL.

7.3 Representação de Ambientes Virtualizados em NSDL

O objetivo deste projeto foi a realização da virtualização automática de cenários de rede descritos em NSDL. Uma vez que a simulação de cenários de rede já foi realizada e integrada com a *Framework* NSDL, foi possível a reutilização de muitos objetos de outros ambientes de simulação para a descrição destes cenários.

A escolha dos elementos para a virtualização de cenários de rede foi a tarefa mais complexa uma vez que o leque de objetos existentes no ambiente real e passíveis de serem descritos é amplo. Assim, nesta secção são apresentados os sistemas operativos, aplicações e outros componentes escolhidos para serem integrados na *Framework* NSDL.

7.3.1 Perfil NSDL/Ambientes Virtualizados

O perfil criado no âmbito deste projeto inclui os objetos necessários para a realização de simulações. Como previsto, os objetos identificados são alguns dos objetos que permitem a sua execução em ambientes virtualizados em um primeiro momento, de forma a validar a proposta do trabalho a ser realizado. A Figura 7.4 apresenta esses objetos.

Como cenário para este perfil seria esperado ter um cenário designado de <Virtualização>, ou algo semelhante, no entanto, foi escolhido um cenário já existente, o <simulation>. A criação de um cenário específico para uma dada ferramenta é justificada se houverem características próprias desse cenário para representar. Neste projecto apenas se pretendia implementar simulações e,

assim, a utilização do cenário <simulation> foi suficiente, não sendo obrigatória a criação de um cenário próprio. A evolução deste projeto poderá conduzir à criação de um cenário relacionado com aspectos pertencentes apenas à virtualização, e.g., gestão de VMs, e então, nesse momento, será necessária a definição de um novo cenário.

Os objetos deste perfil são, essencialmente, os objetos já utilizados no perfil base TCP/IP. Assim, pretendeu-se utilizar os objetos comuns de uma rede real, portanto, os nós, aplicações, protocolos e *interfaces* que surgem são os esperados. Os pormenores da implementação de cada objeto na plataforma virtual são descritos nas subseções seguintes.

7.3.2 Características dos nós

Os nós existentes neste perfil são o <computer> e o <router>. O <computer> toma o papel comum de um computador de trabalho e podem ser executadas as aplicações servidor e/ou cliente. O <router> serve para encaminhar o tráfego entre as redes de VMs de domínios distintos, tal como num ambiente real.

O sistema operativo escolhido para o <computer> e para o <node> foi o *Linux*, e mais precisamente a distribuição *OpenSuSE* 11 [200]. Além de ser um sistema operativo comum, o *Linux* também foi escolhido pela sua flexibilidade na configuração dos *interfaces* e protocolos. O *Linux* permite nativamente a ligação remota via *Telnet* e, por isso, foi a aplicação escolhida para a configuração completa do computador. Apesar das limitações do *Telnet* em termos de segurança não há qualquer problema neste contexto devido ao isolamento que existe entre o ambiente de virtualização e as outras redes. E, se necessário, existe a opção de criar ligações cifradas para o envio das configurações no *Linux* [93].

A VM escolhida como <router> foi o Vyatta [267]. Este contém as funcionalidades necessárias para o projeto, como o encaminhamento de pacotes, e ainda suporta as funções de VPNs, de *firewall*, de deteção de intrusão e de filtragem na camada de aplicação [245]. Além de permitir realizar outros projetos no futuro, é escalável e foi desenhado de forma a suportar eficientemente as funções de rede em ambientes virtualizados.

Como citado anteriormente, as implementações de <computer> e o <router> são sistemas baseados em *Linux*. A plataforma de virtualização *XenServer* também é um sistema *Linux*. Esta característica comum em todos os sistemas permitiu simplificar a criação do código por ser apenas necessária uma linguagem. O código para configurar os <computer> e o <router>, as VMs, o *XenServer*, e a máquina real, é o mesmo, o *shell script* [185] complementado com a extensão *Expect* [154, 155]. O *shell script* é uma linguagem que permite a invocação de comandos de um sistema operativo da família *Unix/Linux* e contém algumas estruturas de programação genéricas. Permite a execução de muitas tarefas sobre o sistema, como manipulação de ficheiros, execução de programa e direcionamento de texto. O *Expect* é uma extensão ao Tcl [201] e permite a automatização de aplicações interativas no ambiente de uma *shell*. Como exemplo, o *Expect* pode automatizar e testar as aplicações FTP, *Telnet*, *passwd* e *rlogin*.

7.3.3 Características das aplicações

De forma a testar o tráfego a circular na rede foi necessário a utilização de um gerador de tráfego que reproduza corretamente o comportamento de diversas aplicações. Uma opção é ter as aplicações

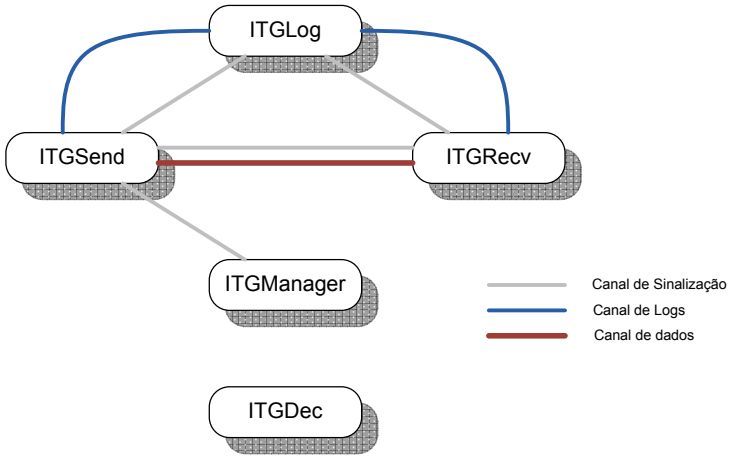


Figura 7.5: Arquitetura da plataforma de geração de pacotes D-ITG

reais, mas a automatização da sua execução é complexa uma vez que a maioria delas não tem uma API simples e uniforme para a sua gestão. Outra opção era utilizar um gerador de tráfego que contemplasse as aplicações necessárias e que, de forma flexível, permitisse os ajustes pretendidos para a simulação.

A opção seguida foi a segunda, um gerador de tráfego, devido à sua simplicidade e flexibilidade, permitindo a geração de padrões de tráfego diversos. Os requisitos mínimos para os geradores de tráfego a pesquisar eram os seguintes: funcionar em modo linha de comando; poder ser executado em Linux; e, operar sobre os protocolos mais conhecidos da arquitetura TCP/IP. De entre os geradores encontrados, apresentam-se de seguidas os geradores que foram analisados:

- Nemesis [181]: é uma ferramenta para a linha de comando que permite a criação e injeção de pacotes numa rede. É adequado para o teste de sistemas de deteção de intrusão, *firewalls*, protocolos da camada IP, entre outros, e;
- D-ITG [30]: o *Distributed Internet Traffic Generator* (D-ITG) é uma plataforma capaz de produzir tráfego ao nível do pacote replicando com precisão os processos estocásticos das variáveis aleatórias para o IDT (*Inter Departure Time*) e para o tamanho do pacote. É possível gerar tráfego IPv4 e IPv6 e das camadas de rede, transporte e aplicação.

Ambos têm as características requeridas e têm funcionalidades semelhantes, cobrindo os protocolos mais conhecidos. O D-ITG foi o gerador de tráfego escolhido no âmbito deste trabalho devido a suportar um maior número de protocolos. Outras características do D-ITG que também contribuíram para a sua escolha foram: uma melhor documentação e suporte; uma maior divulgação na comunidade científica, e; uma longevidade maior e atualizações mais recentes do projeto. Uma lista alargada de ferramentas semelhantes e uma comparação de funcionalidades com o D-ITG pode ser obtida em [64].

Os componentes do D-ITG e as suas relações estão apresentados na Figura 7.5. A função de cada um dos componentes é descrita de seguida:

ITGSend componente responsável pelo envio dos pacotes. É neste componente que se definem os parâmetros para a geração de tráfego. Também suporta o envio de múltiplos fluxos de

```
1 ITGRecv
2
3 ITGSend -a 10.0.0.2 -sp 9400 -rp 9500 -C 100 -c 500 -t 20000 -x recv log file
4
5 cat script_file
6 -a 10.0.0.2 -rp 10001 VoIP -x G.711.2 -h RTP -VAD
7 -a 10.0.0.3 -rp 10002 Telnet
8 -a 10.0.0.3 -rp 10003 DNS
9 ITGSend script_file -l sender log file
```

Figura 7.6: Exemplo de comandos para a geração de tráfego no D-ITG

tráfego;

ITGRecv componente para onde se dirige o tráfego gerado. Consegue receber o tráfego de múltiplos emissores;

ITGLog processo que recebe e armazena as informações de log do emissor e receptor;

ITGDec executável que analisa os resultados (logs) obtidos durante o processo de geração de tráfego. Alguns dos valores obtidos com a análise incluem o atraso, a variação do atraso, e os pacotes perdidos, e;

ITMManager componente que permite o envio de comandos para o controlar o ITGSend. Pode realizar o controlo de vários ITGsend e fazer a coordenação da geração de tráfego de uma rede inteira.

O D-ITG é bastante poderoso e a variedade de parâmetros permitidos são bastante abrangentes para a definição de muitos tipos de tráfego. A Figura 7.6 apresenta alguns exemplos de comandos para a geração de tráfego.

A linha 1 apenas indica o primeiro passo, que deverá ser ativar a recepção do tráfego. Sem a sua inicialização, a geração de tráfego falha. O *host* destino, que irá surgir nas linhas seguintes, deve ser o *host* onde o ITGRecv está a funcionar.

O comando na linha 3 irá gerar um tráfego UDP (escolhido por omissão) com um IDT entre pacotes constante ('-C 100') e tamanho de pacotes constante ('-c 500'). É direcionado ao *host* com o endereço 10.0.0.2 ('-a') e à sua porta 9500 ('-rp'). Por fim, a duração é de 20.000 milissegundos ('–t 20000') e o nome do ficheiro de resultados, a gerar do lado do recetor, é indicado após a opção '-x'.

As linhas 5 a 8 criam um ficheiro de texto que pode ser usado para, em cada linha, conter os parâmetros de diferentes fluxos de tráfego em termos de características e de destinos. A linha 9 é o comando para a geração de tráfego, invocando o ficheiro já referido com a configuração dos fluxos. O parâmetro '-a' indica-nos o destino e verifica-se que existem dois *hosts* como destinatários dos tráfego. O tipo de tráfego é facilmente identificado, por ser indicado o seu nome diretamente (VoIP, *Telnet* e DNS). Além da porta de destino, '-rp', no caso do VoIP são ainda adicionados mais alguns parâmetros: '-x' indica o codificador de áudio usado, '-h' o protocolo de transporte usado, e, '-VAD' ativa a deteção de atividade áudio.

Os exemplos apresentados procuraram representar as possibilidades oferecidas pelo gerador de tráfego D-ITG.

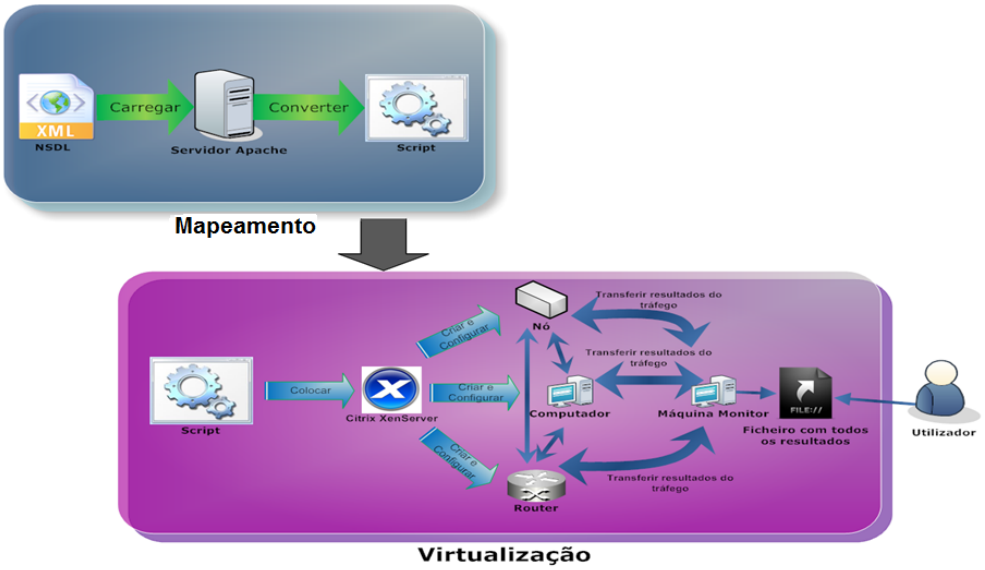


Figura 7.7: Arquitetura de componentes da virtualização da *Framework* NSDL

7.3.4 Características dos Protocolos e *Interfaces*

As características dos objetos dos grupos <protocol> e <interface> são as mesmas que estes já tinham no perfil base TCP/IP. As VMs já contêm todos os componentes necessários destes grupos e apenas falta a sua correta configuração. Os protocolos <ipv4> e <ipv6> associam-se a um *interface* e definem qual o seu endereço de rede e gateway. A escolha de um *interface*, <ethernet> ou <fast-ethernet>, permite definir a largura de banda da ligação da VM nesse *interface*. O <tcp>, <udp> são ligados às aplicações e definem o protocolo de transporte a ser usado pelo tráfego gerado pela aplicação. As opções disponíveis cobrem as configurações esperadas num computador ligado a uma rede real.

Os protocolos e os *interfaces* são configurados nas VMs, mas no caso do *interface*, a adição de algumas políticas de utilização são feitas no *XenServer*, mas especificamente no componente *vSwitch* [266]. Este componente é um *switch* virtual que permite controlar o fluxo de dados que entra e sai de uma VM.

7.3.5 Arquitetura para a virtualização baseada na *Framework* NSDL

A orquestração de todo o sistema de forma a sua execução decorrer dentro do esperado é a tarefa mais complexa. Nesta fase implementaram-se todos os processos de criação e execução de cada componente de uma forma simples. Foram envolvidas algumas das ferramentas da *Framework* NSDL e o servidor *XenServer*. A Figura 7.7 apresenta os componentes para a implementação da virtualização e das relações entre eles.

O processo de realização de uma simulação inicia-se com a construção do cenário em NSDL. A ferramenta VND foi escolhida para essa tarefa e gerou o ficheiro XML com o cenário. As bibliotecas para a validação e, principalmente, para o mapeamento presentes na plataforma web do projeto NSDL produzem o *shell script* que contém todos os comandos necessários para a simulação no ambiente virtualizado. Na Figura 7.7 estas tarefas estão enquadradas no bloco azul.

O segundo passo, bloco rosa na Figura 7.7, acontece no *XenServer* e inicia-se após a receção do *shell script*. É despoletado o processo de criação e configuração das VMs e das ligações entre elas. Os componentes sob o bloco transparente são os objetos virtuais criados para a execução da simulação. Logo de seguida inicia-se a geração de tráfego que tem a duração definida pelo utilizador.

A máquina Monitor tem um papel central na simulação. A criação das VMs é feita pelo *XenServer* (única possibilidade). A configuração das VMs é feita via linha de comandos e após a ligação por *Telnet*. O *XenServer* poderia fazer essa tarefa, mas isso implicava ter as VMs na sua rede. Como o acesso ao *XenServer* é feito via rede, esse fato implicava colocar as VMs também na rede real, uma opção que se pretendia evitar para não existir a possibilidade de conflito entre o tráfego real e o tráfego do ambiente virtual.

A solução deste problema foi conseguida criando uma máquina para a gestão da simulação, designada por Monitor. O Monitor tem duas placas de rede, uma para a ligação com o *XenServer*, outra placa para a ligação às VMs. O tráfego de cada rede está isolado e não há regras de encaminhamento que permitam interligar as redes. É o Monitor que faz a ligação *Telnet* para cada VM e executa os comandos para a sua configuração.

Após a simulação, o Monitor recebe os resultados gerados nas VMs e analisa os seus conteúdos, apresentando após o resultado dessa avaliação. Os valores obtidos são aqueles já referidos para o ITGDec. O ambiente de execução é virtual, mas os dados gerados são iguais aos dados que se obteriam numa rede real. Assim, é simples inserir outras ferramentas para a complementar a análise do tráfego, e um exemplo de ferramenta é o *Wireshark* já descrito na secção 2.3.3.

Como referido no início da secção, em termos de criação e configuração de VMs o processo ainda é muito simples. A sincronização de cada passo está feita com um tempo de intervalo suficiente para que tudo decorra sem falhas. A criação de uma VM é rápida, mas a sua inicialização e configuração completa leva algum tempo (1-2 minutos). Se houver muitas máquinas na simulação este passo inicial pode ser muito demorado. O modelo de eventos necessita de melhorias para permitir o encadeamento mais flexível de todas as ações da simulação, independentemente do equipamento em que seja executado.

A secção seguinte irá apresentar algumas das redes virtualizadas no âmbito deste projeto.

7.4 Caso de estudo

Os testes necessários para validar as bibliotecas foram variados e, nesta secção, é apresentado o cenário com o maior número de VMs. O *hardware* disponibilizado para o projeto (computador com processador *iCore* 5, velocidade 2GHz, disco rígido de 320GB e 8GB de memória) permitiu criar 7 VMs Linux, cada uma com 512MB de RAM. Nas subseções seguintes são apresentadas as características dos objetos do cenário e a sua representação em NSDL e em *shell script*.

7.4.1 Topologia e Máquinas Virtuais

A topologia de rede escolhida é a topologia comum em muitos cenários de simulação. Designa-se por *dumbbell* e serve para a avaliar o efeito de congestão numa ligação partilhada por muitos nós. Um lado da rede contém os nós geradores de tráfego e o outro lado os nós clientes, ou recetores de

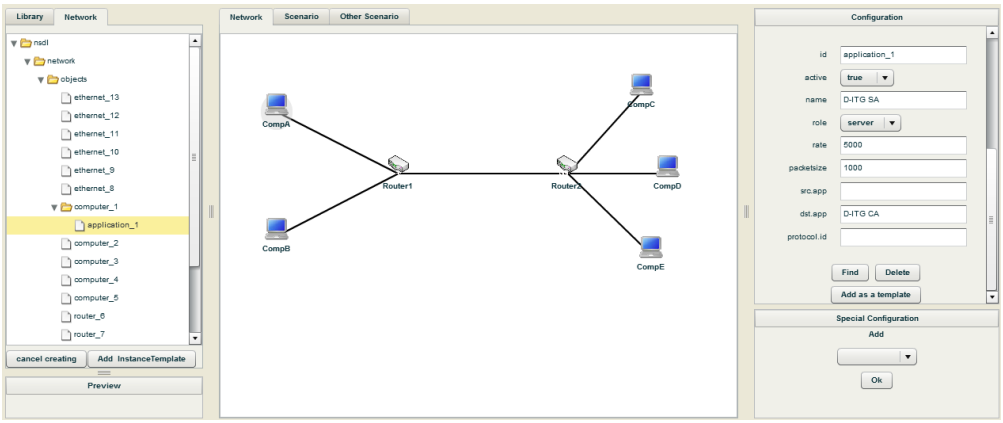


Figura 7.8: Topologia utilizada para o caso de estudo do ambiente de virtualização

```
<computer id="computer_1">
        <name>CompA</name>
        <role>server</role>
        <cbr id="CompAcbr">
                <role>server</role>
                <packetsize>500</packetsize>
                <dst.app>CompCcbr</dst.app>
                <interval>100</interval>
        </cbr>
        <ipv4 id="CAip">
                <address>192.168.2.10</address>
                <mask>255.255.255.0</mask>
                <gateway>192.168.2.254</gateway>
        </ipv4>
        <f-ethernet id="CAinterface" />
                <bandwidth>100Mbps</bandwidth>
        </f-ethernet>
</computer>
```

Figura 7.9: Descrição de um computador em NSDL

tráfego. A interligação é realizada através de uma rede simples com um ou dois *routers* (dois neste caso). A figura 7.8 apresenta a topologia criada com a ferramenta VND.

Os computadores são máquinas Linux completas, mas sem ambiente gráfico para poupar recursos. Para este projeto foram criadas diversas VMs para servirem como *template*, ou seja, um <computer> em NSDL terá uma VM correspondente no *XenServer*. No caso de se pretender testar diferentes sistemas operativos, na NSDL será necessário adicionar o elemento <os> e indicar o sistema pretendido. Nessa situação, a criação da VM é realizada a partir de um *template* diferente. Por omissão as VM utilizam Linux.

Os *routers* são máquinas Vyatta e todo o processo de criação e configuração são semelhantes por estas serem também máquinas Linux. O encaminhamento do tráfego entre redes é gerido automaticamente pelo Vyatta.

A Figura 7.9 apresenta a representação do computador em NSDL, com os seus objetos: aplicação CBR, protocolo IPv4 e *interface Fast-Ethernet*. Esta descrição servirá de base para demonstrar o processo de transformação para o *shell script*.

A criação de uma VM é feita por um processo de clonagem, bastante mais rápido que gerar

```
1 xe vm-clone vm="SUSE 11_Clone" new-name-label=CompA
2 xe vm-start vm=CompA
3 ...
4 xe vm-shutdown vm=CompA
5 xe vm-uninstall vm=CompA
```

Figura 7.10: Criação, arranque e término de uma máquina virtual

```
1 ITGRecv
2 ...
3 ITGSend -a 192.168.2.20 -sp 9400 -rp 9500 -C 100 -c 500 -t 20000 -x logs.txt
```

Figura 7.11: Comandos para a receção e emissão de tráfego

uma máquina nova de raiz. Em poucos instantes está disponível uma máquina completa para ser utilizada. A Figura 7.10 apresenta os comando para a gestão de uma máquina virtual. Estes comandos são invocados no *XenServer*.

A linha 1 da Figura 7.10 contém o comando para a criação de uma nova máquina virtual a partir da máquina "SUSE 11_Clone"designada por "CompA". Neste ponto a máquina já existe no *XenServer*, mas ainda não está operacional. O comando da linha 2 inicia a VM.

A criação e término de máquinas não é um processo complexo. É necessário ter tanto *templates* como *nodes* previstos na NSDL. Após a realização da simulação e a obtenção de todos os resultados as máquinas podem ser apagadas, libertando recursos do servidor. As linhas 4 e 5 da Figura 7.10 contêm os comandos para desligar e para apagar a VM, respetivamente.

Se todas as máquinas fossem criadas em sequência, surgiria um grupo de máquinas com as mesmas configurações. No caso dos parâmetros da rede, existiria um conflito de endereços e a rede ficaria instável e/ou inoperacional. Assim, após a criação e arranque da máquina, é necessário aceder à máquina e especificar as configurações de rede (próxima secção).

A criação dos *routers* é feita pela clonagem de um *template* da máquina *Vyatta*. A configuração das suas placas de rede é feita de forma similar à descrita para o computador.

7.4.2 Aplicações

A geração de tráfego é feita por intermédio da aplicação D-ITG, descrita na secção 7.3.3. De partida, já se encontram instalados todos os seus componentes nos *templates* do *XenServer* e, assim, estão disponíveis em todas as máquinas virtuais criadas a partir desses *templates*. A Figura 7.11 contém o comando para receção e emissão de tráfego da aplicação descrita nas linhas 5 a 10 da Figura 7.9.

A aplicação <cbr> utiliza o protocolo UDP por omissão, e tal caraterística é igual no D-ITG. A linha 1 apresenta o comando para a receção do tráfego do lado do cliente. As restantes opções para a sua execução do lado do servidor (ITGSend) são obtidas da seguinte forma:

- a : endereço IP do recetor. É obtido pesquisando o endereço IP da máquina onde se encontra a aplicação cliente com o @id 'CompCcbr' indicada em <dst.app> (linha 8);

- sp e rp : portas das aplicações. Não sendo indicadas, são usados os valores por omissão;

- C : IDT e é obtido a partir do elemento <interval>;

```
1  # Telnet ao no criado
2  send "telnet 192.168.2.1\r"
3  expect "login:"
4  send "root\r"
5  expect "Senha:"
6  send "123465\r"
7  # Configurar o interface para o novo IP
8  expect "linux:~ #"
9  send "ifconfig eth0 192.168.2.10 netmask 255.255.255.0 \r"
10 (...)
```

Figura 7.12: Configuração do *interface*

```
1  xe vif-param-set uuid=<vif_uuid> qos_algorithm_type=ratelimit
2  xe vif-param-set uuid=<vif_uuid> qos_algorithm_params:kbps=100000
```

Figura 7.13: Definir limites à largura de banda disponível

- c : tamanho do pacote e é obtido a partir do elemento <packetsize>;

- t : duração em milissegundos para o envio de pacotes. É obtido a partir dos eventos da aplicação presentes no elemento <simulation>, e;

- x : ficheiro com os resultados. É obtido a partir das definições feitas em <output> do elemento <simulation>.

7.4.3 Protocolos e *Interfaces*

A configuração dos *interfaces* surge após a máquina ter sido criada e inicializada. As linhas 11 a 18 da Figura 7.9 contêm as definições para o *interface* e para o seu endereçamento em NSDL. O endereço é obtido através dos dados presentes no protocolo <ipv4>. Os protocolos têm neste ambiente, pelo menos para os objetos escolhidos, uma função complementar na configuração. A Figura 7.12 apresenta os comando *shell script* para acesso à VM e para a configuração do *interface*.

O início dos comandos têm send ou expect e são os componentes do *Tcl/Expect* que permitem esperar por um dado momento da execução da máquina (expect) e enviar comandos para a *shell* (send). O acesso à VM é feito via *Telnet* e é usado um endereço genérico configurado no *template* (linhas 2 a 6 da Figura 7.12). Após o acesso é reconfigurada a rede para o endereço definido no cenário NSDL (linhas 8 e 9 da Figura 7.12).

Os *interfaces*, por omissão, utilizam os recursos configurados e disponíveis no *XenServer*. As placas físicas disponibilizam uma largura de banda de 1000Mbps, logo as máquinas virtuais têm, teoricamente, acesso a essa taxa de transmissão de forma partilhada. Para limitar o tráfego de uma VM podemos indicar a taxa máxima de transmissão. A Figura 7.13 apresenta os comandos para realizar essa função.

A largura de banda definida na linha 17 da Figura 7.9 está assim mapeada nas linhas da Figura 7.13. São ainda necessários, previamente, mais alguns passos para obter o identificador da placa de rede virtual (vif_uuid). Estes comandos são invocados no *XenServer*.

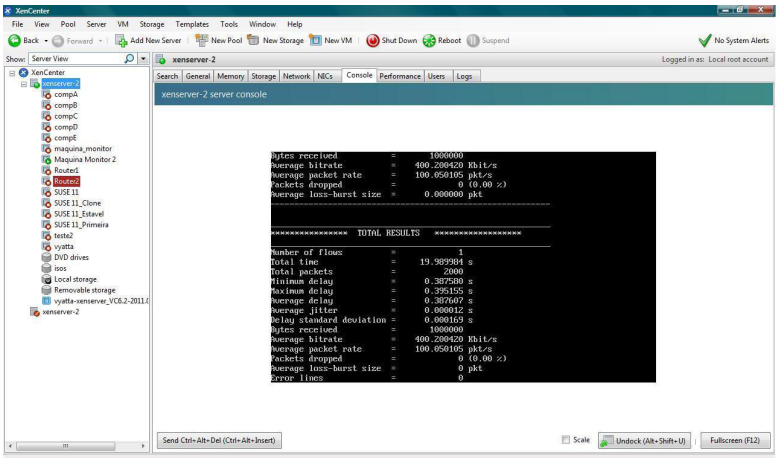


Figura 7.14: Consola com os resultados de uma simulação

Tabela 7.1: Resultados da execução de uma aplicação num ambiente virtual

Descrição	Valor
Número de Fluxos	1
Duração	19,989984 seg
Número de pacotes	2000
Atraso mínimo	0,387580 seg
Atraso máximo	0,395155 seg
Atraso médio	0,387607 seg
Variação média do atraso	0,000012 seg
Desvio Padrão do atraso	0,000169 seg
Bytes recebidos	1.000.000
Taxa média de transmissão	400,200420 Kbps
Taxa média de pacotes	100,050105 pcts/seg
Pacotes perdidos	0% (0 pcts)
Tamanho médio dos *bursts* de perdas	0 pcts

7.4.4 Resultados

A máquina monitor é responsável por obter e agregar todos os resultados construídos. No final do tempo de simulação acede às VMs e copia todos os ficheiros de resultados criados. Apenas após a boa conclusão desse passo se deve proceder ao encerramento e à destruição das VMs da simulação. A Figura 7.14 apresenta uma consola com os resultados obtidos na máquina monitor.

O bloco central da imagem mostra o ecrã do monitor com a impressão dos resultados da execução de uma aplicação. Podemos verificar no bloco da esquerda as diversas VMs criadas (CompA-E e Router1-2) e os *templates* que lhe deram origem ('SUSE 11_Clone' e 'vyatta'). Neste momento as VMs estão inativas e apenas a máquina monitor está a funcionar. Nesta situação foi definido apagar as máquinas manualmente, mas, se requerido, as máquinas poderiam ter já sido apagadas. O bloco central da Figura 7.14 resulta da análise do ficheiro 'logs.txt' através da ferramenta ITGDec. Os valores, pouco visíveis na imagem, são apresentados na Tabela 7.1.

A maioria dos resultados apresenta os valores esperados. Os resultados para o atraso são normais e não existiu qualquer perda de pacotes. As taxas médias de transmissão e de pacotes acabaram por ter os valores previstos, considerando os valores dos parâmetros na descrição.

Os resultados mais importantes nesta fase foram a confirmação da correta geração do código e a sua execução no *XenServer*. As redes testadas foram corretamente implementadas e produziram resultados válidos sobre a execução das aplicações. Todos os cenários testados e todos os resultados obtidos podem ser consultados em [11].

No entanto, a implementação para a virtualização ainda não permite a correta sincronização entre as máquinas de forma a permitir uma gestão mais complexa dos eventos a partir do monitor. Neste momento todas as aplicações iniciam-se ao mesmo tempo (aproximadamente) e não está implementado o recomeço da geração de tráfego após a primeira vez. Os modelos de eventos de simuladores comuns incluem o início, fim e execução paralela de múltiplas aplicações. Num ambiente de múltiplas máquinas virtuais é simples ter várias execuções paralelas, mas a sua sincronização é complexa e necessita um esforço complementar para ser atingido. Para os cenários testados este aspeto não trouxe nenhuma limitação, mas a criação de cenário de teste mais complexos não podem ser realizados sem esses componentes.

Na próxima secção são apresentadas as conclusões deste capítulo.

7.5 Conclusões

A utilização de ambientes virtualizados revelou-se muito interessante devido à flexibilidade que já permite na criação de cenários de rede. A replicação de uma rede real de forma fiel é assim possível e as possibilidades de execução e análise são as mesmas de uma rede a operar num ambiente normal. O uso do *Xen hypervisor* revelou-se muito motivante pelo grande conjunto de funcionalidades para a gestão de máquinas virtuais. No contexto deste projeto outros *hypervisors* poderiam ser usados, permitido uma implementação com igual abrangência e resultados, mas o *XenServer* revelou-se adequado para o suporte às simulações, simples de utilizar e bem documentado.

O propósito principal deste projeto foi a gestão de máquinas virtuais para a execução de simulações simples a partir das suas respetivas descrições NSDL e esse objetivo foi conseguido, principalmente, através de máquinas Linux e do gerador de tráfego D-ITG. Foram utilizados muitos objetos já existentes na *framewok* NSDL e implementadas as bibliotecas para a transformação de uma descrição NSDL num *shell script*.

Os resultados conseguidos foram satisfatórios, mas, em termos de criação e configuração de VM's, o processo é ainda simples. A sincronização de cada passo está feita com um tempo de intervalo suficiente para que tudo decorra sem falhas. Por exemplo, a criação de uma VM é rápida, mas a sua inicialização e configuração completa leva algum tempo (1-2 minutos). Se houver muitas máquinas na simulação este passo inicial pode ser muito demorado. Sobre o modelo de eventos, este ainda necessita de melhorias de forma a conseguir um encadeamento mais flexível para todas as ações da simulação, como o arranque e término de diferentes aplicações.

O objetivo mais importante deste projeto era a integração de um novo domínio de redes na *Framework* NSDL. Os objetos de rede que já existiam na linguagem NSDL foram reutilizados e não foi necessário criar nenhum objeto novo. Esse fato demonstra que se conseguiu alargar as funcionalidades da *framework*, preservando a interoperabilidade entre ferramentas. Os cenários testados poderiam, sem alterações, ser executados nos simuladores presentes nos capítulos 5 e 6.

Outra grande mais-valia deste projeto foi a abertura proporcionada para novas áreas de estudo, principalmente para a gestão de redes e sistemas. O foco até ao momento foi o uso da *framework*

no domínio da simulação, mas, a área da gestão das máquinas virtuais engloba muitas tarefas e ainda contém muitos desafios. Um deles é a criação de *interfaces* gráficos para apoio à gestão de domínios virtuais de grande dimensão [143], e, neste momento, a *Framework* NSDL pode surgir como opção para a integração de novas ferramentas e/ou ferramentas existentes.

Capítulo 8

Conclusões e Perspetivas

O trabalho apresentado nesta tese foi realizado no domínio da gestão de redes. Foi realizado o estudo e análise de muitas das ferramentas de rede existentes, suas principais limitações e como a sua utilização pode ser simplificada de forma a auxiliar a gestão de redes. As secções seguintes concluem este trabalho e estão organizadas da seguinte forma: primeiro, é realizada uma análise aos resultados obtidos, e, por fim, são indicadas algumas linhas possíveis para trabalhos futuros.

8.1 Análise do trabalho realizado

O trabalho apresentado nesta tese procurou contribuir para a área de gestão e projeto de redes, proporcionando uma solução para viabilizar a interoperabilidade entre ferramentas de rede e, assim, otimizar a tarefa do gestor de redes. As ferramentas de rede apresentadas no capítulo 2 raramente permitem repetir e/ou complementar a análise realizada sobre uma determinada rede com uma outra ferramenta, sem ser necessário repetir o esforço a descrever as características da rede. As ferramentas encontradas que são exceções a esta regra, ou seja, que admitem dados de outras ferramentas, continham poucas funcionalidades, não podendo assim serem consideradas adequadas para a análise alargada de uma rede.

A dificuldade em se determinar uma abstração que permita a descrição completa e genérica das características de redes e todas as suas operações é um tópico importante que, sendo ultrapassado, permitirá ao gestor de rede conhecer e melhorar o funcionamento das redes de sua responsabilidade. As linguagens pesquisadas, e apresentadas também no capítulo 2, apenas permitem representar algumas das características de uma rede e foram consideradas como pouco extensíveis e independentes, sendo difícil a sua utilizam na descrição de redes diferentes.

A resolução dos problemas encontrados nesta tese foi enquadrada em três elementos, ou fases, fundamentais de desenvolvimento: uma *framework* onde se integrassem todas as ferramentas necessárias e métodos para a sua integração, uma nova linguagem para a descrição completa de redes de dados, e, por fim a validação da *framework* e da linguagem em diferentes ferramentas e domínios de rede. Em termos gerais, a *framework* desenvolvida permite a utilização de ferramentas independentes, mas complementares na ótica da gestão de rede, e providencia a sua interoperabilidade de uma forma simplificada para os seus utilizadores. Uma opção diferente seria o desenvolvimento de uma ferramenta de raiz para realizar todas as funcionalidades necessárias mas, a dimensão e número das tarefas para a gestão de redes torna esta tarefa muito longa e, exceto para grande

equipas de desenvolvimento, muito difícil de ser conseguida. As ferramentas de rede encontradas durante este projeto, por regra, incluem trabalhos e/ou bibliotecas já existentes e desenvolvidas anteriormente e, essa característica, foi também uma das abordagens escolhidas para este projeto. Foi assim então possível obter uma *framework* com um conjunto de funcionalidades alargadas no período de tempo deste projeto.

A *Framework* NSDL desenvolvida neste trabalho propõe uma estrutura em três camadas para organizar as ferramentas para a gestão das redes. A camada de topo integra as ferramentas que realizam o interface com o utilizador, como as ferramentas gráficas para a modelação e visualização de redes. A camada central é responsável pela interoperabilidade e realiza essa função através da utilização de uma estrutura de dados comum que deverá ser usada pelas ferramentas de rede. Fornece ainda as funções para a transformação de dados para as ferramentas que não utilizem nativamente a linguagem adotada pela *framework*. A camada de base contém as ferramentas que realizam as mais variadas tarefas a pedido do utilizador ou de forma automática, não havendo durante a execução, normalmente, interação com os utilizadores. A estrutura proposta procura suportar qualquer tipo de rede e de ferramenta, servindo como orientação para a estruturação de funcionalidades para a gestão de redes.

As *frameworks* [157, 222] estudadas são utilizadas em problemas específicos e a sua extensão para outros problemas de redes não é simples. A *Framework* NSDL, validada no domínio da simulação, foi definida tendo como princípio facilitar a integração de ferramentas com os mais variados objetivos para os mais diversos domínio das redes. Um qualquer grupo de desenvolvimento pode utilizar a *framework*, adicionando novas funcionalidades e/ou utilizando alguma das ferramentas já integradas na *framework*.

A linguagem proposta para a descrição de redes, *Network Scenarios Description Language* (NSDL), preocupou-se não apenas com a descrição das topologias de redes e os atributos dos objetos, mas também com as várias fases da existência de uma rede, desde o seu projeto até a sua operação, manutenção e atualização. A descrição de cada grupo de informação é realizada de forma independente e esta é uma característica única na linguagem NSDL, sendo que nas restantes linguagens apresentadas a informação da topologia e do domínio de utilização estão agregadas. Uma vantagem importante desta abordagem é permitir a coexistência de diversas informações de utilização sobre uma mesma rede numa única descrição, mas mantendo cada uma independente das restantes.

O suporte à descrição de redes é ainda complementado com os elementos <templates> e <views>. Os <templates> são úteis na descrição de redes com muitos objetos idênticos. As <views> permitem, de forma flexível, a organização dos objetos presentes numa descrição de rede. Outro componente importante da linguagem são os perfis e este serve para agrupar os elementos da linguagem necessário para um determinado contexto ou domínio. O propósito deste componente é facilitar aos utilizadores a utilização da *framework*, oferecendo-lhes apenas os objetos necessários, e, mais importante, facilitar a interoperabilidade entre ferramentas. As descrições realizadas sob o mesmo perfil deverão ser compatíveis e a sua reutilização será direta entre ferramentas.

A linguagem proposta procura ser simples e flexível mas sem deixar de permitir descrições detalhadas. É muito extensível, de forma a acomodar novos objetos, redes e cenários, ultrapassando a principal limitação detetada nas linguagens existentes e reforçando as possibilidades de interoperabilidade. Além da separação da informação da topologia da dos cenários, a linguagem NSDL

ainda possui diversos elementos novos e não presentes em outras linguagens, como os *templates*, as *views* e os perfis.

O primeiro trabalho desenvolvido foi a seleção de um conjunto de ferramentas para suportar um domínio da análise de redes, a simulação. Apesar de existirem simuladores largamente usados na validação e no teste de tecnologias existentes e novas, a sua utilização não é simples e obriga a longos períodos de aprendizagem. As suas linguagens para a descrição de um cenário de simulação são, ou estão próximas de, linguagens de programação genéricas e não existem componentes gráficos na maioria das aplicações para algumas, ou qualquer uma, das suas funções. A melhoria do processo de construção de descrições de simulações de rede passou então pelo desenvolvimento de duas ferramentas gráficas e intuitivas para apoiar o utilizador do simulador a construir descrições.

As aplicações desenvolvidas, *Visual Network Simulator* (VNS) e *Visual Network Descriptor* (VND), foram projetos para o apoio à modelação de cenários de rede heterogéneos, respetivamente para ambiente *desktop* e *online*. Em ambas houve a preocupação em suportar não apenas as redes com fios tradicionais, já suportadas em ferramentas semelhantes, mas apoiar a descrição de outras redes. O VNS destaca-se no suporte à descrição flexível de redes com Qualidade de Serviço, mais especificamente a arquitetura de Serviços Diferenciados. No caso do VND, as redes suportadas são as redes sem fios, nomeadamente as redes Wi-Fi, WiMAX e LTE, e em ambiente *online*. Os utilizadores dos simuladores *Network Simulator 2* e *Network Simulator 3* podem assim, rapidamente, descrever e executar simulações, sem necessidade de aprenderem as linguagens nativas das ferramentas. Algumas das funcionalidades implementadas apenas existem nestas ferramentas, como a construção de cenários DiffServ flexíveis e a possibilidade de construir descrições *online*.

As restantes ferramentas escolhidas e utilizadas permitiram realizar as funções necessárias para todo o processo de simulação de um cenário de rede, tratando-se das ferramentas já existentes e normalmente utilizadas em conjunto com cada simulador. A utilização de todas estas ferramentas em conjunto de uma forma coordenada foi realizada graças à interoperabilidade proporcionada pela *Framework* NSDL.

A integração do simulador *Network Simulator 2* (NS2) agregou valor às funcionalidades da *Framework* NSDL, permitindo a simulação de redes de dados tradicionais e das redes com Qualidade de Serviço (QoS) com um dos simuladores mais utilizados pela comunidade das redes. A ênfase deste trabalho foi a implementação dos objetos das redes de Serviços Diferenciados (DiffServ) e, dessa forma a *framework* permite a construção, numa ferramenta gráfica online, cenários de rede DiffServ, algo não possível em nenhuma outra ferramenta para o NS2.

A extensão da linguagem para os objetos DiffServ foi baseada na norma que define a arquitetura DiffServ [28]. O perfil DiffServ na linguagem é assim independente de qualquer ferramenta, maximizando a interoperabilidade de descrições de redes com objetos DiffServ. Caso fosse seguida a opção de especificar os objetos seguindo apenas a implementação do NS2, a reutilização de uma descrição NSDL em outra ferramenta poderia não ser possível. A tradução implementada dos cenários descritos em NSDL para o código do NS2 apenas está limitada pelas características do NS2, sendo que nem todos os dados da descrição podem ser mapeados, como por exemplo, a marcação de pacotes na NSDL é baseada nos *5-tuples*, mas, no NS2, apenas é usada a informação do nó de origem e de destino. A solução desta situação passaria pela atualização do NS2 para suporte aos *5-tuples*.

O projeto com o NS2, além de ter permitido a simulação de diferentes redes, teve a aplicação da *Framework* NSDL num ambiente didático e onde houve a participação de um grupo de alunos.

A estes foi pedida a realização de simulações com ambas as linguagens, OTcl e NSDL, e, no final, o preenchimento de um inquérito onde foram questionados sobre o processo de simulação com cada linguagem. Uma das principais conclusões retiradas da comparação entre as descrições OTcl, nativa do NS2, e XML, proposta pela *Framework* NSDL, é que em ambos os casos os utilizadores tiveram dificuldades semelhantes para criar as descrições dos cenários de simulação. Os utilizadores, estando a ter o primeiro contato com a simulação com o NS2, referiram a importância em ter existido um ambiente gráfico que facilitasse todo o processo (apenas disponível para segundo grupo, mais pequeno). No geral, os resultados obtidos através da experiência de utilização e do resultado do inquérito mostram a existência de interesse e de muitas vantagens em utilizar uma *framework* com estas características.

A adição do *Network Simulator 3* (NS3) possibilitou a validação da *Framework* NSDL com uma nova classe de simulador, mais flexível e expressiva, uma vez que esta ferramenta consegue simular uma gama diferente de tecnologias e domínios de redes. Em particular, foram definidas e implementadas algumas tecnologias de rede sem fios na linguagem NSDL, como as redes Wi-Fi, Mesh, WiMAX e LTE. Uma limitação grande do simulador NS3 era a inexistência de ambientes gráficos ricos para a descrição de redes e, utilizando as ferramentas gráficas da *Framework* NSDL, foi possível oferecer aos seus utilizadores uma nova forma de especificarem as simulações com este simulador.

Um aspeto importante da implementação das bibliotecas para o NS3, foi a maior proximidade entre as características dos objetos do simulador e as definições dos objetos da NSDL. Apesar de garantir a boa operação do simulador na *framework*, podem surgir eventuais limitações na interoperabilidade das descrições devido a alguns objetos serem únicos ao NS3. Mas a grande maioria dos objetos implementados e testados são comuns a muitas das ferramentas e, pelo menos para esses, não há dificuldade em serem utilizados em diferentes simuladores.

As simulações realizadas com as redes sem fios permitiram testar e melhorar as novas bibliotecas desenvolvidas para *framework*, mas o trabalho foi ainda complementado com a execução de simulações de redes iguais nos dois simuladores já integrados na *framework*, demostrando a capacidade desta em efetuar análises mais ricas sobre uma determinada rede. A execução de simulações em diferentes simuladores era apenas possível se o utilizador fosse experiente nas várias linguagens, algo pouco comum na comunidade de simulação [211]. Com a *Framework* NSDL, os utilizadores apenas necessitam conhecer uma linguagem, a NSDL, ou mesmo utilizarem diretamente uma ferramenta gráfica para criar o seu cenário de redes, e contar com as bibliotecas da *framework* para a obtenção do código nativo de cada simulador.

Como contribuição de novos domínios para a *framework*, foi ainda apresentado um último projeto que a estendeu a *framework* para a sua aplicação aos ambientes virtualizados. Os componentes selecionados e desenvolvidos permitem, a partir de uma descrição de rede NSDL, invocar a criação de máquinas virtuais numa plataforma de virtualização e desencadear todos os eventos definidos para a realização de testes e validações das aplicações, protocolos e equipamentos. Além de estender a *framework* para um novo domínio, procurou-se também obter uma nova forma de testar as redes de computadores mais próxima da realidade.

O problema principal nos simuladores discretos é a validade dos seus modelos e, consequentemente, a qualidade dos seus resultados [147] e, a vantagem de um ambiente virtual é que as implementações de alguns objetos são idênticas às reais, como as aplicações e as camadas proto-

colares de rede numa máquina virtualizada. Assim, a operação de uma rede em ambiente real e a sua simulação num ambiente virtual, produz resultados semelhantes.

No entanto, as ferramentas para a gestão de ambientes virtuais são ainda limitadas [143], colocando a *Framework* NSDL como uma opção possível para a implementação de novas funções de gestão neste domínio.

Uma preocupação no início do projeto foi conhecer qual o tempo necessário para a integração de cada ferramenta ou domínio de rede na *framework*. O desenvolvimento das bibliotecas para o NS2 foi demorado devido a ter servido como base para testar e refinar os processos de definição dos objetos e de criação das bibliotecas de transformação, e como tal não pode ser usado como base de duração normal de um projeto. Após a conclusão da primeira versão mais estável da *framework*, foi então considerado fundamental repetir o processo para diferentes domínios e, de preferência, com utilizadores inexperientes com a NSDL de forma a avaliar (1) a facilidade/dificuldade na compreensão da *Framework* NSDL e (2) qual o tempo necessário para conseguir desenvolver a extensão da *framework* para novos domínios.

Os projetos com o simulador NS3 e com os Ambientes Virtualizados foram implementados com alunos de mestrado, aos quais o autor deste trabalho deu apoio e suporte para a compreensão dos componentes da *framework*. Em ambos os casos considerou-se que os trabalhos foram executados num tempo pequeno, comparativamente ao projeto do NS2. A compreensão e desenvolvimento das bibliotecas não levou mais de 2-3 meses e foram ainda necessários mais 2-3 meses para a compreensão da ferramenta e testes das bibliotecas de tradução. Ambos os alunos não tinham experiência no domínio que estavam a trabalhar e, portanto, foi ainda necessário tempo para essa aprendizagem. A duração de um trabalho de mestrado, aproximadamente 10 meses, foi suficiente para obter implementações alargadas das ferramentas e que corresponderam às expetativas iniciais propostas nos trabalhos.

Apesar de terem sido realizados e apresentados apenas dois trabalhos, a experiência serviu para consolidar a perceção do autor que os processos definidos são simples e possibilitam a integração de uma nova ferramenta na *framework* num intervalo de tempo não muito longo. É evidente que este tempo está sempre dependente da ferramenta, dos domínios de rede que se pretendem cobrir, da complexidade dos cenários que se querem testar e da experiência dos utilizadores nas várias tecnologias envolvidas.

Assim, e no geral, ficou demonstrado que o processo criado, bem como as ferramentas e as bibliotecas desenvolvidas, foi adequado para o ambiente de simulação de redes e pode ser replicado para outros domínios. Conseguiu-se em poucas semanas, com utilizadores sem experiência relevante na área das redes, replicar o processo para novos simuladores. A arquitetura simples e a estrutura clara da *framework*/linguagem permitiram o desenvolvimento para diferentes ferramentas e ambientes de uma forma rápida e eficaz.

A utilização da *Framework* NSDL foi ainda validada no ensino de disciplinas da área das redes. Com a *framework*, o foco da aprendizagem é na rede, nos seus objetos e parâmetros e menos na linguagem programação do simulador. A existência de um repositório de cenários permitirá a realização de diferentes experimentos de rede e pode servir de base para os projetos desenvolvidos nas aulas. A flexibilidade da linguagem permite muitos testes e variações às simulações rapidamente, dando tempo para o aluno compreender de forma mais clara o funcionamento da rede. Na próxima seção são apresentadas algumas perspetivas de continuação deste trabalho.

8.2 Trabalho Futuro

Os trabalhos futuros podem ser divididos em duas categorias: *framework* e linguagem. O desenvolvimento da *framework* deverá passar pela integração de um conjunto maior de ferramentas, principalmente para a análise e visualização de resultados. Os simuladores utilizados permitem já a simulação de muitos cenários, mas ainda impõem aos utilizadores um esforço grande para a análise dos seus resultados. As ferramentas propostas por [111, 272] são opções interessantes para complementar a *framework*, melhorando as possibilidades oferecidas por esta na verificação dos resultados das simulações e na compreensão do funcionamento e estado da rede.

No sentido inverso, as ferramentas de rede também podem utilizar a *framework* para complementar as suas funcionalidades. Qualquer ferramenta sem interface gráfico poderá usar o VND e as bibliotecas da NSDL e incluir essa funcionalidade na sua estrutura. Será importante melhorar a documentação e divulgação do trabalho de forma a motivar outras equipas de investigação da área das redes na utilização da *Framework* NSDL.

Do ponto de vista da linguagem, uma primeira abordagem é utilizar a linguagem NSDL para descrever novos tipos de rede. As redes de sensores sem fios, por exemplo, têm sido amplamente desenvolvidas e verifica-se, à semelhança de outros domínios de redes, a existência de múltiplas e distintas ferramentas [122]. A NSDL deverá ser testada como opção para descrever os objetos deste tipo de redes e ser utilizada nestas ferramentas de forma a facilitar a utilização e a construção de análises sobre este tipo de redes. É também crescente o interesse na utilização de ambiente virtuais para os mais variados fins e o trabalho de [70] propõe uma descrição cuidada do nó virtual e cremos que a NSDL poderá complementar com os restantes elementos da rede e, mais importante, para a descrição dos cenários possíveis nas diversas plataformas de virtualização. Vale a pena referir, a colaboração da Universidade da Madeira com a Universidade Salvador (UNIFACS), Bahia-Brasil, na realização da aplicação da *Framework* NSDL a outros domínios e tipos de redes, tais como *OpenFlow* [170] e MPLS [86], abordando aspetos de Qualidade de Serviço e Engenharia de Tráfego, através dos trabalhos em andamento de alunos de mestrado da UNIFACS.

A proposta de uma nova linguagem e *framework* procurou atender a algumas limitações que existem na interoperabilidade de muitos projetos, de pequena e grande dimensão, em muitos domínios da gestão das redes. A possibilidade de reutilizar trabalhos anteriores e/ou repetir análises são funcionalidades sempre procuradas e importantes para agilizar qualquer processo de desenvolvimento e/ou de implementação, seja nas redes atuais, seja nas redes futuras. A *Framework* NSDL, incluindo as ferramentas e, principalmente, a linguagem de descrição, visa facilitar esse processo aos intervenientes no domínio das redes. Os trabalhos apresentados procuraram confirmar a exequibilidade de um projeto com estas características e demonstrar as vantagens que podem existir para o domínio das redes. Os vários domínios de rede e as múltiplas ferramentas integrados com sucesso na *framework*, pela sua abrangência, confirmam que é possível aprofundar o conhecimento de uma rede sem ser necessário aprender novas linguagens e ferramentas. Necessitará ainda de algum trabalho e, principalmente, divulgação para obter uma aceitação mais ampla, mas os princípios e as implementações já realizadas, apresentadas e detalhadas ao longo desta tese, já permitem a execução de muitas tarefas no domínio da gestão de redes.

Publicações do Autor

Marques, E.M.D.; Sousa J.J.F.; Sampaio, P.M.N., *Providing a Graphical User Interface for the Modelling of Network Simulator 3 Scenarios*, in Proceedings of EUROSIS - The European Multidisciplinary Society for Modelling and Simulation Technology (ESM'2012), Essen, Germany, October 22-24, 2012.

Eduardo M.D. Marques and Paulo N.M. Sampaio, *NSDL: an integration framework for the network modeling and simulation*, International Journal of Modeling and Optimization, Vol. 2, No. 3, June 2012

Marques, E.M.D.; Sousa J.J.F.; Sampaio, P.M.N., *Modeling and Simulation of DiffServ Scenarios with the NSDL Framework*, Proceedings of 7th International Conference on Network and Service Management (CNSM 2011), Paris, France, October 25-28, 2011 (short-paper).

Marques, E.M.D.; Sousa J.J.F.; Sampaio, P.M.N., *NS-3 Simulation and Management of WiMAX and LTE Networks with NSDL*, in Proceedings of EUROSIS - The European Multidisciplinary Society for Modelling and Simulation Technology (ESM'2011), Guimarães, Portugal, October 24-26, 2011.

Marques, E.M.D.; Sampaio, P.N.M., *NSDL: An Integration Framework for the Network Modelling and Simulation.* In Proceedings of the 2010 IEEE International Conference on Modeling, Simulation and Control (ICMSC 2010). Cairo, Egypt. November 2-4, 2010.

Marques, E.M.D.; Sampaio, P.N.M., *A Framework for the Integration of Network Modelling and Simulation Tools.* In Proceedings of EUROSIS - The European Multidisciplinary Society for Modelling and Simulation Technology (ESM'2010). Hasselt, Belgium, October 25th-27th 2010.

Marques, E.M.D.; Brito, L.M.P.L.; Sampaio, P.N.M.; Rodriguez Peralta, L.M. (2010) *An Analysis of Quality of Service Architectures : Principles, Requirements, and Future Trends.* In P. Bhattarakosol (Ed.) *Intelligent Quality of Service Technologies and Network Management: Models for Enhancing Communication* (pp. 15-35). IGI Global

Marques, E.M.D.; Plácido, R.A.S.A.; Sampaio, P.M.N., *Visual Network Simulator (VNS): A GUI to QoS Simulation for the ns-2 Simulator*, In Proceedings of ACS/IEEE International Conference on Computer Systems and Applications, AICCSA'09, Rabat, Morocco, May 10-13, 2009

Marques, E.M.D.; Plácido, R.A.S.A.; Sampaio, P.M.N., *Providing a Graphical User Interface for ns-2 with Visual Network Simulator*, In Proceedings of 2nd International Conference on Simulation Tools and Techniques, SIMUTools'09, Rome, Italy, March 2-6, 2009 (short-paper).

Referências

[1] 3GPP. Evolved universal terrestrial radio access (e-utra); lte physical layer; general description [online], Dec 2010. Release 10. 115

[2] 3GPP. General universal mobile telecommunications system (umts) architecture [online], Mar 2011. Release 10. 115

[3] ACM. Acm digital library [online]. (Acedido em agosto de 2012). 23

[4] Keith Adams and Ole Agesen. A comparison of software and hardware techniques for x86 virtualization. In *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, ASPLOS XII, pages 2–13, New York, NY, USA, 2006. ACM. 141

[5] R.G. Addie, Yu Peng, and M. Zukerman. Netml: Networking networks. In *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, pages 1055 –1060, dec. 2011. 26

[6] Ron Addie, Stephen Braithwaite, and Abdulla Zareer. Netml - a language and website for collaborative work on networks and their algorithms. In *2006 Australian Telecommunication Networks and Applications Conference (ATNAC)*, Melbourne, Australia, dec. 2006. 26

[7] Alfred V. Aho, Brian W. Kernighan, and Peter J. Weinberger. *The AWK programming language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1987. 42

[8] aiSee. Graph visualization [online], 2012. (Acedido em dezembro de 2012). 14

[9] Ehab Al-Shaer, Albert Greenberg, Charles Kalmanek, David A. Maltz, T. S. Eugene Ng, and Geoffrey G. Xie. New frontiers in internet network management. *SIGCOMM Comput. Commun. Rev.*, 39(5):37–39, October 2009. 10

[10] Thomas Anderson, Larry Peterson, Scott Shenker, and Jonathan Turner. Overcoming barriers to disruptive innovation in networking. In *Report of NSF Workshop, 2005*, 2005. 2

[11] Joana Patrícia Mendes Araújo. Virtualização automática de cenários de rede. Master's thesis, Centro de Competências de Ciências Exactas e Engenharias da Universidade da Madeira, Setembro 2011. 135, 151

[12] D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, and X. Xiao. Overview and principles of internet traffic engineering. RFC 3272 (Informational), May 2002. Updated by RFC 5462. 82

[13] Jesuino da Costa Serra de Azevedo. Visual network descriptor: Suporte à simulação de redes através da autoria gráfica de cenários virtuais. Master's thesis, Centro de Competências de Ciências Exactas e Engenharias da Universidade da Madeira, novembro 2010. 41

[14] J. Babiarz, K. Chan, and F. Baker. Configuration guidelines for diffserv service classes. RFC 4594 (Informational), August 2006. Updated by RFC 5865. 81

[15] Sandeep Bajaj, Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, Padma Haldar, Mark Handley, Ahmed Helmy, John Heidemann, Polly Huang, Satish Kumar, Steven McCanne, Reza Rejaie, Puneet Sharma, Kannan Varadhan, Ya Xu, Haobo Yu, and Daniel Zapalla. Improving simulation for network research. Technical report, University of Southern California, 1999. 19, 42, 76

[16] Mike Baker. What is a software framework? and why should you like 'em? [online], October 2009. (Acedido em janeiro de 2012). 36

[17] Carsten Ball. Lte and wimax: Technology and performance comparison [online]. Panel 1: WiMax and 3GPP LTE: How are they related?, April 2007. Release 10. 132

[18] Hitesh Ballani and Paul Francis. Conman: a step towards network manageability. *SIGCOMM Comput. Commun. Rev.*, 37(4):205–216, August 2007. 9

[19] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, 37(5):164–177, October 2003. 140, 141

[20] Rimon Barr. *JiST – Java in Simulation Time: An efficient, unifying approach to simulation using virtual machines*. PhD thesis, Cornell University, May 2004. 19

[21] Rimon Barr, Zygmunt J. Haas, and Robbert van Renesse. Jist: an efficient approach to simulation using virtual machines. *Software: Practice and Experience*, 35(6):539–576, 2005. 19

[22] Rimon Barr, Zygmunt J. Haas, and Robbert van Renesse. *Scalable Wireless Ad hoc Network Simulation*, chapter 19, pages 297–311. CRC Press, August 2005. 19

[23] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. Gephi: An open source software for exploring and manipulating networks. In *International AAAI Conference on Weblogs and Social Media*, 2009. 14

[24] Yigal Bejerano. Taking the skeletons out of the closets: a simple and efficient topology discovery scheme for large ethernet lans. *IEEE/ACM Trans. Netw.*, 17(5):1385–1398, October 2009. 14

[25] Y. Bernet, S. Blake, D. Grossman, and A. Smith. An informal management model for diffserv routers. RFC 3290 (Informational), May 2002. 80

[26] Y. Bernet, P. Ford, R. Yavatkar, F. Baker, L. Zhang, M. Speer, R. Braden, B. Davie, J. Wroclawski, and E. Felstaine. A framework for integrated services operation over diffserv networks. RFC 2998 (Informational), November 2000. 83

[27] M. Bjorklund. YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF). RFC 6020 (Proposed Standard), October 2010. 27

[28] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated service. RFC 2475 (Informational), December 1998. Updated by RFC 3260. 40, 79, 88, 100, 155

[29] Bela Bollobas. *Graph theory : an introductory course*. Springer Verlag, New York, 1979. Includes index. 49

[30] Alessio Botta, Alberto Dainotti, and Antonio Pescapè. A tool for the generation of realistic network workload for emerging networking scenarios. *Computer Networks (Elsevier)*, 2012. 143

[31] Ch. Bouras, D. Primpas, and K. Stamos. Enhancing ns-2 with diffserv qos features. In *Proceedings of the 2007 spring simulaiton multiconference - Volume 1*, SpringSim '07, pages 117–124, San Diego, CA, USA, 2007. Society for Computer Simulation International. 83

[32] R. Braden, D. Clark, and S. Shenker. Integrated services in the internet architecture: an overview. RFC 1633 (Informational), June 1994. 78

[33] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource reservation protocol (rsvp) – version 1 functional specification. RFC 2205 (Proposed Standard), September 1997. Updated by RFCs 2750, 3936, 4495, 5946, 6437. 78

[34] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Ya Xu, and Haobo Yu. Advances in network simulation. *Computer*, 33(5):59 –67, may 2000. 19, 37, 67, 76

[35] R.S. Cahn. *Wide Area Network Design: Concepts and Tools for Optimization*. The Morgan Kaufmann Series in Networking. Morgan Kaufmann, 1998. 31

[36] Zheng Cai. *Maestro: Achieving Scalability and Coordination in Centralized Network Control Plane*. PhD thesis, Rice University, August 2011. 10

[37] Zheng Cai, Alan L. Cox, and T. S. Eugene Ng. Maestro: A system for scalable openflow control. Technical Report TR10-08, Rice University, Dec 2010. 10

[38] Zheng Cai, Alan L. Cox, and T. S. Eugene Ng. Maestro: Balancing fairness, latency and throughput in the openflow control plane. Technical Report TR11-07, Rice University, Dec 2011. 10

[39] Zheng Cai, Florin Dinu, Jie Zheng, Alan L. Cox, and T. S. Eugene Ng. The preliminary design and implementation of the maestro network control platform. Technical Report TR08-13, Rice University, October 2008. 11

[40] Tiago Camilo, Jorge Sá Silva, André Rodrigues, and Fernando Boavida. Gensen: a topology generator for real wireless sensor networks deployment. In *Proceedings of the 5th IFIP WG 10.2 international conference on Software technologies for embedded and ubiquitous systems*, SEUS'07, pages 436–445, Berlin, Heidelberg, 2007. Springer-Verlag. 14

[41] Andrew T. Campbell, Herman G. De Meer, Michael E. Kounavis, Kazuho Miki, John B. Vicente, and Daniel Villela. A survey of programmable networks. *SIGCOMM Comput. Commun. Rev.*, 29(2):7–23, April 1999. 139

[42] A. Campi and F. Callegati. Network resource description language. In *Proc. IEEE GLOBECOM Workshops*, pages 1–6, 2009. 27

[43] R. Canonico, D. Emma, and G. Ventre. Extended nam: An ns2 compatible network topology editor for simulation of web caching systems on large network topologies. *European Simulation and Modelling Conference*, 2003. 2

[44] Roberto Canonico, Donato Emma, and Giorgio Ventre. An xml description language for web-based network simulation. In *Distributed Simulation and Real-Time Applications, 2003. Proceedings. Seventh IEEE International Symposium on*, pages 76 – 81, oct. 2003. 26, 54

[45] Xiaojun Cao, Yang Wang, Adrian Caciula, and Yichuan Wang. Developing a multifunctional network laboratory for teaching and research. In *Proceedings of the 10th ACM conference on SIG-information technology education*, SIGITE '09, pages 155–160, New York, NY, USA, 2009. ACM. 76

[46] Gustavo Carneiro. Python bindings generator [online], 2007. (Acedido em agosto de 2012). 112

[47] Gustavo Carneiro. Pyviz: Live simulation visualizer [online], 2012. (Acedido em julho de 2012). 42

[48] Gustavo João Alves Marques Carneiro. *Transparent Metropolitan Vehicular Network: Design and Fast Prototyping Methodology*. PhD thesis, Universidade do Porto, julho 2012. 30, 113

[49] J.D. Case, M. Fedor, M.L. Schoffstall, and J. Davin. Simple Network Management Protocol (SNMP). RFC 1157 (Historic), May 1990. 43

[50] L. Chappell and G. Combs. *Wireshark Network Analysis: The Official Wireshark Certified Network Analyst Study Guide*. Chappell University. Laura Chappell University, 2010. 17, 42, 114

[51] Jianhua Che, Qinming He, Qinghua Gao, and Dawei Huang. Performance measuring and comparing of virtual machine monitors. In *Embedded and Ubiquitous Computing, 2008. EUC '08. IEEE/IFIP International Conference on*, volume 2, pages 381 –386, dec. 2008. 141

[52] Lechang Cheng, Norm C. Hutchinson, and Mabo R. Ito. Realnet: A topology generator based on real internet topology. In *Proceedings of the 22nd International Conference on Advanced Information Networking and Applications - Workshops*, AINAW '08, pages 526–532, Washington, DC, USA, 2008. IEEE Computer Society. 13

[53] Timothy Chiu. Getting proactive network management from reactive network management tools. *Int. J. Netw. Manag.*, 8(1):12–17, February 1998. 11

[54] N.M. Mosharaf Kabir Chowdhury and Raouf Boutaba. A survey of network virtualization. *Comput. Netw.*, 54(5):862–876, April 2010. 138, 139

[55] N.M.M.K. Chowdhury and R. Boutaba. Network virtualization: state of the art and research challenges. *Communications Magazine, IEEE*, 47(7):20 –26, july 2009. xvi, 138

[56] Christopher A. Chung. *Simulation Modeling Handbook: A Practical Approach*. Industrial and Manufacturing Engineering Series. CRC Press, Inc., Boca Raton, FL, USA, 2003. 18

[57] Selim Ciraci and Bora Akyol. An evaluation of the network simulators in large-scale distributed simulations. In *Proceedings of the first international workshop on High performance computing, networking and analytics for the power grid*, HiPCNA-PG '11, pages 59–66, New York, NY, USA, 2011. ACM. 33

[58] Citrix. Citrix systems, inc. [online]. (Acedido em agosto de 2012). 139

[59] David D. Clark and Wenjia Fang. Explicit allocation of best-effort packet delivery service. *IEEE/ACM Trans. Netw.*, 6(4):362–373, August 1998. 85

[60] David D. Clark, Craig Partridge, J. Christopher Ramming, and John T. Wroclawski. A knowledge plane for the internet. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '03, pages 3–10, New York, NY, USA, 2003. ACM. 9

[61] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol (OLSR). RFC 3626 (Experimental), October 2003. 122

[62] Les Cottrell. Network monitoring tools [online], 2012. (Acedido em dezembro de 2012). 1, 15

[63] Simon Crosby and David Brown. The virtualization reality. *Queue*, 4(10):34–41, December 2006. 136

[64] D-ITG. Distributed internet traffic generator [online]. (Acedido em agosto de 2012). 143

[65] B. Davie, A. Charny, J.C.R. Bennet, K. Benson, J.Y. Le Boudec, W. Courtney, S. Davari, V. Firoiu, and D. Stiliadis. An expedited forwarding phb (per-hop behavior). RFC 3246 (Proposed Standard), March 2002. 81

[66] Zesong Di and H.T. Mouftah. Quips-ii: a simulation tool for the design and performance evaluation of diffserv-based networks. *Computer Communications*, 25(11 - 12):1125 – 1131, July 2002. 110

[67] Dia. Gnu diagramming tool [online]. (Acedido em dezembro de 2012). 12

[68] DML. Domain modeling language [online]. (Acedido em setembro de 2012). 20

[69] DMTF. Common information model (cim) [online]. (Acedido em setembro de 2012). 2, 27

[70] Dalibor Dobrilovic, Zeljko Stojanov, Borislav Odadzic, and Branko Markoski. Using network node description language for modeling networking scenarios. *Adv. Eng. Softw.*, 43(1):53–64, January 2012. 158

[71] Doxygen. Source code documentation generator tool [online]. (Acedido em dezembro de 2012). 112

[72] Ulrich Drepper. The cost of virtualization. *Queue*, 6(1):28–35, January 2008. 138

[73] Dominique Dudkowski, Marcus Brunner, Giorgio Nunzi, Chiara Mingardi, Chris Foley, Miguel Ponce de Leon, Catalin Meirosu, and Susanne Engberg. Architectural principles and elements of in-network management. In *Proceedings of the 11th IFIP/IEEE international conference on Symposium on Integrated Network Management*, IM'09, pages 529–536, Piscataway, NJ, USA, 2009. IEEE Press. 10

[74] Eclipse. Eclipse project [online]. (Acedido em setembro de 2012). 20

[75] E. Egea-Lopez, J. Vales-Alonso, A. Martinez-Sala, P. Pavon-Mario, and J. Garcia-Haro. Simulation scalability issues in wireless sensor networks. *Communications Magazine, IEEE*, 44(7):64 – 73, july 2006. 19

[76] M. Eisler. XDR: External Data Representation Standard. RFC 4506 (Standard), May 2006. 16

[77] C. Eklund, R.B. Marks, K.L. Stanwood, and S. Wang. Ieee standard 802.16: a technical overview of the wirelessman/sup tm/ air interface for broadband wireless access. *Communications Magazine, IEEE*, 40(6):98 –107, june 2002. 115

[78] R. Enns. NETCONF Configuration Protocol. RFC 4741 (Proposed Standard), December 2006. Obsoleted by RFC 6241. 2, 27

[79] Juan M. Estévez-Tapiador, Pedro García-Teodoro, and Jesús E. Díaz-Verdejo. Nsdf: a computer network system description framework and its application to network security. *Comput. Netw.*, 43(5):573–600, December 2003. 26, 31

[80] Deborah Estrin, Mark Handley, John Heidemann, Steven McCanne, Ya Xu, and Haobo Yu. Network visualization with nam, the vint network animator. *Computer*, 33(11):63–68, November 2000. 30, 40, 77

[81] K. Etemad. Overview of mobile wimax technology and evolution. *Communications Magazine, IEEE*, 46(10):31 –40, october 2008. 118

[82] W. Fang, N. Seddigh, and B. Nandy. A time sliding window three colour marker (tswtcm). RFC 2859 (Experimental), June 2000. 85

[83] Farlex. The american heritage©dictionary of the english language [online]. (Acedido em janeiro de 2012). 35

[84] Jahanzeb Farooq and Thierry Turletti. An ieee 802.16 wimax module for the ns-3 simulator. In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, Simutools '09, pages 8:1–8:11, ICST, Brussels, Belgium, Belgium, 2009. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). 122, 124, 130

[85] F. Le Faucheur and W. Lai. Requirements for support of differentiated services-aware mpls traffic engineering. RFC 3564 (Informational), July 2003. Updated by RFC 5462. 83

[86] F. Le Faucheur, L. Wu, B. Davie, S. Davari, P. Vaananen, R. Krishnan, P. Cheval, and J. Heinanen. Multi-protocol label switching (mpls) support of differentiated services. RFC 3270 (Proposed Standard), May 2002. Updated by RFC 5462. 83, 158

[87] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Netw.*, 1(4):397–413, August 1993. 83

[88] Christopher Foley, Sasitharan Balasubramaniam, Eamonn Power, Miguel Ponce Leon, Dmitri Botvich, Dominique Dudkowski, Giorgio Nunzi, and Chiara Mingardi. A framework for in-network management in heterogeneous future communication networks. In *Proceedings of the 3rd IEEE international workshop on Modelling Autonomic Communications Environments*, MACE '08, pages 14–25, Berlin, Heidelberg, 2008. Springer-Verlag. 10

[89] Juan Luis Font, Pablo Iñigo, Manuel Domínguez, José Luis Sevillano, and Claudio Amaya. Architecture, design and source code comparison of ns-2 and ns-3 network simulators. In *Proceedings of the 2010 Spring Simulation Multiconference*, SpringSim '10, pages 109:1–109:8, San Diego, CA, USA, 2010. Society for Computer Simulation International. 114

[90] Brian Foote and Joe Yoder. Big ball of mud. In Brian Foote Neil Harrison and Hans Rohnert, editors, *Pattern Languages of Program Design 4*, pages 277–288. Addison Wesley, 2000. 35, 37

[91] Martin Fowler. *Refactoring: improving the design of existing code.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999. 35

[92] FreeBSD. The freebsd project [online]. (Acedido em dezembro de 2012). 19

[93] Fuxiang Gao, Fengyun Li, Shengfei Bao, and Xiaojing Wang. Analysis and implementation of secure console server based on embedded linux. In *Industrial Technology, 2008. ICIT 2008. IEEE International Conference on*, pages 1 –6, april 2008. 142

[94] Matthew S Gast. *802.11 Wireless Networks: The Definitive Guide, Second Edition.* O'Reilly Media, Inc., 2005. 115, 116

[95] M. Ghijsen, J. van der Ham, P. Grosso, and C. de Laat. Towards an infrastructure description language for modeling computing infrastructures. In *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*, pages 207 –214, july 2012. 30

[96] A. Ghosh, R. Ratasuk, B. Mondal, N. Mangalvedhe, and T. Thomas. Lte-advanced: next-generation wireless broadband technology [invited paper]. *Wireless Communications, IEEE*, 17(3):10 –22, june 2010. 119

[97] Gliffy. Online diagramming application [online]. (Acedido em dezembro de 2012). 13

[98] Robert P. Goldberg. Survey of virtual machine research. *Computer*, 7(9):34–45, September 1974. 136

[99] Albert Greenberg, Gisli Hjalmtysson, David A. Maltz, Andy Myers, Jennifer Rexford, Geoffrey Xie, Hong Yan, Jibin Zhan, and Hui Zhang. A clean slate 4d approach to network control and management. *SIGCOMM Comput. Commun. Rev.*, 35(5):41–54, October 2005. 9

[100] D.B. Grossman. An overview of frame relay technology. In *Computers and Communications, 1991. Conference Proceedings., Tenth Annual International Phoenix Conference on*, pages 539 –545, mar 1991. 42

[101] Guifen Gu and Guili Peng. The survey of gsm wireless communication system. In *Computer and Information Application (ICCIA), 2010 International Conference on*, pages 121 –124, dec. 2010. 115

[102] Diwaker Gupta, Ludmila Cherkasova, Rob Gardner, and Amin Vahdat. Enforcing performance isolation across virtual machines in xen. In Maarten van Steen and Michi Henning, editors, *Middleware*, volume 4290 of *Lecture Notes in Computer Science*, pages 342–362. Springer, 2006. 140

[103] R. Händel, M.N. Huber, and S. Schröder. *ATM networks: concepts, protocols, applications*. Electronic systems engineering series. Addison-Wesley, 1998. 42

[104] Richard C. Harlan. Network management with nagios. *Linux J.*, 2003(111):3–, July 2003. 15

[105] D. Harrington, R. Presuhn, and B. Wijnen. An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks. RFC 3411 (Standard), December 2002. Updated by RFCs 5343, 5590. 14, 26

[106] Jeff Heer. Flare - data visualization for the web [online]. (Acedido em dezembro de 2012). 14

[107] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. Assured forwarding phb group. RFC 2597 (Proposed Standard), June 1999. Updated by RFC 3260. 81

[108] J. Heinanen and R. Guerin. A single rate three color marker. RFC 2697 (Informational), September 1999. 85

[109] J. Heinanen and R. Guerin. A two rate three color marker. RFC 2698 (Informational), September 1999. 86

[110] Thomas R. Henderson, Sumit Roy, Sally Floyd, and George F. Riley. ns-3 project goals. In *Proceeding from the 2006 workshop on ns-2: the IP network simulator*, WNS2 '06, New York, NY, USA, 2006. ACM. 18, 20, 37, 112, 113

[111] Rick Hofstede and Tiago Fioreze. Surfmap: a network monitoring tool based on the maps api. In *Proceedings of the 11th IFIP/IEEE international conference on Symposium on Integrated Network Management*, IM'09, pages 676–690, Piscataway, NJ, USA, 2009. IEEE Press. 158

[112] Hyper-V. Microsoft hyper-v server [online]. (Acedido em agosto de 2012). 140

[113] Gianluca Iannaccone, Martin May, and Christophe Diot. Aggregate traffic performance with active queue management and drop from tail. *SIGCOMM Comput. Commun. Rev.*, 31(3):4–13, July 2001. 83

[114] IEEE. Ieee xplore digital library [online]. (Acedido em agosto de 2012). 23

[115] IEEE. Ieee standard for telecommunications and information exchange between systems - lan/man - specific requirements - part 15: Wireless medium access control (mac) and physical layer (phy) specifications for wireless personal area networks (wpans). *IEEE Std 802.15.1-2002*, pages 1 –473, 14 2002. 114

[116] IEEE. Ieee standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks specific requirements part 15.4: Wireless medium access control (mac) and physical layer (phy) specifications for low-rate wireless personal area networks (lr-wpans). *IEEE Std 802.15.4-2003*, pages 1 –670, 2003. 114

[117] IEEE. Ieee standard for information technology-telecommunications and information exchange between systems-local and metropolitan area networks-specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. *IEEE Std 802.11-2007 (Revision of IEEE Std 802.11-1999)*, pages C1–1184, 12 2007. 114

[118] IEEE. Ieee standard for local and metropolitan area networks–media access control (mac) bridges and virtual bridged local area networks. *IEEE Std 802.1Q-2011 (Revision of IEEE Std 802.1Q-2005)*, pages 1 –1365, 31 2011. 139

[119] IEEE. Ieee standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, pages 1 –2793, 29 2012. 116

[120] IEEE. Ieee802.11ac: The next evolution of wi-fi standards, May 2012. White Paper. 116

[121] IEEE. Official ieee 802.11 working group project timelines [online], February 2013. (Acedido em fevereiro 2013). 116

[122] M. Imran, A.M. Said, and H. Hasbullah. A survey of simulators, emulators and testbeds for wireless sensor networks. In *Information Technology (ITSim), 2010 International Symposium in*, volume 2, pages 897 –902, june 2010. 1, 19, 158

[123] M. A. Ismail, G. Piro, L. A. Grieco, and T. Turletti. An improved ieee 802.16 wimax module for the ns-3 simulator. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, SIMUTools '10, pages 63:1–63:10, ICST, Brussels, Belgium, Belgium, 2010. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). 124

[124] ISO. ISO/IEC 7498-4 Information technology - Open Systems Interconnection - Basic Reference Model – Part 4: Management framework. Technical report, International Organization for Standardization, June 1989. 8

[125] ISO. ISO/IEC 7498-1 Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model. Technical report, International Organization for Standardization, June 1994. 8

[126] ISO. ISO/IEC 10040 Information technology - Information technology – Open Systems Interconnection – Systems management overview. Technical report, International Organization for Standardization, 1998. 8

[127] Teerawat Issariyakul and Ekram Hossain. *Introduction to Network Simulator NS2*. Springer, 2008. 77

[128] C. Issariyapat, P. Pongpaibool, S. Mongkolluksame, and K. Meesublak. Using nagios as a groundwork for developing a better network monitoring system. In *Technology Management for Emerging Technologies (PICMET), 2012 Proceedings of PICMET '12:*, pages 2771 –2777, 29 2012-aug. 2 2012. 15

[129] ITU-T. M.3010 : Principles for a telecommunications management network [online], Nov 2000. (Acedido em dezembro de 2012). 8, 9

[130] J-Sim. J-sim (formerly known as javasim) [online], 2006. (Acedido em agosto de 2012). 19

[131] Van Jacobson, Craig Leres, and Steven McCanne. tcpdump: Command-line packet analyzer [online], 2012. (Acedido em março de 2012). 46

[132] Ralph E. Johnson. Frameworks = (components + patterns). *Commun. ACM*, 40:39–42, October 1997. 36, 37

[133] Xi Ju, Hongwei Zhang, Wenjie Zeng, Mukundan Sridharan, Jing Li, Anish Arora, Rajiv Ramnath, and Yufeng Xin. Lens: resource specification for wireless sensor network experimentation infrastructures. In *Proceedings of the 6th ACM international workshop on Wireless network testbeds, experimental evaluation and characterization*, WiNTECH '11, pages 35–42, New York, NY, USA, 2011. ACM. 27

[134] Jaroslav Kačer. Discrete event simulations with j-sim. In *Proceedings of the inaugural conference on the Principles and Practice of programming, 2002 and Proceedings of the second workshop on Intermediate representation engineering for virtual machines, 2002*, PPPJ '02/IRE '02, pages 13–18, Maynooth, County Kildare, Ireland, Ireland, 2002. National University of Ireland. 67

[135] Mike Kershaw. Kismet - 802.11 wireless network detector [online]. (Acedido em dezembro de 2012). 14

[136] S. Keshav. *An Engineering Approach To Computer Networking: Atm Networks, The Internet, And The Telephone Network*. Pearson Education, 1997. 76

[137] Dagmar Köhn and Nicolas Le Novère. Sed-ml - an xml format for the implementation of the miase guidelines. In Monika Heiner and Adelinde Uhrmacher, editors, *Computational Methods in Systems Biology*, volume 5307 of *Lecture Notes in Computer Science*, pages 176–190. Springer Berlin / Heidelberg, 2008. 10.1007/978-3-540-88562-7_15. 27

[138] R. Khondoker, E.M. Veith, and P. Mueller. A description language for communication services of future network architectures. In *Network of the Future (NOF), 2011 International Conference on the*, pages 68 –75, nov. 2011. 30

[139] C. Kiddle, R. Simmonds, D.K. Wilson, and B. Unger. Anml: A language for describing networks. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2001. Proceedings. Ninth International Symposium on*, pages 135 –141, 2001. 25

[140] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. kvm: the linux virtual machine monitor. OLS '07: The 2007 Ottawa Linux Symposium, July 2007. 140

[141] Donald E. Knuth. backus normal form vs. backus naur form. *Commun. ACM*, 7(12):735–736, December 1964. 26

[142] Leonhard Korowajczuk. *LTE, WiMAX and WLAN Network Design, Optimization and Performance Analysis*. John Wiley & Sons, 2011. 118

[143] Evangelos Kotsovinos. Virtualization: blessing or curse? *Commun. ACM*, 54(1):61–65, January 2011. 137, 152, 157

[144] Stein Kristiansen and Thomas Plagemann. Accuracy and scalability of ns-2's distributed emulation extension. *Simulation*, 87(1-2):45–65, January 2011. 78

[145] Nino Kubinidze, Ivan Ganchev, and Máirtin O'Droma. Network simulator ns2: Shortcomings, potential development and enhancement strategies. In A. Nejat Ince and Ercan Topuz, editors, *Modeling and Simulation Tools for Emerging Telecommunication Networks*, pages 263–277. Springer US, 2006. 10.1007/0-387-34167-6_12. 87

[146] Stuart Kurkowski, Tracy Camp, and Michael Colagrosso. Manet simulation studies: the incredibles. *SIGMOBILE Mob. Comput. Commun. Rev.*, 9(4):50–61, October 2005. 1, 18, 76, 78

[147] Mathieu Lacage. *Outils d'expérimentation pour la recherche en réseaux*. PhD thesis, Université de Nice-Sophia Antipolis, November 2010. 135, 156

[148] Mathieu Lacage and Thomas R. Henderson. Yet another network simulator. In *Proceeding from the 2006 workshop on ns-2: the IP network simulator*, WNS2 '06, New York, NY, USA, 2006. ACM. 112, 122

[149] Pero Latkoski, Valentin Rakovic, Ognen Ognenoski, Vladimir Atanasovski, and Liljana Gavrilovska. Sdl+qualnet: a novel simulation environment for wireless heterogeneous networks. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, SIMUTools '10, pages 25:1–25:10, ICST, Brussels, Belgium, Belgium, 2010. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). 20

[150] S. M. Ibrahim Lavlu and Dinangkur Kundu. *Cacti 0.8 Network Monitoring*. Packt Publishing, 2009. 16

[151] G. Leduc, H. Abrahamsson, Simon Balon, S. Bessler, M. D'Arienzo, O. Delcourt, J. Domingo-Pascual, S. Cerav-Erbas, I. Gojmerac, X. Masip, A. Pescaph, B. Quoitin, S.F. Romano, E. Salvatori, Fabian Skivée, H.T. Tran, S. Uhlig, and H. mit. An open source traffic engineering toolbox. *Computer Communications*, 29(5):593–610, March 2006. 67

[152] Y.J. Lee and G.F. Riley. Dynamic nix-vector routing for mobile ad hoc networks. In *Wireless Communications and Networking Conference, 2005 IEEE*, volume 4, pages 1995 – 2001 Vol. 4, march 2005. 122

[153] Miklós Lengyel and János Sztrik. Performance comparison of traditional schedulers in diffserv architecture using ns. *Proceedings of the 16th European Simulation Symposium*, pages 261–266, oct 2004. 87

[154] Don Libes. *Exploring Expect*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1st edition, 1996. 142

[155] Don Libes. Writing a tcl extension in only 7 years. In *Proceedings of the 5th conference on Annual Tcl/Tk Workshop 1997 - Volume 5*, TCLTK'97, pages 8–8, Berkeley, CA, USA, 1997. USENIX Association. 142

[156] Dr Zheng Lu and Hongji Yang. *Unlocking the Power of OPNET Modeler*. Cambridge University Press, New York, NY, USA, 2012. 12, 20

[157] A. Luntovskyy, S. Uhlig, D. Gütter, and A. Schill. Load and capacity simulation for design of combined wired and wireless lan. In *Proceedings of the 2008 Spring simulation multiconference*, SpringSim '08, pages 16:1–16:4, San Diego, CA, USA, 2008. Society for Computer Simulation International. 30, 154

[158] Andriy Luntovskyy, Dietbert Gutter, and Alexander Schill. Network design and optimization under use of candy framework. In *Microwave and Telecommunication Technology, 2006. CriMiCO '06. 16th International Crimean Conference*, volume 1, pages 394 –397, sept. 2006. xiii, 26, 30, 31

[159] Andriy Luntovskyy, Taissia Trofimova, Natalia Trofimova, Dietbert Guetter, and Alexander Schill. To a proposal towards standardization of network design markup language. International Network Optimization Conference 2007, INOC 2007 by EURO/ENOG, Spa, Belgium, 22-25 April 2007, p.54 (paper ID=7), 4 2007. 26

[160] Gordon Fyodor Lyon. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure, USA, 2009. 14, 37, 67

[161] Joylan Nunes Maciel and Cristina Duarte Murta. Nit: A new internet topology generator. In *Proceedings of the 15th Open European Summer School and IFIP TC6.6 Workshop on The Internet of the Future*, EUNICE '09, pages 108–117, Berlin, Heidelberg, 2009. Springer-Verlag. 13

[162] Eduardo Magaña, Edurne Izkue, and Jesús Villadangos. Review of traffic scheduler features on general purpose platforms. *SIGCOMM Comput. Commun. Rev.*, 31(2 supplement):50–79, April 2001. 83

[163] Daniel Mahrenholz and Svilen Ivanov. Real-time network emulation with ns-2. In *Proceedings of the 8th IEEE International Symposium on Distributed Simulation and Real-Time Applications*, DS-RT '04, pages 29–36, Washington, DC, USA, 2004. IEEE Computer Society. 78

[164] Jeremy Ethridge Mandeep, Peter Pieda, Jeremy Ethridge, Mandeep Baines, and Farhan Shallwani. A network simulator differentiated services implementation open ip. Technical report, Open IP, Nortel Networks, August 2000. 83, 84, 85

[165] Eduardo M. D. Marques, Lina M. P. L. de Brito, Paulo N. M. Sampaio, and Laura M. Rodríguez Peralta. *An Analysis of Quality of Service Architectures : Principles, Requirements, and Future Trends*, volume Intelligent Quality of Service Technologies and Network Management. IGI Global, April 2010. 79, 83

[166] Eduardo M.D. Marques, Ricardo A.S.A. Plácido, and Paulo N.M. Sampaio. Visual network simulator (vns): A gui to qos simulation for the ns-2 simulator. In *Computer Systems and Applications, 2009. AICCSA 2009. IEEE/ACS International Conference on*, pages 342 –349, may 2009. 40

[167] Eduardo Miguel Dias Marques, José Jorge F. Sousa, and Paulo Nazareno Maia Sampaio. Ns-3 simulation and management of wimax and lte networks with nsdl. In *Proceedings of the 2011 European Simulation and Modelling Conference*, ESM'2011, Guimarães, Portugal, 2011. xv, 129

[168] M. Massie, B. Li, B. Nicholes, V. Vuksan, R. Alexander, J. Buchbinder, F. Costa, A. Dean, D. Josephsen, P. Phaal, et al. *Monitoring with Ganglia*. O'Reilly Media, 2012. 16

[169] M. L. Massie, B. N. Chun, and D. E. Culler. The ganglia distributed monitoring system: Design, implementation, and experience. *Parallel Computing*, 30(7), July 2004. 16

[170] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008. 10, 158

[171] A. Medina, A. Lakhina, I. Matta, and J. Byers. Brite: an approach to universal topology generation. In *Proc. Ninth Int Modeling, Analysis and Simulation of Computer and Telecommunication Systems Symp*, pages 346–353, 2001. 13, 31

[172] S. Mehta, Niamat Ullah, Md. Humaun Kabir, Mst. Najnin Sultana, and Kyung Sup Kwak. A case study of networks simulation tools for wireless networks. In *Proceedings of the 2009 Third Asia International Conference on Modelling & Simulation*, AMS '09, pages 661–666, Washington, DC, USA, 2009. IEEE Computer Society. 1, 23

[173] Marco Mezzavilla, Marco Miozzo, Michele Rossi, Nicola Baldo, and Michele Zorzi. A lightweight and accurate link abstraction model for the simulation of lte networks in ns-3. In *Proceedings of the 15th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems*, MSWiM '12, pages 55–60, New York, NY, USA, 2012. ACM. 132

[174] Microsoft. Visio [online]. (Acedido em agosto de 2012). 12

[175] Marius Milner. Netstumbler [online]. (Acedido em dezembro de 2012). 14

[176] José Morais, Paulo Carvalho, and Solange Lima. Plataforma de configuração e monitorização de qos numa rede diffserv. In *Actas da CRC'2003 - 6 Conferência sobre Redes de Computadores - Protocolos, Tecnologias e Aplicações para Ambientes Móveis*, Bragança, Portugal, September 2003. 38, 40, 110

[177] Ammar Muqaddas. Heterogeneous grooming optical network simulator [online], 2007. (Acedido em janeiro de 2012). 38

[178] Nagios. Nagios ain't gonna insist on sainthood [online]. (Acedido em dezembro de 2012). 15

[179] Thomas Nagy. The waf book [online], 2011. (Acedido em agosto de 2012). 112

[180] Hemanth Narra, Yufei Cheng, Egemen K. Çetinkaya, Justin P. Rohrer, and James P. G. Sterbenz. Destination-sequenced distance vector (dsdv) routing protocol implementation in ns-3. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, SIMUTools '11, pages 439–446, ICST, Brussels, Belgium, Belgium, 2011. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). 122

[181] Jeff Nathan. Projeto nemesis [online]. (Acedido em 2012). 143

[182] Shekar Nethi, Mikael Pohjola, Lasse Eriksson, and Riku Jäntti. Simulation case studies of wireless networked control systems. In *Proceedings of the 2nd ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*, PM2HW2N '07, pages 100–104, New York, NY, USA, 2007. ACM. 83

[183] NetSurveyor. 802.11 network discovery / wi-fi scanner [online]. (Acedido em dezembro de 2012). 14

[184] Scalable Networks. Qualnet® communications simulation platform (qualnet) [online]. (Acedido em 2009). 20

[185] Cameron Newham. *Learning the bash shell, third edition*. O'Reilly Media, Inc., 3 edition, 2005. 142

[186] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the differentiated services field (ds field) in the ipv4 and ipv6 headers. RFC 2474 (Proposed Standard), December 1998. Updated by RFCs 3168, 3260. 79

[187] K. Nichols, V. Jacobson, and L. Zhang. A two-bit differentiated services architecture for the internet. RFC 2638 (Informational), July 1999. 81

[188] David M. Nicol. Scalability of network simulators revisited. In *Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conference*, Orlando, FL, February 2003. 67

[189] J. Nin Marco Miozzo Nicolas Baldo, M. Requena. A new model for the simulation of the lte-epc data plane. In *Proceedings of the Workshop on ns-3 (WNS3 2012)*, Sirmione, Italy, March 2012. 132

[190] Nmap. Network mapper [online]. (Acedido em dezembro de 2012). 14

[191] NML-WG. Network mark-up language working group [online]. (Acedido em 2012). 29

[192] NS2. Differentiated services module in ns [online]. (Acedido em 2009). 86

[193] NSNAM. ns-3 lte model [online]. (Acedido em julho de 2012). 125

[194] NSNAM. ns-3 wifi model [online]. (Acedido em julho de 2012). 123

[195] Tobi Oetiker. Monitoring your it gear: the mrtg story. *IT Professional*, 3(6):44 –48, nov/dec 2001. 17, 37

[196] OGF. Open grid forum [online]. (Acedido em 2012). 29

[197] Andy Ogielski. Domain modeling language (dml) reference manual [online]. (Acedido em setembro de 2012). 25

[198] Rihards Olups. *Zabbix 1.8 Network Monitoring*. Packt Publishing, 2010. 16

[199] Eng Hwee Ong, J. Kneckt, O. Alanen, Zheng Chang, T. Huovinen, and T. Nihtila. Ieee 802.11ac: Enhancements for very high throughput wlans. In *Personal Indoor and Mobile Radio Communications (PIMRC), 2011 IEEE 22nd International Symposium on*, pages 849–853, Sept. 116

[200] openSUSE. opensuse project [online]. (Acedido em agosto de 2012). 142

[201] John K. Ousterhout. *Tcl and the Tk Toolkit*. Flatbrain Com, 1996. 142

[202] Sudeep Palat and Philippe Godin. *Network Architecture*, pages 21–50. John Wiley & Sons, Ltd, 2009. 120

[203] Christos Papadopoulos and John Heidemann. Using ns in the classroom and lab. In *Proceedings of the ACM SIGCOMM Workshop on Computer Networking: Curriculum Designs and Educational Challenges*, pages 45–46, Pittsburgh, PA, USA, August 2002. ACM. 76

[204] Parallels. Parallels ip holdings gmbh [online]. (Acedido em agosto de 2012). 140

[205] D. Pareit, B. Lannoo, I. Moerman, and P. Demeester. The history of wimax: A complete survey of the evolution in certification and standardization for ieee 802.16 and wimax. *Communications Surveys Tutorials, IEEE*, PP(99):1 –29, 2011. 115, 118

[206] Subharthi Paul, Jianli Pan, and Raj Jain. Architectures for the future networks and the next generation internet: A survey. *Computer Communications*, 34(1):2 – 42, 2011. 10

[207] K. Pawlikowski, H.-D.J. Jeong, and J.-S.R. Lee. On credibility of simulation studies of telecommunication networks. *Communications Magazine, IEEE*, 40(1):132 –139, jan 2002. 18

[208] C. Dennis Pegden, Randall P. Sadowski, and Robert E. Shannon. *Introduction to Simulation Using SIMAN*. McGraw-Hill, Inc., New York, NY, USA, 2nd edition, 1995. 18

[209] P. Muditha Perera and Chamath Keppitiyagama. A performance comparison of hypervisors. In *Advances in ICT for Emerging Regions (ICTer), 2011 International Conference on*, page 120, sept. 2011. 141

[210] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561 (Experimental), July 2003. 122

[211] L. Felipe Perrone, Claudio Cicconetti, Giovanni Stea, and Bryan C. Ward. On the automation of computer network simulators. In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, Simutools '09, pages 49:1–49:10, ICST, Brussels, Belgium, Belgium, 2009. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). 2, 67, 156

[212] D. Petrovic and A. Schiper. Implementing virtual machine replication: A case study using xen and kvm. In *Advanced Information Networking and Applications (AINA), 2012 IEEE 26th International Conference on*, pages 73 –80, march 2012. 141

[213] PHP. Php: Hypertext preprocessor [online]. (Acedido de 2009 a 2011). 95

[214] G. Piro, L.A. Grieco, G. Boggia, F. Capozzi, and P. Camarda. Simulating lte cellular systems: An open-source framework. *Vehicular Technology, IEEE Transactions on*, 60(2):498 –513, feb. 2011. 130

[215] Ricardo Alexandre Silva Alves Plácido. Simulação de redes com multicasting e garantias de qualidade de serviço. Master's thesis, Universidade da Madeira, July 2008. 40

[216] J. Postel. Internet Control Message Protocol. RFC 792 (Standard), September 1981. Updated by RFCs 950, 4884. 14

[217] J. Postel. Transmission control protocol. RFC 793 (Standard), sep 1981. Updated by RFCs 1122, 3168, 6093, 6528. 78

[218] Python. Python programming language [online]. (Acedido em setembro de 2012). 20, 112

[219] Quest. Big brother pro system monitor [online]. (Acedido em dezembro de 2012). 16

[220] Z.A. Qun and Wang Jun. Application of ns2 in education of computer networks. In *Advanced Computer Theory and Engineering, 2008. ICACTE '08. International Conference on*, pages 368 –372, dec. 2008. xiv, 76, 77

[221] Muhammad Azizur Rahman, Algirdas Pakštas, and Frank Zhigang Wang. An approach to integration of network design and simulation tools. In *Telecommunications, 2005. ConTEL 2005. Proceedings of the 8th International Conference on*, volume 1, pages 173 – 180, 15-17, 2005. 25

[222] Muhammad Azizur Rahman, Algirdas Pakštas, and Frank Zhigang Wang. Nedase: A bridge between network topology generator, network design and simulation tools. In *Computer as a Tool, 2005. EUROCON 2005.The International Conference on*, volume 2, pages 1675 –1678, nov. 2005. xiii, 25, 31, 32, 154

[223] Muhammad Azizur Rahman, Algirdas Pakštas, and Frank Zhigang Wang. Network modelling and simulation tools. *Simulation Modelling Practice and Theory*, 17(6):1011 – 1031, 2009. 1, 44

[224] Trygve Reenskaug and James O. Coplien. The dci architecture: A new vision of object-oriented programming [online], March 2009. (Acedido em julho de 2012). 127

[225] Ferozuddin Riaz and Khidir M. Ali. Applications of graph theory in computer science. In *2011 Third International Conference on Computational Intelligence, Communication Systems and Networks (CICSyN)*, pages 142–145. IEEE, July 2011. 49

[226] George Riley and Alfred Park. Pdns - parallel/distributed ns [online], 2004. (Acedido em setembro de 2012). 78

[227] George F. Riley. The georgia tech network simulator. In *Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research*, MoMeTools '03, pages 5–12, New York, NY, USA, 2003. ACM. 112

[228] George F. Riley. Netanim [online], 2012. (Acedido em julho de 2012). 30, 113

[229] George F. Riley, Richard M. Fujimoto, and Mostafa H. Ammar. A generic framework for parallelization of network simulations. In *Proceedings of the 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, MASCOTS '99, pages 128–135, Washington, DC, USA, 1999. IEEE Computer Society. 78

[230] George F. Riley, Ellen Zegura, and Mostafa Ammar. Efficient routing using Nix-Vectors. Technical Report GIT-CC-00-27, Georgia Tech, 2000. 122

[231] François Rioux, François Bernier, and Denis Laurendeau. Design and implementation of an xml-based, technology-unified data pipeline for interactive simulation. In *Proceedings of the 40th Conference on Winter Simulation*, WSC '08, pages 1130–1138. Winter Simulation Conference, 2008. 28

[232] Fulvio Risso and Mario Baldi. Netpdl: an extensible xml-based language for packet header description. *Comput. Netw.*, 50(5):688–706, April 2006. 27, 28

[233] Russ Rogers and Russ Rogers. *Nessus Network Auditing*. Syngress Publishing, 2 edition, 2008. 14

[234] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol label switching architecture. RFC 3031 (Proposed Standard), January 2001. Updated by RFC 6178. 82

[235] Margaret Rouse. Platform definition [online], September 2006. (Acedido em março 2012). 37

[236] F.D. Sacerdoti, M.J. Katz, M.L. Massie, and D.E. Culler. Wide area cluster monitoring with ganglia. In *Cluster Computing, 2003. Proceedings. 2003 IEEE International Conference on*, pages 289 – 298, dec. 2003. 16

[237] J. Sahoo, S. Mohapatra, and R. Lath. Virtualization: A survey on concepts, taxonomy and associated security issues. In *Computer and Network Technology (ICCNT), 2010 Second International Conference on*, pages 222 –226, april 2010. 139

[238] Chris Sanders. *Practical Packet Analysis: Using Wireshark to Solve Real-World Network Problems*. No Starch Press, San Francisco, CA, USA, 2011. 17

[239] M. Sawahashi, Y. Kishiyama, H. Taoka, M. Tanno, and T. Nakamura. Broadband radio access: Lte and lte-advanced. In *Intelligent Signal Processing and Communication Systems, 2009. ISPACS 2009. International Symposium on*, pages 224 –227, jan. 2009. 115

[240] Saxon. The xslt and xquery processor [online]. (Acedido em 2009). 95

[241] Gregor Schaffrath, Stefan Schmid, Ishan Vaishnavi, Ashiq Khan, and Anja Feldmann. A resource description language with vagueness support for multi-provider cloud networks. In *Proceedings of International Conference on Computer Communication Networks (ICCCN)*, 2012. 30

[242] Han Albrecht Schmid. Systematic framework design by generalization. *Commun. ACM*, 40(10):48–51, October 1997. 37

[243] Douglas C Schmidt, Aniruddha Gokhale, and Balachandran Natarajan. Leveraging application frameworks. *Queue*, 2:66–75, July 2004. 36, 37, 47

[244] Kaveh Shafiee, Jinwoo Brian Lee, Victor C.M. Leung, and Garland Chow. Modeling and simulation of vehicular networks. In *Proceedings of the first ACM international symposium on Design and analysis of intelligent vehicular networks and applications*, DIVANet '11, pages 77–86, New York, NY, USA, 2011. ACM. 19

[245] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. Making middleboxes someone else's problem: network processing as a cloud service. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '12, pages 13–24, New York, NY, USA, 2012. ACM. 142

[246] John R. Smith and Peter Schirling. Metadata standards roundup. *Multimedia, IEEE*, 13(2):84 – 88, april-june 2006. 52

[247] Karen R. Sollins. An architecture for network management. In *Proceedings of the 2009 workshop on Re-architecting the internet*, ReArch '09, pages 67–72, New York, NY, USA, 2009. ACM. 10

[248] José Jorge Fernandes Sousa. Autoria e simulação de cenários de redes em ns-3. Master's thesis, Centro de Competências de Ciências Exactas e Engenharias da Universidade da Madeira, Setembro 2011. xv, 115, 128, 132, 133

[249] Spiceworks. Social network and it management software [online]. (Acedido em dezembro de 2012). 16

[250] Springer. Springer link [online]. (Acedido em agosto de 2012). 23

[251] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 2000. 19

[252] Paul Tader. Server monitoring with zabbix. *Linux J.*, 2010(195), July 2010. 16

[253] Andrew S. Tanenbaum. *Computer networks*. Prentice-Hall International, Englewood Cliffs, 4th edition edition, 2003. 85

[254] Sasu Tarkoma. *Overlay Networks: Toward Information Networking*. Auerbach Publications, Boston, MA, USA, 1st edition, 2010. 139

[255] Joanna Tomasik and Marc-Antoine Weisser. ashiip: Autonomous generator of random internet-like topologies with inter-domain hierarchy. In *Proceedings of the 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, MASCOTS '10, pages 388–390, Washington, DC, USA, 2010. IEEE Computer Society. 14

[256] Linus Torvalds. Linux operating system [online], 2012. (Consultado em março de 2012). 42

[257] Hung-ying Tyan. *Design, Realization and Evakuation of a Component-based Compositional Software Arquitecture For Network Simulation*. PhD thesis, Ohio State University, 2002. 19

[258] Thomas Urban. *Cacti 0.8 Beginner's Guide*. Packt Publishing, 2011. 16

[259] Jeroen Van Der Ham, Freek Dijkstra, Paola Grosso, Ronald Van Der Pol, Andree Toonk, and Cees De Laat. A distributed topology information system for optical networks based on the semantic web. *Opt. Switch. Netw.*, 5(2-3):85–93, June 2008. 25

[260] Jeroen Johannes van der Ham. *A Semantic Model for Complex Computer Networks: The Network Description Language*. PhD thesis, Universiteit van Amsterdam, April 2010. 26

[261] András Varga. Parameterized topologies for simulation programs. In *WMC'98: Western Multiconference on Simulation, CNDS'98: Communication Networks and Distributed Systems*, 1998. 20, 26

[262] András Varga. Omnet++. In Klaus Wehrle, Mesut G´unes, and James Gross, editors, *Modeling and Tools for Network Simulation*, pages 35–59. Springer Berlin Heidelberg, 2010. 20, 46

[263] András Varga and Rudolf Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, Simutools '08, pages 60:1–60:10, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). 20

[264] Alexei Vladishev. Zabbix - monitoring solution [online]. (Acedido em dezembro de 2012). 16

[265] VMware. Vmware vsphere [online]. (Acedido em agosto de 2012). 140

[266] vSwitch. Virtual switch [online]. (Acedido em agosto de 2012). 145

[267] Vyatta. Vyatta software middlebox [online]. (Acedido em agosto de 2012). 142

[268] W3C. Extensible markup language (xml) [online]. (Acedido em março de 2012). 16, 26

[269] W3C. Xml schema (xsd) [online]. (Acedido em março de 2012). 44

[270] W3C. Xsl transformations (xslt). (Acedido em março de 2012). 44

[271] W3C. Rdf/xml syntax specification [online], 2004. (Acedido em 2009). 26

[272] Haoxiang Wang. Nevml: A markup language for network topology visualization. In *Proceedings of the 2010 Second International Conference on Future Networks*, ICFN '10, pages 119–123, Washington, DC, USA, 2010. IEEE Computer Society. 30, 158

[273] S. Y. Wang and H. T. Kung. A new methodology for easily constructing extensible and high-fidelity tcp/ip network simulators. *Comput. Netw.*, 40(2):257–278, October 2002. 19

[274] Shie-Yuan Wang and Yu-Ming Huang. Nctuns distributed network emulator. In Inc. Nova Science Publishers, editor, *Internet Journal*, volume 4, pages 61–94, 2012. 19

[275] Zheng Wang. *Internet QoS: Architectures and Mechanisms for Quality of Service*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2001. 82

[276] Elias Weingärtner, Hendrik Vom Lehn, and Klaus Wehrle. A performance comparison of recent network simulators. In *Proceedings of the 2009 IEEE international conference on Communications*, ICC'09, pages 1287–1291, Piscataway, NJ, USA, 2009. IEEE Press. 33, 112, 114

[277] Pierre Weiss and Sebastien Vincent. The ns-3 topology generator [online], 2010. (Acedido em julho de 2012). 113

[278] Michael Welzl, Muhammad Ali, and Sven Hessler. Network simulation by mouse (nsbm): A gui approach for teaching computer networks with the ns simulator. In *Proceedings of the International Conference on Interactive Computer Aided Learning (ICL 2006)*, pages 26–28, Villach, Austria, September 2006. 40

[279] David Wetherall and Christopher J. Lindblad. Extending tcl for dynamic object-oriented programming. In *Proceedings of the 3rd Annual USENIX Workshop on Tcl/Tk - Volume 3*, TCLTK '98, pages 19–19, Berkeley, CA, USA, 1995. USENIX Association. 19, 77

[280] Peng-Jung Wu. Ns2 scenarios generator 2 [online], 2008. (Acedido em 2009). 12, 40

[281] xGraph. xgraph [online]. (Acedido em 2008). 30, 42, 77

[282] Yu Xiangzhan, Wu Gang, Jin Dailiang, and Li Wei. Research on performance estimation model of distributed network simulation based on pdns conservative synchronization mechanism in complex environment. In *Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology*, CIT '10, pages 2576–2580, Washington, DC, USA, 2010. IEEE Computer Society. 78, 83

[283] Xianghua Xu, Feng Zhou, Jian Wan, and Yucheng Jiang. Quantifying performance properties of virtual machine. In *Information Science and Engineering, 2008. ISISE '08. International Symposium on*, volume 1, pages 24 –28, dec. 2008. 141

[284] Hong Yan, David A. Maltz, T. S. Eugene Ng, Hemant Gogineni, Hui Zhang, and Zheng Cai. Tesseract: a 4d network control plane. In *Proceedings of the 4th USENIX conference on Networked systems design & implementation*, NSDI'07, pages 27–27, Berkeley, CA, USA, 2007. USENIX Association. 9

[285] Leo Yi, Kai Miao, and A. Liu. A comparative study of wimax and lte as the next generation mobile enterprise network. In *Advanced Communication Technology (ICACT), 2011 13th International Conference on*, pages 654 –658, feb. 2011. 119

[286] Sunghyun Yoon and Young Boo Kim. A design of network simulation environment using ssfnet. In *Advances in System Simulation, 2009. SIMUL '09. First International Conference on*, pages 73 –78, sept. 2009. 20

[287] Jakub Wszolek Zdzislaw Kowalczuk. Networked object monitor - a distributed system for monitoring, diagnostics and control of complex industrial facilities. In Romuald Zielonko, editor, *Metrology and Scientific Instrumentation*, volume XIX, pages 521–530. Polish Academy of Sciences, October 2012. 30

[288] Marko Zec and Miljenko Mikuc. Operating system support for integrated network emulation in imunes. In *in Proceedings of the 1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure / ASPLOS-XI*, Boston, USA, October 2004. 19

[289] Xiang Zeng, Rajive Bagrodia, and Mario Gerla. Glomosim: a library for parallel simulation of large-scale wireless networks. *SIGSIM Simul. Dig.*, 28(1):154–161, July 1998. 20

[290] Zenoss. Zenoss core [online]. (Acedido em dezembro de 2012). 16

[291] Shengying Zhao, Kai Xiong, and Lina Shang. Research on nsbench' application in computer network teaching. In *Consumer Electronics, Communications and Networks (CECNet), 2012 2nd International Conference on*, pages 1244 –1246, april 2012. 40

[292] Tammy Zitello, Deborah Williams, and Paul Weber. *HP OpenView System Administration Handbook: Network Node Manager, Customer Views, Service Information Portal, OpenView Operations*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2003. 14

Apêndices

Apêndice A

XML *Schemas* do perfil base da linguagem NSDL

O código apresentado neste apêndice é o conteúdo dos ficheiros principais para a definição dos objetos da linguagem NSDL. A Figura A.1 apresenta graficamente a arquitetura de ficheiros para definição e validação de cenários NSDL. Cada uma das seções seguintes contém o código de cada um dos ficheiros.

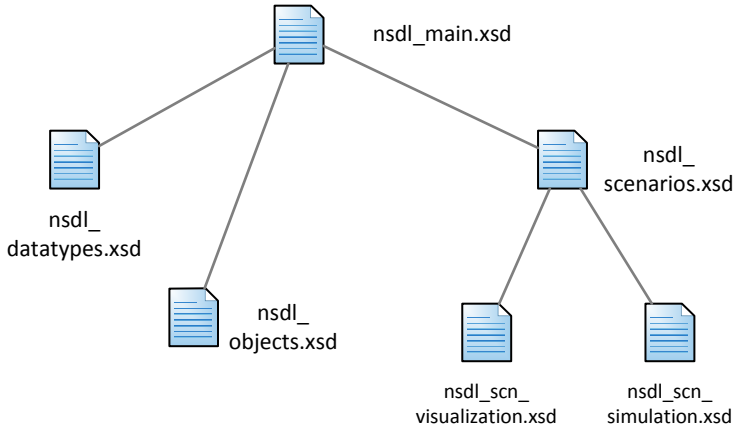


Figura A.1: Arquitetura de ficheiros XML *Schemas* para a definição da linguagem NSDL

A.1 Ficheiro nsdl_main.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <!--
        Author: emarques@uma.pt
        Date: 1th quarter 2010
        Name: nsdl_main.xsd
        Purpose:
        This is the root schema to validate a standard NSDL file.
        The first lines shows the other included files to provide the complete validation.

        Can be used to execute an extended validation on a nsdl file.

    -->

    <xs:include schemaLocation="nsdl_objects.xsd"/>
    <xs:include schemaLocation="nsdl_scenarios.xsd"/>

    <xs:element name="nsdl">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="metadata" minOccurs="0"/>
                <xs:element name="network">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="templates" minOccurs="0">
                                <xs:complexType>
                                    <xs:choice maxOccurs="unbounded" minOccurs="0">
                                        <xs:element ref="node" minOccurs="0"/>
                                        <xs:element ref="link" minOccurs="0"/>
                                        <xs:element ref="domain" minOccurs="0"/>
                                        <xs:element ref="interface" minOccurs="0"/>
                                        <xs:element ref="protocol" minOccurs="0"/>
                                        <xs:element ref="application" minOccurs="0"/>
                                    </xs:choice>
                                </xs:complexType>
                            </xs:element>
                            <xs:element name="objects">
                                <xs:complexType>
                                    <xs:choice minOccurs="0" maxOccurs="unbounded">
                                        <xs:element ref="node" minOccurs="0"/>
                                        <xs:element ref="link" minOccurs="0"/>
                                        <xs:element ref="domain" minOccurs="0"/>
                                    </xs:choice>
                                </xs:complexType>
                            </xs:element>
                            <xs:element name="views" minOccurs="0">
                                <xs:complexType>
                                    <xs:choice maxOccurs="unbounded" minOccurs="0">
                                        <xs:element ref="set" maxOccurs="unbounded"/>
                                    </xs:choice>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element ref="scenarios"/>
            </xs:sequence>
        </xs:complexType>

        <!-- Constrains of NSDL -->

        <xs:unique name="objID">
            <xs:selector xpath=".//*"/>
            <xs:field xpath="@id"/>
        </xs:unique>

        <xs:unique name="netObjID">
            <xs:selector xpath="./network/objects/* | ./network/objects/*/*"/>
            <xs:field xpath="@id"/>
```

```
        </xs:unique>

        <xs:unique name="firstLevelObjID">
            <xs:selector xpath="./network/objects/*"/>
            <xs:field xpath="@id"/>
        </xs:unique>

        <xs:keyref refer="objID" name="netObjId_Val">
            <xs:selector xpath="./scenarios/simulations/simulation/description/events/event"/>
            <xs:field xpath="@objectid"/>
        </xs:keyref>

        <xs:keyref refer="objID" name="viewObjId_Val">
            <xs:selector xpath="./network/views/set/listobjects/objectid"/>
            <xs:field xpath="."/>
        </xs:keyref>

        <!-- End of Constrains of NSDL -->

    </xs:element>
</xs:schema>
```

A.2 Ficheiro nsdl_datatypes.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <!--
        Author: emarques@uma.pt
        Date: 2th quarter 2010
        Name: nsdl_datatypes.xsd
        Purpose:
        This is the schema that contains the available datatypes to all NSDL schemas.

        Can be used to execute an extended validation on a nsdl file.

        Updates:
        15/Mar/2011: Adition of an attribute to base object -> base
    -->

    <xs:attribute name="id" type="nsdlID"/>
    <xs:attribute name="object" type="baseObjects"/>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="notes" type="xs:string"/>
    <xs:element name="active" type="xs:boolean" default="true"/>

    <xs:simpleType name="nsdlID">
        <xs:restriction base="xs:ID">
            <xs:minLength value="2" />
            <xs:maxLength value="30" />
            <xs:pattern value="[a-zA-Z][a-zA-Z0-9_]+"></xs:pattern>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="baseObjects">
        <xs:restriction base="xs:string">

            <xs:enumeration value="nsdl"/>
            <xs:enumeration value="network"/>
            <xs:enumeration value="scenarios"/>
            <xs:enumeration value="templates"/>
            <xs:enumeration value="views"/>
            <xs:enumeration value="objects"/>

            <xs:enumeration value="node"/>
            <xs:enumeration value="link"/>
            <xs:enumeration value="domain"/>
            <xs:enumeration value="application"/>
            <xs:enumeration value="protocol"/>
            <xs:enumeration value="interface"/>
```

```
47          </xs:restriction>
48      </xs:simpleType>
49
50
51      <xs:simpleType name="linkType">
52          <xs:restriction base="xs:string">
53              <xs:enumeration value="simplex"/>
54              <xs:enumeration value="duplex"/>
55          </xs:restriction>
56      </xs:simpleType>
57
58      <xs:simpleType name="queueType">
59          <xs:restriction base="xs:string">
60              <xs:enumeration value="DropTail"/>
61              <xs:enumeration value="RED"/>
62              <xs:enumeration value="FQ"/>
63              <xs:enumeration value="DRR"/>
64              <xs:enumeration value="SFQ"/>
65              <xs:enumeration value="CBQ"/>
66          </xs:restriction>
67      </xs:simpleType>
68
69      <xs:simpleType name="tcpType">
70          <xs:restriction base="xs:string">
71              <xs:enumeration value="TCP"/>
72              <xs:enumeration value="TCP/Reno"/>
73              <xs:enumeration value="TCP/Vegas"/>
74              <xs:enumeration value="TCP/Sack1"/>
75              <xs:enumeration value="TCP/Fack"/>
76              <xs:enumeration value="TCP/FullTcp"/>
77              <xs:enumeration value="TCPSink"/>
78          </xs:restriction>
79      </xs:simpleType>
80
81      <xs:simpleType name="udpType">
82          <xs:restriction base="xs:string">
83              <xs:enumeration value="UDP"/>
84              <xs:enumeration value="Null"/>
85          </xs:restriction>
86      </xs:simpleType>
87
88      <xs:simpleType name="rtpType">
89          <xs:restriction base="xs:string">
90              <xs:enumeration value="RTP"/>
91              <xs:enumeration value="RTCP"/>
92          </xs:restriction>
93      </xs:simpleType>
94
95      <xs:simpleType name="roleType">
96          <xs:restriction base="xs:string">
97              <xs:enumeration value="server"/>
98              <xs:enumeration value="client"/>
99              <xs:enumeration value="both"/>
100             <xs:enumeration value="peer"/>
101         </xs:restriction>
102     </xs:simpleType>
103
104     <xs:simpleType name="outputFormat">
105         <xs:restriction base="xs:string">
106             <xs:enumeration value="trace"/>
107             <xs:enumeration value="namtrace"/>
108         </xs:restriction>
109     </xs:simpleType>
110
111 </xs:schema>
```

A.3 Ficheiro nsdl_objects.xsd

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

```
3
4     <!--
5         Author: emarques@uma.pt
6         Date: 1th quarter 2010
7         Name: nsdl_objects.xsd
8         Purpose:
9         This is the schema to validate the objects of a standard NSDL file.
10        The first lines shows the other included files to provide the complete validation.
11        Each file pointed contains the particular basic object file validation rules.
12
13        Can be used to execute an extended validation on a nsdl file.
14        Updates:
15
16        15/Mar/2010: Adition of an attribute to base object -> base
17    -->
18
19    <xs:include schemaLocation="nsdl_datatypes.xsd"/>
20
21    <xs:include schemaLocation="nsdl_nodes.xsd"/>
22    <xs:include schemaLocation="nsdl_links.xsd"/>
23    <xs:include schemaLocation="nsdl_domains.xsd"/>
24
25    <xs:include schemaLocation="nsdl_interfaces.xsd"/>
26    <xs:include schemaLocation="nsdl_protocols.xsd"/>
27    <xs:include schemaLocation="nsdl_applications.xsd"/>
28
29    <!-- Declaration of objects for the 'objects' and 'templates' -->
30    <xs:element name="node" type="type_node"/>
31    <xs:element name="link" type="type_link"/>
32    <xs:element name="domain" type="type_domain"/>
33    <xs:element name="interface" type="type_interface"/>
34    <xs:element name="protocol" type="type_protocol"/>
35    <xs:element name="application" type="type_application"/>
36
37    <!-- Definition of the general object -->
38    <xs:element name="metadata" type="type_metadata"/>
39
40    <xs:complexType name="type_nsdlobject">
41        <xs:attribute ref="id" use="required"/>
42        <xs:sequence minOccurs="0">
43            <xs:element ref="active" minOccurs="0" maxOccurs="1"/>
44            <xs:element ref="name" minOccurs="0" maxOccurs="1"/>
45            <xs:element ref="notes" minOccurs="0" maxOccurs="1"/>
46            <xs:element ref="metadata" minOccurs="0" maxOccurs="1"/>
47        </xs:sequence>
48    </xs:complexType>
49
50    <!-- Types for the ... -->
51    <xs:complexType name="type_object">
52        <xs:complexContent>
53            <xs:extension base="type_nsdlobject">
54                <xs:attribute ref="object"/>
55                <xs:attribute name="template">
56                    <xs:simpleType>
57                        <xs:restriction base="xs:string"/>
58                    </xs:simpleType>
59                </xs:attribute>
60            </xs:extension>
61        </xs:complexContent>
62    </xs:complexType>
63
64    <xs:complexType name="type_metadata">
65        <xs:sequence minOccurs="0">
66            <xs:element name="title" type="xs:string"/>
67            <xs:element minOccurs="0" name="description" type="xs:string"/>
68            <xs:element name="author" maxOccurs="1" type="xs:string"/>
69            <xs:element minOccurs="0" name="created" type="xs:dateTime"/>
70            <xs:element minOccurs="0" name="copyright"/>
71        </xs:sequence>
72    </xs:complexType>
73
74
75    <!-- Types for the views -->
76    <xs:element name="set" type="type_set"/>
```

```
77
78      <xs:complexType name="type_set">
79          <xs:complexContent>
80              <xs:extension base="type_nsdlobject">
81                  <xs:sequence>
82                      <xs:element minOccurs="1" name="listobjects">
83                          <xs:complexType>
84                              <xs:sequence>
85                                  <xs:element name="objectid" maxOccurs="unbounded" type="xs:IDREF"/>
86                              </xs:sequence>
87                          </xs:complexType>
88                      </xs:element>
89                  </xs:sequence>
90              </xs:extension>
91          </xs:complexContent>
92      </xs:complexType>
93
94      <!-- Types for the objects and templates -->
95      <!-- Definition of each type -->
96      <xs:complexType name="type_node">
97          <xs:complexContent>
98              <xs:extension base="type_object">
99                  <xs:choice maxOccurs="unbounded" minOccurs="0">
100                     <xs:element ref="application" minOccurs="0"/>
101                     <xs:element ref="protocol" minOccurs="0"/>
102                     <xs:element ref="interface" minOccurs="0"/>
103                 </xs:choice>
104             </xs:extension>
105         </xs:complexContent>
106     </xs:complexType>
107
108     <xs:complexType name="type_link">
109         <xs:complexContent>
110             <xs:extension base="type_object">
111                 <xs:sequence minOccurs="0">
112                     <xs:element name="connection" minOccurs="0">
113                         <xs:complexType>
114                             <xs:attribute name="source" type="xs:IDREF"/>
115                             <xs:attribute name="destination" type="xs:IDREF"/>
116                         </xs:complexType>
117                     </xs:element>
118                     <xs:element name="type" type="linkType" minOccurs="0"/>
119                     <xs:element name="bandwidth" minOccurs="0"/>
120                     <xs:element name="delay" minOccurs="0"/>
121                     <xs:element name="loss" minOccurs="0"/>
122                 </xs:sequence>
123             </xs:extension>
124         </xs:complexContent>
125     </xs:complexType>
126
127     <xs:complexType name="type_domain">
128         <xs:complexContent>
129             <xs:extension base="type_object">
130                 <xs:sequence minOccurs="0">
131                     <xs:element name="bandwidth" minOccurs="0"/>
132                     <xs:element name="delay" minOccurs="0"/>
133                     <xs:element name="loss" minOccurs="0"/>
134                 </xs:sequence>
135             </xs:extension>
136         </xs:complexContent>
137     </xs:complexType>
138
139     <xs:complexType name="type_interface">
140         <xs:complexContent>
141             <xs:extension base="type_object">
142                 <xs:sequence minOccurs="0">
143                     <xs:element name="type" type="linkType" minOccurs="0"/>
144                     <xs:element name="bandwidth" minOccurs="0"/>
145                     <xs:element name="delay" minOccurs="0"/>
146                     <xs:element minOccurs="0" name="loss"/>
147                     <xs:element minOccurs="0" name="linkedto"/>
148                 </xs:sequence>
149             </xs:extension>
150         </xs:complexContent>
```

```
151     </xs:complexType>
152
153     <xs:complexType name="type_protocol">
154         <xs:complexContent>
155             <xs:extension base="type_object">
156                 <xs:sequence minOccurs="0">
157                     <xs:choice minOccurs="0">
158                         <xs:element name="tcpiplayer" minOccurs="0"/>
159                         <xs:element minOccurs="0" name="isolayer"/>
160                     </xs:choice>
161                     <xs:element minOccurs="0" name="interface.id" type="xs:IDREF"/>
162                 </xs:sequence>
163             </xs:extension>
164         </xs:complexContent>
165     </xs:complexType>
166
167     <xs:complexType name="type_application">
168         <xs:complexContent>
169             <xs:extension base="type_object">
170                 <xs:sequence minOccurs="0">
171                     <xs:element name="role" minOccurs="0" type="roleType"> </xs:element>
172                     <xs:element name="rate" minOccurs="0"/>
173                     <xs:element name="packetsize" minOccurs="0"/>
174                     <xs:element name="src.app" maxOccurs="unbounded" minOccurs="0" type="xs:IDREF"/>
175                     <xs:element name="dst.app" maxOccurs="unbounded" minOccurs="0" type="xs:IDREF"/>
176                     <xs:element minOccurs="0" name="protocol.id" type="xs:IDREF"/>
177                     <!-- <xs:element minOccurs="0" name="class" type="dsClass"/> -->
178                 </xs:sequence>
179             </xs:extension>
180         </xs:complexContent>
181     </xs:complexType>
182
183 </xs:schema>
```

A.4 Ficheiro nsdl_scenarios.xsd

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3
4      <!--
5          Author: emarques@uma.pt
6          Date: 1th quarter 2010
7          Name: nsdl_scenarios.xsd
8
9          Purpose:
10         This is the schema that lists the already defined scenarios in NSDL.
11         Each file pointed contains the particular basic scenario file validation rules.
12
13         Can be used to execute an extended validation on a nsdl file.
14     -->
15
16     <xs:include schemaLocation="nsdl_scn_simulation.xsd"/>
17     <xs:include schemaLocation="nsdl_scn_visualization.xsd"/>
18
19 </xs:schema>
```

A.5 Ficheiro nsdl_scn_simulation.xsd

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3      <!-- Files included -->
4      <xs:include schemaLocation="nsdl_objects.xsd"/>
5
6      <xs:element name="simulations">
7          <xs:complexType>
8              <xs:choice maxOccurs="unbounded" minOccurs="0">
9                  <xs:element ref="simulation" minOccurs="0"/>
```

```
10          </xs:choice>
11      </xs:complexType>
12  </xs:element>
13
14  <xs:element name="simulation" type="type_simulation"/>
15
16  <xs:complexType name="type_simulation">
17      <xs:complexContent>
18          <xs:extension base="type_nsdlobject">
19              <xs:sequence minOccurs="0">
20                  <xs:element name="general">
21                      <xs:complexType>
22                          <xs:sequence>
23                              <xs:element name="duration">
24                                  <xs:complexType>
25                                      <xs:choice>
26                                          <xs:element name="time"/>
27                                          <xs:element name="condition"/>
28                                      </xs:choice>
29                                  </xs:complexType>
30                              </xs:element>
31                              <xs:element name="randnumbers" minOccurs="0">
32                                  <xs:complexType>
33                                      <xs:sequence minOccurs="0">
34                                          <xs:element minOccurs="0" name="function"/>
35                                          <xs:element minOccurs="0" name="seed"/>
36                                      </xs:sequence>
37                                  </xs:complexType>
38                              </xs:element>
39                              <xs:element name="runs" minOccurs="0"/>
40                              <xs:element name="simulator" minOccurs="0"/>
41                          </xs:sequence>
42                      </xs:complexType>
43                  </xs:element>
44                  <xs:element name="description">
45                      <xs:complexType>
46                          <xs:sequence minOccurs="0">
47                              <xs:element name="events">
48                                  <xs:complexType>
49                                      <xs:sequence>
50                                          <xs:element maxOccurs="unbounded" ref="event"/>
51                                      </xs:sequence>
52                                  </xs:complexType>
53                              </xs:element>
54                              <xs:element minOccurs="0" name="outputs" maxOccurs="unbounded">
55                                  <xs:complexType>
56                                      <xs:sequence>
57                                          <xs:element maxOccurs="unbounded" name="output">
58                                              <xs:complexType>
59                                                <xs:sequence>
60                                                  <xs:element name="path" minOccurs="0"/>
61                                                  <xs:element name="filename" minOccurs="0"/>
62                                                  <xs:element minOccurs="0" name="filter">
63                                                <xs:complexType>
64                                                  <xs:sequence>
65                                                    <xs:element name="objectid"/>
66                                                    <xs:element name="parameter">
67                                                    </xs:element>
68                                                  </xs:sequence>
69                                                </xs:complexType>
70                                                </xs:element>
71                                                </xs:sequence>
72                                                <xs:attribute name="outputid" use="required"/>
73                                                <xs:attribute name="format" use="required"
74                                                type="outputFormat">
75                                                </xs:attribute>
76                                              </xs:complexType>
77                                          </xs:element>
78                                      </xs:sequence>
79                                  </xs:complexType>
80                              </xs:element>
81                          </xs:sequence>
82                      </xs:complexType>
83                  </xs:element>
```

```
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:element name="event" type="eventtype"/>

<xs:complexType name="eventtype">
    <xs:sequence maxOccurs="unbounded">
        <xs:element name="parameter">
            <xs:complexType>
                <xs:attribute name="name"> </xs:attribute>
                <xs:attribute name="value"/>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
    <xs:attribute name="objectid"/>
    <xs:attribute name="time"/>
</xs:complexType>
</xs:schema>
```

A.6 Ficheiro nsdl_scn_visualization.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:include schemaLocation="nsdl_objects.xsd"/>

    <xs:element name="visualizations">
        <xs:complexType>
            <xs:choice maxOccurs="unbounded" minOccurs="0">
                <xs:element ref="visualization" minOccurs="0"/>
            </xs:choice>
        </xs:complexType>
    </xs:element>

    <xs:element name="visualization" type="type_visualization"/>

    <xs:complexType name="type_visualization">
        <xs:complexContent>
            <xs:extension base="type_nsdlobject">
                <xs:sequence minOccurs="0">
                    <xs:element name="general" minOccurs="0">
                        <xs:complexType>
                            <xs:sequence minOccurs="0">
                                <xs:element name="bg.image" minOccurs="0"/>
                                <xs:element name="init.zoom" minOccurs="0"/>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="description">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element name="objects">
                                    <xs:complexType>
                                        <xs:sequence>
                                            <xs:element maxOccurs="unbounded" name="object">
                                                <xs:complexType>
                                                  <xs:sequence>
                                                  <xs:element name="x.position"/>
                                                  <xs:element name="y.position"/>
                                                  <xs:element name="z.position"/>
                                                  <xs:element name="image" minOccurs="0"/>
                                                  <xs:element minOccurs="0" name="color"/>
                                                  <xs:element minOccurs="0" name="zoom"/>
                                                  </xs:sequence>
                                                  <xs:attribute name="objectid" type="xs:IDREF"/>
                                                </xs:complexType>
                                            </xs:element>
                                        </xs:sequence>
```

```
48                                  </xs:complexType>
49                              </xs:element>
50                          </xs:sequence>
51                      </xs:complexType>
52                  </xs:element>
53              </xs:sequence>
54          </xs:extension>
55      </xs:complexContent>
56  </xs:complexType>
57 </xs:schema>
```

Apêndice B

Código OTcl para descrever nós DiffServ para o NS2

B.1 Configuração das funções de borda

A Figura B.1 contém todas as funções utilizadas para a configuração de borda num domínio de Serviços Diferenciados (DiffServ). A implementação destas funções é realizada numa ligação *simplex* entre o nó de borda e um nó de núcleo.

```
1  $ns simplex-link $edge $core 10Mb 10ms dsRED/edge
2  ...
3  set fEdCo [[$ns link $edge $core] queue]
4  ...
5  $fEdCo set numQueues_ 2
6  $fEdCo setNumPrec 1
7  $fEdCo addPolicyEntry [$srvA id] [$cltA id] TSW2CM 10 100000
8  $fEdCo addPolicyEntry [$srvB id] [$cltB id] TSW2CM 0 500000
9  $fEdCo addPolicerEntry TSW2CM 10 0;
10 $fEdCo addPolicerEntry TSW2CM 0 0
11 $fEdCo configQ 0 0 5 15 0.3
12 $fEdCo configQ 1 0 15 30 0.7
13 $fEdCo addPHBEntry 10 0 0
14 $fEdCo addPHBEntry 0 1 0
15 ...
16 $fEdCo printPolicyTable
17 $fEdCo printPolicerTable
18 $fEdCo printPHBTable
```

Figura B.1: Extrato de código Tcl para a configuração das funções de borda

A linha 1 apresenta a definição da ligação entre os nós $edge e $core. A criação dos nós foi omitida na figura por ser simples e não contribuir para o entendimento das funções de borda. A linha 3 cria a fila $fEdCo, associada ao *link* da linha 1. A definição dos DiffServ, funções de borda, é feita a partir da linha 5. Nas linhas 5 e 6 temos a definição das filas físicas (numQueues) e virtuais (setNumPrec). No NS2 não é possível especificar um número de filas virtuais diferente para cada fila física. Este facto pode levar a serem definidas mais filas que as necessárias, mas em termos de operação não causa nenhuma diferença, apenas limita por vezes a opção de organização das filas.

Após a definição das filas, nas linhas 7 e 8 são realizadas duas funções de borda simultaneamente: a marcação e o policiamento. O tráfego entre os nós $srvA e $cltA é marcado com o código 10 e policiado com um CIR de 100Kb. O tráfego entre os nós $srvB e $cltB é marcado com o código 0 e policiado com um CIR de 500Kb. No caso do o CIR ser ultrapassado, medido pelo algoritmo TSW2CM, o tráfego é remarcado. A linha 9 mostra a remarcação do tráfego com o código 10 para

o código 0. A linha 10 remarca com o mesmo valor (não tem efeito, mas é necessário para efeito de código OTcl).

A configuração das duas filas é feita nas linhas 11 e 12. Na linha 11 os valores '0 0' indicam a fila física 1 e a fila virtual 1 da fila física 1. Os valores 5, 15 e 0.3 são os parâmetros $\min_{th}$, $\max_{th}$ e Max_{drop}, respetivamente. A configuração na linha 12 segue a mesma estrutura. Por fim, a cada fila é feita a associação do DSCP correspondente. As linhas 13 e 14 contêm os comandos que relacionam os códigos 10 e 0 com cada uma das filas.

As linhas 16-18 não são comandos para a configuração, mas para a consulta aos valores guardados nas estruturas de dados do NS2 relacionados com as configurações DiffServ. As três tabelas apresentam, de forma agrupada, as configurações que existem para uma dada fila (apontada pela variável no início de cada linha) e podem servir para detetar configurações incorretas.

B.2 Configuração das funções de núcleo

A Figura B.2 contém todas as funções utilizadas para a configuração de núcleo num domínio de Serviços Diferenciados (DiffServ). A implementação destas funções é realizada numa ligação *simplex* entre o nó de núcleo e um nó de núcleo e/ou de borda.

```
1  $ns simplex-link $core $edge 10Mb 10ms dsRED/core
2  ...
3  set fCoEd [[$ns link $core $edge] queue]
4  ...
5  $fCoEd set numQueues 2
6  $fCoEd setNumPrec 1
7  $fCoEd setSchedularMode WRR
8  $fCoEd addQueueWeights 0 2.5
9  $fCoEd addQueueWeights 1 7.5
10 $fCoEd addPHBEntry 10 0 0
11 $fCoEd addPHBEntry 0 1 0
12 $fCoEd configQ 0 0 10 20 0.25
13 $fCoEd configQ 1 0 20 40 0.75
```

Figura B.2: Extracto de código Tcl para a configuração das funções de núcleo

As linhas 1-6 têm o mesmo objetivo das linhas 1-6 das funções de borda. Na linha 1 surge a criação da ligação. Na linha 3 é criada a fila para o *link*. Nas linhas 5 e 6 são criadas as filas físicas e virtuais.

Na linha 7 já temos a caraterização de função apenas do nó de núcleo, e que é o agendamento, WRR neste caso. Nas linhas 8 e 9 temos a definição dos pesos de cada fila física, comprovado pela identificação da fila feita apenas por um valor. Ou seja, o agendamento apenas é realizado entre filas físicas. As restantes linhas têm configurações com os mesmos propósitos já descritos para o extrato de código da Figura B.2, mas com valores diferentes. Nas linhas 10 e 11 temos a associação dos *phb's* às filas físicas e virtuais. Nas linhas 12 e 13 temos a configuração das filas virtuais.

Apêndice C

Inquérito utilizado no projeto NS2

Utilização de Network Scenarios Description Language

Inquérito no âmbito da disciplina Tecnologias Avançadas de Redes

O objectivo deste inquérito é, relativamente à linguagem NSDL, (i) avaliar o grau de compreensão adquirido e (ii) quais as suas vantagens e desvantagens comparativamente à linguagem de programação Tcl, usada pelo *Network Simulator* 2.

O inquérito seguinte é anónimo e é realizado pelo conjunto dos alunos da disciplina Tecnologias Avançadas de Redes, ano lectivo de 2009/2010.

Após aceder ao inquérito terá de o preencher completamente, visto não lhe ser permitido voltar a preencher ou completar. Em *itálico* e à direita em cada questão, está a indicação do significado de cada valor numérico.

Muito obrigado!

As questões desta secção procuram conhecer qual o grau do seu contacto com um conjunto de matérias previamente a Tecnologias Avançadas de Redes.

1 Identifique na seguinte tabela quais os seus conhecimentos prévios em...

Varia de "1 = Muito Poucos" até "5 = Bastantes". O valor intermédio é "3 = Alguns".

	1	2	3	4	5
Qualidade de Serviço	☐	☐	☐	☐	☐
Serviços Diferenciados	☐	☐	☐	☐	☐
Simulação de redes	☐	☐	☐	☐	☐
Tcl/OTcl	☐	☐	☐	☐	☐
XML	☐	☐	☐	☐	☐

As questões desta secção são relativas à linguagem usada pelo Network Simulator 2, o Tcl/OTcl.

2 Identifique na seguinte tabela, para cada tarefa, o grau de dificuldade no **PRIMEIRO** projecto para criar em Tcl...

Varia de "1 = Muito Simples" até "5 = Bastante Complexo". O valor intermédio é "3 = Nem Simples nem Complexo".

	Não usou	1	2	3	4	5
... os nós (computadores, routers)	☐	☐	☐	☐	☐	☐
... as ligações (simplex, duplex)	☐	☐	☐	☐	☐	☐
... as aplicações (e os agentes)	☐	☐	☐	☐	☐	☐
... as topologias (os objectos em geral)	☐	☐	☐	☐	☐	☐
... os eventos da simulação	☐	☐	☐	☐	☐	☐
... as estatísticas e os resultados	☐	☐	☐	☐	☐	☐
... todos os componentes no geral	☐	☐	☐	☐	☐	☐

3 Identifique na seguinte tabela, para cada tarefa, o grau de dificuldade no **SEGUNDO** projecto para criar em Tcl...

Varia de "1 = Muito Simples" até "5 = Bastante Complexo". O valor intermédio é "3 = Nem Simples nem Complexo".

	Não usou	1	2	3	4	5
... os nós (computadores, routers)	☐	☐	☐	☐	☐	☐
... as ligações (simplex, duplex)	☐	☐	☐	☐	☐	☐
... as aplicações (e os agentes)	☐	☐	☐	☐	☐	☐
... as topologias (os objectos em geral)	☐	☐	☐	☐	☐	☐
... os eventos da simulação	☐	☐	☐	☐	☐	☐
... as estatísticas e os resultados	☐	☐	☐	☐	☐	☐
... todos os componentes no geral	☐	☐	☐	☐	☐	☐

As questões desta secção são relativas à linguagem *Network Scenarios Description Language*.

4 Identifique na seguinte tabela, para cada tarefa, o grau de dificuldade no **PRIMEIRO** projecto para criar em NSDL...

Varia de "1 = Muito Simples" até "5 = Bastante Complexo". O valor intermédio é "3 = Nem Simples nem Complexo".

	Não usou	1	2	3	4	5
... os nós (computadores, routers)	☐	☐	☐	☐	☐	☐
... as ligações (simplex, duplex)	☐	☐	☐	☐	☐	☐
... as aplicações (e protocolos)	☐	☐	☐	☐	☐	☐
... os templates	☐	☐	☐	☐	☐	☐
... as views	☐	☐	☐	☐	☐	☐
... as topologias (os objectos em geral)	☐	☐	☐	☐	☐	☐
... os eventos da simulação	☐	☐	☐	☐	☐	☐
... as estatísticas e os resultados	☐	☐	☐	☐	☐	☐
... os cenários (Visualização e Simulação)	☐	☐	☐	☐	☐	☐
... todos os componentes no geral	☐	☐	☐	☐	☐	☐

5 Identifique na seguinte tabela, para cada tarefa, o grau de dificuldade no **SEGUNDO** projecto para criar em NSDL...

Varia de "1 = Muito Simples" até "5 = Bastante Complexo". O valor intermédio é "3 = Nem Simples nem Complexo".

	Não usou	1	2	3	4	5
... os nós (computadores, routers)	☐	☐	☐	☐	☐	☐
... as ligações (simplex, duplex)	☐	☐	☐	☐	☐	☐
... as aplicações (e protocolos)	☐	☐	☐	☐	☐	☐
... os templates	☐	☐	☐	☐	☐	☐
... as views	☐	☐	☐	☐	☐	☐
... as topologias (os objectos em geral)	☐	☐	☐	☐	☐	☐
... os eventos da simulação	☐	☐	☐	☐	☐	☐
... as estatísticas e os resultados	☐	☐	☐	☐	☐	☐
... os cenários (Visualização e Simulação)	☐	☐	☐	☐	☐	☐
... todos os componentes no geral	☐	☐	☐	☐	☐	☐

6 Qual é, na sua opinião, a utilidade dos seguintes aspectos e componentes presentes no NSDL para representar as redes de computadores...

Varia de "1 = Pouco Útil" até "5 = Muito Útil". O valor intermédio é "3 = Alguma Utilidade".

	1	2	3	4	5
Templates para os objectos de rede	☐	☐	☐	☐	☐
Views para os objectos de rede	☐	☐	☐	☐	☐
Separação entre a rede e os cenários	☐	☐	☐	☐	☐
Possível múltiplos cenários	☐	☐	☐	☐	☐

As questões desta secção referem-se a ambas as linguagens.

7 Para cada um dos aspectos abaixo, indique qual a linguagem onde foi **MAIS SIMPLES** definir...

Varia de "1 = Claramente no Tcl" até "5 = Claramente no NSDL". O valor intermédio é "3 = Igual Dificuldade".

	+Tcl	2	3	4	+NSDL
... os nós (computadores, routers)	☐	☐	☐	☐	☐
... as ligações (simplex, duplex)	☐	☐	☐	☐	☐
... as aplicações (e protocolos)	☐	☐	☐	☐	☐
... as topologias (os objectos em geral)	☐	☐	☐	☐	☐
... os cenários (Visualização e Simulação)	☐	☐	☐	☐	☐
... os eventos da simulação	☐	☐	☐	☐	☐
... as estatísticas e os resultados	☐	☐	☐	☐	☐

8 Na sua opinião, uso de duas linguagens no âmbito de TAR foi para a compreensão...

Varia de "1 = Pouco Útil" até "5 = Muito Útil". O valor intermédio é "3 = Alguma Utilidade".

	1	2	3	4	5
... dos mecanismos de QoS	☐	☐	☐	☐	☐
... da simulação de redes	☐	☐	☐	☐	☐
... da simulação de redes com QoS	☐	☐	☐	☐	☐
... do tópico descrição de redes	☐	☐	☐	☐	☐

9 Na caixa de texto seguinte pode deixar todas as suas opiniões e observações sobre o uso NSDL em TAR e sobre o próprio NSDL.

Apêndice D

Código completo de um cenário WiMAX

O código neste apêndice apresenta um cenário de simulação completo com objetos da rede sem fios WiMAX. São apresentados os códigos nos formatos NSDL e C++ (para o NS3). A rede descrita neste cenário pode ser visualizada na Figura 6.14.

D.1 Código no formato NSDL/XML

```
<nsdl><!-- ns.11 verified (8/7/2011) & Schema Validation .:Exemplo WiMAX:. -->
<network>
<objects>

<node id="node0">
       <wlan802.16 id="wimaxN0">
              <type>ss</type>
              <scheduler.type>simple</scheduler.type>
              <modulation.type>qam16_12</modulation.type>
              <service.flow>
                     <direction>up</direction>
                     <scheduler.type>be</scheduler.type>
                     <ipcs.classifier>
                            <src.port.lower>0</src.port.lower>
                            <src.port.upper>65000</src.port.upper>
                            <dst.port.lower>100</dst.port.lower> <!-- CBR1 Server -->
                            <dst.port.upper>100</dst.port.upper>
                            <protocol.id>17</protocol.id>
                            <priority>1</priority>
                     </ipcs.classifier>
              </service.flow>
       </wlan802.16>
       <cbr id="cbr1Server">
              <role>server</role>
              <dst.app>cbr1Client</dst.app>
       </cbr>
</node>

 <node id="node1">
       <wlan802.16 id="wimaxN1">
              <type>ss</type>
              <scheduler.type>simple</scheduler.type>
              <modulation.type>qam16_12</modulation.type>
              <service.flow>
                     <direction>up</direction>
                     <scheduler.type>be</scheduler.type>
                     <ipcs.classifier>
                            <src.port.lower>0</src.port.lower>
                            <src.port.upper>65000</src.port.upper>
                            <dst.port.lower>200</dst.port.lower> <!-- FTP2 Server -->
                            <dst.port.upper>200</dst.port.upper>
                            <protocol.id>6</protocol.id>
```

```
43                              <priority>2</priority>
44                      </ipcs.classifier>
45              </service.flow>
46      </wlan802.16>
47      <ftp id="ftp2Server">
48              <role>server</role>
49              <dst.app>ftp2Client</dst.app>
50      </ftp>
51  </node>
52
53  <node id="node2">
54      <wlan802.16 id="wimaxN2">
55              <type>ss</type>
56              <scheduler.type>simple</scheduler.type>
57              <modulation.type>qam16_12</modulation.type>
58              <service.flow>
59                      <direction>down</direction>
60                      <scheduler.type>be</scheduler.type>
61                      <ipcs.classifier>
62                              <src.port.lower>0</src.port.lower>
63                              <src.port.upper>65000</src.port.upper>
64                              <dst.port.lower>100</dst.port.lower> <!-- CBR1 Client -->
65                              <dst.port.upper>100</dst.port.upper>
66                              <protocol.id>17</protocol.id>
67                              <priority>1</priority>
68                      </ipcs.classifier>
69              </service.flow>
70      </wlan802.16>
71      <cbr id="cbr1Client">
72              <role>client</role>
73              <src.app>cbr1Server</src.app>
74      </cbr>
75  </node>
76
77  <node id="node3">
78      <wlan802.16 id="wimaxN3">
79              <type>ss</type>
80              <scheduler.type>simple</scheduler.type>
81              <modulation.type>qam16_12</modulation.type>
82              <service.flow>
83                      <direction>up</direction>
84                      <scheduler.type>be</scheduler.type>
85                      <ipcs.classifier>
86                              <src.port.lower>0</src.port.lower>
87                              <src.port.upper>65000</src.port.upper>
88                              <dst.port.lower>300</dst.port.lower> <!-- FTP1 Server -->
89                              <dst.port.upper>300</dst.port.upper>
90                              <protocol.id>6</protocol.id>
91                              <priority>2</priority>
92                      </ipcs.classifier>
93              </service.flow>
94      </wlan802.16>
95      <ftp id="ftp1Server">
96              <role>server</role>
97              <dst.app>ftp1Client</dst.app>
98      </ftp>
99  </node>
100
101 <node id="node4">
102     <ipv4 id="ipv4node3">
103             <interface.id>wimaxN4</interface.id>
104             <mask>255.255.255.0</mask>
105             <net.address>10.1.4.0</net.address>
106     </ipv4>
107     <wlan802.16 id="wimaxN4">
108             <type>bs</type>
109             <scheduler.type>simple</scheduler.type>
110     </wlan802.16>
111 </node>
112
113 <node id="node5">
114     <wlan802.16 id="wimaxN5">
115             <type>ss</type>
116             <scheduler.type>simple</scheduler.type>
```

```
117                 <modulation.type>qam16_12</modulation.type>
118                 <service.flow>
119                         <direction>up</direction>
120                         <scheduler.type>be</scheduler.type>
121                         <ipcs.classifier>
122                                 <src.port.lower>0</src.port.lower>
123                                 <src.port.upper>65000</src.port.upper>
124                                 <dst.port.lower>400</dst.port.lower> <!-- Pareto1 Server -->
125                                 <dst.port.upper>400</dst.port.upper>
126                                 <protocol.id>17</protocol.id>
127                                 <priority>1</priority>
128                         </ipcs.classifier>
129                 </service.flow>
130         </wlan802.16>
131         <pareto id="pareto1Server">
132                 <role>server</role>
133                 <dst.app>pareto1Client</dst.app>
134                 <shape>2</shape>
135         </pareto>
136 </node>
137
138 <node id="node6">
139         <wlan802.16 id="wimaxN6">
140                 <type>ss</type>
141                 <scheduler.type>simple</scheduler.type>
142                 <modulation.type>qam16_12</modulation.type>
143                 <service.flow>
144                         <direction>down</direction>
145                         <scheduler.type>be</scheduler.type>
146                         <ipcs.classifier>
147                                 <src.port.lower>0</src.port.lower>
148                                 <src.port.upper>65000</src.port.upper>
149                                 <dst.port.lower>400</dst.port.lower> <!-- Pareto1 Client -->
150                                 <dst.port.upper>400</dst.port.upper>
151                                 <protocol.id>17</protocol.id>
152                                 <priority>1</priority>
153                         </ipcs.classifier>
154                 </service.flow>
155         </wlan802.16>
156         <pareto id="pareto1Client">
157                 <role>client</role>
158                 <src.app>pareto1Server</src.app>
159         </pareto>
160 </node>
161
162 <node id="node7">
163         <wlan802.16 id="wimaxN7">
164                 <type>ss</type>
165                 <scheduler.type>simple</scheduler.type>
166                 <modulation.type>qam16_12</modulation.type>
167                 <service.flow>
168                         <direction>down</direction>
169                         <scheduler.type>be</scheduler.type>
170                         <ipcs.classifier>
171                                 <src.port.lower>0</src.port.lower>
172                                 <src.port.upper>65000</src.port.upper>
173                                 <dst.port.lower>200</dst.port.lower> <!-- FTP2 Client -->
174                                 <dst.port.upper>200</dst.port.upper>
175                                 <protocol.id>6</protocol.id>
176                                 <priority>2</priority>
177                         </ipcs.classifier>
178                 </service.flow>
179         </wlan802.16>
180         <ftp id="ftp2Client">
181                 <role>client</role>
182                 <src.app>ftp2Server</src.app>
183         </ftp>
184 </node>
185
186 <node id="node8">
187         <wlan802.16 id="wimaxN8">
188                 <type>ss</type>
189                 <scheduler.type>simple</scheduler.type>
190                 <modulation.type>qam16_12</modulation.type>
```

```
191                 <service.flow>
192                         <direction>down</direction>
193                         <scheduler.type>be</scheduler.type>
194                         <ipcs.classifier>
195                                 <src.port.lower>0</src.port.lower>
196                                 <src.port.upper>65000</src.port.upper>
197                                 <dst.port.lower>300</dst.port.lower> <!-- FTP1 Client -->
198                                 <dst.port.upper>300</dst.port.upper>
199                                 <protocol.id>6</protocol.id>
200                                 <priority>2</priority>
201                         </ipcs.classifier>
202                 </service.flow>
203         </wlan802.16>
204         <ftp id="ftp1Client">
205                 <role>client</role>
206                 <src.app>ftp1Server</src.app>
207         </ftp>
208 </node>
209 <!-- links -->
210
211 <link.wimax id="wimaxLink">
212 </link.wimax>
213
214 </objects>
215 </network>
216 <scenarios>
217 <visualizations>
218                         <ns3visualization id="vis01">
219                                 <name>Visual01</name>
220                                 <description>
221                                         <object id="node0">
222                                                 <x.position>10</x.position>
223                                                 <y.position>5</y.position>
224                                                 <z.position>0</z.position>
225                                         </object>
226                                         <object id="node1">
227                                                 <x.position>50</x.position>
228                                                 <y.position>5</y.position>
229                                                 <z.position>0</z.position>
230                                         </object>
231                                         <object id="node2">
232                                                 <x.position>130</x.position>
233                                                 <y.position>5</y.position>
234                                                 <z.position>0</z.position>
235                                         </object>
236                                         <object id="node3">
237                                                 <x.position>5</x.position>
238                                                 <y.position>50</y.position>
239                                                 <z.position>0</z.position>
240                                         </object>
241                                         <object id="node4">
242                                                 <x.position>50</x.position>
243                                                 <y.position>50</y.position>
244                                                 <z.position>0</z.position>
245                                         </object>
246                                         <object id="node5">
247                                                 <x.position>90</x.position>
248                                                 <y.position>50</y.position>
249                                                 <z.position>0</z.position>
250                                         </object>
251                                         <object id="node6">
252                                                 <x.position>10</x.position>
253                                                 <y.position>90</y.position>
254                                                 <z.position>0</z.position>
255                                         </object>
256                                         <object id="node7">
257                                                 <x.position>50</x.position>
258                                                 <y.position>90</y.position>
259                                                 <z.position>0</z.position>
260                                         </object>
261                                         <object id="node8">
262                                                 <x.position>130</x.position>
263                                                 <y.position>90</y.position>
264                                                 <z.position>0</z.position>
```

```
                                </object>
                        </description>
                </ns3visualization>
        </visualizations>
        <simulations>
                <ns3simulation id="sim01">
                        <description>
                                <general>
                                        <duration>
                                                <time>8</time>
                                        </duration>
                                        <runs>1</runs>
                                        <simulator>ns3.11</simulator>
                                </general>
                                <events>
                                        <event objectid="ftp1Server" time="0.2">
                                                <parameter name="action" value="Start"/>
                                        </event>
                                        <event objectid="ftp2Server" time="0.2">
                                                <parameter name="action" value="Start"/>
                                        </event>
                                        <event objectid="pareto1Server" time="0.6">
                                                <parameter name="action" value="Start"/>
                                        </event>
                                        <event objectid="cbr1Server" time="0.6">
                                                <parameter name="action" value="Start"/>
                                        </event>
                                        <event objectid="ftp1Server" time="10.0">
                                                <parameter name="action" value="Stop"/>
                                        </event>
                                        <event objectid="ftp2Server" time="10.0">
                                                <parameter name="action" value="Stop"/>
                                        </event>
                                        <event objectid="pareto1Server" time="10.0">
                                                <parameter name="action" value="Stop"/>
                                        </event>
                                        <event objectid="cbr1Server" time="10.0">
                                                <parameter name="action" value="Stop"/>
                                        </event>
                                </events>
                                <outputs>
                                </outputs>
                                <extra>
                                </extra>
                        </description>
                </ns3simulation>
        </simulations>
    </scenarios>
</nsdl>
```

D.2 Código na linguagem C++

```
/***** Inclusion of main libraries *****
**************************************/
#include <iostream>
#include <fstream>
#include <string>
#include <cassert>
/*** Minimum set of libraries for last stable version: ns-3.11 (May 25, 2011) ***/
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/mobility-module.h"
#include "ns3/config-store-module.h"
// Routing support
#include "ns3/aodv-module.h"
#include "ns3/olsr-helper.h"
#include "ns3/dsdv-helper.h"
#include "ns3/olsr-routing-protocol.h"
```

```
20 #include "ns3/ipv4-static-routing-helper.h"
21 #include "ns3/ipv4-list-routing-helper.h"
22 #include "ns3/ipv4-nix-vector-helper.h"
23 #include "ns3/ipv4-static-routing-helper.h"
24 #include "ns3/ipv4-list-routing-helper.h"
25 #include "ns3/flow-monitor-module.h"
26 #include "ns3/csma-module.h"
27 #include "ns3/wifi-module.h"
28 // Wimax support
29 #include "ns3/wimax-module.h"
30 #include "ns3/mesh-module.h"
31 #include "ns3/mesh-helper.h"
32 #include "ns3/lte-module.h"
33 #include <ns3/spectrum-helper.h>
34 #include <ns3/lte-helper.h>
35 #include <ns3/enb-phy.h>
36 #include <ns3/ue-phy.h>
37 #include <ns3/packet-burst.h>
38 #include <ns3/constant-position-mobility-model.h>
39 #include <ns3/constant-velocity-mobility-model.h>
40 #include "ns3/single-model-spectrum-channel.h"
41 #include "ns3/lte-spectrum-phy.h"
42 #include "ns3/enb-lte-spectrum-phy.h"
43 #include "ns3/ue-lte-spectrum-phy.h"
44 #include "ns3/ue-net-device.h"
45 #include "ns3/enb-net-device.h"
46 #include "ns3/ue-manager.h"
47 #include "ns3/spectrum-propagation-loss-model.h"
48 #include "ns3/lte-propagation-loss-model.h"
49 #include "ns3/global-route-manager.h"
50
51 using namespace ns3;
52 NS_LOG_COMPONENT_DEFINE ("My_NS3_Scenario");
53 /***** Standard header for NS3 programs *****/
54 static uint32_t m_bytesTotal;
55 static void
56 Throughput (){ // in Mbps calculated every 0.2s
57   Time time = Simulator::Now ();
58   double mbps = (((m_bytesTotal * 8.0) / 1000000)/0.2);
59   cout << time.GetSeconds() << " " << mbps << endl;
60   m_bytesTotal = 0;
61   Simulator::Schedule (Seconds (0.2), &Throughput);
62 }
63 void SendPacket( std::string context, Ptr<const Packet> p) {
64  m_bytesTotal += p->GetSize ();
65 }
66 void ReceivedPacket (std::string context, Ptr<const Packet> p, const
67 Address& addr) {
68  m_bytesTotal += p->GetSize ();
69 }
70 void DevTxTrace (std::string context, Ptr<const Packet> p) {
71   Time time = Simulator::Now ();
72   cout << time.GetSeconds() << " TX Packet " << "ID: " << p->GetUid() << endl;
73 }
74 void DevRxTrace (std::string context, Ptr<const Packet> p, const
75 Address& addr) {
76   Time time = Simulator::Now ();
77   cout << time.GetSeconds() << " RX Packet " << "ID: " << p->GetUid() << endl;
78 }
79 int main (int argc, char *argv[]){
80   CommandLine cmd;
81   bool enableFlowMonitor = true;
82   bool verbose = false;
83   bool verboseLTE = false;
84   cmd.AddValue ("verbose", "turn on all WimaxNetDevice log components", verbose);
85   cmd.AddValue ("verboseLTE", "turn on all LteNetDevice log components", verboseLTE);
86   cmd.AddValue("EnableMonitor", "Enable Flow Monitor", enableFlowMonitor);
87   cmd.Parse (argc, argv);
88   // Enabling Packet Metadata avoids further errors in ascii recording files
89   ns3::PacketMetadata::Enable ();
90   WimaxHelper wimax;
91   // CONFIGURE DL and UL SUB CHANNELS
92   // Define a list of sub channels for the downlink
93   std::vector<int> dlSubChannels;
```

```
94    for (int i = 0; i < 25; i++){ dlSubChannels.push_back (i); }
95    // Define a list of sub channels for the uplink
96    std::vector<int> ulSubChannels;
97    for (int i = 50; i < 100; i++){ulSubChannels.push_back (i);}
98
99  /***** Definition of ports to use on Applications *****/
100 NS_LOG_INFO ("Create Applications.");
101
102 Ptr<Node> node0 = CreateObject<Node> (); /* *** Node node0 *** */
103 Ptr<Ipv4> Ipv4node0 = node0->GetObject<Ipv4> (); /**** Getting node ready with Ipv4 ***/
104 Ptr<Ipv6> Ipv6node0 = node0->GetObject<Ipv6> (); /**** Getting node ready with Ipv6 ***/
105
106 Ptr<Node> node1 = CreateObject<Node> (); /* *** Node node1 *** */
107 Ptr<Ipv4> Ipv4node1 = node1->GetObject<Ipv4> (); /**** Getting node ready with Ipv4 ***/
108 Ptr<Ipv6> Ipv6node1 = node1->GetObject<Ipv6> (); /**** Getting node ready with Ipv6 ***/
109
110 Ptr<Node> node2 = CreateObject<Node> (); /* *** Node node2 *** */
111 Ptr<Ipv4> Ipv4node2 = node2->GetObject<Ipv4> (); /**** Getting node ready with Ipv4 ***/
112 Ptr<Ipv6> Ipv6node2 = node2->GetObject<Ipv6> (); /**** Getting node ready with Ipv6 ***/
113
114 Ptr<Node> node3 = CreateObject<Node> (); /* *** Node node3 *** */
115 Ptr<Ipv4> Ipv4node3 = node3->GetObject<Ipv4> (); /**** Getting node ready with Ipv4 ***/
116 Ptr<Ipv6> Ipv6node3 = node3->GetObject<Ipv6> (); /**** Getting node ready with Ipv6 ***/
117
118 Ptr<Node> node4 = CreateObject<Node> (); /* *** Node node4 *** */
119 Ptr<Ipv4> Ipv4node4 = node4->GetObject<Ipv4> (); /**** Getting node ready with Ipv4 ***/
120 Ptr<Ipv6> Ipv6node4 = node4->GetObject<Ipv6> (); /**** Getting node ready with Ipv6 ***/
121
122 Ptr<Node> node5 = CreateObject<Node> (); /* *** Node node5 *** */
123 Ptr<Ipv4> Ipv4node5 = node5->GetObject<Ipv4> (); /**** Getting node ready with Ipv4 ***/
124 Ptr<Ipv6> Ipv6node5 = node5->GetObject<Ipv6> (); /**** Getting node ready with Ipv6 ***/
125
126 Ptr<Node> node6 = CreateObject<Node> (); /* *** Node node6 *** */
127 Ptr<Ipv4> Ipv4node6 = node6->GetObject<Ipv4> (); /**** Getting node ready with Ipv4 ***/
128 Ptr<Ipv6> Ipv6node6 = node6->GetObject<Ipv6> (); /**** Getting node ready with Ipv6 ***/
129
130 Ptr<Node> node7 = CreateObject<Node> (); /* *** Node node7 *** */
131 Ptr<Ipv4> Ipv4node7 = node7->GetObject<Ipv4> (); /**** Getting node ready with Ipv4 ***/
132 Ptr<Ipv6> Ipv6node7 = node7->GetObject<Ipv6> (); /**** Getting node ready with Ipv6 ***/
133
134 Ptr<Node> node8 = CreateObject<Node> (); /* *** Node node8 *** */
135 Ptr<Ipv4> Ipv4node8 = node8->GetObject<Ipv4> (); /**** Getting node ready with Ipv4 ***/
136 Ptr<Ipv6> Ipv6node8 = node8->GetObject<Ipv6> (); /**** Getting node ready with Ipv6 ***/
137
138 /***** Definition of internet behavior *****/
139 InternetStackHelper internet;
140 internet.InstallAll ();
141 Ptr<SimpleOfdmWimaxChannel> wimaxLink;
142 wimaxLink = CreateObject<SimpleOfdmWimaxChannel> ();
143
144 /***** Interfaces Setting *****/
145 Ipv4InterfaceContainer iwimaxN0;
146 Ipv6InterfaceContainer iwimaxN0v6;
147 WimaxHelper::SchedulerType wimaxN0scheduler = WimaxHelper::SCHED_TYPE_SIMPLE;
148 NetDeviceContainer dnode0d = wimax.Install ( NodeContainer (node0) ,
149                       WimaxHelper::DEVICE_TYPE_SUBSCRIBER_STATION ,
150                       WimaxHelper::SIMPLE_PHY_TYPE_OFDM , wimaxN0scheduler);
151 Ptr<SubscriberStationNetDevice> node0SS;
152 node0SS = dnode0d.Get(0) ->GetObject<SubscriberStationNetDevice> ();
153  /***** Modulation Type Setting *****/
154 node0SS->SetModulationType (WimaxPhy::MODULATION_TYPE_QAM16_12 );
155
156  /***** Interfaces Setting *****/
157 Ipv4InterfaceContainer iwimaxN1;
158 Ipv6InterfaceContainer iwimaxN1v6;
159 WimaxHelper::SchedulerType wimaxN1scheduler = WimaxHelper::SCHED_TYPE_SIMPLE;
160 NetDeviceContainer dnode1d = wimax.Install ( NodeContainer (node1) ,
161                       WimaxHelper::DEVICE_TYPE_SUBSCRIBER_STATION ,
162                       WimaxHelper::SIMPLE_PHY_TYPE_OFDM , wimaxN1scheduler);
163 Ptr<SubscriberStationNetDevice> node1SS;
164 node1SS = dnode1d.Get(0) ->GetObject<SubscriberStationNetDevice> ();
165  /***** Modulation Type Setting *****/
166 node1SS->SetModulationType (WimaxPhy::MODULATION_TYPE_QAM16_12 );
167
```

```
168  /***** Interfaces Setting *****/
169 Ipv4InterfaceContainer iwimaxN2;
170 Ipv6InterfaceContainer iwimaxN2v6;
171 WimaxHelper::SchedulerType wimaxN2scheduler = WimaxHelper::SCHED_TYPE_SIMPLE;
172 NetDeviceContainer dnode2d = wimax.Install ( NodeContainer (node2) ,
173                       WimaxHelper::DEVICE_TYPE_SUBSCRIBER_STATION ,
174                       WimaxHelper::SIMPLE_PHY_TYPE_OFDM , wimaxN2scheduler);
175 Ptr<SubscriberStationNetDevice> node2SS;
176 node2SS = dnode2d.Get(0) ->GetObject<SubscriberStationNetDevice> ();
177  /***** Modulation Type Setting *****/
178 node2SS->SetModulationType (WimaxPhy::MODULATION_TYPE_QAM16_12 );
179
180  /***** Interfaces Setting *****/
181 Ipv4InterfaceContainer iwimaxN3;
182 Ipv6InterfaceContainer iwimaxN3v6;
183 WimaxHelper::SchedulerType wimaxN3scheduler = WimaxHelper::SCHED_TYPE_SIMPLE;
184 NetDeviceContainer dnode3d = wimax.Install ( NodeContainer (node3) ,
185                       WimaxHelper::DEVICE_TYPE_SUBSCRIBER_STATION ,
186                       WimaxHelper::SIMPLE_PHY_TYPE_OFDM , wimaxN3scheduler);
187 Ptr<SubscriberStationNetDevice> node3SS;
188 node3SS = dnode3d.Get(0) ->GetObject<SubscriberStationNetDevice> ();
189  /***** Modulation Type Setting *****/
190 node3SS->SetModulationType (WimaxPhy::MODULATION_TYPE_QAM16_12 );
191
192  /***** Interfaces Setting *****/
193 Ipv4InterfaceContainer iwimaxN4;
194 Ipv6InterfaceContainer iwimaxN4v6;
195 WimaxHelper::SchedulerType wimaxN4scheduler = WimaxHelper::SCHED_TYPE_SIMPLE;
196 NetDeviceContainer dnode4d = wimax.Install ( NodeContainer (node4) ,
197                       WimaxHelper::DEVICE_TYPE_BASE_STATION ,
198                       WimaxHelper::SIMPLE_PHY_TYPE_OFDM , wimaxN4scheduler);
199 Ptr<BaseStationNetDevice> node4BS;
200 node4BS = dnode4d.Get(0) ->GetObject<BaseStationNetDevice> ();
201
202  /***** Interfaces Setting *****/
203 Ipv4InterfaceContainer iwimaxN5;
204 Ipv6InterfaceContainer iwimaxN5v6;
205 WimaxHelper::SchedulerType wimaxN5scheduler = WimaxHelper::SCHED_TYPE_SIMPLE;
206 NetDeviceContainer dnode5d = wimax.Install ( NodeContainer (node5) ,
207                       WimaxHelper::DEVICE_TYPE_SUBSCRIBER_STATION ,
208                       WimaxHelper::SIMPLE_PHY_TYPE_OFDM , wimaxN5scheduler);
209 Ptr<SubscriberStationNetDevice> node5SS;
210 node5SS = dnode5d.Get(0) ->GetObject<SubscriberStationNetDevice> ();
211  /***** Modulation Type Setting *****/
212 node5SS->SetModulationType (WimaxPhy::MODULATION_TYPE_QAM16_12 );
213
214  /***** Interfaces Setting *****/
215 Ipv4InterfaceContainer iwimaxN6;
216 Ipv6InterfaceContainer iwimaxN6v6;
217 WimaxHelper::SchedulerType wimaxN6scheduler = WimaxHelper::SCHED_TYPE_SIMPLE;
218 NetDeviceContainer dnode6d = wimax.Install ( NodeContainer (node6) ,
219                       WimaxHelper::DEVICE_TYPE_SUBSCRIBER_STATION ,
220                       WimaxHelper::SIMPLE_PHY_TYPE_OFDM , wimaxN6scheduler);
221 Ptr<SubscriberStationNetDevice> node6SS;
222 node6SS = dnode6d.Get(0) ->GetObject<SubscriberStationNetDevice> ();
223  /***** Modulation Type Setting *****/
224 node6SS->SetModulationType (WimaxPhy::MODULATION_TYPE_QAM16_12 );
225
226  /***** Interfaces Setting *****/
227 Ipv4InterfaceContainer iwimaxN7;
228 Ipv6InterfaceContainer iwimaxN7v6;
229 WimaxHelper::SchedulerType wimaxN7scheduler = WimaxHelper::SCHED_TYPE_SIMPLE;
230 NetDeviceContainer dnode7d = wimax.Install ( NodeContainer (node7) ,
231                       WimaxHelper::DEVICE_TYPE_SUBSCRIBER_STATION ,
232                       WimaxHelper::SIMPLE_PHY_TYPE_OFDM , wimaxN7scheduler);
233 Ptr<SubscriberStationNetDevice> node7SS;
234 node7SS = dnode7d.Get(0) ->GetObject<SubscriberStationNetDevice> ();
235  /***** Modulation Type Setting *****/
236 node7SS->SetModulationType (WimaxPhy::MODULATION_TYPE_QAM16_12 );
237
238  /***** Interfaces Setting *****/
239 Ipv4InterfaceContainer iwimaxN8;
240 Ipv6InterfaceContainer iwimaxN8v6;
241 WimaxHelper::SchedulerType wimaxN8scheduler = WimaxHelper::SCHED_TYPE_SIMPLE;
```

```
242 NetDeviceContainer dnode8d = wimax.Install ( NodeContainer (node8) ,
243                     WimaxHelper::DEVICE_TYPE_SUBSCRIBER_STATION ,
244                     WimaxHelper::SIMPLE_PHY_TYPE_OFDM , wimaxN8scheduler);
245 Ptr<SubscriberStationNetDevice> node8SS;
246 node8SS = dnode8d.Get(0) ->GetObject<SubscriberStationNetDevice> ();
247  /***** Modulation Type Setting *****/
248 node8SS->SetModulationType (WimaxPhy::MODULATION_TYPE_QAM16_12 );
249 if (verbose){
250   wimax.EnableLogComponents (); // Turn on all wimax logging
251 }
252 if (verboseLTE){
253   lte.EnableLogComponents (); // Turn on all lte logging
254 }
255 /*** Protocol creation ***/
256 /*** WiMAX Ipv4 Addressing ***/
257 Ipv4AddressHelper ipv4node3;
258 ipv4node3.SetBase("10.1.4.0", "255.255.255.0");
259 iwimaxN4 = ipv4node3.Assign(dnode4d);iwimaxN0 = ipv4node3.Assign(dnode0d);
260 iwimaxN1 = ipv4node3.Assign(dnode1d);iwimaxN2 = ipv4node3.Assign(dnode2d);
261 iwimaxN3 = ipv4node3.Assign(dnode3d);iwimaxN5 = ipv4node3.Assign(dnode5d);
262 iwimaxN6 = ipv4node3.Assign(dnode6d);iwimaxN7 = ipv4node3.Assign(dnode7d);
263 iwimaxN8 = ipv4node3.Assign(dnode8d);
264
265 /** Application creation **
266 ***************************/
267 /* *** Application "cbr1Server"(node0) to "cbr1Client" *** */
268 OnOffHelper onoffcbr1Server ("ns3::UdpSocketFactory",
269              InetSocketAddress (iwimaxN2.GetAddress(0), 100));
270 onoffcbr1Server.SetAttribute ("OnTime", RandomVariableValue (ConstantVariable (1)));
271 onoffcbr1Server.SetAttribute ("OffTime", RandomVariableValue (ConstantVariable (0)));
272 ApplicationContainer appscbr1Server = onoffcbr1Server.Install (node0);
273 appscbr1Server.Start (Seconds (0.6)); appscbr1Server.Stop (Seconds (10.0));
274
275 /** Application creation **
276 ***************************/
277 /* *** Application "ftp2Server"(node1) to "ftp2Client" *** */
278 OnOffHelper onoffftp2Server ("ns3::TcpSocketFactory",
279              InetSocketAddress (iwimaxN7.GetAddress(0), 200));
280 onoffftp2Server.SetAttribute ("OnTime", RandomVariableValue (ConstantVariable (1)));
281 onoffftp2Server.SetAttribute ("OffTime", RandomVariableValue (ConstantVariable (0)));
282 ApplicationContainer appsftp2Server = onoffftp2Server.Install (node1);
283 appsftp2Server.Start (Seconds (0.2)); appsftp2Server.Stop (Seconds (10.0));
284
285 /** Application creation **
286 ***************************/
287 /* *** Application "ftp1Server"(node3) to "ftp1Client" *** */
288 OnOffHelper onoffftp1Server ("ns3::TcpSocketFactory",
289              InetSocketAddress (iwimaxN8.GetAddress(0), 300));
290 onoffftp1Server.SetAttribute ("OnTime", RandomVariableValue (ConstantVariable (1)));
291 onoffftp1Server.SetAttribute ("OffTime", RandomVariableValue (ConstantVariable (0)));
292 ApplicationContainer appsftp1Server = onoffftp1Server.Install (node3);
293 appsftp1Server.Start (Seconds (0.2)); appsftp1Server.Stop (Seconds (10.0));
294
295 /** Application creation **
296 ***************************/
297 /* *** Application "pareto1Server"(node5) to "pareto1Client" *** */
298 OnOffHelper onoffpareto1Server ("ns3::UdpSocketFactory",
299              InetSocketAddress (iwimaxN6.GetAddress(0), 400));
300 onoffpareto1Server.SetAttribute ("OnTime", RandomVariableValue (ParetoVariable (1,2)));
301 onoffpareto1Server.SetAttribute ("OffTime", RandomVariableValue (ParetoVariable (1,2)));
302 ApplicationContainer appspareto1Server = onoffpareto1Server.Install (node5);
303 appspareto1Server.Start (Seconds (0.6)); appspareto1Server.Stop (Seconds (10.0));
304
305 /** Sink Creation **
306 ***************************/
307 /* *** Receiver of application "cbr1Server" at node2 *** */
308 PacketSinkHelper sinkcbr1Servernode2 ("ns3::UdpSocketFactory",
309              InetSocketAddress (Ipv4Address::GetAny (), 100));
310 appscbr1Server = sinkcbr1Servernode2.Install (node2);
311 appscbr1Server.Start (Seconds (0.6)); appscbr1Server.Stop (Seconds (10.0));
312
313 /** Sink Creation **
314 ***************************/
315 /* *** Receiver of application "pareto1Server" at node6 *** */
```

```
316 PacketSinkHelper sinkpareto1Servernode6 ("ns3::UdpSocketFactory",
317              InetSocketAddress (Ipv4Address::GetAny (), 400));
318 appspareto1Server = sinkpareto1Servernode6.Install (node6);
319 appspareto1Server.Start (Seconds (0.6)); appspareto1Server.Stop (Seconds (10.0));
320
321 /** Sink Creation **
322 ***************************/
323 /* *** Receiver of application "ftp2Server" at node7 *** */
324 PacketSinkHelper sinkftp2Servernode7 ("ns3::TcpSocketFactory",
325              InetSocketAddress (Ipv4Address::GetAny (), 200));
326 appsftp2Server = sinkftp2Servernode7.Install (node7);
327 appsftp2Server.Start (Seconds (0.2)); appsftp2Server.Stop (Seconds (10.0));
328
329 /** Sink Creation **
330 ***************************/
331 /* *** Receiver of application "ftp1Server" at node8 *** */
332 PacketSinkHelper sinkftp1Servernode8 ("ns3::TcpSocketFactory",
333              InetSocketAddress (Ipv4Address::GetAny (), 300));
334 appsftp1Server = sinkftp1Servernode8.Install (node8);
335 appsftp1Server.Start (Seconds (0.2));
336 appsftp1Server.Stop (Seconds (10.0));
337 std::string context1 = "/NodeList/*/DeviceList/*/Phy/State/RxOk";
338 std::string context2 = "/NodeList/*/DeviceList/*/$ns3::BaseStationNetDevice/";
339 std::string context5 = "/NodeList/*/DeviceList/*/$ns3::BaseStationNetDevice/BSTx"; // Works
340 std::string context4 = "/NodeList/*/ApplicationList/*/$ns3::PacketSink/Rx"; // Works
341 std::string context41 = "/NodeList/6/ApplicationList/*/$ns3::PacketSink/Rx"; // Works
342 std::string context3 = "/NodeList/*/DeviceList/*/$ns3::PacketSink/Rx";
343 Config::Connect ("/NodeList/*/ApplicationList/*/$ns3::OnOffApplication/Tx",
344               MakeCallback(&DevTxTrace));
345 Config::Connect ("/NodeList/*/ApplicationList/*/$ns3::PacketSink/Rx",
346               MakeCallback (&DevRxTrace));
347 Simulator::Stop(Seconds(12));
348 IpcsClassifierRecord UpwimaxN0100(iwimaxN0.GetAddress(0),
349                                   Ipv4Mask ("255.255.255.255"),Ipv4Address ("0.0.0.0"),
350                                   Ipv4Mask ("0.0.0.0"),0,65000,100,100,17,1);
351 ServiceFlow UpServiceFlowwimaxN0100 =
352                                   wimax.CreateServiceFlow (ServiceFlow::SF_DIRECTION_UP,
353                                   ServiceFlow::SF_TYPE_BE,UpwimaxN0100);
354 node0SS->AddServiceFlow (UpServiceFlowwimaxN0100);
355 IpcsClassifierRecord UpwimaxN1200(iwimaxN1.GetAddress(0),
356                                   Ipv4Mask ("255.255.255.255"),Ipv4Address ("0.0.0.0"),
357                                   Ipv4Mask ("0.0.0.0"), 0,65000,200,200,6,2);
358 ServiceFlow UpServiceFlowwimaxN1200 =
359                                   wimax.CreateServiceFlow (ServiceFlow::SF_DIRECTION_UP,
360                                   ServiceFlow::SF_TYPE_BE,UpwimaxN1200);
361 node1SS->AddServiceFlow (UpServiceFlowwimaxN1200);
362 IpcsClassifierRecord DownwimaxN2100 (Ipv4Address ("0.0.0.0"),
363                                   Ipv4Mask ("0.0.0.0"),iwimaxN2.GetAddress(0),
364                                   Ipv4Mask ("255.255.255.255"),0,65000,100,100,17,1);
365 ServiceFlow DownServiceFlowwimaxN2100 =
366                                   wimax.CreateServiceFlow (ServiceFlow::SF_DIRECTION_DOWN,
367                                   ServiceFlow::SF_TYPE_BE,DownwimaxN2100);
368 node2SS->AddServiceFlow (DownServiceFlowwimaxN2100);
369 IpcsClassifierRecord UpwimaxN3300(iwimaxN3.GetAddress(0),
370                                   Ipv4Mask ("255.255.255.255"),Ipv4Address ("0.0.0.0"),
371                                   Ipv4Mask ("0.0.0.0"),0,65000,300,300,6,2);
372 ServiceFlow UpServiceFlowwimaxN3300 =
373                                   wimax.CreateServiceFlow (ServiceFlow::SF_DIRECTION_UP,
374                                   ServiceFlow::SF_TYPE_BE,UpwimaxN3300);
375 node3SS->AddServiceFlow (UpServiceFlowwimaxN3300);
376 IpcsClassifierRecord UpwimaxN5400(iwimaxN5.GetAddress(0),
377                                   Ipv4Mask ("255.255.255.255"),Ipv4Address ("0.0.0.0"),
378                                   Ipv4Mask ("0.0.0.0"),0,65000,400,400,17,1);
379 ServiceFlow UpServiceFlowwimaxN5400 =
380                                   wimax.CreateServiceFlow (ServiceFlow::SF_DIRECTION_UP,
381                                   ServiceFlow::SF_TYPE_BE,UpwimaxN5400);
382 node5SS->AddServiceFlow (UpServiceFlowwimaxN5400);
383 IpcsClassifierRecord DownwimaxN6400 (Ipv4Address ("0.0.0.0"),
384                                   Ipv4Mask ("0.0.0.0"),iwimaxN6.GetAddress(0),
385                                   Ipv4Mask ("255.255.255.255"),0,65000,400,400,17,1);
386 ServiceFlow DownServiceFlowwimaxN6400 =
387                                   wimax.CreateServiceFlow (ServiceFlow::SF_DIRECTION_DOWN,
388                                   ServiceFlow::SF_TYPE_BE,DownwimaxN6400);
389 node6SS->AddServiceFlow (DownServiceFlowwimaxN6400);
```

```
390 IpcsClassifierRecord DownwimaxN7200 (Ipv4Address ("0.0.0.0"),
391                                      Ipv4Mask ("0.0.0.0"),iwimaxN7.GetAddress(0),
392                                      Ipv4Mask ("255.255.255.255"),0,65000,200,200,6,2);
393 ServiceFlow DownServiceFlowwimaxN7200 =
394                                      wimax.CreateServiceFlow (ServiceFlow::SF_DIRECTION_DOWN,
395                                      ServiceFlow::SF_TYPE_BE,DownwimaxN7200);
396 node7SS->AddServiceFlow (DownServiceFlowwimaxN7200);
397 IpcsClassifierRecord DownwimaxN8300 (Ipv4Address ("0.0.0.0"),
398                                      Ipv4Mask ("0.0.0.0"),iwimaxN8.GetAddress(0),
399                                      Ipv4Mask ("255.255.255.255"),0,65000,300,300,6,2);
400 ServiceFlow DownServiceFlowwimaxN8300 =
401                                      wimax.CreateServiceFlow (ServiceFlow::SF_DIRECTION_DOWN,
402                                      ServiceFlow::SF_TYPE_BE,DownwimaxN8300);
403 node8SS->AddServiceFlow (DownServiceFlowwimaxN8300);
404 /** Definitions to the visualization ****************************************/
405 /** Objects Position ********************************/
406 MobilityHelper mobility;
407 Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocator> ();
408 mobility.SetPositionAllocator (positionAlloc);
409 //nodes will move with constant speed
410 mobility.SetMobilityModel ("ns3::ConstantVelocityMobilityModel");
411 positionAlloc->Add (Vector (10, 5, 0)); // node0 node position
412 positionAlloc->Add (Vector (50, 5, 0)); // node1 node position
413 positionAlloc->Add (Vector (130, 5, 0)); // node2 node position
414 positionAlloc->Add (Vector (5, 50, 0)); // node3 node position
415 positionAlloc->Add (Vector (50, 50, 0)); // node4 node position
416 positionAlloc->Add (Vector (90, 50, 0)); // node5 node position
417 positionAlloc->Add (Vector (10, 90, 0)); // node6 node position
418 positionAlloc->Add (Vector (50, 90, 0)); // node7 node position
419 positionAlloc->Add (Vector (130, 90, 0)); // node8 node position
420 mobility.Install (node0); mobility.Install (node1);
421 mobility.Install (node2); mobility.Install (node3);
422 mobility.Install (node4); mobility.Install (node5);
423 mobility.Install (node6); mobility.Install (node7);
424 mobility.Install (node8);
425
426 /***** Definition of Ascii for trace events *****/
427 AsciiTraceHelper ascii;
428
429 /** Definitions to the output **
430 *********************************/
431 /***** Flow Monitor Definition and Visualization with PyViz *****/
432 // Flow Monitor
433 Ptr<FlowMonitor> flowmon;
434 if (enableFlowMonitor){
435        FlowMonitorHelper flowmonHelper;
436        flowmon = flowmonHelper.InstallAll ();
437 }
438 Simulator::Run ();
439
440 if (enableFlowMonitor){
441        flowmon->SerializeToXmlFile ("MyNS3Scenario.xml", true, true);
442 }
443 Simulator::Destroy ();
444 return 0;
445 }
```

A Nossa Universidade

Colégio dos Jesuítas
Rua dos Ferreiros - 9000-082, Funchal

Tel: +351 291 209400
Fax: +351 291 209410
Email: gabinetedareitoria@uma.pt