Universidade Federal de Santa Catarina
Departamento de Informática e Estatística

Uma Ferramenta de Suporte a Simulações de Redes com o ns-2

Trabalho de Conclusão de Curso submetido à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção de grau como
Bacharel em Sistemas de Informação

Adriano Orlando Campestrini

Florianópolis, Julho de 2005

Resumo do Trabalho de Conclusão de Curso submetido à
Universidade Federal de Santa Catarina
como parte dos requisitos para a obtenção de grau como
Bacharel em Sistemas de Informação

Uma Ferramenta de Suporte a Simulações de Redes com o ns-2

Adriano Orlando Campestrini

Julho 2005

Orientador: Roberto Alexandre Dias, Dr.
Co-orientador: Frank Siqueira, Dr.
Curso: Bacharelado em Sistema de Informação
Palavras-chave: NS4D, *Network Simulator*, Simulação de Redes de Computares, C.
Páginas: 71

Resumo

O ns-2 (*Network Simulator* versão 2) é uma ferramenta poderosa e flexível para realizar simulação de redes de computadores, entretanto apresenta algumas limitações e possui problemas de usabilidade. Como forma de suprir deficiências encontradas no ns-2, o presente trabalho propõe e desenvolve uma ferramenta de suporte a simulação de redes que torna mais amigável a utilização do ns-2: o NS4D (*Network Simulator for Dummies*). O NS4D atua basicamente como um gerador de scripts OTcl configurável. Como parte desta solução, foram desenvolvidos módulos para geração de carga de trabalho, configuração MPLS, incorporação de algoritmos de roteamento, análise estatística de resultados e transformações úteis. Um estudo de caso mostra que a utilização da ferramenta amplia as possibilidades e a produtividade dos usuários que a empregam, além de tornar possíveis tarefas inviáveis apenas com o ns-2.

Abstract of do Course Conclusion Works presents to
Universidade Federal de Santa Catarina
as a partial fulfillment of the requirements for the degree of
Bachelor in Systems of Information

A Network Simulations Support Tool with ns-2

Adriano Orlando Campestrini

July 2005

Advisor: Roberto Alexandre Dias, Dr.
Co-advisor: Frank Siqueira, Dr.
Course: Bacharelado em Sistema de Informação
Keywords: NS4D, *Network Simulator*, C.
Number of Pages: 71

Abstract

The ns-2 (Network Simulator version 2) is a powerful and flexible tool to build network simulations, however it presents some limitations and possess usability problems. As form to supply deficiencies found int the ns-2, this work proposes and develops a network simulation support tool that makes friendlier the use of ns-2: the NS4D (Network Simulator for Dummies). The NS4D acts basically like a changeable generator of scripts. As part of this solution, modules for work load generation, for MPLS configuration, routing algorithms, statistical analysis and useful transformations had been developed. A use case sample that the tool extends the users possibilities and productivity, besides becoming possible some tasks that were impracticable only with ns-2.

Agradecimentos

Agradeço especialmente aos meus pais por me educarem com tanto amor e à minha irmã Daniela pela presença, pelo apoio e pelo carinho. Amo vocês!

Sou muito grato também ao professor Rubens Carvalho, ao professor Roberto Dias e ao amigo Gerson Vaz que colaboraram significativamente com o meu crescimento profissional. Muito obrigado pela oportunidade de aprender com vocês!

Sumário

Capítulo 1

Introdução

A evolução tecnológica dos últimos tempos impulsiona um novo paradigma a aplicações na internet, principalmente aplicações multimídia e em tempo real, como vídeo sob demanda, videoconferência, voz sobre IP e computação em grupo de trabalho. Aliado a estas novas aplicações está o aumento da demanda por recursos de redes de computadores, exigidos para que o usuário final realize suas tarefas em rede, como se estivesse realizando-as localmente.

O emergente crescimento da demanda por infraestrutura e recursos de largura de banda traz novos desafios a projetistas e pesquisadores da área de redes de computadores. Tradicionalmente, as quatro principais técnicas para análise de desempenho de redes de computadores são *medição*, *experimentação*, *modelagem analítica* e *simulação por computador.*

Devido às diversas restrições e variações de comportamento em redes de computadores, como atraso de processamento, limitação de carga de trabalho e tolerância à falhas, os algoritmos de modelagem analítica tendem a ficar cada vez mais complexos e limitados, fugindo da realidade das redes reais. A estratégia de medições, por sua vez, é útil apenas para estudos com sistemas e protocolos já implantados, além de não fornecer recursos para controle do cenário medido. Da mesma forma, a prototipagem e experimentação de topologias de redes reais, com serviços configurados a agir de forma controlada, muitas vezes exigem recursos computacionais não disponíveis, seja pela complexidade da rede ou pela infraestrutura disponível para o pesquisador. Neste sentido, há um consenso entre projetistas e pesquisadores que a utilização de ferramentas de simulação é a mais completa e flexível abordagem para análise quantitativa de redes de computadores.

No contexto das simulações de redes por computador, o ns-2 (*Network Simulator* versão 2) se destaca como uma poderosa ferramenta que permite configurar e o monitorar cenários fictícios utilizando implementações de diversos algoritmos e serviços, principalmente, da arquitetura TCP/IP. Nesta ferramenta, as simulações são geradas em scripts na linguagem OTcl (uma linguagem orientada a objetos baseada em TCL - *Tool Command Language*) e a resposta do simulador é inserida em arquivos de log (*trace files*).

O presente trabalho visa fornecer aos usuários do ns-2 uma ferramenta que facilite a sua utilização, desde a criação dos scripts OTcl e a geração da carga de trabalho até a análise quantitativa do resultado das simulações. A ferramenta desenvolvida também tem como objetivo facilitar a criação de simulações utilizando alguns recursos avançados do ns-2, entre eles configurar explicitamente rotas fim a fim e utilizar protocolos de roteamento convencionais. Além disso, é objetivo da ferramenta estender algumas funcionalidades ao simulador, como: atraso de processamento nos nodos, interação do simulador com implementações isoladas de algoritmos de roteamento e algoritmos para mensuração de parâmetros de QoS (Qualidade de Serviço) dos ambientes simulados.

1.1 – Objetivos

O objetivo principal do presente trabalho é desenvolver uma ferramenta para facilitar simulações de redes usando o ns-2, tornando a interação com o simulador mais amigável para o usuário.

Os objetivos específicos são relacionados a seguir:

- Criação de uma ferramenta que facilite a definição de caminhos para fluxos de dados na rede;
- Criação de uma ferramenta que possibilite inserir atraso de processamento nos nodos de uma rede, que permita fazer reserva de recursos para fluxos específicos e agregações de fluxos com características semelhantes;
- Criação de uma ferramenta que possibilite o acoplamento do ns-2 com implementações isoladas de algoritmos de roteamento;
- Criação de uma ferramenta de geração de fluxos de dados em massa para as simulações no ns-2;
- Criação de ferramentas para análise de desempenho de simulações.

1.2 – Metodologia

A fim de atender os objetivos elencados anteriormente, o presente trabalho está organizado nas seguintes etapas:

- Levantamento bibliográfico sobre redes de computadores, simulação de redes de computadores e metodologias de desenvolvimento de software;
- Estudo das funcionalidades e comportamento dos simuladores de redes e análise de suas limitações;

- Projeto de desenvolvimento de uma ferramenta amigável de suporte a simulações de redes com o ns-2;

- Avaliação da utilização da ferramenta através de um estudo de caso.

1.3 - Resultados Esperados

Através deste trabalho espera-se desenvolver uma ferramenta que torne a utilização do ns-2 mais amigável ao usuário, no que diz respeito à criação pequenas simulações, à simulações utilizando recursos especiais do simulador e à obtenção de resultados quantitativos dos cenários simulados.

Capítulo 2

Fundamentos de Simulação de Redes de Computadores

Este capítulo apresenta brevemente os conceitos de redes de computares utilizados no decorrer do trabalho. Em seguida, apresenta-se o conceito de simulação de redes de computadores e a ferramenta ns-2 (*Network Simulator* versão 2). Finalmente, com o objetivo de justificar a adoção do ns-2, apresenta-se um estudo comparativo entre os principais simuladores de redes utilizados no mercado.

2.1 - Fundamentos de Rede de Computadores

A Internet é uma rede de computadores pública mundial na qual milhares de equipamentos estão conectados (Kurose 2003). Todos os equipamentos conectados a ela são chamados de *hosts*, ou sistemas finais. Os *hosts* se comunicam através de protocolos de envio e recebimento de informações. Dentre estes protocolos, o TCP (*Transmission Control Protocol*) e o IP (*Internet Protocol*) são os mais importantes e, por isto, a família de protocolos utilizados na Internet é comumente conhecida como TCP/IP. Os *hosts* são conectados entre si através de enlaces de comunicação e a velocidade de transmissão destes enlaces, medida em bits por segundo (bps), é chamada de largura de banda.

Aplicações de rede são “a razão de ser” das redes de computadores, inclusive da Internet. Apenas com a possibilidade de implementar aplicações distribuídas úteis é que surgiu a necessidade de projetar algoritmos, equipamentos e protocolos de rede. Grande parte de uma aplicação de rede abrange o tratamento do seu protocolo de comunicação. Dentro do conceito da Internet e, mais especificamente, das redes TCP/IP estes protocolos são os chamados protocolos da camada de aplicação.

A Internet e outras redes TCP/IP fornecem dois tipos de serviços às suas aplicações: um serviço orientado à conexão e outro não orientado à conexão (Tanenbaum 1996). O serviço orientado a conexão é implementado pelo TCP e garante que os dados transmitidos da origem ao destino sejam entregues em ordem e completos. O serviço da Internet não orientado à conexão é fornecido pelo UDP (*User Datagram Protocol*). O UDP não oferece nenhuma garantia quanto à entrega dos dados ao destinatário. Estes serviços são na verdade protocolos da camada de transporte.

A maioria das aplicações conhecidas na Internet utiliza o TCP, entre elas, as aplicações FTP (*File Transfer Protocol*) e HTTP (*Hiper Text Transfer Protocol*). O UDP, no entanto, é utilizado principalmente por aplicações multimídia, como VoIP (voz sobre IP), áudio e vídeo sob demanda e videoconferências.

O endereçamento da Internet segue um padrão hierárquico, onde *hosts* periféricos recebem endereços de menor nível e *hosts* centrais - que normalmente pertencem a grandes sub-redes conectadas a Internet - recebem endereços de maior hierarquia. O controle deste endereçamento é realizado através do IP. O IP é o único protocolo da camada de rede da Internet, daí o nome de *Internet Protocol*, ou protocolo da Internet. Na camada de rede os dados são transmitidos em pacotes desde a origem até o seu destino e cada um destes pacotes percorre enlaces e *hosts* intermediários. Estes *hosts* intermediários são chamados de roteadores e tem a tarefa de decidir o caminho que os pacotes irão percorrer até o seu destino.

A cada *host* intermediário existe um enlace também intermediário que interliga individualmente cada *host*. O acesso aos meios físicos de comunicação, como placas de redes e cabos de conexão, e a transmissão dos dados através dos enlaces individuais é tarefa da camada de enlace na arquitetura TCP/IP. Os protocolos desta camada definem as ações necessárias para enviar e receber dados de/para um *host* vizinho, bem como o formato das unidades de dados trocadas entre eles.

2.1.1 – Qualidade de Serviço - QoS

Atualmente, a Internet não fornece meios pelos quais se possa fazer garantias quanto ao tempo que se gastará para transportar os dados da origem ao destino. Da mesma forma, não é possível pagar mais para se obter um serviço privilegiado. Considera-se, portanto, que a Internet oferece um serviço de melhor esforço a todas as aplicações. Onde tudo é prioridade nada é prioridade. O serviço de melhor esforço, portanto, não se compromete com a qualidade do serviço oferecido.

Dentro do conceito de QoS (Quality of Service – qualidade de serviço), para medir a qualidade de um serviço existem alguns parâmetros que podem ser considerados:

- Perda de pacotes: quantidade de dados que são descartados ao longo do caminho em decorrência da limitação dos *buffers* dos roteadores;

- Vazão final: taxa de recebimento (em bps) dos dados no destinatário;

- Atraso fim a fim: tempo que o pacote leva para trafegar na rede desde a origem até o destino;

- Variação de atraso: variação do tempo decorrido desde que o pacote é gerado na fonte até o recebimento no destinatário.

Na área de redes de computadores há um grande esforço para se criar arquiteturas de redes que, no futuro, forneçam suporte explícito às exigências de serviços principalmente às aplicações multimídia. Ao invés de oferecer apenas uma classe de serviço de melhor esforço, arquiteturas de nova geração deverão fornecer classes de serviço com garantias de QoS. As propostas mais consolidadas no meio acadêmico são as arquiteturas Intserv (serviços integrados) e Diffserv (serviços diferenciados), entretanto diversas propostas variantes estão surgindo, inclusive mesclando vantagens de ambas (Xiao 1999).

Esforços em prol da melhor utilização dos recursos de redes e, como conseqüência, da QoS, também são realizados no estudo das áreas de roteamento e Engenharia de Tráfego, abordados nas próximas sessões.

2.1.2 – Roteamento

Para transferir os pacotes de um *host* remetente a um *host* destinatário, a camada de rede deve determinar o caminho ou a rota que os pacotes devem seguir. No coração de qualquer protocolo de roteamento está o algoritmo de roteamento. A tarefa dos algoritmos de roteamento é simples: dada uma série de roteadores conectados por enlaces, deve-se descobrir um caminho entre uma fonte e um destino. Esta tarefa pode ser complicada quando se espera do algoritmo de roteamento a solução de rotas de modo a aproveitar adequadamente, quando não de forma ótima, os recursos da rede.

Apenas dois tipos de algoritmos de roteamento são tipicamente usados na Internet: o algoritmo d*istance-vector* e o *link-state* (Perlman 2000). O algoritmo *distance-vector* se baseia em uma tabela de distâncias que informa a qual *host* vizinho deve-se encaminhar os pacotes de todos os destinatários da rede. Já no algoritmo *link-state*, o cômputo das rotas é baseado em um peso ou custo que pode ser definido para cada enlace da rede.

Completando a tarefa de roteamento da camada de rede, os protocolos de roteamento são as formas de comunicação dos roteadores para troca de informações de roteamento. Dentre as implementações de protocolos de roteamento mais utilizadas na Internet estão o RIP (*Routing Information Protocol*) e BGP (*Border Gateway Protocol*), que utilizam algoritmos *distance-vector*; além de OSPF (*Open Shortest Path First*) e IS-IS (*Intermediate System to Intermediate System*) que utilizam algoritmos de roteamento do tipo *link-state*.

2.1.3 – Engenharia de Tráfego

Com o crescimento da demanda por recursos de redes, apenas o aumento da capacidade dos enlaces não é suficiente. Do ponto de vista de QoS nem sempre o caminho mais curto é o melhor caminho a se trafegar por uma rede. Neste cenário, a ET (Engenharia de Tráfego) vem a contribuir com a evolução da tecnologia de redes de computadores. Em cenários com tendência a congestionamento, o resultado do emprego da ET é o mapeando os fluxos de tráfego de forma balancear a carga entre os componentes de uma infra-estrutura de rede.

O balanceamento de carga é uma das principais motivações para a implantação da ET. Balancear a carga de trabalho dentre os *hosts* de uma rede otimiza a utilização dos recursos, valoriza investimentos em infraestrutura, reduz os pontos de congestionamento e traz uma melhora significativa nos parâmetros de qualidade de serviço, pois reduz o índice de perdas de dados e a variação do atraso.

Os algoritmos de roteamento tradicionalmente encontrados em operação são algoritmos descentralizados, ou seja, possuem uma visão limitada da rede, apenas consideram seus vizinhos na resolução de rotas. Como não poderia ser diferente, rotas “não ótimas” comumente são escolhidas para um fluxo trafegar. Normalmente este problema ocorre pois as rotas escolhidas são as que apresentam menor número de hosts intermediários, mesmo que outras rotas estejam ociosas.

A seguinte sessão apresenta um protocolo a adoção de algoritmos de roteamento globais, ou seja, que tem o conhecimento completo da rede instalada e até dos fluxos em operação nela. No objetivo de alcançar um maior nível de balanceamento de carga, estes esquemas sofisticados de roteamento são baseados na capacidade de estabelecimento explícito de rotas fim a fim.

2.1.4 – MPLS

Os protocolos de roteamento atuais, como OSPF e IS-IS, utilizam protocolos de roteamento do tipo *link-state*, portanto não consideram banda máxima e banda residual dos enlaces ao computar o destino dos fluxos da rede. Apesar de estarem previstas extensões aos algoritmos *link-state*, tornando-os capacitados a tratar estes parâmetros, não é recomendado o uso de novos protocolos de rede, pois estes podem causar inundação (ou *flood*) de mensagens de sinalização.

Em ambientes de roteamento convencionais, a decisão sobre o roteamento de cada pacote é realizada em cada roteador da infra-estrutura. Para permitir o estabelecimento prévio da rota a ser seguida pelos pacotes o MPLS, *Multiprotocol Label Switching* (Callon 2001), insere rótulos nos pacotes que entram no domínio. Estes rótulos são associados a uma FEC (*Forward Equivalence Class*), que representa o caminho fim a fim que o pacote deverá percorrer na rede. Uma FEC é definida como sendo uma classe correspondente a um conjunto de pacotes encaminhados de uma mesma forma através da rede. Cada FEC define explicitamente os nodos que os pacotes de cada classe irá percorrer. Com as FEC’s é possível criar e manter tabelas de rótulos, ou LIB (*Label Information Base*), nos roteadores MPLS.

MPLS é denominado multiprotocolo, pois pode ser empregado de forma independente dos protocolos da camada de rede e enlace, por exemplo, ATM, *Frame Relay*, PPP com IP, IPX, *Appletalk*, etc. Eis alguns conceitos relacionados a MPLS:

- Rótulo: conjunto de informações resumido e de tamanho fixo utilizado para busca na tabela de encaminhamento;

- LIB: é a tabela presente em cada roteador MPLS (chamados de LSR – *Label Switch Routers*) que contém informações necessárias ao encaminhamento dos pacotes;

- LSP (*Label Switched Path*): é uma rota pré-definida na qual um conjunto de pacote de uma mesma FEC é encaminhado através de uma rede MPLS;

- Protocolos de distribuição de rótulos: Para que decisões de encaminhamento sejam definidas em um domínio MPLS, é necessário que informações sejam intercambiadas entre todos os LSR's vizinhos.

No âmbito da ET e dos algoritmos de roteamento, existe o CBR (*Constraint Based Routing)*. CBR é um conjunto de processos voltados ao cômputo de rotas de rede sujeito ao atendimento de uma ou mais restrições (regras). Estas regras normalmente são relacionadas às especificações de QoS das aplicações, informações de segurança e/ou tarifação. A partir disto, as rotas configuradas podem ser as que melhor atendem tais restrições.

Para possibilitar a configuração explícita de rotas fim-a-fim - requisito de CBR - é necessário utilizar protocolos de distribuição de rótulos MPLS. No mercado existem duas propostas: RSVP-TE (extensões para engenharia de tráfego ao *Resource Reservation Protocol*) e CR-LDP (*Constraint Rounting - Label Distribution Protocol*).

2.2 – Simulação de Redes de Computadores

Simulação é uma técnica muito utilizada para avaliação de desempenho de sistemas em geral. Quando o sistema a ser avaliado não está disponível, caso comum em muitas situações, uma simulação é o caminho mais fácil de prever o comportamento ou comparar soluções alternativas. No caso de simulações de redes de computadores (Bajaj 1999), existem quatro técnicas para análise de desempenho: (1) *medição*, (2) *experimentação*, (3) *modelagem analítica* e (4) *simulação por computador*.

Medição é uma técnica utilizada para compreender o comportamento vigente em um sistema. Ao realizar medições na Internet, por exemplo, é possível tentar inferir sobre o tamanho da Internet. É possível também obter informações sobre o volume de tráfego, protocolos em uso, tamanho de pacote, perda de pacotes e horários de pico. Esta técnica, entretanto, é uma forma passiva de observação. Não temos o controle sobre o que está sendo medido, tornando praticamente inviável a validação de novas tecnologias.

Aproveitando o mesmo exemplo, ao analisar o desempenho da Internet, apesar do sistema já estar implementado e funcionando, uma alteração num ponto crítico pode gerar resultados desagradáveis, o que torna impraticável a experimentação. Existe ainda a alternativa de realizar experimentos em ambientes menores e controlados, os chamados *testbeds*. Nestes casos, é possível ter maior controle e segurança sobre os elementos a serem estudados, porém os testbeds apenas imitam um ambiente real de forma muito limitada. Além disso, nem sempre é possível ter acesso a infraestrutura de redes, equipamentos e configurações de software.

A utilização de métodos analíticos, apesar de ser uma técnica barata e com o ambiente analisado sob controle, esbarra na complexidade da Internet. As simplificações são tantas que os algoritmos criados para entender o comportamento dos elementos da rede acabam fugindo da realidade. Desta forma, considera-se arriscado confiar em resultados obtidos por métodos analíticos.

Enquanto medição e experimentação exploram eventos reais, as simulações e a metodologia analítica se restringem a um modelo abstrato. A simulação diferencia-se da experimentação e da medição por permitir maior flexibilidade ao estudo e diferencia-se em relação aos métodos analíticos por permitir a construção de modelos mais complexos e representativos do mundo real. Logo, a realização das simulações é o melhor caminho para obter boas estimativas do comportamento do sistema após a possível modificação de algum elemento da rede.

Nenhuma das técnicas discutidas é realmente eficiente se a metodologia de pesquisa não expressar realmente o que está se querendo medir, experimentar, provar ou simular. Como as simulações são fruto de cenários programaticamente montados, qualquer pequena distorção nas configurações do comportamento da rede, pode produzir resultados totalmente diferentes do esperado. Alguns dos erros mais comuns na configuração de simulações são carga de trabalho ou topologia não representativa, escolha de métricas incorretas, simulações muito simplificadas e parâmetros incorretos devido ao nível exagerado de detalhes na simulação.

2.3 – O Network Simulator

Esta sessão apresenta o simulador de redes ns-2 (VINT 2005), o *Network Simulator* versão 2, que está sendo desenvolvido dentro do projeto VINT, por algumas universidades e por centros de pesquisa norte americanos. O ns-2 possui funcionalidades específicas para simular redes de computadores com diversos protocolos e serviços comuns na internet, o que faz dele uma poderosa ferramenta para configurar simulações complexas e também para comparação de resultados de pesquisas.

O ns-2 recebe constantes atualizações de seus mantenedores do projeto VINT, além de receber várias contribuições de pessoas e grupos de pesquisa de diferentes partes do mundo, inclusive o NS4D, do Brasil. O ns-2 abrange um grande número de aplicações, de protocolos, de tipos de redes, de elementos de redes e modelos de tráfego, inclusive a grande maioria dos protocolos e serviços existentes na internet, como o IP, variantes do TCP, UDP, FTP, HTTP e protocolos de roteamento.

Dada a dificuldade da obtenção e configuração de equipamentos adequados para o experimento de pesquisadores da área de redes, o ns-2 torna-se uma importante ferramenta para validar estudos neste segmento. O ns-2 também provê avaliações sucintas de topologias de redes comerciais. Neste sentido, consultores e projetistas de redes poderão usufruir do ns-2 em seus diagnósticos. Da mesma forma, os investimentos em redes corporativas poderão ser direcionados de forma mais segura e eficaz. A aplicação do *Network Simulador* também abrange sua utilização como guia no ensino de redes de computadores, utilizando o “animador de redes” como forma de tornar mais claro aos alunos o comportamento das redes de computadores e seus elementos.

Após a execução das simulações, o ns-2 pode apresentar seus resultados em vários arquivos de *trace* que, devidamente interpretados e processados, podem resultar em tabelas e gráficos. Junto com o ns-2 é distribuído um software para animação das simulações, o Nam (*Network Animator*), que pode ser executado após o termino das simulações para a sua visualização. O Nam utiliza um dos arquivos de *trace* gerados pelo ns-2 para apresentar a animação e tem como intuito dar um respaldo visual ao cenário simulado. A representação visual gerada pelo Nam apresenta a topologia (com nodos e enlaces), o envio de pacotes, seu alojamento nas possíveis filas e o descarte dos pacotes.

A figura 2.1 apresenta um fluxo normal de operação com o ns-2 e o Nam.

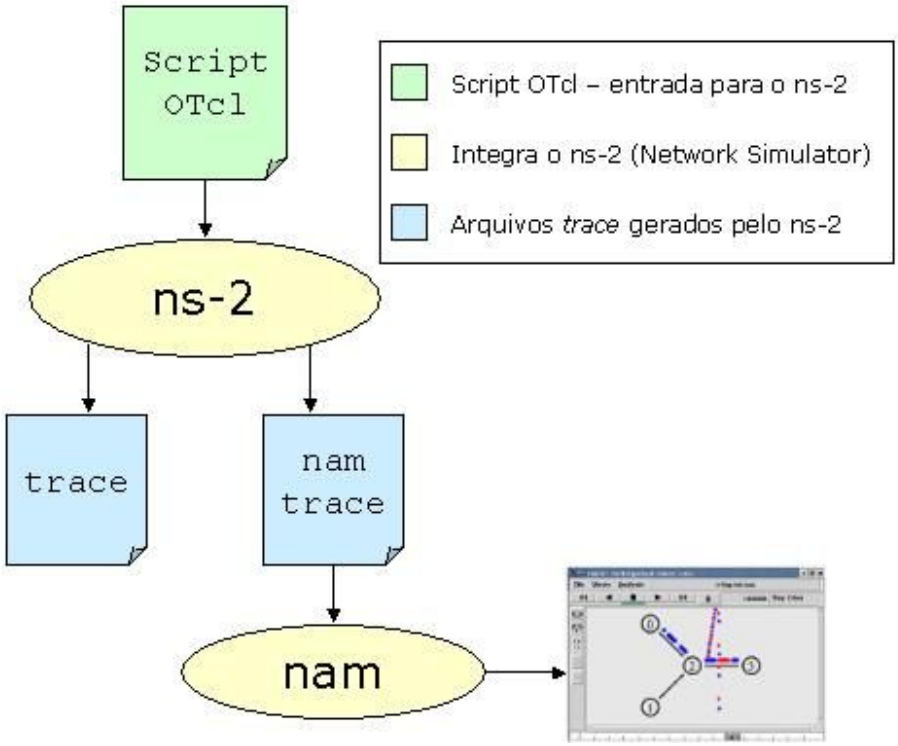

Figura 2.1: O ns-2 e o nam em funcionamento

O presente trabalho adotou como plataforma base para as simulações o ns-2, por este ser conhecido, desde suas primeiras versões, como o simulador de redes padrão da comunidade acadêmica em todo o mundo. Este título traz consigo diversas outras características, como a robustez, a confiabilidade dos resultados, as funcionalidades oferecidas e o fato de ser um software livre - com uma grande comunidade de usuários, listas de discussão e contribuidores espalhados pelo mundo.

2.3.1 – Arquitetura

O desenvolvimento deste simulador de redes com alto grau de flexibilidade resultou na escolha de uma arquitetura multi-linguagem empregada no simulador. O ns-2 funciona baseado em scripts OTcl (uma extensão orientada a objetos da linguagem TCL - *Tool Command Language*), porém seu núcleo foi desenvolvido em C++ e uma grande parcela dos módulos de suporte a tecnologias mais específicas em OTcl.

O ns-2 é um simulador de eventos discreto, no qual o avanço do tempo nas simulações depende da temporização do eventos que são gerenciados por um escalonador de eventos. Um evento é um objeto na arquitetura C++, com um identificador único, um tempo de escalonamento e um ponteiro para um objeto que trata o evento. O escalonador mantém uma

estrutura de dados ordenada (na verdade existem quatro opções de estruturas de dados, mas o ns-2 utiliza uma lista encadeada como padrão) com os eventos a serem executados e os dispara no tempo configurado chamando o objeto responsável por seu tratamento.

A instalação do ns-2 pode ser realizada de modo segmentada – com a instalação de cada módulo separadamente – ou a partir de um pacote chamado *all-in-one* (todos em um) que engloba módulos necessários para rodar o ns-2 e módulos adicionais freqüentemente usados em simulações. Dentre eles destacam-se: Tcl/Tk, TclCL, OTcl, NS, Nam e Xgraph.

2.3.2 - Interface OTcl

Apesar de não ser uma linguagem muito difundida comercialmente, TCL é a terceira linguagem de programação *open source* mais utilizada no mundo, atrás apenas de Perl e C/C++. Sua sintaxe possibilita a criação de programas inteiros na linguagem ou, como no caso do ns-2, roda embutida em softwares escritos em outras linguagens.

O ns-2 disponibiliza duas hierarquias de classes para o usuário realizar suas simulações: uma em C/C++ e outra em OTcl. A API (Aplication Programming Interface) em C++ possibilita ao usuário rodar simulações com mais eficiência e melhores tempos de execução, entretanto o uso desta API requer um estudo detalhado sobre a estrutura interna do simulador.

Apesar de OTcl ter sua própria API, os usuários do ns-2 acabam conhecendo apenas a API OTcl disponível pelo simulador, necessária para criar suas simulações. As classes em OTcl podem fazer uso das classes em C++ através de uma conexão entre as linguagens, implementada no pacote TclCL (uma interface TCL/C++). Para cada classe em C++ há uma correspondente em OTcl.

É inevitável ao usuário avançado do simulador, com o passar do tempo, deparar-se com a necessidade de simular cenários que, por algum motivo, não são possíveis de serem simulados com os recursos da API OTcl. Esta necessidade é a causa principal pelo interesse crescente no estudo do funcionamento interno do simulador. De forma análoga a este interesse, o núcleo do ns-2 e sua API em C++ são dotados de grande complexidade, dificultando o entendimento e a utilização até mesmo a usuários já iniciados. Por este motivo, o presente trabalho não atua internamente na arquitetura do ns-2, apenas complementa externamente suas interfaces.

2.3.3 – Componentes básicos

Eis alguns elementos comumente encontrados em simulações no ns-2:

- Nó ou nodo: são elementos que implementam a lógica associada aos computadores na rede, sejam eles *hosts* ou roteadores. Os nodos implementam protocolo IP, possuindo desta forma as funcionalidades da camada de rede da

arquitetura TCP/IP. Os nodos não implementam nenhum protocolo em outra camada, abstraindo o funcionamento de protocolos de camadas inferiores, como *Ethernet*, FDDI e ATM;

- Enlace: é o elemento que conecta dois nodos. O enlace é independente de implementação de algoritmo de transporte, sendo esta funcionalidade designada para os agentes;

- Agente: responsável por implementar protocolos da camada de transporte. Os dois principais agentes presentes no ns-2 implementam TCP e UDP, entretanto o simulador suporta vários outros agentes;

- Aplicação: responsável por gerar tráfego de dados para a simulação, através de alguns modelos de dados, como CBR (*Constant Bit Rate*), *Paretto*, Exponencial e *On-Off*. Este elemento implementa a camada de aplicação da arquitetura TCP/IP.

Outras ferramentas de simulação utilizam componentes e interfaces distintos, inclusive abstraindo as conceitos de rede de forma diferente do ns-2. A sessão seguinte apresenta uma avaliação dos, considerados, principais simuladores de redes de computadores.

2.4 – Análise de Simuladores de Redes de Computadores

O J-Sim (INDEX Group 2005) é um pacote desenvolvido pelo *Illinois Network Design and Experimentation* (INDEX) *Group,* com base na *University of Illinois,* implementado em uma arquitetura baseada em componentes relacionados à modelagem de protocolos da Internet. O J-Sim pode executar tanto de simulações discretas quanto simulações baseadas em processos com requisitos de tempo-real.

O SSF (SSF Research Network 2005), *Scalable Simulation Framework,* é uma API para simulação discreta de eventos, resultado do esforço colaborativo entre a Rutgers University, Dartmouth College, Georgia Tech e Boston University. Duas implementações independentes da API SSF são fornecidas: a DaSSF escrita em C++ e a JSSF escrita em Java.

Com o foco em redes de computadores, o SFFNet é uma coleção de componente baseados em SSF. Os componentes do SSFNet modelam e simulam protocolos de rede da arquitetura TCP/IP (sinônimo de Internet). As simulações com SSFNet são realizadas através de arquivos de configuração no formato DML, *Domain Modeling Language*, uma linguagem de alto-nível com a intenção de ser simples e de padronizar a configuração das simulações.

Como experimento para medir requisitos de escalabilidade, um cenário comum foi simulado nos quatro ambientes: ns-2, J-Sim, SSFNet em Java e o SSFNet em C++. Diversas rodadas de simulações foram realizadas (David 2002). A cada rodada mais fluxos foram injetados no cenário, aumentando assim a complexidade da simulação.

Os parâmetros utilizados para mensurar a escalabilidade foram: consumo de memória e tempo para resolução das simulações. Constatou-se o seguinte:

- O ns-2 é o mais rápido nos problemas que consegue resolver, entretanto o consumo de memória é elevado, tornando o crescimento do tempo de simulação exponencial à medida que a memória *swap* é utilizada;

- O J-Sim é em geral o mais lento, apesar de ter capacidade de resolver as simulações mais complexas em menos tempo e realizar pequenas simulações com baixo consumo de memória;

- A implementação do SSFNet em Java é a que menos consome memória, mas roda na metade da velocidade do ns-2;

- A implementação do SSFNet em C++ é a que mais demanda memória e tem velocidade semelhante ao ns-2.

Em contato com a equipe de desenvolvedores das ferramentas analisadas, concluiu-se que, nos casos do ns-2 e do SSFNet, a infraestrutura alocada para as simulações não varia muito de acordo com o modelo simulado, ou seja, a infraestrutura destes simuladores é projetada para suportar diversos modelos. No caso do J-Sim, por ter uma arquitetura de componentes menos acoplada, provê funcionalidades específicas para cada cenário simulado, consumindo menos memória principalmente em pequenos cenários. O ponto fraco do J-Sim é que, por ser menos acoplado, a comunicação entre seus componentes sacrifica o tempo de execução.

Capítulo 3

Uma Abordagem para Utilização Amigável do ns-2

Este capítulo tem como objetivo descrever a problemática do trabalho, incluindo a descrição do problema atacado e a justificativa da solução proposta. Além disso, é apresentado o ambiente utilizado para o desenvolvimento da solução.

3.1 – Definição do Problema

A utilização de uma interface adequada deve ser uma das principais preocupações no desenvolvimento de qualquer software que interaja com algum usuário. Da mesma forma, simuladores de redes com interfaces inadequadas podem gerar problemas relacionados a falhas no modelo simulado e até ao tempo de desenvolvimento de uma simulação.

As principais formas de interação com simuladores de redes são através de arquivos de configuração ou através de linguagens de programação. Arquivos de configuração podem ser mais simples de manipular, entretanto nem sempre espelham todas as funcionalidades disponíveis pelo simulador. Ainda no caso dos arquivos de configuração, estes deverão seguir uma sintaxe e semântica para serem interpretados, como no caso do SSFNet que funciona baseado em arquivos de configuração que seguem a sintaxe da linguagem DML. Esta característica faz com que interfaces baseadas em arquivos de configuração compartilhem grande parte das vantagens e desvantagens dos simuladores baseados em linguagens de programação.

A escolha de uma linguagem de programação como interface de entrada para o simulador pode agregar valor ou invalidar o esforço empregado na implementação das funcionalidades de fato. Seja ela uma linguagem específica para simulações ou uma linguagem

de propósito geral, como C e Java, o usuário terá total flexibilidade em desenvolver suas aplicações, podendo usufruir todo o potencial da linguagem para enriquecer suas simulações. Esta liberdade, entretanto, acarreta em uma onerosa curva de aprendizado a ser vencida e requer do usuário conhecimentos tanto da área de redes de computadores quanto da linguagem de programação. Além disso, linguagens de programação são mais um fator a ser considerado ao portar o simulador para outros sistemas operacionais.

O ns-2 adotou a OTcl como a linguagem para criar os scripts das simulações. Esta escolha resolve satisfatoriamente a questão da portabilidade, já que existem implementações de interpretadores OTcl para diversas plataformas. Quanto à curva de aprendizado, OTcl apenas ameniza este quesito, pois não é uma linguagem específica de simulação e sim de abrangência geral. Por outro lado, OTcl não deixa de ser uma linguagem de programação e seu aprendizado é um requisito para realizar as simulações.

A forma com que um software apresenta seus resultados é a sua interface de saída e a esta se aplicam requisitos semelhantes à interface de entrada. A interface de saída do ns-2 deve apresentar seus resultados de forma a satisfazer as necessidades e expectativas do usuário. Ao realizar uma simulação, o ns-2 possibilita a criação de uma animação a ser apresentada pelo Nam. Esta animação apresenta a movimentação dos pacotes sobre os enlaces e nodos da topologia, entretanto ignora a medição estatística de características da simulação. Apenas com as ferramentas disponíveis pelo ns-2, não é possível realizar análises de quesitos de QoS, como vazão, índice de perdas, atraso e variação de atraso (*jitter*).

Outra forma com que o ns-2 mostra seus resultados é através do arquivo *trace*, que contém o registro de todos os eventos ocorridos durante a simulação. Este arquivo de *trace*, entretanto, não possui nenhuma informação tratada, apenas dados relacionados a eventos como: a chegada de um pacote em um nodo, a saída de um pacotes de uma fila, etc. O ns-2 permite configurar *traces* específicos para uma fila ou para a simulação como um todo. Em ambas as situações, cada linha do arquivo representa um evento e segue o seguinte formato:

```
AÇÃO TEMPO ORG DST TIPO TAM FLAGS FID END_ORG END_DST SEQ PID
```

Listagem 3.1: Sintaxe do arquivo *trace* gerado pelo ns-2

A listagem 3.1 mostra em seqüência a denominação das colunas de cada linha no arquivo *trace*. Eis o significado dos dados presentes em cada uma destas colunas:

1. AÇÃO: Este é o primeiro campo e indica a ação, ou evento, que está ocorrendo. Cada evento é representado por um dos símbolos: *r*, +, -, *d* que significam, respectivamente, pacotes recebido, pacote entrou na fila, pacote saiu da fila e pacote descartado;

2. TEMPO: Este campo indica o momento em que ocorreu o evento, de acordo com o escalonador do simulador;

3. ORG: Este campo indica o nodo de origem do enlace onde ocorreu o evento;

4. DST: Este campo indica o nodo de destino do enlace onde ocorreu o evento;

5. TIPO: Indica o tipo de fluxo de dados a que pertence o pacote que realiza o evento;

6. TAM: Tamanho do pacote que realiza o evento;

7. FLAGS: Algumas *flags* de controle que indicam se o pacote é de reenvio ou um reconhecimento (*acknowledgment*) TCP, por exemplo;

8. FID: Número identificador do fluxo de dados (*flow id*);

9. END_ORG: Endereço de origem do fluxo de dados ao qual pertence o pacote que realiza o evento;

10. END_DST: Endereço de destino do fluxo de dados ao qual pertence o pacote que realiza o evento;

11. SEQ: Número seqüencial do pacote dentro do fluxo de dados;

12. PID: Identificador do pacote dentro da simulação (*packet id*).

Percebe-se que existe uma frustração do usuário final ao realizar uma simulação e ter com resposta um arquivo de milhares de linhas contendo diversos símbolos que só fazem sentido no ns-2. Este arquivo de *trace*, mesmo em pequenas simulações, costuma ter dezenas de *kilobytes*, chegando a *gibabytes* em simulações de grande porte.

As deficiências nas interfaces de entrada e saída do simulador são ainda mais perceptíveis quando da realização de simulações de grande porte. Nestes casos os scripts OTcl que descrevem a simulação chegam a ter mais de 25.000 (vinte e cinco mil) linhas, sendo que a partir de 300 linhas os scripts começam a se tornar complexos e de difícil manutenção. Outra limitação encontrada no ns-2 quando em simulações de grande porte é a capacidade de processamento. Como descrito na sessão 3.4 sobre comparação de simuladores, o ns-2 ocupa muito espaço na memória e quando não há mais memória disponível a simulação não pode ser realizada, sendo abortada no momento em que a memória se esgota.

Os diversos problemas apresentados com relação à forma de iteração do ns-2 com o usuário motivou uma avaliação do simulador seguindo critérios de usabilidade de software.

3.1.1 – Usabilidade

Segundo (Nielsen 1993), usabilidade é o conceito de "*qualidade de uma aplicação sob uma perspectiva de uso, tradicionalmente relacionado a cinco atributos: facilidade de aprendizado, eficiência, facilidade de reter o conhecimento sobre a aplicação obtido em usos*

anteriores (memorização), baixo índice de erros e satisfação do usuário". O processo de projetar e avaliar a usabilidade de um software se denomina *engenharia de usabilidade*. Os métodos de avaliação podem ser classificados das seguintes formas:

1. Testes com usuários: onde usuários reais testam a usabilidade das aplicações;

2. Inspeção: onde avaliadores especialistas conferem se uma aplicação provê um conjunto predefinido de princípios de projetos de interface.

Avaliação através de testes com usuário "*é o método de avaliação de usabilidade mais fundamental e é praticamente insubstituível, pois prevê informação direta sobre como as pessoas usam os computadores e quais são os problemas exatos dos usuários com a interface concreta que é testada*" de acordo com (Nielsen 1997). Os testes de usabilidade visam principalmente verificar, de forma quantitativa, o cumprimento dos princípios de usabilidade da aplicação. Neles, são estabelecidos fatores críticos e faixas de valores que determinam a usabilidade da aplicação. Em seguida, são realizados testes para verificar se os valores obtidos para cada fator crítico são ou não satisfatórios, e como resultado são fornecidas indicações sobre que aspectos devem ser modificados.

Com o objetivo de avaliar a usabilidade do ns-2 devem ser selecionados alguns fatores considerados críticos para o sucesso no uso de um software simulador de redes. Esta escolha de fatores críticos, entretanto, não pode ser levada adiante sem antes se identificar o público alvo desta categoria de software: os simuladores de redes. Esta identificação irá nos guiar na escolha e na avaliação dos fatores críticos.

O público alvo de um sistema são os usuários do sistema. O sistema em questão deve estar voltado a atender os objetivos destes usuários. Pode-se dizer que classes de pessoas interessadas em simular redes de computadores são pesquisadores e projetistas de redes. Estes profissionais são especialistas nas áreas em que atuam, neste caso em redes de computadores, portanto, para as demais áreas devemos considerá-los como leigos. Obviamente haverá usuários especialistas em diversos assuntos, entretanto devemos generalizar esta classe para abranger o maior porção possível dos usuários de simuladores. Estes usuários, ao interagir com simuladores de redes, estarão familiarizados apenas com conceitos de redes de computadores e conceitos gerais de informática (já que a área de redes de computadores é uma especialização dentro da informática).

Baseado no fato que simuladores de redes têm como público alvo especialistas em redes de computadores, podemos listar os seguintes fatores críticos aos simuladores:

- Acessibilidade: a instalação do ns-2 requer muitas bibliotecas de desenvolvimento, nem sempre disponíveis em muitas distribuições de Linux, por exemplo;

- Número de erros: o ns-2 possui uma extensa implementação de testes unitários e conhecidamente apresenta poucos erros;

- Aprendizado: por apresentar conceitos de redes de computadores embutidos na API de uma linguagem comercialmente pouco difundida, a OTcl, apresenta uma severa curva de aprendizado;

- Facilidade de uso: comandos fáceis executam o ns-2 pelo *Shell*, entretanto a confecção dos scripts OTcl compromete a produtividade;

- Satisfação subjetiva: as animações no Nam fornecem uma visão geral sobre o cenário simulado, entretanto os resultados apresentados pelo ns-2 não são quantitativamente relevantes;

- Desempenho: apesar de consumir bastante memória, seu tempo de processamento é baixo;

- Comunicabilidade: as interfaces de entrada e de saída não são intuitivas e podem não expressar corretamente a informação desejada;

- Controle sobre a atividade: o usuário nem sempre recebe do simulador um feedback de erros ou sucesso;

- Nível de configuração: o ns-2, por ser programado, é altamente configurável;

- Confiança de que o software executa o que é esperado: as animações do Nam previnem grande parte das distorções mais sensíveis;

- Conformidade com as expectativas do usuário: um ambiente de simulação deve ter pelo uma estratégia automatizada de geração de carga e apresentar resultados quantitativos, o que não é fornecido pelo ns-2;

- Autodescrição: exemplos e tutoriais são distribuídos junto com o ns-2, o que facilita a descrição do seu funcionamento.

Outra limitação é a falta de documentação adequada para iniciantes. Apesar dos scripts OTcl capacitarem a criação de redes com características próximas da realidade, o desenvolvimento de simples modelos requer o conhecimento da linguagem, que não é complexa, mas pouco conhecida. Na medida que se necessita desenvolver novos protocolos ou mecanismos não presentes no ns-2, deve-se ter não apenas intimidade com o desenvolvimento em C++, mas desvendar a estrutura de dados do ns-2 e seu esquema de hierarquias de classes para saber de onde novas classes podem ser derivadas e encaixadas, o que é uma tarefa para especialistas.

3.2 – Solução Proposta

Os problemas apresentados na sessão anterior, sobretudo com relação à usabilidade do ns-2, motivaram a criação de uma ferramenta que minimize ou elimine:

- A dificuldade de aprendizado e utilização do simulador;
- A complexidade de elaborar simulações de grande porte;
- A falta de clareza e objetividade dos resultados.

O presente trabalho propõe a criação de uma ferramenta que torne mais amigável a utilização do ns-2, especificamente as tarefas acima relacionadas.

Muitos projetos de pesquisa e desenvolvimento se iniciam dispostos a implementar novos módulos e funcionalidades ao simulador, mesmo sem ter realmente uma visão sistêmica dos processos que ocorrem no ns-2. Tendo este conhecimento, o presente trabalho atua sobre uma camada independente da estrutura interna do simulador, tornando a ferramenta proposta mais portável em relação às diferentes versões e releases do *Network Simulator*.

Devido à dificuldade no aprendizado da linguagem OTcl, a solução proposta atua como um gerador de scripts, tornando transparente a API do simulador. Para realizar esta tarefa, a ferramenta proposta deve possuir alguns mecanismos simples de configuração da simulação. A idéia é que a descrição de toda a simulação possa ser feita por de forma simples e objetiva. A descrição da simulação, portanto, pode ser divida em 3 (três) etapas:

- Definição da topologia física: descreve o cenário da simulação, abrangendo os nodos, enlaces e suas configurações, entre elas a largura de banda, a política de enfileiramento e o atraso inserido pelo enlace;
- Definição dos fluxos de dados: relação de todos os fluxos de dados que serão inseridos na simulação, incluindo detalhes como o protocolo do fluxo de dado, o tipo de distribuição da amostragem, o tamanho dos pacotes, a vazão inicial dos fluxos e endereços de origem e destino;
- Configurações gerais: configurações referentes à simulação em geral e à própria ferramenta proposta, como o protocolo de roteamento utilizado, o tempo de simulação, o nível de log do gerador, alguns tratamentos especiais e quais parâmetros se desejam analisar na simulação.

Outro problema encontrado no ns-2 é a dificuldade em criar simulações com grande porte. A limitação de capacidade de processamento só pode ser resolvida com uma modificação interna no ns-2, entretanto é possível atacar a complexidade da criação destas simulações.

Com uma estrutura de arquivos de interface, a criação de grandes simulações é simplificada, pois os arquivos de configurações e de topologia físicos normalmente são fáceis de elaborar. Desta forma, a complexidade fica apenas no arquivo de fluxo de dados. Para uma simulação de 500 fluxos de dados, apesar da geração automática do script OTcl, ainda assim deve-se configurar cada um dos fluxos de dados separadamente.

Pensando nisto, a solução proposta engloba um gerador de carga de trabalho. Este gerador de carga tira do usuário a responsabilidade de criar o arquivo de fluxos de dados. Através de outro arquivo de configurações, o gerador de carga cria automaticamente arquivos dos diversos fluxos de dados. O gerador de carga utiliza diversas configurações de parâmetros limites mínimo e máximo e gera valores aleatórios entre estes limites. Dentre as configurações do gerador de carga de trabalho estão os limites de prioridade mínimo e máximo de cada fluxo, os limites de vazão inicial, os limites de duração de cada fluxos, o tamanho dos pacotes, o tempo de simulação e a quantidade de fluxos.

De nada adianta ter a capacidade de criar boas simulações em pouco tempo sem conhecer a linguagem OTcl, se não for possível analisar os resultados desta simulação. Para uma boa análise quantitativa de parâmetros de QoS da simulação deve ser possível realizar medições tanto em escopo global da simulação quanto em medições de fluxos específicos. Como forma de possibilitar esta análise de parâmetros de QoS, a ferramenta proposta neste trabalho inclui scripts com algoritmos para análise de vazão (*throughput*), índice de perdas, atraso e variação de atraso (*jitter*).

Embasado no ambiente de simulação que esta ferramenta propicia, um professor pode mostrar na prática os elementos de redes de computadores em funcionamento, sem conhecer a linguagem OTcl e sem se preocupar com uma rede real. No ensino de conceitos de ET (Engenharia de Tráfego) pode-se mostrar o diferencial entre técnicas avançadas em relação a políticas de redes ou algoritmos de roteamento convencionais, contando ainda com a análise de parâmetros de QoS.

Outra categoria de profissional da informática que pode ser satisfeito é o desenvolvedor de algoritmos de roteamento. Normalmente a implementação de um algoritmo só é validada matematicamente ou através de experimentos que não abrangem as teorias de redes de computadores. A proposta é que a ferramenta desenvolvida tenha uma interface para comunicação com implementações externas de algoritmos de roteamento. Além dos algoritmos já fornecidos pelo ns-2 (DV, LS, ISIS e CR-LDP), qualquer outro algoritmo poderá se conectar ao ambiente de simulações, mesmo sem implementar nenhuma lógica de redes de computadores.

Esta proposta vem de encontro justamente aos quesitos de usabilidade que o ns-2 deixa a desejar. Todo o esforço empregado no ns-2 é reaproveitado, pois a ferramenta proposta atua de forma complementar no ambiente de simulação. Eis a relação dos quesitos de usabilidade atacados:

- Aprendizado: na proposta o usuário do simulador precisa conhecer apenas os conceitos de redes de computadores e não mais a API OTcl;

- Facilidade de uso: a definição da simulação em arquivos em forma de tabela aproveita as facilidades fornecidas pelo *Shell* e abstrai a confecção manual dos scripts OTcl;

- Satisfação subjetiva: ainda é possível realizar as animações no Nam, porém na proposta as rotinas de análise de desempenho complementam-nas com resultados quantitativos;

- Desempenho: o tempo de processamento não é afetado pela estrutura proposta, a não ser que se acoplem algoritmos de roteamento com baixo desempenho;

- Comunicabilidade: as interfaces de entrada e de saída são intuitivas e facilitam ao usuário expressar e receber corretamente a informação desejada;

- Número de funções: a configuração da proposta não poderá ter mais funções que a própria API OTcl;

- Nível de configuração: a princípio qualquer configuração pode ser colocada nos arquivos de definição de recursos ou no arquivo de configuração, desde que não de tornem muito extensos;

- Confiança de que o software executa o que é esperado: é possível visualizar os scripts OTcl e ainda pode-se visualizar as animações do Nam;

- Conformidade com as expectativas do usuário: o gerador de carga e os analisadores de desempenhos, principalmente, satisfazem as expectativas de usuários de simuladores de redes;

Com a proposta bem especificada e os riscos avaliados, pode-se iniciar o desenvolvimento da ferramenta. Para um melhor embasamento sobre como gerenciar o projeto de desenvolvimento do software, diversas metodologias foram estudas e algumas de suas práticas adotadas. O estudo e a escolha das práticas estão descritos na próxima sessão.

3.3 – Metodologia de Desenvolvimento

Antes de iniciar o desenvolvimento do software, é necessário realizar um estudo sobre as metodologias de desenvolvimento que possam vir a ser adotadas.

Code-and-fix foi um modelo básico proposto nos primórdios do desenvolvimento de software contendo dois passos: (1) escrever algum código e (2) debugar aquele código. A ordem dos passos, portanto, era codificar alguma coisa e só então pensar em requisitos, design, testes e manutenção.

Identificados problemas na metodologia *code-and-fix*, surgiu a proposta do modelo em estágios (*stagewise*). Este modelo pressupõe que o desenvolvimento de software ocorre em estágios bem definidos (planejamento, especificação, codificação, teste...). O modelo em cascata evoluiu da em estágios e tornou-se padrão na indústria de desenvolvimento realçando dois pontos:

1. Reconhecimento de *feedback* entre os estágios;

2. Incorporação de prototipagem no ciclo de desenvolvimento.

Algumas dificuldades mostraram que uma nova alternativa deveria ser proposta, apesar das inúmeras revisões e refinamentos no modelo em cascata.

A metodologia de desenvolvimento evolucionária consiste em incrementar o software ao longo dos estágios de desenvolvimentos, na medida em que se tem mais experiência operacional. Apesar de trazer a filosofia de desenvolvimento incremental, a codificação parece ficar sem rumo, semelhante ao *code-and-fix*, e o planejamento inicial é uma barreira para a produtividade, da mesma forma que no modelo em cascata.

A metodologia proposta a seguir foi a metodologia de transformação. Esta parte do princípio que é possível transformar uma especificação formal de software em um produto realmente capaz de atender os requisitos. Eis os passos desta metodologia:

1. Especificação formal;

2. Transformação automática da especificação em código fonte;

3. Um *looping* iterativo, se necessário, para prover melhor desempenho;

4. Exercício do produto resultante;

5. Um último *looping* iterativo para ajustar a especificação inicial de acordo com o resultado operacional.

Este modelo resolve diversos problemas de metodologias anteriores, porém ainda apresenta dificuldades, como por exemplo a transformação automática de especificação formal em código fonte, que só é possível em pequenos projetos e em áreas limitadas.

A seguir veio a metodologia de desenvolvimento espiral, que foi discutida por diversos anos baseada na experiência com modelos e refinamentos anteriores (Boehm 2000). O modelo espiral pode acomodar a maioria dos modelos anteriores como casos especiais. Este modelo reflete a idéia de que cada ciclo, ou iteração, da espiral realiza a mesma seqüência de passos para cada porção do produto ou nível de elaboração, partindo de conceitos abrangentes a codificação individual de cada programa. Cada ciclo da espiral começa com a identificação de:

1. Objetivos da porção do produto sendo elaborada;

2. Os meios alternativos de implementação desta porção;

3. As restrições impostas à aplicação das alternativas.

Na próxima etapa são identificadas fontes de risco significantes e formulado o custo para tratar estes riscos ao projeto. Mensurados os riscos, é possível, na etapa seguinte, elaborar um

protótipo abrangendo o principal risco dominante, como desempenho e interface gráfica, por exemplo. Se o protótipo for suficientemente operacional e robusto, pode ser utilizado como parâmetro para agir sobre o risco mensurado. Novos protótipos podem ser criados à medida que novos riscos são identificados. Se todos os riscos foram resolvidos através do esforço com os protótipos na etapa anterior, o próximo passo segue a convencional metodologia em cascata (concepção, análise de requisitos, design...).

O ciclo segue com uma etapa de validação e é finalizada com o planejamento do próximo ciclo. Esta etapa de planejamento é muito importante, pois abrange também revisão de tudo que foi desenvolvido no ciclo anterior e remanejamento do pessoal e recursos envolvidos. Cada novo ciclo de desenvolvimento é iniciado assim que é elaborada alguma hipótese de que alguma tarefa operacional pode ser melhorada com o emprego de algum software. Se em algum momento a hipótese falha (um risco vem à tona ou um software superior surge no mercado) o ciclo é abortado imediatamente. Caso contrário, o ciclo termina com a instalação do software novo ou modificado e a hipótese é testada de acordo com o resultado operacional alcançado.

As vantagens da metodologia espiral incluem:

- Foco inicial no reuso de softwares já existentes;
- Acomodação do preparo da evolução, do crescimento e das mudanças no produto nos ciclos seguintes;
- Provimento de mecanismos para incorporação de objetivos de qualidade no desenvolvimento do software;
- Foco inicial em eliminar os erros e as piores alternativas;
- Para cada gasto de recurso ou tempo do projeto é feita a pergunta: "Quanto é o suficiente?";
- Separação entre desenvolvimento e manutenção de software;
- Provimento de uma infra-estrutura viável para desenvolvimento de software e hardware.

Até a metodologia ser dominada pelo desenvolvedor, algumas dificuldades devem ser consideradas:

- Cumprimento do contrato de software. O mundo dos contratos de aquisição de software normalmente torna a especificação do software pobre. O que se tem feito é utilizar contratos flexíveis, tornando-os confortáveis para a equipe de desenvolvimento;
- Confiança na perícia da avaliação de risco. O modelo espiral aposta na habilidade do desenvolvedor de identificar eventuais riscos ao projeto. Um

desenvolvedor com pouca experiência, entretanto, tende a não identificar ou especificar de forma ruim estes riscos. O que se costuma fazer, é especificar e documentar o sistema da forma que pessoas fora do projeto, normalmente não-especialistas, possam também tentar identificar e solucionar os riscos do projeto;

- Investimento excessivo em especificação e documentação, em detrimento da rapidez nos resultados.

3.3.1 – A metodologia adotada

Recentemente, surgiu um forte movimento de apoio às metodologias de desenvolvimento ágil (Martin 2002). Estas metodologias procuram encontrar maneiras melhores de desenvolver software atacando, principalmente, a falta de agilidade das metodologias de desenvolvimento tradicionais. Neste sentido, as metodologias ágeis têm como lema valorizar:

- Indivíduos e interação mais que processos e ferramentas;
- Software em funcionamento mais que documentação abrangente;
- Colaboração com o cliente mais que negociação de contratos;
- Responder a mudanças mais que seguir um plano.

Dentre as metodologias ágeis, XP (*Extreme Programming*) é uma metodologia ágil para equipes pequenas e médias desenvolvendo software com requisitos vagos e em constante mudança. XP é baseada em mínimo de documentação e máximo de comunicação. Seu principal objetivo é reduzir o custo de modificações no software ao longo do ciclo de vida do projeto, possibilitando respostas rápidas ao usuário. Respostas rápidas significam que novas funcionalidades devem estar em funcionamento o mais rápido possível.

Os requisitos do presente projeto mudam na medida em que novas necessidades surgem ou novas possibilidades são visualizadas, não sendo possível realizar estimativas em longo prazo. Por este motivo, foi descartada a utilização integral de qualquer metodologia de desenvolvimento tradicional, com base em longas etapas de especificação. Foram adotadas, portanto, diversas práticas da metodologia XP (a listagem de práticas abaixo se refere a um suposto cliente, pois o presente projeto foi desenvolvido como parte de uma tese de doutorado - vide capítulo 4), entre elas:

- Iterações curtas: o software é entregue em pequenos *releases*. Com esta prática, o cliente obtém retorno o mais cedo possível e os riscos de migração são minimizados. Esta é a prática que melhor exprime o sentimento de XP sobre dar respostas rápidas às mudanças de requisitos;

- Cliente sempre disponível: o cliente está sempre disponível, não somente para auxiliar, mas para fazer parte da equipe resolvendo dúvidas, alterando o escopo de uma iteração, redefinindo prioridades e detectando possíveis erros o mais cedo possível;

- Jogo de planejamento: cada iteração é como se fosse uma etapa de um jogo. O objetivo deste jogo é colocar em produção as funcionalidades de maior valor/prioridade possível em menos tempo;

- Testes de aceitação: são critérios definidos pelo cliente para aceitação de software. Ao final de cada iteração um novo release, antes de ser lançado, deve passar por estes testes;

- Implementação simples: o software é implementado da forma mais simples possível para satisfazer as necessidades do cliente;

- Metáfora: o desenvolvedor se comunica com outros membros do projeto através de metáforas. A Metáfora é uma poderosa ferramenta de comunicação que elimina rapidamente a complexidade da idéia e constrói a informação para o interlocutor;

- *Stand Up Meeting*: reuniões rápidas no início de cada dia de trabalho;

- Otimizações por último: Tarefas de otimização de funcionalidades são realizadas apenas quando sobra tempo, a não ser que façam parte dos requisitos dos testes de aceitação.

A cada início de iteração, diversas tarefas são selecionadas e ordenadas por prioridade. As iterações não devem passar de duas semanas e deve-se ter o controle sobre o tempo estimado e o realizado de cada tarefa. Ao final de uma iteração, as práticas de XP possibilitam ter novas funcionalidades prontas para o uso. Está resposta rápida à demanda do usuário ocorre sem grandes perdas de tempo em planejamento ou documentação de software - que nem sempre é bem aproveitada.

Para um bom andamento do projeto, a adoção de práticas de metodologias de desenvolvimento deve ser complementada com a utilização de ferramentas que automatizam tarefas rotineiras no desenvolvimento de software. Neste sentido, a questão do ambiente de desenvolvimento é abordada na sessão seguinte.

3.4 – Ambiente de Desenvolvimento

O desenvolvimento do presente projeto foi realizado utilizando um computador Pentium IV de 1.2 GHz e 256 MBytes de memória RAM. Sobre este hardware, foi instalada a distribuição Mandrake (versão 8.2) do sistema operacional Linux. Além disso, outras tecnologias e ferramentas foram empregadas neste projeto. Este ambiente de desenvolvimento é descrito em detalhes nas próximas sessões.

3.4.1 – A Linguagem C e o compilador GCC

A linguagem C (Schildt 1997) é uma linguagem de alto nível, genérica. Foi desenvolvida por programadores para programadores tendo como meta características de flexibilidade e portabilidade. C é uma linguagem que nasceu juntamente com o advento da teoria de linguagem estruturada e do computador pessoal e, assim, tornou-se rapidamente uma linguagem "popular" entre os programadores. O C foi usado para desenvolver diversos sistemas operacionais, entre eles o UNIX e o Linux, e hoje esta sendo usada para desenvolver novas linguagens, por exemplo, as linguagens C++ e Java.

Programas em C são compilados, gerando programas executáveis. Dentre os diversos compiladores disponíveis para esta tarefa, o compilador utilizado no presente trabalho foi o GCC (GNU *Compiler Collection*). GCC é um conjunto de compiladores de linguagens de programação produzido pelo projeto GNU. O GCC é um software livre, tem seu código fonte aberto e é distribuído pela FSF (*Free Software Foundation*). É o compilador padrão para sistemas operativos UNIX e Linux e certos sistemas operativos derivados tais como o Mac OS X. Originalmente chamado de GNU C *Compiler*, por só trabalhar com a linguagem de programação C, foi mais tarde extendido para compilar C++, Fortran, Ada, Java, Objective-C e outros.

3.4.2 – O Linux e o padrão POSIX

O Linux é um sistema operacional criado em 1991 por Linus Torvalds na universidade de Helsinki na Finlândia. É um sistema operacional de código aberto distribuído gratuitamente pela Internet. Seu código fonte é liberado como software livre. O aviso de *copyright* do kernel, ou

núcleo, feito por Linus descreve detalhadamente isto e mesmo ele está proibido de fazer a comercialização do sistema.

O Linux segue o padrão POSIX, que é o mesmo usado por sistemas Unix e seus variantes. Assim, aprendendo o Linux não se encontrará muita dificuldade em operar um sistema do tipo Unix, FreeBSD, HPUX ou SunOS, bastando apenas aprender alguns detalhes encontrados em cada sistema. O código fonte aberto permite que qualquer pessoa veja como o sistema funciona, corrija algum problema ou faça alguma sugestão para sua melhoria (Mitchell 2001). Estes são alguns dos motivos do seu rápido crescimento, da sua popularidade, do aumento da compatibilidade de periféricos e de sua estabilidade.

POSIX (IEEE 2004), *Portable Operating System Interface*, não é um simples padrão, mas uma família de padrões sendo desenvolvidos pela IEEE (*Institute* for Electrical and Electronics Engineers). O padrão POSIX também é adotado por outras entidades padronizadoras internacionais, como as ISO (*International Organization for Standardization*) e IEC (*International Electrotechnical Commission*), chamadas de ISO/IEC.

O primeiro dos padrões POSIX foi elaborado em 1988 e especificou a interface da linguagem C em sistemas baseados no kernel Unix cobrindo as seguintes áreas: primitivas de processos (comandos fork, `exec`, sinais e temporizadores), o ambiente de um processo (identificadores de usuário, grupos de processos), funções de entrada e saída, a base de dados do sistema (senhas, permissões e grupos) e os formatos de arquivamento `tar` e `cpio`. Desde então, o padrão POSIX sofreu diversas atualizações, numa freqüência média de quase uma nova versão por ano. Atualmente o padrão POSIX abrange, além das áreas cobertas pela versão inicial, as seguintes áreas: Shell, utilitários (desde `awk` e `basename` até `vi` e `yacc`), extensões tempo-real, sincronização de arquivos, entrada e saída assíncrona, semáforos, gerenciamento de memória, escalonamento, *clocks* e temporizadores, filas de mensagem, suporte a *threads* e a sua API.

A grande maioria dos sistemas baseada em Unix está conforme alguma versão do POSIX e a quantidade de sistemas fora deste padrão vem diminuindo. As principais diferenças ainda existem em relação à administração de sistemas, uma área na qual o POSIX ainda vem evoluindo para também abrangê-la.

3.4.3 – Linguagem AWK

AWK é uma linguagem de programação interpretada que surgiu em 1977 nos Laboratórios Bell da AT&T. O nome da linguagem é proveniente das iniciais dos sobrenomes de seus criadores, Alfred V. Aho, Peter J. Weinberger e Brian W. Kernigham.

Em casos onde há necessidade de se tratar informações contidas em arquivos textos onde estas obedecem algum padrão, a linguagem AWK é uma boa escolha. Em situações como esta, pode-se necessitar incluir alguma informação neste arquivo, ou extrair apenas parte dela para ser aproveitada em outros arquivos ou para emissão de relatórios. Fazer um programa de computador que execute essa tarefa em uma linguagem de programação como C, por exemplo, seria muito trabalhoso. Isso não acontece quando se faz um programa desses em

AWK. Com AWK pode-se manipular pequenas bases de dados em arquivos, validar dados e realizar experimentos com algoritmos que poderão ser utilizados posteriormente em outras linguagens de programação.

```
padrão_1 { ação_1 }
padrão_2 { ação_2 }
padrão_n { ação_n }
```

Listagem 3.2: Forma geral de um script AWK

A listagem 3.2 apresenta a estrutura geral de um script AWK. Na listagem, os padrões são condições lógicas e as ações são comandos em AWK. Para cada linha do arquivo processado é verificado se algum padrão se corresponde. Para estes casos a ação correspondente é processada. Caso não seja especificado o padrão, a ação será executada em todas as linhas. Os padrões especiais “BEGIN” e “END” indicam, respectivamente, que a ação será executada no início do processamento do arquivo e no final do processamento do arquivo.

3.4.4 – Biblioteca spConfig

A API dos sistemas desenvolvidos sob o padrão IEEE POSIX define, portanto, a interface pela qual os aplicativos poderão acessar recursos do sistema operacional. A API POSIX é bastante abrangente e nos casos mais usuais pode-se considerá-la simples, entretanto alguns recursos não convencionais são difíceis de implementar ou necessitam de bibliotecas complementares.

No âmbito do presente projeto foi verificada a necessidade de utilização de um destes recursos não contemplados pela API: a manipulação de arquivos XML (Deitel 2001), e*Xtensible Markup Language*. A solução encontrada foi a utilização da *spConfig*, uma biblioteca de manipulação de arquivos de configuração com o intuito de permitir externalizar configurações do sistema em um arquivo com sintaxe amigável, semelhante a XML. Além de elementos e atributos comumente encontrados em um documento XML, a spConfig também permite comentários no estilo Shell, ou seja, iniciando pelo caractere ‘#’.

3.4.5 – IDE Anjuta

Para gerenciar tantas características heterogêneas do projeto e facilitar operações comuns na tarefa de desenvolvimento de software, foi adotada a IDE Anjuta. O conceito de IDE (*Integrated Development Environment*) vem sendo recentemente adotado na linguagem C e significa que todo o projeto pode ser realizado em um único ambiente.

Anjuta é um versátil ambiente de desenvolvimento integrado para C e C++, que roda em ambientes GNOME e GTK. O Anjuta é desenvolvido e mantido pelo *SourceForge* (SourceForge 2005), um site de gerenciamento de projetos de código fonte aberto (o*pen source*). Oferece diversos recursos que não são encontrados sequer em IDE’s comerciais. Entre os recursos oferecidos pelo Anjuta estão: um editor de códigos, gerenciador de projetos, *debugger*

interativo, gerenciador de *deployer* e suporte a C/C++, Java, Perl e Pascal. Nas comunidades Linux e Unix é comum se utilizar simples editores de texto para escrever código em C e C++, entretanto o Anjuta organiza e projeta uma visualização global interessante para programas de médio e grande porte.

3.4.6 – Electric Fence

Desenvolver software com qualidade requisita mais do que apenas conhecer os recursos da linguagem e o funcionamento do sistema operacional, deve-se também utilizar recursos para encontrar erros em tempo de execução, como uso ilegal de memória dinamicamente alocada e determinar que partes do programa estão tomando mais tempo de execução.

No desenvolvimento de um software normalmente não se sabe quanta memória o programa irá consumir quando rodar. Por exemplo, a leitura de uma linha de arquivo em tempo de execução pode conter qualquer comprimento finito. A linguagem C utiliza as chamadas de sistema `malloc` e `free` (e suas variantes) para alocar memória dinamicamente enquanto o programa executa. Existem, entretanto, regras a serem seguidas para correta utilização do recurso de alocação de memória dinâmica:

- O número de chamadas a `malloc` deve ser exatamente igual ao número de chamadas a `free`;
- Leituras e escritas na memória alocada devem acontecer dentro do alcance da porção de memória alocada;
- A memória não pode ser usada depois de desalocada.

Ferramentas de checagem de memória realizam a tarefa de identificar se uma destas regras é violada. Dentre as diversas ferramentas de checagem de memória (*malloc checking*, `mtrace`, `ccmalloc`) a ferramenta que satisfaz o maior número de checagens é a *Electric Fence* (Mitchell 2001). Esta ferramenta foi escolhida para checagem de memória no presente trabalho.

A *Electric Fence* interrompe a execução do programa toda a vez que uma escrita ou leitura é realizada fora das regras de alocação de memória. Esta é a única ferramenta desta categoria capaz de identificar leituras ilegais. Para esta checagem, o software deve ser ligado à biblioteca `libefence` durante a compilação. A utilização de um debugger, como o gdb ou o debugger do Anjuta, pode identificar a linha exata onde o acesso ilegal ocorreu. A *Electric Fence* diagnostica acessos ilegais à memória gravando cada alocação em pelo menos duas páginas de memória. As alocações de memória são colocadas ao final da primeira página. Qualquer acesso depois do final da alocação, ou seja, na segunda página, causa uma falha de segmentação interrompendo o programa. Como são alocadas duas páginas de memória a cada chamada ao `malloc`, a *Electric Fence* utiliza muita memória, sendo utilizada somente para *debugging*.

Capítulo 4

NS4D – Network Simulator for Dummies

A proposta descrita no capítulo anterior tornou-se realidade com o desenvolvimento do *Network Simulator for Dummies*, o NS4D. O presente capítulo tem como objetivo descrever as funcionalidades e a forma como cada solução foi projetada e implementada nesta ferramenta.

A concepção do NS4D como ele se encontra atualmente, entretanto, não se deu de uma hora para outra. Várias iterações foram necessárias e nelas diversas versões e *releases* foram testados e utilizados até a próxima versão ser disponibilizada.

O desenvolvimento do NS4D foi motivado inicialmente pelas atividades do Projeto UCER (Uso Controlado e Eficiente de Recursos de Redes de Computadores), uma parceria a UFSC e a empresa W2B. Ao final do projeto UCER, em meados de 2003, foi iniciada uma segunda etapa de desenvolvimento. Esta etapa foi realizada no CEFET-SC (Centro Educacional Tecnológico de Santa Catarina) e as novas funcionalidades implementadas na ferramenta atenderam ao seu principal cliente: a tese de doutorado do professor Roberto Alexandre Dias (Dias 2004b). Esta parceria da ferramenta com o trabalho científico da tese renderam ainda algumas publicações (Dias 2003a) (Dias 2003b) (Dias 2003c) (Dias 2004a).

A primeira versão do NS4D foi lançada em meados de 2002 e apelidada de Penta, como homenagem a então eminente conquista do futebol brasileiro: o pentacampeonato mundial de futebol. Esta versão gerava scripts OTcl simples e possuía poucas facilidades de configuração. O Penta chegou a ser integrado com um sistema web para realização de simulações via internet.

Rapidamente novas idéias e novos módulos surgiram. A versão 2 do NS4D, chamada de Hexa, já abrangia um arquivo de configuração global, arquivos de definição de topologia e de carga de trabalho. Diversos *releases* do Hexa foram lançados, cada uma com uma nova carac-

terística e, é claro, correções. Dentre as funcionalidades presentes em *releases* mais completos do Hexa estavam os scripts AWK do módulo de análise de desempenho, o módulo de configuração MPLS e o módulo Plus, contendo algumas transformações controladas na simulação.

Além do módulo gerador de carga, o grande salto para o lançamento da versão 3 do NS4D - esta chamada apenas de NS4D - foi o módulo de roteamento, possibilitando a integração estática e dinâmica com implementações externas de algoritmos de roteamento. A figura 4.1 abaixo demonstra a estrutura interna do NS4D:

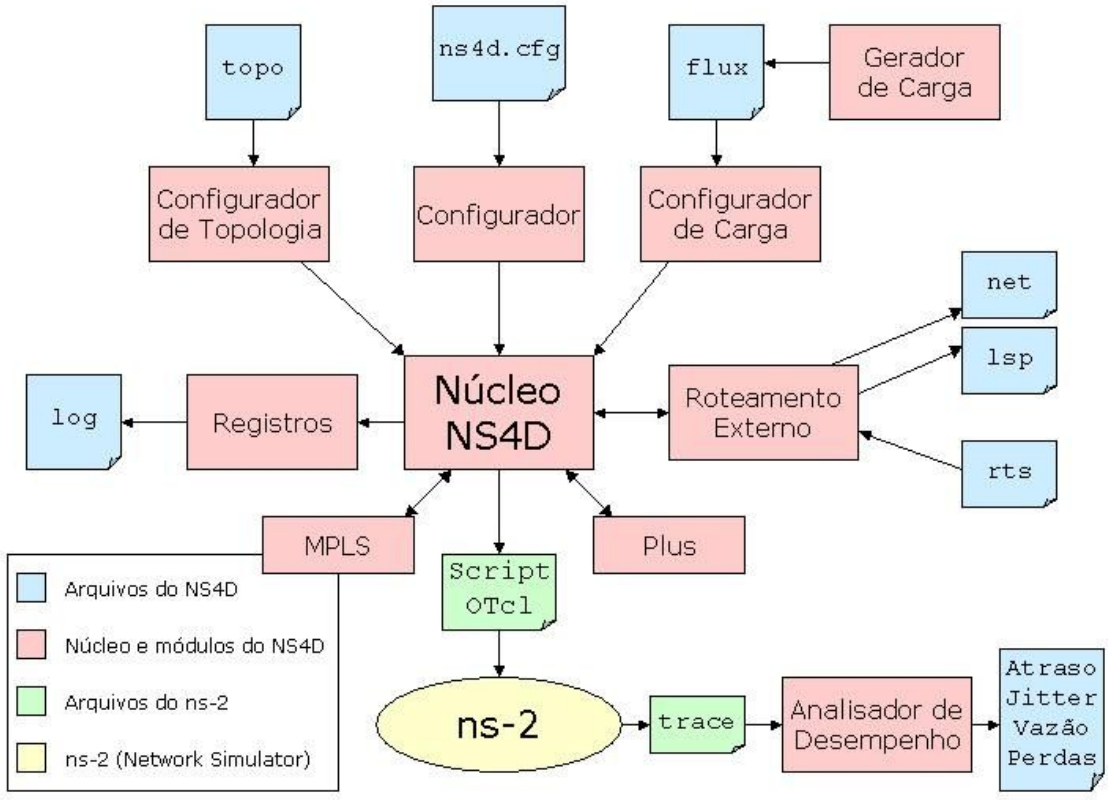

Figura 4.1: Estrutura interna do NS4D, incluindo núcleo, módulos e arquivos utilizados

As sessões seguintes descrevem em detalhes cada um dos módulos presentes na figura 4.1 e na versão atual do NS4D.

4.1 – O Núcleo: Gerador de Scripts OTcl

A primeira e mais importante etapa do projeto do NS4D foi o desenvolvimento do núcleo do programa. Este núcleo é o responsável pela tarefa mais importante do NS4D: gerar os

scripts OTcl. Esta funcionalidade de gerar scripts adicionada a alguns módulos de configuração já são suficientes para satisfazer cerca de 90% das necessidades dos usuários iniciantes no ns-2.

O núcleo detém o domínio do fluxo de execução. Qualquer outro módulo acoplado ao núcleo será invocado por ele, caracterizando o chamado princípio de Hollywood: “Não ligue para nós. Nós ligaremos para você!”. Para executar o núcleo, ou o NS4D, basta executar na linha de comando “`ns4d <ID>`”, onde ID é um número escolhido pelo usuário que será usado para identificação dos eventos relacionados a esta simulação. O ID da simulação possibilita executar diversas simulações em um mesmo diretório organizadamente.

Para gerar os scripts OTcl, alguns comandos dos que sempre se repetem foram automatizados pelo NS4D. Dentre eles a instanciação do objeto gerente do ns-2 e o comando de iniciar do escalonador. Em outros comandos OTcl variam apenas os valores informados, como a definição do algoritmo de roteamento, a habilitação dos arquivos de *trace* do ns-2 e configuração do tempo de simulação.

```
set ns [new Simulator]

#-----------------TRACE -- ARQUIVOS DE SAÍDA-----------------
set tf [open simulacao.tr w]
$ns trace-all $tf

set nf [open simulacao.nam w]
$ns namtrace-all $nf

#---------------------------CORES-----------------------------
$ns color 0 Black
$ns color 01 DarkBlue
$ns color 02 DarkRed
$ns color 03 Blue
$ns color 04 Green
$ns color 05 Purple
$ns color 06 Red
$ns color 07 Orange
$ns color 08 DarkGreen
$ns color 09 Yellow

#-------------------------ROTEAMENTO-------------------------
$ns rtproto LS
```

Listagem 4.1: Trecho de script OTcl comumente encontrado no início das simulações

```
#---------------------------FINISH---------------------------
proc fim {} {
 global ns tf
 $ns flush-trace
 close $nf
 close $tf
 exec nam simulacao.nam &
 exit 0
```

```
}
$ns at 20.0 "fim"

#-------------------------ANOTAÇÕES-------------------------
$ns at 0.2 "$ns trace-annotate Gerado\\ pelo\\ NS4D(v3)"
$ns at 0.2 "$ns trace-annotate Adriano\\ Orlando\\ Campestrini"

#---------------------------RODAR---------------------------
$ns run
```

Listagem 4.2: Trecho de script OTcl comumente encontrado no final das simulações

As listagens 4.1 e 4.2 apresentam os trechos inicias e finais dos scripts gerados pelo NS4D. Neste exemplo, o simulador é configurado para criar os arquivos de *trace* e *namtrace*. Com o intuito de deixar as animações do Nam mais agradáveis, algumas cores são definidas para, posteriormente, serem associadas a nodos e fluxos. Além disso, a simulação é programada para utilizar o protocolo de roteamento LS e ter 20 segundos de intervalo.

A próxima sessão explana sobre o módulo de configurações e apresenta as configurações necessárias para rodar pequenas simulações e satisfazer as configurações iniciais do NS4D.

4.2 – Módulo de Configuração

Para todos os módulos e também para o núcleo do sistema, existe a necessidade de disponibilizar alguma forma de configuração. Com exceção da configuração da topologia e da carga de trabalho, qualquer outra configuração global da simulação é feita através de um arquivo de configuração chamado de “ns4d<ID>.cfg”, onde ID é número inteiro passado como parâmetro na chamada do NS4D. Este arquivo deve ser colocado no diretório onde se deseja montar o ambiente para a simulação.

Para alguns módulos, certas configurações adicionais devem ser realizadas, e serão discutidas nas próximas sessões. Em pequenas simulações, basta definir um arquivo de configurações simples.

```
<CFG_GERAIS>
  <ALGORITMO
    ## Define o algoritmo da simulação
    #  "0" = Static - Distance Vector sem sinalização
    #  "1" = DV     - Distance Vector com sinalização - RIP
    #  "2" = LS     - Link State - OSPF
    #  "3" = ISIS   - Link State - IS-IS
    ativo=2
  />
  <DURACAO
    ## Define limites para duração dos fluxos e da simulação
```

```
        tempo_simulacao=20 #20s
        ## Escala de tempo
        escala=1
      />
      <REGISTROS
        ## Define o nome do arquivo de registro
        arquivo_log=log
      />
      <TOPOLOGIA
        ## Define o número de enlaces da topologia
        n_links=31
      />
      <CARGA_DE_TRABALHO
        ## Define o número de fluxos
        n_fluxos=15
      />
    </CFG_GERAIS>
```

Listagem 4.3: Exemplo de arquivo de configuração “ns4d.cfg” mínimo

A listagem 4.3 mostra o arquivo de configurações mínimo para o NS4D. Nota-se que alguns comentários estão inseridos no arquivo, iniciando com o caractere ‘#’. Qualquer trecho iniciando em ‘#’ e finalizando em uma quebra de linha é ignorado pelo módulo de configuração. Para analisar léxica e sintaticamente este tipo de arquivo, o NS4D utilizou a API da biblioteca spConfig.

Para entender como a spConfig trabalha com os arquivos de configurações, faz-se uma simples analogia com a hierarquia de um sistema de arquivos. Arquivos de configuração são agrupados em uma estrutura tipo árvore, composta por nodos ou elementos (análogos a diretórios), atributos (arquivos) e texto (os dados). Além disso, comentários podem ser adicionados à árvore de configuração. Com exceção dos comentários, a sintaxe é semelhante à sintaxe de XML. Como prova disto, o analisador sintático da spConfig é capaz de reconhecer a grande maioria dos arquivos XML.

Analisando a semântica do arquivo “ns4d.cfg”, existem os seguintes de elementos e valores:

- `CFG_GERAIS/ALGORITMO/ativo`: define o algoritmo de roteamento utilizado na simulação. Os algoritmos testados com o NS4D foram: Static, DV, LS e ISIS. O módulo de roteamento adiciona a este atributo mais valores que serão vistos adiante em detalhes;

- `CFG_GERAIS/DURACAO/tempo_simulacao`: define o tempo total da simulação. Além disso, se houver algum fluxo configurado com tempo final após o fim da simulação, o tempo final do fluxo é ajustado para o tempo da simulação configurado;

- `CFG_GERAIS/DURACAO/escala`: define o fator de divisão para o tempo da simulação. Por exemplo, para tempo_simulacao=5 e escala=1, o tempo simulado no ns-2 será de 5s; para tempo_simulacao=5000 e escala=1000 o tempo simulado no ns-2 também será de 5s; e tempo_simulacao=100 e escala=10 o tempo real simulado será de 10s.

- `CFG_GERAIS/REGISTROS/arquivo_log`: define o nome do arquivo de registro, em detalhes na sessão do módulo de registro de logs;

- `CFG_GERAIS/TOPOLOGIA/n_links`: define o número de enlaces da simulação, em detalhes na sessão do módulo de configuração de topologia;

- `CFG_GERAIS/CARGA_DE_TRABALHO/n_fluxos`: define a quantidade de fluxos de dados a serem injetados no simulador, em detalhes na sessão do módulo de configuração de carga.

O arquivo “ns4d.cfg” é o principal arquivo de configurações do NS4D e é elaborado tão intuitivamente quanto uma simples descrição da simulação desejada. As configurações apresentadas nesta sessão são suficientes para satisfazer 90% das necessidades dos usuários. A sintaxe simples e já conhecida de XML, aliada à flexibilidade do analisador sintático da spConfig, transformam positivamente a interface com o simulador. Estas facilidades prevêem um significativo aumento de produtividade ao realizar simulações no ns-2.

4.3 – Módulo de Registro de Logs

A tarefa de gravar informações sobre o comportamento da execução de um programa é a melhor forma para o usuário entender o seu funcionamento. Um dos grandes problemas no ns-2 é a falta de feedback ao usuário justamente quanto ao andamento da execução. Ao realizar uma simulação, em caso de sucesso o ns-2 não apresenta ao usuário nenhum tipo de informação e, em caso de erro, nem sempre as mensagens são significativas.

O nome arquivo de log é configurado no “ns4d.cfg”, entretanto, como todos os outros arquivos do NS4D, seu nome é concatenado ao ID fornecido na linha de comando ao executar a ferramenta. Este log pode servir para realizar auditorias nas simulações e também é essencial para o módulo de Roteamento no modo dinâmico, visto mais adiante.

O módulo de registro de log do NS4D registra quais nodos estão sendo configurados, quais enlaces entre são criados e quais fluxos são injetados na rede. Também é fornecida ao usuário, como forma de controle, todas as configurações extraídas pelo NS4D do arquivo de

configuração. Em simulações complexas ou computadores antigos, as simulações podem ser lentas e, nestes casos, o acompanhamento do arquivo de log é útil como *feedback* ao usuário sobre a execução da ferramenta. Erros de configuração podem ser rapidamente identificados através de registros no arquivo de log, por exemplo:

```
ERRO: /CFG_GERAIS/DURACAO/tempo_simulacao nao encontrado.
ERRO: Algoritmo "XYZ" desconhecido.
```

Listagem 4.4: Exemplos de erros apresentados no arquivo de log

Os erros apresentados na listagem 4.4 indicam, respectivamente, que o atributo tempo_simulacao não foi encontrado no arquivo de configurações, e que o algoritmo roteamento configurado não é suportado pelo NS4D.

Os módulos de configuração e de registro são essências para o funcionamento do NS4D, entretanto nada pode ser feito sem a definição do cenário e dos fluxos injetados na simulação. As próximas sessões respondem por estas funcionalidades.

4.4 – Módulo de Configuração de Topologia

Nenhuma rede de computador é simulada sem ter um cenário definido. Nenhum fluxo de dados pode trafegar através de uma rede sem esta rede possuir nodos e enlaces. Como não poderia ser diferente, as simulações realizadas pelo ns-2 devem obrigatoriamente definir estes elementos da simulação.

```
set primeiro_nodo [$ns node]
set segundo_nodo  [$ns node]
set terceiro_nodo [$ns node]
```

Listagem 4.5: Trecho de script OTcl contendo a declaração de nodos

A listagem 4.5 mostra como os nodos são definidos na API OTcl do ns-2. Nela são definidos 3 (três) nodos com os nomes “primeiro_nodo”, “segundo_nodo” e “terceiro_nodo”. Pode-se dizer que, cada um deles, é instanciado através da classe node.

Analogamente a uma rede real de computadores, os nodos, ou roteadores, devem ser conectados através de enlaces. No ns-2, os enlaces são encarregados de traçar uma rota entre um nodos origem e um nodo destino seguindo alguns parâmetros:

```
$ns duplex-link  $primeiro_nodo $segundo_nodo   500Kb    1s  DropTail
$ns simplex-link $segundo_nodo  $terceiro_nodo    2Mb   50ms SFQ
$ns simplex-link $terceiro_nodo $primeiro_nodo   10Mb    2ms RED
```

Listagem 4.6: Trecho de script OTcl contendo a declaração de enlaces

Observando a listagem 4.6 podemos identificar alguns novos conceitos. Cada linha representa um enlace inserido na simulação. Algumas variações foram propositalmente inseridas para podermos analisar algumas possibilidades de configuração de enlaces. Para cada enlace há definições de: tipo (bidirecional ou unidirecional), nodos de origem e destino, largura de banda do enlace, atraso do enlace e tipo de enfileiramento.

Enlaces unidirecionais (*simplex*) conectam os nodos apenas do sentido origem/destino, enquanto, enlaces bidirecionais (*full-duplex*) podem ser comparados a dois enlaces unidirecionais em sentidos opostos: origem/destino e destino/origem. As definições de largura de banda e de atraso do enlace seguem o padrão de e valor concatenado à métrica. A largura de banda aceita valores com as métricas Kb (o mesmo que Kbps), Mb (Mbps) e Gb (Gbps). O atraso dos enlaces aceita valores com as métricas s (segundo), ms (milisegundo) e us (microsegundo).

Nos enlaces, as filas são lugares onde os pacotes devem permanecer antes de serem processados e encaminhados ou simplesmente descartados. As filas simulam o comportamento de um roteador real dentro da rede. O termo usado para referir-se ao processo de decisão usado para escolher quais pacotes devem ser processados ou descartados é "escalonamento de pacotes", no ns-2 "enfileiramento". No caso do enfileiramento do tipo *droptail*, os pacotes são descartados usando a disciplina FIFO (*First In First Out*), na qual o primeiro a chegar é o primeiro a sair, e se a fila enfrentar problemas de falta de espaço para armazenamento de pacotes, os pacotes do fim da fila serão descartados. Outros tipos de enfileiramento implementados no ns-2 são WFQ (*Weigth, Fair Queueing*), SFQ (*Stochastic Fair Queueing*) e RED (*Randon Early Detect*). A WFQ e SFQ implementam um tratamento considerado justo no descarte de pacote, entretanto o diferencial da WFQ é a possibilidade de definir pesos, ou importância, para cada fluxo de dados. O enfileiramento do tipo RED detecta que a quantidade de pacotes na fila tende a ultrapassar seu limite e previamente descarta alguns pacotes, ativando o controle de congestionamento dos fluxos de dados TCP.

Esta etapa de definição dos nodos e enlaces, no NS4D, é encapsulada pelo módulo de configuração de topologia. Para isto, as configurações realizadas no script OTcl são transferidas para um arquivo de definição de topologia, chamado de "topo<ID>", onde ID é número inteiro passado como parâmetro na chamado do NS4D. O mesmo cenário criado nas listagens anteriores é representado da seguinte forma no arquivo "topo":

```
Origem  Destino  Banda       Atraso      Fila        Tipo
0       1        500Kb        1s         DropTail    duplex-link
1       2          2Mb       50ms        SFQ         simplex-link
2       0         10Mb        2ms        RED         simplex-link
```

Listagem 4.7: Exemplo de arquivo "topo"

O arquivo "topo" segue um formato baseado em linhas e colunas. Cada linha representa um enlace e possui suas configurações e nodos associados dispostos nas colunas. O NS4D sempre ignora a primeira linha deste arquivo, possibilitando a inserção de um cabeçalho contendo informações sobre o conteúdo de cada coluna. As linhas podem ser separadas no padrão Unix – através do caractere '\n' - ou pelo padrão da internet – através da seqüência de caracteres '\r\n'. As colunas podem ser separadas por um ou mais espaços ou o caractere '\t' – correspondente à tecla <TAB>.

Como visto no módulo de configuração, no arquivo "ns4d.cfg" existem o atributo `CFG_GERAIS/TOPOLOGIA/n_links`. Este atributo normalmente coincide com o número de linhas do arquivo "topo", entretanto existe a possibilidade do usuário não utilizar todos os enlaces definidos no arquivo, apenas os primeiros `n_links`.

Resumidamente, a tarefa do módulo de configuração de topologia é ler o arquivo “topo” e popular em memória as estruturas de dados do NS4D. Estas estruturas são utilizadas em todo o restante da ferramenta. Caso haja a necessidade de utilizar outro formato para os arquivos de definição de topologia, por exemplo, XML, ou até mesmo carregar a topologia de outra forma, por exemplo, via *WebService* ou RPC (*Remote Procedure Call*), basta substituir a implementação deste módulo de configuração de topologia. Este desacoplamento preserva a integridade e o esforço empregado no desenvolvimento do restante do sistema, bastando reimplementar um pedaço específico do NS4D e recompilá-lo.

Este desacoplamento vale também para o módulo de configuração de carga, descrito na sessão seguinte.

4.5 – Módulo de Configuração de Carga

A simulação de redes sem a injeção de fluxos de dados é tão inexpressiva quanto à realização de medições em uma rede real com todos os seus computadores desligados. Um fluxo de dados é um conjunto seqüencial de pacotes que possuem a mesma origem e o mesmo destino e pertencem a uma mesma conexão da camada de transporte na arquitetura TCP/IP.

A injeção de fluxos caracteriza a configuração da carga de trabalho da simulação. Os fluxos de dados serão os alvos das análises de desempenho e é o grande beneficiário das possíveis melhorias trazidas pelas simulações. Aplicações multimídia, conferências e a navegação pela internet configuram um ambiente bastante heterogêneo. O ns-2 apresenta robustez e flexibilidade ao lidar com este ambiente heterogêneo, possibilitando diversas combinações entre Agentes e Aplicações.

```
set agente_tcp [new Agent/TCP]
$ns attach-agent $primeiro_nodo $agente_tcp

set agente_receptor [new Agent/TCPSink]
$ns attach-agent $segundo_nodo $agente_receptor

$ns connect $agente_tcp $agente_receptor
```

Listagem 4.8: Trecho de script OTcl contendo a declaração de Agentes TCP

A listagem 4.8 acima apresenta a definição de dois agentes: um transmissor e outro receptor. O agente transmissor foi chamado de “agente_tcp” e é uma instância de Agent/TCP. Este agente é responsável por gerar um fluxo de dados TCP com origem no nodo associado a ele, o “primeiro_nodo“. O chamado “agente_receptor”, associado ao nodo “segundo_nodo”, tem a responsabilidade de receber o fluxo e, no caso de transporte TCP, responder adequadamente ao transmissor com *acknowledgments*. Na última linha, o script OTcl informa ao ns-2 que este transmissor e o receptor formam uma dupla, através do comando `$connect`.

Feita a conexão da camada de transporte, deve-se ainda inserir uma aplicação para efetivar a transmissão do fluxo de dados:

```
set aplicacao_ftp [new Application/FTP]
```

```
$aplicacao_ftp set packetSize_ 210
$aplicacao_ftp attach-agent $agente_tcp

$ns at 7.0 "$aplicacao_ftp start"
$ns at 20.0 "$aplicacao_ftp stop"
```

Listagem 4.9: Trecho de script OTcl contendo a definição de um fluxo de dados

A listagem 4.9 apresenta o trecho de script OTcl necessário para ativar uma aplicação sobre a conexão de agentes previamente instanciados. A aplicação foi chamada de "aplicacao_ftp" e o escalonador do simulador foi programado para rodar a aplicação do 7º (sétimo) ao 20º (vigésimo) segundo contados desde o início da simulação. Além disso, o tamanho dos pacotes deste fluxo de dados foi configurado na 2º linha para 210 bytes.

Todos estes passos são substituídos no NS4D por apenas uma linha em um arquivo de configuração de carga. Este arquivo foi chamado de "flux<ID>", onde ID é número inteiro passado como parâmetro na chamada do NS4D. O arquivo "flux", da mesma forma que o arquivo "topo", é disposto em forma de tabela, onde cada linha representa um fluxo de dados e cada coluna representa um atributo dos fluxos de dados. Eis a configuração do fluxo de dados demonstrado no formato do arquivo "flux":

```
FID Origem Dest Cliente Início Fim Tam Atraso Setup Hold Nível
    1   4     0   1       7     2  210 0     0     0    0
```

Listagem 4.10: Exemplo de arquivo "flux"

Na primeira linha da listagem 4.10, o NS4D novamente possibilita ao usuário criar um cabeçalho com informações sobre o conteúdo semântico das colunas. Com apenas uma linha o mesmo fluxo é configurado. Neste caso, algumas colunas não são significativas e serão aproveitadas pelos módulos de Engenharia de Tráfego com MPLS e de Roteamento. A colunas significativas para este módulo são as seguintes: identificador do fluxo (cabeçalho `FID`), origem (`Origem`), destino (`Dest`), inicio do fluxo (`Início`), final do fluxo (`Fim`) e tamanho dos pacotes (`Tam`).

Este exemplo demonstrou uma aplicação do tipo FTP sobre agentes TCP e TCPSink, entretanto, outros agentes e outras aplicações poderiam ter sido utilizadas. O NS4D suporta os agentes transmissores TCP e UDP, e as aplicações FTP, CBR, *Paretto*, Exponencial e outras que também não requerem nenhuma configuração especial.

Como visto no módulo de configuração, no arquivo "ns4d.cfg" existe o atributo `CFG_GERAIS/CARGA_DE_TRABALHO/n_fluxos`. Este atributo normalmente coincide com o número de linhas do arquivo "flux", entretanto existe a possibilidade do usuário não utilizar todos os fluxos de dados definidos no arquivo, apenas os primeiros `n_fluxos`.

A tarefa do módulo de configuração de carga, de forma resumida, é interpretar o arquivo "flux" e popular em memória as estruturas de dados referentes aos fluxos de dados do NS4D. Semelhante ao módulo de configuração de topologia descrito na sessão anterior, em caso de troca do módulo de configuração de carga (motivado pelo emprego de outro formato de arquivo "flux", por exemplo) a integridade e o esforço empregado na implementação do restante do sistema é preservada.

Com os módulos descritos até esta sessão, incluindo o núcleo, já é possível realizar simulações. Todas as etapas para a criação de um script OTcl já foram satisfeitas. A figura 4.2 mostra o NS4D em funcionamento com a sua estrutura mais básica:

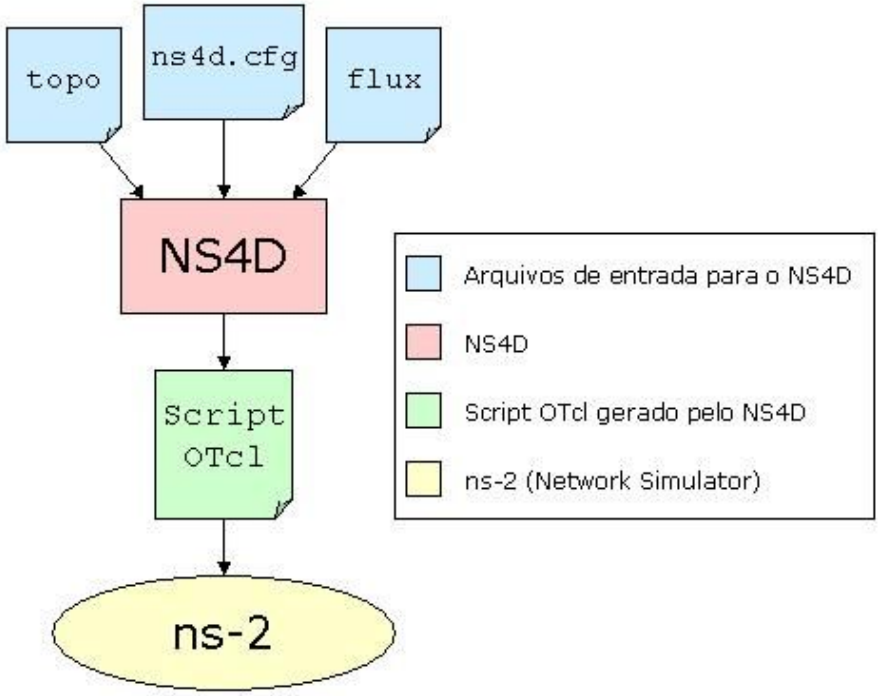

Figura 4.2: O NS4D em funcionamento

A partir desta sessão, apenas módulo implementando funcionalidades periféricas são descritos. Um destes módulos é o gerador de carga de trabalho, que apesar de ser opcional no ambiente de simulações, desempenha um papel essencial quando há a necessidade de gerar médios ou grandes conjuntos de fluxos.

4.6 – Módulo de Geração de Carga de Trabalho

O aumento da demanda por recursos de redes intensifica a expansão das redes de computadores e gera cenários cada vez maiores a serem simulados. Na medida que a largura de banda dos enlaces e os nodos da rede aumentam, a quantidade de fluxos de dados necessária para saturá-la aumenta exponencialmente.

No intuito de facilitar ainda mais a criação de grandes quantidades de fluxos de dados, o NS4D possui um gerador de carga de trabalho. Este gerador é distribuído como uma ferramenta separada do núcleo do NS4D, portanto deve ser executada manualmente. O gerador de carga de trabalho, a partir de um arquivo de configurações, gera de forma aleatoriamente controlada um arquivo “flux” com os fluxos de dados, como mostra a figura 4.3, abaixo:

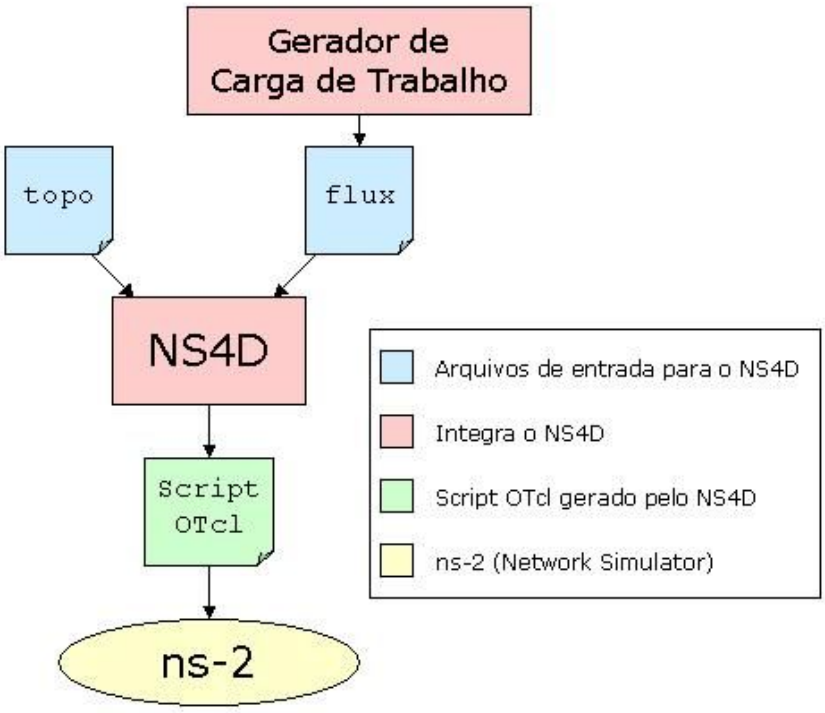

Figura 4.3: Módulo Gerador de Carga de Trabalho

A biblioteca spConfig também foi utilizada como interpretador das configurações do gerador de carga de trabalho. Por este motivo, o arquivo de configuração, chamado de "gera_carga.cfg", segue o padrão do arquivo "ns4d.cfg".

A listagem 4.11 apresenta um exemplo completo de arquivo de configuração do gerador de carga de trabalho. Este arquivo de configurações utiliza os conceitos de prioridade de *setup* e *holding*, de requisitos de atraso e níveis de admissão que serão abordados na descrição do módulo de roteamento. Prioridades *setup* e *holding* significam, respectivamente, qual a prioridade do fluxo para entrar na rede e qual a prioridade do fluxo para se manter na rede (Banerjee 2002).

Eis a descrição semântica dos atributos deste arquivo:

- `CFG_GERAIS/PRIORIDADES/max_alto_setup`: define a máxima prioridade de *setup* para fluxos de alta prioridade;

- `CFG_GERAIS/PRIORIDADES/min_alto_setup`: define a mínima prioridade de *setup* para fluxos de alta prioridade;

- `CFG_GERAIS/PRIORIDADES/max_baixo_setup`: define a máxima prioridade de *setup* para fluxos de baixa prioridade;

- `CFG_GERAIS/PRIORIDADES/min_baixo_setup`: define a mínima prioridade de *setup* para fluxos de baixa prioridade;

- `CFG_GERAIS/PRIORIDADES/razao_setup_holding_alta_prio`: define quantas vezes maior será a prioridade *holding* em relação à prioridade *setup* dos fluxos de alta prioridade;

- `CFG_GERAIS/PRIORIDADES/razao_setup_holding_baixa_prio`: define quantas vezes maior será a prioridade *holding* em relação à prioridade *setup* dos fluxos de baixa prioridade;

```
<CFG_GERAIS>
  <PRIORIDADES
    ## Define limites para a prioridade de setup
    max_alto_setup=7
    min_alto_setup=5
    max_baixo_setup=4
    min_baixo_setup=1
    ## Define a razão entre as prioridades setup e holding
    # holding = razao * setup
    razao_setup_holding_alta_prio=2
    razao_setup_holding_baixa_prio=2
  />
  <LARGURA_DE_BANDA
    ## Define limites para requisitos de largura de banda
    max_bw_alta_prioridade=880
    min_bw_alta_prioridade=380
    max_bw_baixa_prioridade=160
    min_bw_baixa_prioridade=20
  />
  <ATRASO
    ## Define limites para requisitos de atraso
    max_delay_alta_prioridade=100
    min_delay_alta_prioridade=70
    max_delay_baixa_prioridade=150
    min_delay_baixa_prioridade=100
  />
  <DURACAO
    ## Define limites para a duração dos fluxos
    max_duracao_alta_prioridade=2400 #40m
    min_duracao_alta_prioridade=300  #5m
    max_duracao_baixa_prioridade=900 #15m
    min_duracao_baixa_prioridade=5   #5s
    ## Define o tempo da simulação em segundos
    tempo_simulacao=18000            #5h
  />
  <TOPOLOGIA
    ## Define quantidade de nodos de acesso
    total_nodos=8
    ## Define o tamanho dos pacotes
    tam_pacote=210
```

```
        ## Define o número de níveis de admissão
        n_niveis=7
      />
      <CARGA_DE_TRABALHO
        ## Percentual de fluxos priorizados
        # Máximo de duas casas decimais
        porcentagem_priorizados=1.5
        media_fluxos_simultaneos=1300
        media_fluxos_por_cliente=40
      />
    </CFG_GERAIS>
```

Listagem 4.11: Exemplo de arquivo de configuração do gerador de carga

- `CFG_GERAIS/LARGURA_DE_BANDA/max_bw_alta_prioridade`: define a máxima largura de banda dos fluxos de alta prioridade, em Kbps;

- `CFG_GERAIS/LARGURA_DE_BANDA/min_bw_alta_prioridade`: define a mínima largura de banda dos fluxos de alta prioridade, em Kbps;

- `CFG_GERAIS/LARGURA_DE_BANDA/max_bw_baixa_prioridade`: define a máxima largura de banda dos fluxos de baixa prioridade, em Kbps;

- `CFG_GERAIS/LARGURA_DE_BANDA/min_bw_baixa_prioridade`: define a mínima largura de banda dos fluxos de baixa prioridade, em Kbps;

- `CFG_GERAIS/ATRASO/max_delay_alta_prioridade`: define o máximo requisito de atraso para os fluxos de alta prioridade, em milisegundos;

- `CFG_GERAIS/ATRASO/min_delay_alta_prioridade`: define o mínimo requisito de atraso para os fluxos de alta prioridade, em milisegundos;

- `CFG_GERAIS/ATRASO/max_delay_baixa_prioridade`: define o máximo requisito de atraso para os fluxos de baixa prioridade, em milisegundos;

- `CFG_GERAIS/ATRASO/min_delay_baixa_prioridade`: define o mínimo requisito de atraso para os fluxos de baixa prioridade, em milisegundos;

- `CFG_GERAIS/DURACAO/max_duracao_alta_prioridade`: define a máxima duração dos fluxos de alta prioridade, em segundos;

- `CFG_GERAIS/DURACAO/min_duracao_alta_prioridade`: define a mínima duração dos fluxos de alta prioridade, em segundos;

- `CFG_GERAIS/DURACAO/max_duracao_baixa_prioridade`: define a máxima duração dos fluxos de baixa prioridade, em segundos;

- `CFG_GERAIS/DURACAO/min_duracao_baixa_prioridade`: define a mínima duração dos fluxos de baixa prioridade, em segundos;

- `CFG_GERAIS/DURACAO/tempo_simulacao`: define o tempo total de simulação, em segundos;

- `CFG_GERAIS/TOPOLOGIA/total_nodos`: define a quantidade de nodos que podem gerar fluxos de dados;

- `CFG_GERAIS/TOPOLOGIA/tam_pacote`: define o tamanho dos pacotes;

- `CFG_GERAIS/TOPOLOGIA/n_niveis`: define o número de níveis de admissão;

- `CFG_GERAIS/CARGA_DE_TRABALHO/porcentagem_priorizados`: define o percentual de fluxos de dados considerados de alta prioridade;

- `CFG_GERAIS/CARGA_DE_TRABALHO/media_fluxos_simultaneos`: define a média de fluxos de dados presentes simultaneamente na simulação;

- `CFG_GERAIS/CARGA_DE_TRABALHO/media_fluxos_por_cliente`: define a média de fluxos de dados por cliente.

Com estas configurações, o gerador de carga está habilitado a criar um arquivo “flux” no formato reconhecido pelo núcleo do NS4D. Nota-se que para valores como requisito de atraso, largura de banda, duração e prioridades existem sempre definidos valores máximos e mínimos. O gerador de carga define estes valores através de uma escolha aleatória dentro destes intervalos. Os nodos origem e destino sempre vão de 0 (zero) até o valor definido em `total_nodos`. Como o gerador de carga de trabalho não gera nenhum arquivo “topo”, deve-se ter o cuidado de manter os nodos que geram fluxos de dados sempre entre os primeiros declarados no arquivo “topo”.

Para simular melhor o tráfego real na internet, o gerador de carga de trabalho divide os fluxos de dados em duas categorias: os fluxos de baixa prioridade e os fluxos de alta prioridade. Os fluxos de baixa prioridade são os mais comumente encontrados na internet. São transmissões de dados via conexão TCP, normalmente *downloads* de arquivos e navegação na internet. Os fluxos de alta prioridade são aqueles que necessitam melhores condições e tratamento do sistema autônomo (domínio), normalmente transmissões de vídeo, rádio ou até tele-conferências.

A proporção entre a quantidade de fluxos de alta prioridade e a de baixa prioridade fica a carga da configuração do parâmetro `porcentagem_priorizados`. Para cada 100 fluxos, a quantidade definida em `porcentagem_priorizados` é de fluxos de alta prioridade. Recomenda-se, como forma de se proximidade com a realidade, que os fluxos de alta prioridade não passem de 10% do total de fluxos, já que transmissões de vídeo e conferências compõem uma pequena minoria dos fluxos injetados na rede. Para outros parâmetros do gerador de carga, além da configuração de valores mínimos e máximos, há a configuração independente de valores para os fluxos de alta e para os de baixa prioridade.

A geração de carga de trabalho completa uma lacuna extremamente importante do NS4D. A definição de estratégias automatizadas para esta geração serve como base para demonstrar que as simulações realizadas pelos usuários não possuem uma carga de trabalho tendenciosa. Mesmo em pequenas simulações, com por exemplo 30 fluxos de dados, o tempo gasto para gerá-los pode inviabilizar o projeto. Este argumento é reforçado por normalmente ser necessário realizar seguidas simulações no mesmo cenário com diversos conjuntos diferentes de fluxos de dados para validá-las. Este módulo, portanto, é indispensável para qualquer ferramenta de suporte a simulação de redes.

Neste sentido, para satisfazer as expectativas do usuário quanto a uma ferramenta de suporte a simulação de redes, esta deve ser capaz de medir quantitativamente os resultados da sua simulação. Neste cenário, surgem os scripts AWK do módulo de análise, descrito na sessão seguinte.

4.7 – Módulo de Análise de Desempenho

Nas metodologias de desenvolvimento ágeis, muito se fala sobre satisfazer a expectativa do usuário quando novas versões de software são lançadas. Concomitantemente, a teoria da usabilidade prega que os resultados da execução do software devem ser os esperados pelo usuário. De nada adianta ao usuário possuir ferramentas que detalham como os dados devem ser inseridos no sistema se, após a execução, os resultados apresentados não o servem de nada.

No ambiente de simulação composto pelo ns-2 e pelo NS4D, o mesmo conceito se aplica. Existe uma ferramenta de suporte à geração de carga de trabalho, módulos de configuração e logs de registro, entretanto até a presente sessão, as únicas formas com que os resultados da simulação são comunicados ao usuário são através de dados no arquivo *trace* e de animação no Nam.

As animações realizadas pelo Nam são visualizações da rede em ação, entretanto nenhuma forma de análise quantitativa de parâmetros de redes é possível ser realizada. Um arquivo *trace* (descrito na sessão 3.1) fornece dados detalhados sobre cada evento ocorrido na simulação. Um evento pode ser a entrada ou saída de um pacote em uma fila ou em um nodo e o descarte deste pacote. Apesar de poder ser considerada uma base de dados completa sobre a simulação, um arquivo *trace* contêm simplismente dados brutos – geralmente não representativos para o usuário.

No papel de usuário, o pesquisador ou projetista de redes pode contar com o módulo de análise de desempenho do NS4D. Este módulo tem como objetivo principal extrair da base de dados da simulação, o arquivo *trace*, informações relevantes sobre a simulação. Para isto, existem rotinas que fornecem os principais parâmetros de desempenho QoS: vazão, índice de perdas, atraso e *jitter* (variação de atraso).

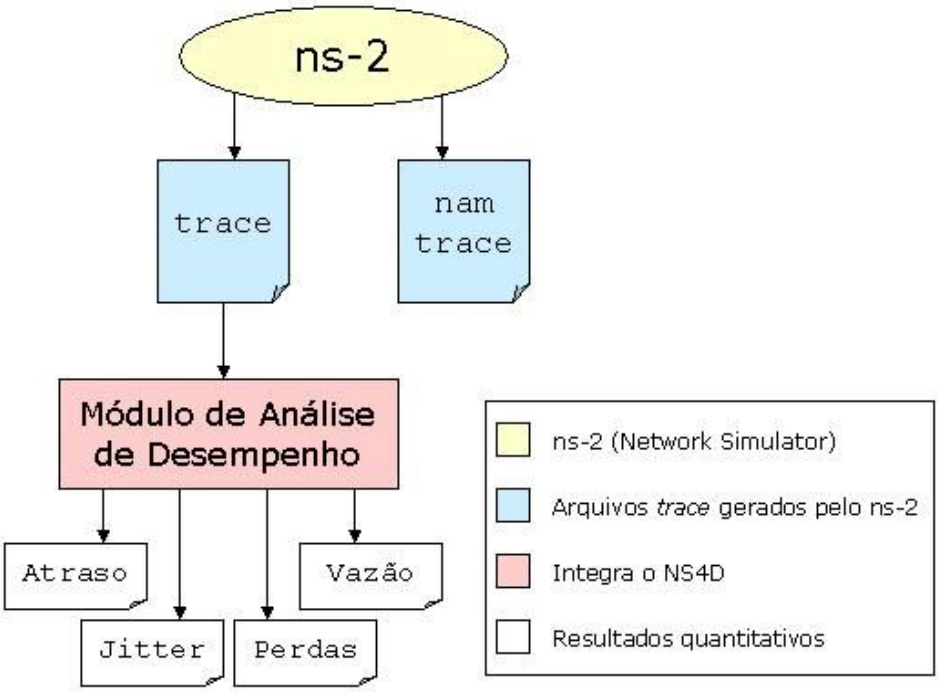

Figura 4.4: Módulo de Análise de Desempenho

A figura 4.4 mostra o funcionamento do módulo de análise de desempenho. As rotinas que compõem este módulo foram desenvolvidas na forma de scripts, na linguagem AWK. A linguagem AWK foi escolhida por possuir familiaridade com o padrão POSIX e, principalmente, por sua estrutura fornecer suporte a processamento de arquivos em formato de tabela – diversas linhas com quantidades iguais de colunas separadas por um ou mais espaços em branco.

A escolha da linguagem AWK possibilita o processamento de grandes arquivos, pois otimiza a alocação do conteúdo do arquivo em memória. Ao invés de carregar todo o arquivo e processá-lo em memória, apenas uma porção de dados é carregada e processada de cada vez. Esta estratégia eleva o tempo de processamento, entretanto nenhum tipo de limitação é imposta à capacidade de processamento da ferramenta.

Para análise dos quatro principais parâmetros de desempenho (vazão, índice de perdas, atraso e *jitter*), três arquivos com scripts AWK foram criados:

- `delay_jitter.awk`: este script implementa o algoritmo para calcular o atraso e o *jitter* fim a fim de um fluxo de dados;

- `thput.awk`: este script implementa o algoritmo para cálculo da vazão de algum fluxo de dados. O nome "`thput`" vem da expressão *throughput*, que significa vazão em inglês;

- `indice.awk`: este script implementa o algoritmo que calcula a porcentagem de perdas na transmissão de algum fluxo de dados.

Todos os scripts extraem seus resultados a partir do arquivo de *trace* da simulação. Para o cálculo do atraso, o script AWK varre o arquivo *trace* a procura do tempo inicial e final de transmissão de um pacote na simulação. Ao obter o atraso de todos os pacotes de um fluxo faz-se uma média aritmética. O *jitter* é calculado a partir do atraso de cada pacote subtraído do atraso do pacote anterior. Novamente, o resultado é a média aritmética do *jitter* de todos os pacotes. Para o cálculo da vazão, o tamanho de todos os pacotes que chegam no seu destino é acumulado para cada segundo de simulação. O resultado é a transformação deste valor acumulado de bytes para bits, transformando a medida em bytes por segundo. O índice de perdas é taxa de descarte de pacotes em relação à vazão inicial do fluxo. O tamanho de todos os pacotes descartados é acumulado e confrontado com o tamanho de todos os pacotes que foram enviados pelo nodo de origem.

Todos os algoritmos possuem filtros que são enviados como parâmetros na execução dos scripts. Ao enviar o parâmetro FID é definido qual fluxo será medido, sendo que pacotes pertencentes a outros fluxos de dados são descartados. Para o cálculo da vazão e do índice de perdas é possível configurar a granularidade da taxa de amostragem enviando o parâmetro FREQ. Por padrão FREQ é 0.1s, o que indica que haverá um resultado a cada 100ms de transmissão do fluxo de dados.

Eis as linhas de comando para execução dos scripts:

```
awk -f delay_jitter.awk -v FID=(fluxo) (arquivo trace)
awk -f thput.awk  -v FID=(fluxo) FREQ=(amostragem) (arquivo trace)
awk -f indice.awk -v FID=(fluxo) FREQ=(amostragem) (arquivo trace)
```

Listagem 4.12: Sintaxe dos comandos para utilização dos scripts de análise de desempenho

Na listagem 4.12 pode-se observar os parâmetros "–f" e "-v" do AWK. Eles significam, respectivamente, os scripts que serão executados e as opções passadas para eles. Ao final é passado o arquivo de entrada para os scripts, no caso o arquivo *trace* da simulação. Para maior segurança, é possível encaminhar os resultados dos scripts para um arquivo qualquer concatenando a linha de comando à instrução "> (arquivo de resultado)". Eis alguns exemplos de comandos executados:

```
awk -f delay_jitter.awk -v FID=2              simulacao.tr > delay_jitter.tr
awk -f thput.awk        -v FID=2 FREQ=10 simulacao.tr > thput.tr
```

```
awk -f indice.awk -v FID=2 FREQ=10 simulacao.tr > indice.tr
```

Listagem 4.13: Exemplo de chamadas aos scripts de análise de desempenho

A listagem 4.13 demonstra a chamada dos scripts. Todos os dados estão sendo coletados do fluxo com identificador 2 (FID=2) e a freqüência de amostragem está ajustada para uma mediação a cada 10s. Os resultados estão sendo escritos nos arquivos `delay_jitter.tr`, `thput.tr` e `indice.tr`.

Para cada um dos scripts há um formato para apresentação dos resultados:

```
[número de pacotes] [atraso médio] [jitter médio]
```

Listagem 4.14: Sintaxe dos resultados de atraso e *jitter*

O script de cálculo de atraso e jitter apresenta o resultados conforme a listagem 4.14, com a quantidade de pacotes transmitidos, o atraso e o *jitter* médio.

```
[momento da amostragem] [vazão] [tipo de fluxo]
```

Listagem 4.15: Sintaxe do resultado de vazão

O script “thput.awk” tem como resultado, para cada amostragem, o momento da amostragem, a vazão medida e o tipo de transmissão de dados, como mostra a listagem 4.15.

```
[momento da amostragem] [índice de perdas] [bytes descartados] [bytes transmitidos] [id do fluxo] [tipo de fluxo]
```

Listagem 4.16: Sintaxe do resultado de índice de perdas

A listagem 4.16 acima apresenta o resultado do script “índice.awk”: momento da amostragem, índice de descarte, total de bytes descartados, total de bytes transmitidos, identificador do fluxo e tipo de fluxo transmitido. No estudo de caso apresentado no capítulo 5 há mais detalhes sobre os arquivos de resultados gerados por este módulo.

Esta série de scripts por si só já é uma grande contribuição à sociedade acadêmica. Independente da utilização do NS4D, qualquer usuário pode utilizar este módulo para obter resultados quantitativos da sua simulação (exceto *wireless*) com o ns-2.

O módulo de análise de desempenho é a principal forma com que o NS4D ataca o quesito “satisfação subjetiva” da usabilidade de software. O preenchimento desta extensa lacuna deixada pelo ns-2 eleva o potencial do ambiente de simulação. Os usuários que antes se preocupavam em colher estas informações agora podem investir seu tempo propondo e validando modelos cada vez melhores.

Este módulo de análise de desempenho pode ser utilizado em qualquer simulação *wired* no ns-2, entretanto outros módulos implementam funcionalidades mais específicas. A sessão seguinte descreve o módulo que permite simular cenários com MPLS.

4.8 – Módulo de Configuração MPLS

Imergindo na área de roteamento, dentro das redes de computadores, o NS4D apresenta o módulo de configuração MPLS. Como o próprio nome diz este módulo habilita o usuário a realizar simulações utilizando MPLS e faz uso da implementação do protocolo MPLS para a arquitetura do ns-2: o MNS, ou *MPLS for Network Simulator* (Ahn 2002). Na prática, porém, sua principal funcionalidade é a possibilidade de configurar rotas explícitas, fim a fim. A chamada configuração de rotas explícitas é a capacidade de configurar os fluxos a seguir uma trajetória previamente definida dentro do cenário simulado. Outra funcionalidade é a utilização do protocolo de distribuição de rótulos, o LDP (*Label Distribution Protocol*) em conjunto com um algoritmo de roteamento global baseado em restrições, o chamado CR-LDP (*Constraint Routing – Label Distribution Protocol*).

No ns-2 há o comando “setup-erlsp” para a configuração da rota com LDP e outro comando “bind-flow-erlsp” para efetivar a utilização da rota pelo fluxo. Estas configurações de rota explícitas são mostradas na listagem 4.17:

```
$ns at 12 "[$nodo1 get-module "MPLS"] setup-erlsp 1 4_1_3_0 6"
$ns at 15 "[$nodo4 get-module "MPLS"] bind-flow-erlsp      1 5 6"
```

Listagem 4.17: Trecho de script OTcl contendo a configuração de uma rota explícita com MPLS

No NS4D, o módulo de configuração MPLS requer que alguns novos dados sejam inseridos no arquivo “flux”. Para utilizar a configuração de rotas explícitas ou o algoritmo CR-LDP deve-se adicionar ao arquivo “flux” uma nova coluna, que podemos chamar de MPLS.

No caso de utilizar CR-LDP na simulação, todos os fluxos devem ser configurados desta forma. O ambiente não foi projetos para aceitar mais de um tipo de roteamento na mesma simulação. Sendo assim, para cada fluxo de dados do arquivo “flux”, a coluna MPLS deve receber como valor “CRLDP”, como mostra a listagem 4.18 abaixo:

```
FID Origem Dest Cliente Início Fim Tam Atraso Setup Hold Nível MPLS
    1   4     0   1        7     2  210 0     0     0    0   CRLDP
```

Listagem 4.18: Exemplo de arquivo “flux” contendo um fluxo de dados roteado com CRLDP

Para a configuração de rotas explícitas, existe a necessidade do usuário configurar uma rota válida dentro da topologia. Faz-se isto mostrando nodo por nodo o caminho onde os pacotes daquele fluxo de dados irão trafegar. O formato utilizado deve ser os identificadores dos nodos intercalados com o caractere “_”. A listagem 4.19 demonstra como utilizar este recurso:

```
FID Origem Dest Cliente Início Fim Tam Atraso Setup Hold Nível MPLS
    1   4     0   1        7     2  210 0     0     0    0  4_1_3_0
```

Listagem 4.19: Exemplo de arquivo “flux” com uma rota explícita configurada para um fluxo de dados

Neste exemplo, o fluxo irá trafegar pelos nodos 4, 1, 3 e 0. Nota-se que a configuração da rota explícita deve ter seu início coincidindo com a origem do fluxo e seu final deve coincidir com o destino configurado para o fluxo de dados.

Este módulo também torna amigável o emprego de MPLS nas simulações. No ns-2 o uso de rotas explícitas pela API OTcl, por exemplo, é muito complicado e delicado. Qualquer erro na configuração das rotas explícitas, seja um erro de origem e destino ou de algum enlace inexistente na rede, invalida totalmente a configuração das demais rotas. Com o NS4D, estes parâmetros são bem mais visíveis já que não estão no meio de nenhuma linguagem de programação.

O emprego de MPLS e suas rotas explícitas expandem o potencial das simulações, já que fornece ao usuário todo o controle dos fluxos de dados. Aproveitando este potencial e satisfazendo outras áreas interessadas inclusive em engenharia de tráfego, a seguinte sessão descreve o funcionamento do módulo de roteamento.

4.9 – Módulo de Roteamento

O módulo de configuração MPLS, apesar de possibilitar a configuração de rotas explícitas, não automatiza esta tarefa. Ao usuário cabe a tarefa de definir para cada um dos fluxos de dados qual será o seu trajeto dentro da topologia. Da mesma forma, não é tarefa do módulo de configuração MPLS otimizar a distribuição dos fluxos dentre as rotas possíveis. Estas tarefas de automatização e otimização podem ser alcançadas pelo presente módulo, o módulo de roteamento.

O módulo de roteamento age dentro do NS4D antes da definição das rotas nos scripts OTcl. Definidas a topologia e a carga de trabalho, o NS4D gera uma representação nos arquivos chamados “net” e “lsp”. O arquivo “net” tem o mesmo objetivo do arquivo “topo”: descrever a topologia física da simulação. O arquivo “lsp” tem objetivo simular ao arquivo “flux”: descrever a carga de trabalho. De posse destes arquivos, o NS4D realiza uma chamada de sistema para executar uma implementação externa de algoritmo de roteamento. Esta execução deve gerar um arquivo chamado “rts”, que contém as rotas definidas para cada fluxo injetado no algoritmo. A seguir, de posse do arquivo “rts”, o NS4D configura as rotas explícitas para cada fluxo injetado na simulação.

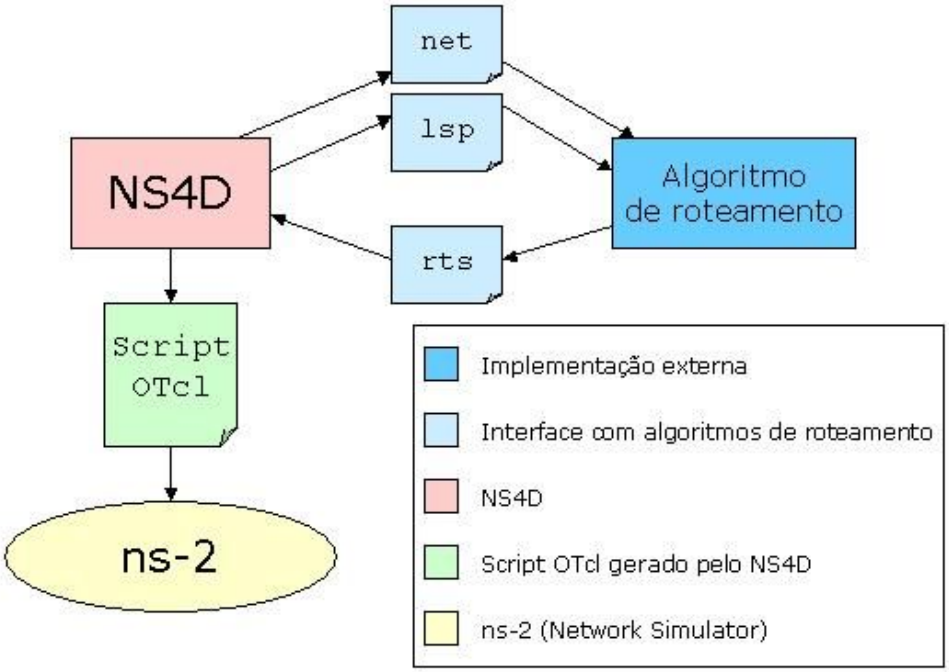

Figura 4.5: O NS4D interagindo com um algoritmo de roteamento

A figura 4.5 acima mostra o NS4D funcionando em conjunto com um algoritmo de roteamento externo. Esta implementação de algoritmo de roteamento é chamada de externa, pois não é fornecida pelo NS4D. Para integração entre o NS4D e este outro programa que implementa o algoritmo, a forma de comunicação definida foi através de uma chamada de sistema e dos arquivos “net”, “lsp” e “rts”.

O NS4D pode se acoplar a algoritmos de roteamento de duas maneiras: a estática e a dinâmica. No acoplamento estático o algoritmo de roteamento é executado apenas uma vez, no início da simulação, e independente da configuração de momento inicial do fluxo, todos os fluxos são injetados no algoritmo. Outra estratégia, mais coerente, é a de acoplamento dinâmico. No decorrer da simulação, o NS4D identifica quais fluxos estão ativos e os encaminha para o algoritmo de roteamento. O tempo para a solução das rotas é identificado pelo NS4D que avança seu escalonador até o próximo momento e novamente identifica os fluxos ativos para rotear. Nesta estratégia o algoritmo de roteamento pode ser chamado diversas vezes, o que pode prejudicar o desempenho do NS4D.

Assim como outras configurações adicionais necessárias pelo módulo de roteamento, a chamada de sistema para executar o algoritmo é configurada no arquivo de configuração. A listagem 4.20 abaixo apresenta as configurações necessárias pelo módulo de roteamento, além das configurações comuns no arquivo “ns4d.cfg”:

```
<CFG_GERAIS>
```

```
    <ALGORITMO
      ## Define o algoritmo do experimento
      #  "4" = Acoplamento com algoritmo externo estático
      #  "5" = Acoplamento com algoritmo externo dinâmico
      ativo=4

      ## Define a chamada de sistema para rodar algoritmo externo
      plug_system_call="lgr"

      ## Define o intervalo mínimo entre rodadas do algoritmo
      #  Somente para o caso de algoritmos externos dinâmicos
      #  (10000 = 0.1s) MAXIMO=10000
      intervalo_minimo=1
    />
    <DURACAO
      ## Define o tempo máximo que um fluxo irá aguardar aprovação
      #  Somente para o caso de algoritmos externos dinâmicos
      timeout_rejeicoes=10 #10s
    />
    <REGISTROS
      ## Nome dos arquivos de comunicação.
      arquivo_net=net
      arquivo_lsp=lsp
      arquivo_rts=rts
    />
    <CARGA_DE_TRABALHO
      ## Define o número de níveis
      n_niveis=7
    />
  </CFG_GERAIS>
```

Listagem 4.20: Configurações adicionais necessárias pelo módulo de roteamento

Eis a descrição semântica de cada um dos atributos configuráveis:

- `CFG_GERAIS/ALGORITMO/ativo`: este é um atributo já presente nas configurações habituais. Ele define o algoritmo de roteamento a ser utilizado. Além dos tradicionais Static, DV, LS e ISIS, com o módulo de roteamento é possível utilizar implementações de algoritmos de roteamento estáticos ou dinâmicos;

- `CFG_GERAIS/ALGORITMO/plug_system_call`: define qual será a chamada de sistema para executar o algoritmo externo na forma de um *plugin*;

- `CFG_GERAIS/ALGORITMO/intervalo_minimo`: caso o algoritmo de roteamento execute-se muito rapidamente, pode-se definir este intervalo de tempo mínimo para executá-lo;

- `CFG_GERAIS/DURAÇÃO/timeout_rejeicoes`: no caso de roteamento dinâmico, este atributo define quanto tempo algum fluxo irá aguardar para ser injetado na rede, caso o algoritmo de roteamento adote alguma estratégia de rejeição de fluxos;

- `CFG_GERAIS/REGISTROS/arquivo_net`: é o nome do arquivo “net”;

- `CFG_GERAIS/REGISTROS/arquivo_lsp`: é o nome do arquivo “lsp”;

- `CFG_GERAIS/REGISTROS/arquivo_rts`: é o nome do arquivo “rts”;

- `CFG_GERAIS/CARGA_DE_TRABALHO/n_niveis`: define o número de níveis de demanda de vazão requisitados ao algoritmo de roteamento.

Aos algoritmos de roteamento são fornecidos diversos parâmetros que podem ser considerados no momento de escolher as rotas. Existem estratégias de demanda de atraso máximo, níveis de demanda de largura de banda e prioridades *setup* e *holding*:

- A estratégia de demanda de atraso máximo é uma restrição que pode ser inserida em cada fluxo de dados, que indica o tempo máximo que o fluxo aceita levar para chegar ao seu destino;

- A estratégia de níveis é uma forma de controlar a demanda de um fluxo de dados pelos recursos da rede, quando a rede está praticamente saturada. No arquivo “lsp” os níveis de demanda são informados para o algoritmo de roteamento externo. O último e mais alto nível é o realmente desejado pelo fluxo, já no primeiro nível a demanda é 0 (zero). Os níveis intermediários formam uma curva que ascende exponencialmente;

- As prioridades *setup* e *holding* são gerenciadas pelo NS4D, pois é ele quem sabe quais fluxos estão ativos no momento. O algoritmo de roteamento recebe a prioridade válida no momento pelo arquivo “lsp” e pode ou não considerá-la para a definição da rota de cada fluxo de dados.

Cada um dos arquivos de comunicação entre o NS4D e seus *plugins* algoritmos deve seguir uma sintaxe. Abaixo um exemplo de arquivo “net”

```
3           4

0     1     10    2000
1           0     10    2000
1     2     10    2000
2     1     10    2000
```

Listagem 4.21: Exemplo de arquivo “net” contendo a descrição da topologia

O arquivo “net” demonstrado na listagem 4.21 tem seu corpo disposto em forma de tabela, entretanto há uma linha de cabeçalho que informa o número de nodos e o número de enlaces respectivamente. O corpo do arquivo contém 4 (quatro) colunas que significam, respectivamente, origem, destino, atraso e largura de banco do enlace em Kbps. Este arquivo aceita somente enlaces *simplex*, ou seja, cada definição de enlace vale apenas no sentido demonstrado na origem/destino.

O arquivo “lsp” define a carga de trabalho e os valores para as estratégias de níveis de demanda, prioridades e demanda de atraso máximo. Eis um exemplo de arquivo “lsp”:

```
3 8

0 2 12 39 0 10 40  91 163 255 367 500
```

```
0 2 4 107 0 1   2   5  10  16  23  32
0 1 4 117 0 17 69 156 278 435 626 853
```

Listagem 4.22: Exemplo de arquivo “lsp” contendo a descrição dos fluxos

Na listagem 4.22 também há um cabeçalho com 2 (dois) valores: o primeiro indica o número de fluxos de dados que seguem, e o segundo indica a quantidade de níveis de demanda para cada um destes fluxos. O corpo do arquivo “lsp” possui quantidade de colunas variável, pois depende de quantos níveis são configurados. No exemplo estão definidos em cada coluna, respectivamente, origem do fluxo, destino do fluxo, prioridade, demanda de atraso e os 8 (oito) níveis – iniciando com demanda 0 (zero) até a demanda realmente requerida.

O arquivo “rts” faz o caminho inverso. Deve ser gerado pelo algoritmo de roteamento para ser interpretado pelo NS4D. Á seguir a sintaxe utilizada para representar os resultados do algoritmo:

```
1 0_2   500k 8
2 0_1_2  32k 8
3 0_1   853k 8

0.127  17680
```

Listagem 4.23: Exemplo de arquivo “rts” contendo a resolução das rotas

O arquivo “rts”, apresentado na listagem 4.23, apresenta um rodapé ao invés do cabeçalho dos arquivos anteriores. O cabeçalho na última linha possibilita ao algoritmo apresentar o seu tempo de execução e um valor de função objetivo, respectivamente. No corpo do arquivo são apresentados, respectivamente, o identificador do fluxo, a sua rota explícita, a largura de banda aceita pelo algoritmo e o nível correspondente a esta largura de banda.

É interessante lembrar que o módulo de registro de logs fornece informações detalhadas sobre a execução deste módulo. Apesar de sua complexidade e da variação que suas configurações possibilitam, a auditoria na busca de erros na execução das simulações é possibilitada por estes registros.

Para realizar transformações não triviais em simulações de redes foi criado um módulo chamado de módulo Plus. Este módulo é descrito na sessão seguinte.

4.10 – Módulo Plus

Para realizar algumas tarefas avançadas nos cenários simulados, o NS4D fornece ao usuário o módulo Plus. Normalmente, o excesso de funções em um software atrapalha seu aprendizado, entretanto o módulo Plus realiza de forma transparente estas transformações. Isto torna o ambiente poderoso sem dificultar o uso do NS4D nem tornar seu aprendizado mais demorado.

Como parte da rota fim a fim entre sua origem e o destino, um pacote pode ser enviado por meio de diversos roteadores intermediários. Cada um destes roteadores representa um salto, ou um passo, a mais realizado pelo pacote e a cada novo salto maior é o atraso para a

chegada dos dados. Neste sentido, é possível identificar 4 tipos de atrasos em redes de computadores:

- O atraso de processamento: é o tempo requerido para examinar o cabeçalho do pacote de determinar para onde direcioná-lo, além de outras questões como o tempo necessário para verificação de erros em bits. Este atraso ocorre dentro dos nodos roteadores;

- O atraso de fila: ao sair de um *host*, um pacote pode sofrer atraso esperando em uma fila para ser transmitido no enlace;

- O atraso de transmissão: é a quantidade de tempo exigida para a transmissão de todos os bits do pacote no enlace;

- O atraso de propagação: é o tempo que leva para um pacote se propagar desde a origem do enlace até seu destino.

O ns-2 possui mecanismos para simular apenas o atraso de fila, transmissão e propagação. O atraso de processamento nos nodos da simulação é desconsiderado tanto pelo ns-2, quanto pela maioria dos ambientes de simulação. Uma das funcionalidades do módulo Plus é a inserção de atraso de processamento nos nodos, um recurso até então pouco explorado pelos simuladores.

Este módulo pode realizar 3 (três) tipos de transformações no cenário simulado: divisão de recursos, inserção de atraso de processamento nos nodos e agregação de fluxos. Estas transformações são descritas em detalhes a seguir:

- Divisão de recursos: toda a topologia é dividida como se fossem duas topologias diferentes. No arquivo de configuração, informa-se qual um fator de divisão (atributos `fator_xy`), de 0 a 100%, e todas as larguras de banda da simulação são divididas seguindo este fator. Os enlaces com menos capacidade ficam reservados para os fluxos de alta prioridade e os de baixa complexidade ficam com a maioria dos enlaces;

- Inserção de atraso de processamento nos nodos: uma grande limitação do ns-2 é que seus nodos não consomem tempo dos pacotes que neles trafegam. Apenas as filas e os enlaces atrasam os pacotes. Esta transformação faz com que haja um pequeno enlace dentro de cada nodo que, para o ns-2, é o responsável pelo atraso. Este enlace é configurado pelo NS4D para ter um atraso de 5ms. Para cada nodo, o NS4D gera outros dois nodos: o primeiro nodo gerado recebe todos os enlaces *simplex* do nodo original e o segundo nodo saem todos os enlaces *simplex* que saíam do nodo original. O enlace que gera o atraso de processamento é o enlace *simplex* que liga o primeiro nodo ao segundo nodo gerado;

- A agregação de fluxos é uma estratégia utilizada para diminuir a quantidade de fluxos. Potencialmente, todos os fluxos com origem e destino iguais podem ser agregados em um só. O arquivo de configuração deve informar quais critérios serão

considerados para agregar estes fluxos. A agregação dos fluxos de dados formam os chamados troncos de tráfego, ou *traffic trunks*.

As configurações adicionais necessárias por este módulo estão todas dentro do sub-elemento <PLUS>, como demonstra a listagem 4.24. Eis a descrição semântica de cada um dos atributos configuráveis no módulo Plus:

- `CFG_GERAIS/PLUS/tipo_execucao`: define a utilização das transformações de divisão de recursos e inserção de atraso de processamento nos nodos e a ordem de execução destas transformações, já que podem ser realizadas em conjunto;

- `CFG_GERAIS/PLUS/fator_xy`: define o fator de divisão para a divisão de recursos. O nome `xy` indica que os recursos serão separados em duas variáveis, cada uma contendo uma porcentagem da largura de banda de cada enlace;

- `CFG_GERAIS/PLUS/agregacao`: este atributo indica se será realizada agregação e, no caso de utilizar agregação, quais serão os critérios indicados para definir se dois fluxos irão pertencer ao mesmo *traffic trunk*;

- `CFG_GERAIS/PLUS/setup_prio_media`: define a prioridade setup média. Acima desta média os fluxos são considerados de alta prioridade e abaixo dela, de baixa prioridade;

- `CFG_GERAIS/PLUS/trata_atraso`: define qual tratamento será dado para a demanda de atraso dos fluxos agregados. Pode prevalecer a demanda mínima, a demanda máxima ou ainda uma média entre as demandas dos fluxos do *traffic trunk*;

- `CFG_GERAIS/PLUS/trata_prioridade`: da mesma forma, define qual tratamento será dado para as prioridades dos fluxos agregados. Pode prevalecer a prioridade mínima, a prioridade máxima ou ainda uma média entre as prioridades de todos os fluxos do *traffic trunk*.

```
    <CFG_GERAIS>
      <PLUS
        ## Define o fluxo de execução das transformações
        #  "0" = Normal - sem transformação
        #  "1" = Realiza a divisão de recursos
        #  "2" = Insere atraso de processamento
        #  "3" = Divide recursos e, a seguir, insere atraso de proc.
        #  "4" = Insere atraso de proc. e, a seguir, divide  recursos
        tipo_execucao=3

        ## Define fator utilizado na divisão de recursos
        fator_xy=5

        ## Define o protocolo de roteamento
        #  "0" = Sem agregação   - Desabilita traffic trunks
        #  "1" = Fonte/Destino   - Agrega fluxos por fonte e destino
          #   "2" = Cliente/Destino - Agrega fluxos por cliente e
destino
```

```
        agregacao=1

        ## Define a prioridade de setup media
        setup_prio_media=5

        ## Opcional: Define a estratégia de agregação para o atraso
        #  "0" = Mínimo - Demanda de atraso do traffic trunks igual a
        #                 menor demanda entre seus fluxos
        #  "1" = Médio  - Demanda de atraso do traffic trunks igual a
        #                 média das demandas dos seus fluxos
        #  "2" = Máximo - Demanda de atraso do traffic trunks igual a
        #                 maior demanda entre seus fluxos
        trata_atraso=0

        ## Opcional: Define a estratégia de agregacao para prioridade
         #   "0" = Mínimo - Prioridades do traffic trunks iguais a
menor
        #                 prioridade entre seus fluxos
         #   "1" = Médio   - Prioridades do traffic trunks iguais a
média
        #                 das prioridades dos seus fluxos
         #   "2" = Máximo - Prioridades do traffic trunks iguais a
maior
        #                 prioridade entre seus fluxos
        trata_prioridade=2
      />
    </CFG_GERAIS>
```

Listagem 4.24: Configurações adicionais necessárias pelo módulo Plus

Definidos quais módulos e quais as funcionalidades presentes no NS4D, deve-se definir uma estratégia para distribuí-lo. Esta estratégia inclui a forma de instalação e informações adicionais para outros desenvolvedores interessados em prosseguir neste projeto. Neste contexto, a sessão seguinte explana sobre a forma de distribuição do NS4D.

4.11 – Distribuição do NS4D

Portabilidade de software e implantação efetiva de sistemas são aspectos cruciais de práticas modernas de engenharia de software. É inaceitável iniciar hoje um projeto de software que tem como expectativa rodar em apenas uma plataforma. Como prova da importância da portabilidade há o recente crescimento da procura por linguagens que rodam sobre máquinas virtuais ou interpretadores, como Perl, PHP e principalmente Java. Restrições de hardware podem mudar a escolha da plataforma, novos usuários podem surgir com um novo tipo de plataforma ou uma nova versão de um sistema operacional proprietário pode trazer novidades incompatíveis. Além disso, software que possuem um mecanismo de implantação fácil e com menos chances de erro são valiosos.

Pensando desta forma, o NS4D adotou a mesma forma de distribuição que a comunidade de software livre: utilizando as ferramentas *Autoconf* e *Automake* (Vaugham 2000). Estes pacotes têm o intuito de deixar um software mais portável e simplificar a compilação e implantação dele.

Autoconf é uma ferramenta que torna softwares mais portáveis através da realização de testes para descobrir características do sistema antes do software ser compilado. O código fonte do software pode, então, se adaptar a diferentes características.

Automake é uma ferramenta para gerar *Makefile's* (arquivos que descrevem o que compilar) dentro de diversos padrões. *Automake* simplifica o processo de descrever a organização de um software e realiza funções como a checagem e ligação de dependências entre os arquivos de código fonte.

A utilização destas duas ferramentas reduz o esforço em duas etapas do projeto: no desenvolvimento e na instalação do software. Ao desenvolvedor não é mais necessário criar scripts para compilação e implantação da ferramenta. Já ao usuário final, basta ter acesso a um arquivo compactado via internet ou qualquer outra fonte, em seguida descompactá-lo, e rodar os scripts `configure` e `Makefile` – com isto, o NS4D estará instalado.

Para utilizar as ferramentas *Autoconf* e *Automake*, no papel de desenvolvedor, o autor do presente trabalho precisou criar dois arquivos:

1. `configure.in`: é uma entrada para o Autoconf. Este arquivo é um modelo de script que contém macros e fragmentos de código Shell que são utilizados para o *Autoconf* gerar o script chamado `configure`. O *Autoconf* copia o conteúdo de `configure.in` para o `configure` substituindo as macros à medida que elas ocorrem na entrada;

2. `Makefile.am`: é uma entrada para o Automake que descreve em auto-nível os requisitos para compilar um projeto: o que é necessário compilar e para onde vão quando instalados. Seu produto final é um arquivo `Makefile` contendo uma série de tarefas *make*.

Para gerar os scripts é necessário rodar os comandos `aclocal` e `autoconf`. O comando `aclocal` produz um arquivo `aclocal.m4`, que contêm a definição das macros do `configure.in` e também é utilizado como entrada para o *Automake*. Finalmente, para gerar o `Makefile`, deve-se rodar o comando `automake`.

O `Makefile` possibilita ao desenvolvedor executar algumas tarefas pré-definidas como tarefas dentro do arquivo. Para executar uma tarefa basta executar o comando *make* seguido do nome da tarefa. O `Makefile` gerado pelo *Automake* tem como tarefa padrão a compilação do software, portanto executando *make* o desenvolvedor consegue construir sua aplicação. A opção `automake –add-missing` é útil para gerar alguns arquivos comuns em distribuições de software livre, dentre eles os arquivos:

- `AUTHORS`: Uma lista de nomes e, geralmente, endereços de e-mail das pessoas que trabalharam no software;

- `INSTALL`: Contêm uma explicação "passo-a-passo" de como instalar o software;

- `NEWS`: É visível ao usuário final e contêm as novidades do software;

- `README`: O primeiro lugar que o usuário deve olhar para ter uma visão geral sobre o propósito e o funcionamento do software;

- `COPYING`: Este arquivo traz os direitos de cópia do software que pode ser complementar à licença do software;

- `ChangeLog`: Este arquivo recorda as mudanças que são feitas em um software à medida que este amadurece e avançam suas versões.

O usuário de um software distribuído seguindo os padrão de *Autoconf* e *Automake* podem implantar a aplicação com segurança executando, primeiramente o script `configure` para configurar o Makefile de acordo com as características do ambiente do usuário. Em seguida, é necessário compilar e empacotar a aplicação, com o comando *make*. Caso haja a necessidade de implantar a aplicação em locais visíveis a outros os usuários do sistema operacional, pode-se rodar a tarefa `make install`.

O NS4D utiliza todo este potencial das ferramentas *Autoconf* e *Automake*, possibilitando ao usuário a instalação em diversas plataformas, de forma padronizada, controlada e fácil. Este diferencial na instalação do software também influencia nos requisitos gerais de usabilidade do NS4D.

Como prova de sucesso no desenvolvimento do NS4D, o capítulo seguinte narra um estudo de caso que valida o emprego desta ferramenta como uma ferramenta amigável de suporte a simulação de redes com o ns-2.

Capítulo 5

Estudo de Caso

Como forma de verificar a eficiência do NS4D conforme proposto, este capítulo apresenta os resultados de um experimento que empregou a ferramenta. Este experimento é parte das atividades realizadas durante a elaboração da tese de doutorado do professor Roberto Alexandre Dias (Dias 2004b)

O objetivo do experimento foi testar o desempenho de um algoritmo de roteamento global baseado em heurísticas e confrontá-lo com o desempenho de algoritmos de roteamento convencionais. Também foi acoplado ao módulo de roteamento um pacote de programação matemática – o Xpress-MP – no intuito de contrastar a capacidade do algoritmo de roteamento em estudo em encontrar a solução de mais alta qualidade.

Os experimento foram realizados utilizando um computador Pentium IV de 1.2 GHz e 256 MBytes de memória RAM. A exemplo do ambiente de utilizado para o desenvolvimento do NS4D, foi instalada a distribuição Mandrake (versão 8.2) do sistema operacional Linux.

Foram utilizadas duas topologias de redes para os experimentos. A primeira similar à topologia utilizada por (Banerjee 2002), possui complexidade média, com 17 roteadores e 31 enlaces. Dentre os roteadores, apenas oito são geradores ou receptores de tráfego. Os enlaces foram divididos em duas categorias: enlaces de acesso, com 2kbps de largura de banda e atraso de 10ms; e (2) enlaces de núcleo, com 8kbps de largura de banda e atraso de 5ms. Esta topologia está apresentada na figura 5.1:

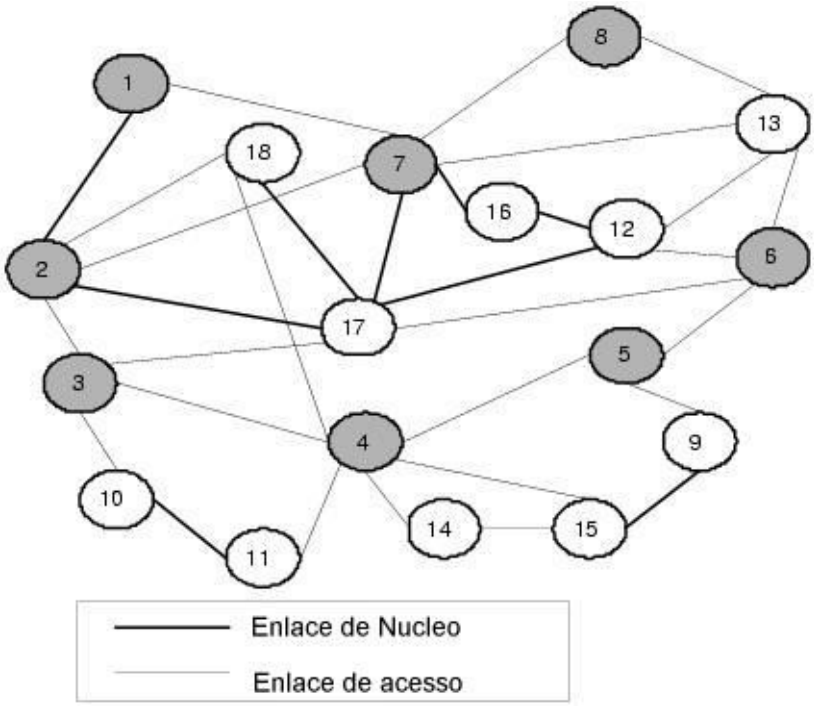

Figura 5.1: A primeira topologia de rede

A listagem 5.1 apresenta o arquivo “topo” utilizado para descrever a primeira topologia:

Origem	Destino	BW	Atraso	Fila	Tipo
0	1	2000kb	10ms	DropTail	duplex-link
0	6	2000kb	10ms	DropTail	duplex-link
1	2	2000kb	10ms	DropTail	duplex-link
1	6	2000kb	10ms	DropTail	duplex-link
1	17	2000kb	10ms	DropTail	duplex-link
1	16	8000kb	5ms	DropTail	duplex-link
2	3	2000kb	10ms	DropTail	duplex-link
2	9	2000kb	10ms	DropTail	duplex-link
2	16	2000kb	10ms	DropTail	duplex-link
3	4	2000kb	10ms	DropTail	duplex-link
3	10	2000kb	10ms	DropTail	duplex-link
3	13	2000kb	10ms	DropTail	duplex-link
3	14	2000kb	10ms	DropTail	duplex-link
3	17	2000kb	10ms	DropTail	duplex-link
4	5	2000kb	10ms	DropTail	duplex-link
4	8	2000kb	10ms	DropTail	duplex-link
5	11	2000kb	10ms	DropTail	duplex-link
5	12	2000kb	10ms	DropTail	duplex-link
5	16	2000kb	10ms	DropTail	duplex-link
6	7	2000kb	10ms	DropTail	duplex-link
6	12	2000kb	10ms	DropTail	duplex-link
6	15	8000kb	5ms	DropTail	duplex-link
6	16	8000kb	5ms	DropTail	duplex-link
7	12	2000kb	10ms	DropTail	duplex-link
8	14	8000kb	5ms	DropTail	duplex-link
9	10	8000kb	5ms	DropTail	duplex-link
11	12	2000kb	10ms	DropTail	duplex-link
11	15	8000kb	5ms	DropTail	duplex-link
11	16	8000kb	5ms	DropTail	duplex-link
13	14	2000kb	10ms	DropTail	duplex-link
16	17	8000kb	5ms	DropTail	duplex-link

Listagem 5.1: Arquivo “topo” que descreve a primeira topologia

A segunda topologia de rede utilizada, extremamente complexa e aleatória, possui 50 roteadores e 200 enlaces. A segunda topologia é apresentada na figura 5.2:

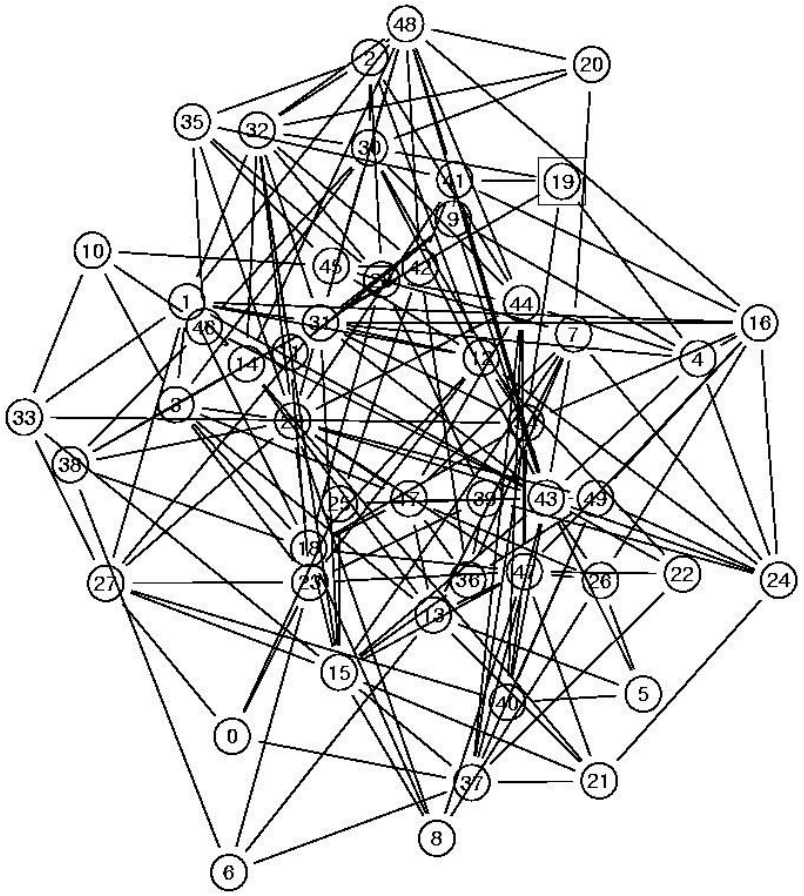

Figura 5.2: A segunda topologia utilizada

Para configurar a carga de trabalho injetada e cada uma das topologias, o módulo gerador de carga foi utilizado com a seguinte configuração:

1. Fluxos de baixa prioridade, com taxa de transmissão entre 20 e 150Kbps e atraso fim-a-fim entre 100 e 150ms; e

2. Fluxos de alta prioridade, com taxa de transmissão entre 380 e 870Kbps e atraso fim-a-fim entre 70 e 100ms.

Ainda em relação aos fluxos de dados, para cada um deles foram gerados 7 (sete) níveis de transmissão, variando de zero até um valor máximo desejado dentro das faixas correspondentes. Para a primeira topologia foram injetados até 500 fluxos de dados e para a segunda topologia foram gerados, aproximadamente, 20.000 (vinte mil) fluxos. Em ambos os cenários, os fluxos de alta prioridade correspondem a 1,5% do total de fluxos injetados nas topologias. A listagem 5.2 apresenta o arquivo de configuração do gerador de carga utilizado na definição da carga de trabalho para a segunda topologia:

```
<CFG_GERAIS>
  <PRIORIDADES
    max_alto_setup=7
    min_alto_setup=5
    max_baixo_setup=4
    min_baixo_setup=1
    razao_setup_holding_alta_prio=2
    razao_setup_holding_baixa_prio=2
  />
  <LARGURA_DE_BANDA
    max_bw_alta_prioridade=870
    min_bw_alta_prioridade=380
    max_bw_baixa_prioridade=150
    min_bw_baixa_prioridade=20
  />
  <ATRASO
    max_delay_alta_prioridade=100
    min_delay_alta_prioridade=70
    max_delay_baixa_prioridade=150
    min_delay_baixa_prioridade=100
  />
  <DURACAO
    max_duracao_alta_prioridade=2400 #40m
    min_duracao_alta_prioridade=300  #5m
    max_duracao_baixa_prioridade=900 #15m
    min_duracao_baixa_prioridade=5   #5s
    tempo_simulacao=18000            #5h
  />
  <TOPOLOGIA
    total_nodos=8
    tam_pacote=210
    n_niveis=7
  />
  <CARGA_DE_TRABALHO
    porcentagem_priorizados=1.5
    media_fluxos_simultaneos=1300
    media_fluxos_por_cliente=40
  />
</CFG_GERAIS>
```

Listagem 5.2: Configuração do gerador de carga para a segunda topologia

Após a execução do gerador de carga, o arquivo “flux” está pronto para ser injetado no núcleo do NS4D. A listagem 5.3 abaixo apresenta um trecho do arquivo “flux” gerado, com exatamente 19.300 (dezenove mil e trezentos) fluxos de dados definidos:

```
FID Org Dst Cliente Início Fim  Tam Atraso Setup Hold Níveis[1 a 7]
00001      1 3 18          1  10341959 210 142 2  4  3 12 28  50  79
113 155
00002      2 3 24          1  11567691 210 122 1  2  2 10 23  41  64
92 126
00003      5 7 23          1   8719927 210  71 6 12  9 39 88 157 245
353 482
00004      2 0 15          1  18705995 210 137 1` 2  1  4 10  19  30
43  59
00005      0 7  8          1  17744691 210  73 6 12  9 36 81 144 226
326 444
...
```

```
19297      7 0   9 179539466 180000000 210 131 1   2   1   6 13   24   38
55  75
19298      6 7   6 149933257 167757718 210   98 6 12 12 48 99 195 305
439 598
19299      0 3 11 149040012 151279801 210 109 3   6   1   4   10 18   28
41  56
19300      2 3   3 164287330 180000000 210   79 5 10 11 46 97 190 295
412 585
```

Listagem 5.3: Arquivo “flux” com a carga de trabalho gerada para a segunda topologia

Com os arquivo “topo” e “flux” prontos, o NS4D foi rodado para gerar o script OTcl. Pode-se verificar a eficácia da ferramenta quando se analisa a quantidade de linhas de código que o pesquisador não precisou escrever. Para o primeiro experimento, o mais simples, o script OTcl gerado pelo NS4D ficou com 10.750 (dez mil setecentos e cinqüenta) linhas. No segundo experimento, com a topologia mais complexa e milhares fluxos de dados definidos pelo gerador de cargas, o pesquisador foi poupado de escrever 420.452 (quatrocentos e vinte mil quatrocentos e cinqüenta e duas) linhas de script OTcl – ocupando 488MB de espaço em disco.

Corroborando com a afirmação de que seria inviável realizar estas simulações sem o NS4D, deve-se lembrar que para cada experimento foram geradas diversas simulações. Cada simulação serviu para averiguar o comportamento da rede com a variação do tamanho dos pacotes, prioridades, restrições e principalmente com diferentes algoritmos de roteamento. Cada uma destas simulações possui praticamente a mesma quantidade de linhas apresentada. Além disso, o NS4D apresentou um bom desempenho com relação a tempo de execução e consumo de memória. Descontando o tempo empregado na resolução das rotas pelo algoritmo de roteamento externos, o NS4D levou em média 9 segundos e utilizou no máximo 82MB de memória RAM para gerar o script OTcl do segundo e maior experimento.

Devido à complexidade do segundo experimento, este não pode ser executado no ns-2. No ambiente utilizado para estas simulações a execução do ns-2 consumiu além da memória disponível para tanto. Desta forma, para o segundo experimento, foram coletadas apenas métricas de otimalidade da solução e desempenho do algoritmo de roteamento, que não fazem parte da ferramenta e do presente trabalho. Para o primeiro experimento, entretanto, a simulação foi realizada no ns-2 e foi possível analisar quantitativamente os resultados da simulação com o módulo de desempenho do NS4D.

Os diagramas a seguir foram gerados pela ferramenta GNUPlot, a partir dos resultados obtidos pelo módulo de análise de desempenho.

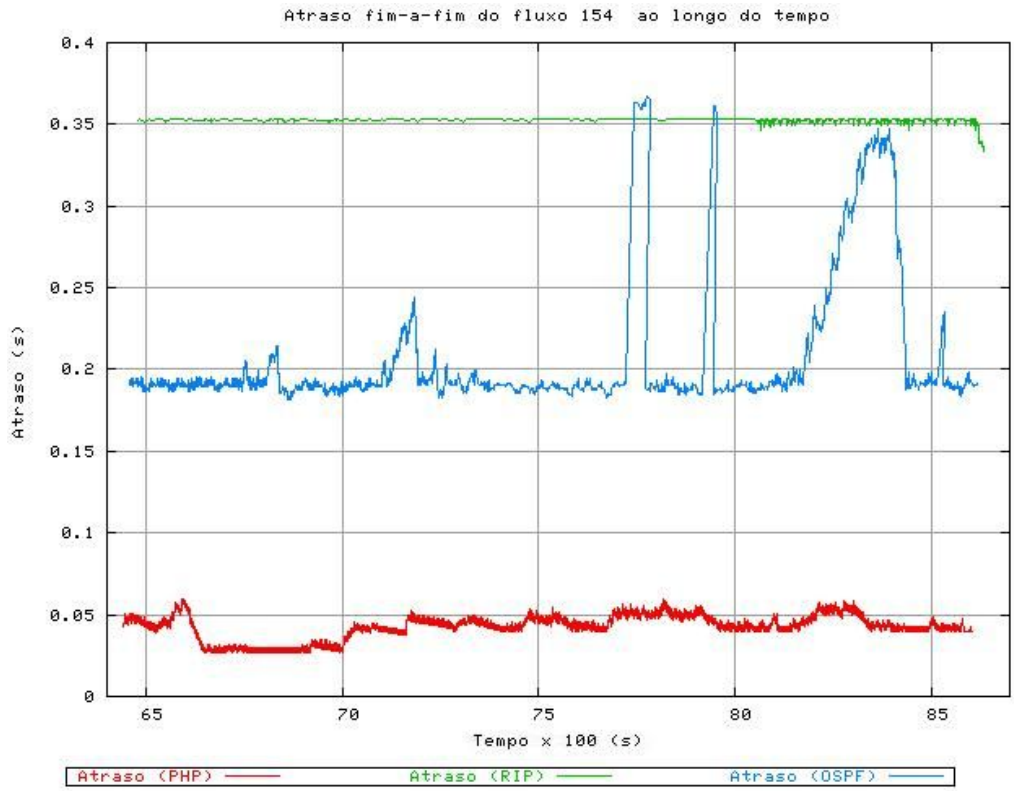

Figura 5.3: Gráfico de atraso fim a fim de um fluxo amostrado em três simulações diferentes

A figura 5.3 apresenta o resultado da medição de atraso fim a fim de uma amostra de fluxo em três simulações: utilizando o algoritmo proposto pelo estudo de caso (chamado de PHP), o protocolo RIP e o protocolo OSPF. Os resultados do módulo de análise de desempenho plotados no gráfico indicam claramente que, para o fluxo amostrado, o algoritmo PHP (em vermelho) possui menor atraso quando comparado aos algoritmos RIP (em verde) e OSPF (em azul).

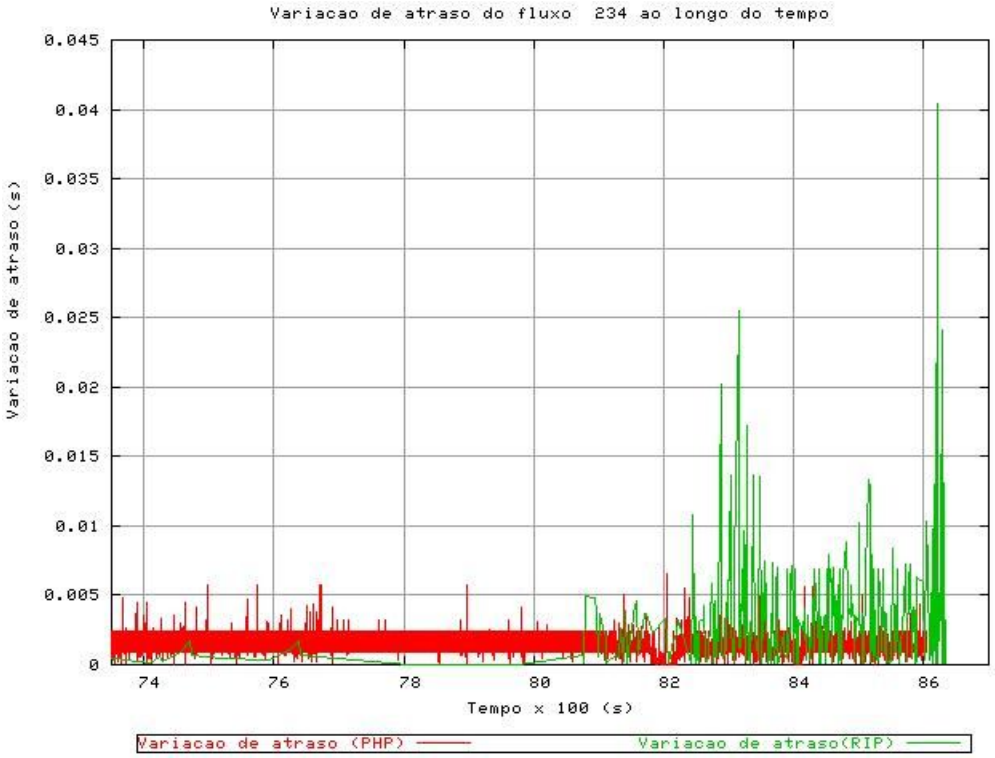

Figura 5.4: Gráfico de *jitter* fim a fim de um fluxo amostrado em duas simulações diferentes

A exemplo do gráfico de atraso, a figura 5.4 apresenta o gráfico de *jitter* fim a fim do fluxo de dados identificado pelo número 234 ao longo do tempo em duas simulações. Nota-se que em ambas as simulações o *jitter* se mantêm abaixo de 0,005 segundo, entretanto a partir do 0,82 segundo de simulação esta marca é ultrapassada na simulação com o protocolo RIP (em verde).

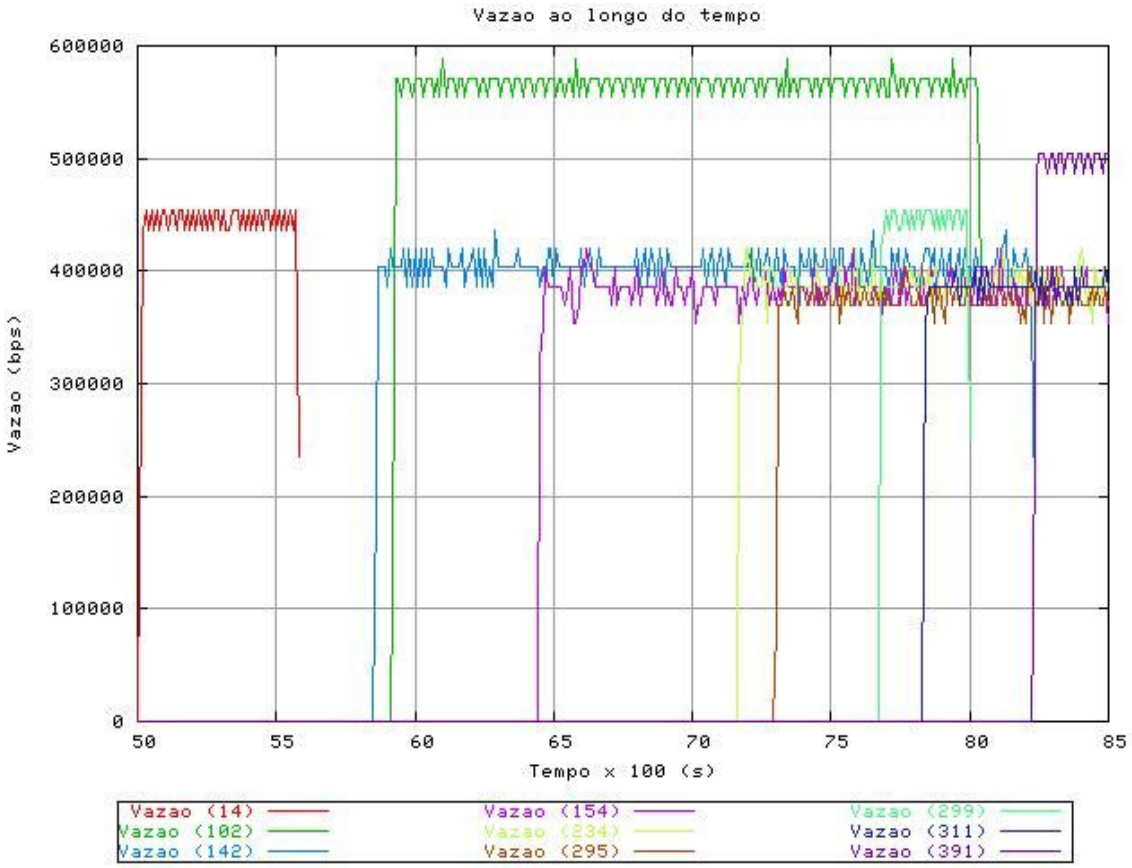

Figura 5.5: Gráfico de vazão de diversos fluxos em uma simulação

A figura 5.5 contém o gráfico de vazão de diversos fluxos de dados em uma simulação. Nota-se que os fluxos de dados iniciam e terminam em momentos diferentes – conforme a configurados no arquivo “flux”. O módulo de análise de desempenho permite este nível de detalhamento do comportamento do fluxo na simulação, sendo possível verificar se a vazão final medida para cada um dos fluxos é a mesma vazão inicial injetada na rede.

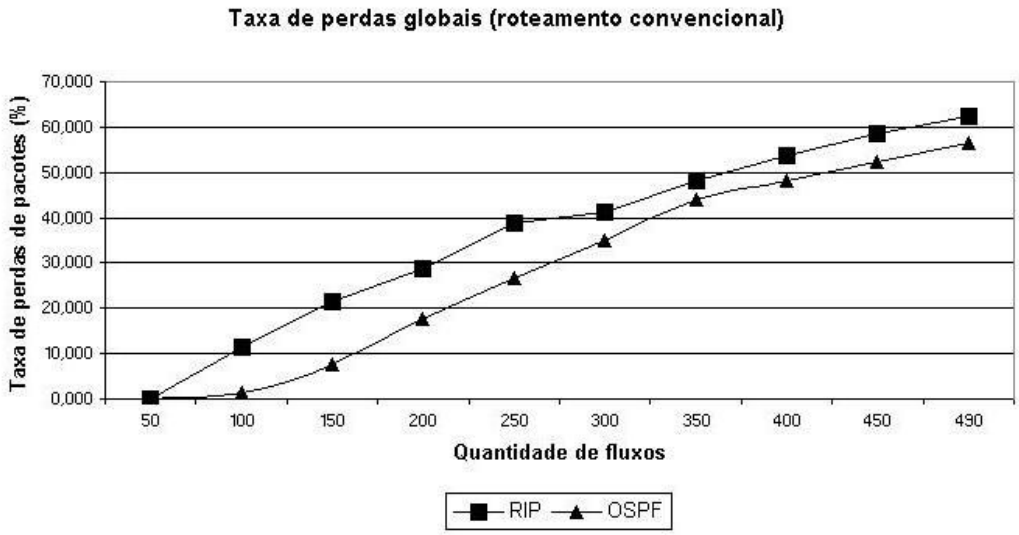

Figura 5.6: Gráfico de índice de perdas para simulações com RIP e OSPF

Finalmente, é apresentado o gráfico de índice de perdas na figura 5.6. As duas linhas do gráfico se referem a medições em duas simulações: uma com RIP e outra com OSPF. No gráfico, o índice de perdas que cresce na medida que mais fluxos são injetados nas simulações, mantendo sempre o melhor desempenho do protocolo OSPF.

Este estudo de caso serviu como teste de aceitação para o NS4D. Este, por sua vez, alcançou o resultado esperado, tanto na melhoria da usabilidade do ambiente de simulações quanto na execução eficaz das tarefas a que se propôs. Resumidamente, o NS4D alcançou seu objetivo, pois ampliou os horizontes e possibilitou ao trabalho que o empregou alcançar seus objetivos.

Capítulo 6

Conclusão

Como forma de suprir algumas das deficiências do ns-2, a proposta de uma ferramenta para facilitar a elaborações de simulações de redes foi materializada no NS4D (*Network Simulator for Dummies*). Esta ferramenta, além de tornar a interação com o simulador mais amigável, ampliou a sua carteira de usuários, já que não é mais necessário conhecer programação para realizar simulações.

O objetivo de criar uma ferramenta que facilite a definição de caminhos para fluxos de dados na rede foi alcançado com o módulo de configuração MPLS. Com ele, o usuário tem total controle sobre o comportamento da carga de trabalho, já que os fluxos podem ter suas rotas explicitamente configuradas.

O módulo Plus do NS4D tornou possível estender o ns-2, atacando de forma transparente para o usuário funcionalidades não previstas na sua arquitetura. Este módulo permite inserir atraso de processamento nos nodos de uma rede, fazer reserva de recursos para fluxos específicos e agregações de fluxos com características semelhantes.

O NS4D também permite o acoplamento do ns-2 com implementações isoladas de algoritmos de roteamento. Esta tarefa é realizada pelo módulo de roteamento. Com ele é possível configurar as rotas do simulador de acordo com o que é definido por uma implementação externa e isolada de um algoritmo de roteamento.

Requisitos para qualquer ambiente de simulação, ferramentas para geração de fluxos de dados em massa e para análise de quantitativa das simulações são implementadas, respectivamente pelos módulos de geração de carga de trabalho e de análise de desempenho. A geração de carga de trabalho capacita ao pesquisador utilizar estratégias neutras e aleatoriamente controladas para definir seus fluxos de dados. O módulo de análise de desempenho completa o ciclo da simulação. Este módulo dá ao usuário o *feedback* esperado da simulação, materializado pelos parâmetros: vazão, índice de perdas, atraso e *jitter*.

6.1 – Contribuições

O conhecimento agregado neste trabalho é uma conquista de anos de experiência com simulações de redes e o maior fruto disto é o NS4D. Este vem sendo amplamente utilizada em trabalhos relacionados com simulação de redes. Não obstante, mesmo antes da sua publicação, já estão certas algumas referências e citações ao presente trabalho.

O que antes era viável apenas para pesquisadores com conhecimento aprofundado e redes de computadores e programação, agora é possível para uma comunidade muito mais abrangente. Basta conhecer o domínio da solução para realizar simulações e obter os resultados desejados. O uso do NS4D em conjunto com o ns-2 amplia os horizontes das pesquisas em redes de computadores.

De forma particular, cada um dos módulos desenvolvidos tiveram seu papel na contribuição social. O núcleo gerador de scripts em conjunto com os módulos de configuração, possibilitou a criação de scripts OTcl de forma simples e configurável. O módulo MPLS ampliou este benefício aos pesquisadores desta área. O gerador de carga implementou estratégias neutras para geração de carga de trabalho, requisito para qualquer ambiente experimental. Transformações criativas possibilitaram a implementação de novas funcionalidades ao ns-2 sem afetar a sua estrutura, no módulo Plus. Da mesma forma, a essencial tarefa de analisar quantitativamente os resultados da simulação não era facilmente realizada apenas com o ns-2. O módulo de roteamento, por sua vez, agregou ainda mais valor ao ambiente de simulação ao permitir o uso de qualquer algoritmo de roteamento, independente da integração com o simulador.

Também vale salientar, que o estudo do ns-2 proporcionou ao autor do presente trabalho conhecimentos com demanda crescente dentro da comunidade acadêmica. O presente autor tornou-se capacitado a realizar diversos tipos de simulações com o ns-2. Estas aptidões foram empregadas no suporte e consultoria a diversos outros projetos, tanto na comunidade local quanto remotamente em outros centros de pesquisa espalhados pelo mundo. De forma direta, portanto, foi possível interagir com outros pesquisadores, contribuindo para o desenvolvimento da comunidade científica.

6.2 – Trabalhos Futuros

Como em todo trabalho de desenvolvimento de software, sempre há o que melhorar ou o que agregar ao produto. Como a própria prática de *eXtreme Programming* tem como regra, dada uma lista de funcionalidades, são relacionadas diversas tarefas em ordem de prioridade. Como era de se esperar, algumas funcionalidades de baixa prioridade restaram a ser desenvolvidas, entre elas:

- Módulo GUI (*Grafical User Interface*): como forma de investir ainda mais na questão da usabilidade do ns-2 em conjunto com o NS4D, uma interface gráfica deverá ser desenvolvido. Este módulo terá como principal tarefa facilitar a criação dos arquivos “topo”, “flux” e “ns4d.cfg”. O módulo de interface gráfica também deverá se comunicar com o núcleo do NS4D para apresentar mensagens de sucesso e erros

da aplicação. Além disso, deve-se apresentar graficamente os elementos que compõem as simulações e fornecer meios de configurá-las de forma amigável;

- Arquitetura Cliente/Servidor: em conjunto com o módulo GUI, existe a proposta de desenvolvimento de uma *middleware* de suporte a execução distribuída. Utilizando a arquitetura cliente/servidor, com o cliente acoplado ao módulo GUI e o servidor junto ao restante do NS4D, este módulo possibilitará que diversos clientes leves executem as simulações sem se preocupar com a instalação do ns-2 e com o seu consumo de memória;

- Módulo de *DiffServ*: semelhante ao módulo MPLS, o módulo *DiffServ* capacitará o núcleo do NS4D a gerar código OTcl para configurações de simulações empregando o conceito de diferenciação de serviços;

- Módulo *Wireless*: o NS4D da forma como foi projetado é útil apenas para simulações com redes *wired*. O ns-2 entretanto implementa uma série de funcionalidade de redes *wireless*, entre elas roteamento *ad hoc*, movimentação dos nodos, etc. Este módulo, portanto, possibilitará ao núcleo do NS4D gerar código OTcl para configurar simulações *wireless*.

Outro artefato importante que deve ser considerado no desenvolvimento de software para uso de terceiros é a confecção de um manual para o usuário. Este manual deve a descrição detalhada das funcionalidades e interfaces, além de conter roteiros para um bom aproveitamento do software em tarefas cotidianas, como realizar uma simples simulação.

Satisfazendo mais uma das práticas da metodologia adotada (*eXtreme Programming*) deve-se considerar os testes unitários. Completando a lista de possíveis trabalhos futuros, a elaboração de testes unitários, apesar do NS4D ter passado pelos testes de aceitação, melhoram a qualidade do software, transmitem ao desenvolvedor tranqüilidade e a garantia que o sistema funciona sem erros e da forma como esperado. Para tanto, é prevista a utilização da CUnit - uma biblioteca de funções que auxiliam a elaboração de testes unitários e C e C++.

6.3 – Comentários Finais

O sentimento deixado por este trabalho é de que se pode contribuir com o desenvolvimento tecnológico tornando viável a realização de tarefas antes encaradas como de grande esforço. Ferramentas como o NS4D abstraem a complexidade de softwares de pesquisa, o que impulsiona a comunidade científica a um novo patamar de possibilidades. Quanto maior a abstração, maior é a produtividade e a capacidade de se pensar globalmente. Que venha o “NS4D *for dummies*”!

Anexo A - Glossário

- API: *Application Programming Interface*;
- AWK: Nome da linguagem de programação composto pelos sobrenomes dos seus autores: Aho, Weinberger e Kernigham;
- CR-LDP: *Constraint Rounting - Label Distribution Protocol*;
- DV: *distance-vector*;
- ET: Engenharia de Tráfego;
- FTP: *File Transfer Protocol*;
- GCC: *GNU Compiler Collection*;
- HTTP: *Hiper Text Transfer Protocol*;
- IDE: *Integrated Development Environment*;
- IEEE: *Institute for Electrical and Electronics Engineers*;
- IP: *Internet Protocol*;
- IS-IS: *Intermediate System to Intermediate System*;
- LS: *link-state*;
- LSP: *Label Switched Path*;
- MPLS: *Multiprotocol Label Switching*;
- Nam: *Network Animator*;
- ns-2: *Network Simulator* versão 2;
- NS4D: *Network Simulator for Dummies*;
- OSPF: *Open Shortest Path First*;
- Otcl: *Object Tool Command Language*;
- POSIX: *Portable Operating System Interface*;
- QoS: *Quality of Service* - qualidade de serviço;
- RIP: *Routing Information Protocol*;
- SSFNet: *Network Scalable Simulation Framework*;
- TCL: *Tool Command Language*;
- TCP: *Transmission Control Protocol*;
- UDP: *User Datagram Protocol*;
- VoIP: voz sobre IP;
- XML: *eXtensible Markup Language*;
- XP: *eXtreme Programming*;

Anexo B – Artigo

Uma Ferramenta de Suporte a Simulação de Redes com o ns-2

Adriano Orlando Campestrini
campes@gmail.com

1 – Introdução

A emergente demanda por infraestrutura e recursos de largura de banda trazem novos desafios a projetistas e pesquisadores da área de redes de computadores. No contexto das simulações de redes por computador, o ns-2 (*Network Simulator* versão 2) se destaca como uma poderosa ferramenta que permite configurar e o monitorar cenários fictícios utilizando implementações de diversos algoritmos e serviços, principalmente, da arquitetura TCP/IP. Nesta ferramenta, as simulações são geradas em scripts na linguagem OTcl (uma linguagem orientada a objetos baseada em TCL - *Tool Command Language*) e a resposta do simulador é inserida em arquivos de log (*trace files*).

O presente trabalho visa fornecer aos usuários do ns-2 uma ferramenta que facilite a sua utilização, desde a criação dos scripts OTcl e a geração da carga de trabalho até a análise quantitativa do resultado das simulações. A ferramenta desenvolvida também tem como objetivo facilitar a criação de simulações utilizando alguns recursos avançados do ns-2, entre eles configurar explicitamente rotas fim a fim e utilizar protocolos de roteamento convencionais. Além disso, é objetivo da ferramenta estender algumas funcionalidades ao simulador, como: atraso de processamento nos nodos, interação do simulador com implementações isoladas de algoritmos de roteamento e algoritmos para mensuração de parâmetros de QoS (Qualidade de Serviço) dos ambientes simulados.

1.1 – Objetivos

O objetivo principal do presente trabalho é desenvolver uma ferramenta para facilitar simulações de redes usando o ns-2, tornando a interação com o simulador mais amigável para o usuário.

2 – Fundamentos de Simulação de Redes de Computadores

Este capítulo apresenta brevemente os conceitos de redes de computares utilizados no decorrer do trabalho. Em seguida, apresenta-se o conceito de simulação de redes de computadores e a ferramenta ns-2 (*Network Simulator* versão 2). Finalmente, com o objetivo de justificar a adoção do ns-2, apresenta-se um estudo comparativo entre os principais simuladores de redes utilizados no mercado.

2.1 – Fundamentos de Redes de Computadores

A Internet é uma rede de computadores pública mundial na qual milhares de equipamentos estão conectados (Kurose 2003). Todos os equipamentos conectados a ela são chamados de *hosts*, ou sistemas finais. Os *hosts* se comunicam através de protocolos de envio e recebimento de informações. Dentre estes protocolos, o TCP (*Transmission Control Protocol*) e o IP (*Internet Protocol*) são os mais importantes e, por isto, a família de protocolos utilizados na Internet é comumente conhecida como TCP/IP. Os *hosts* são conectados entre si através de enlaces de comunicação e a velocidade de transmissão destes enlaces, medida em bits por segundo (bps), é chamada de largura de banda.

Aplicações de rede são “a razão de ser” das redes de computadores, inclusive da Internet. Apenas com a possibilidade de implementar aplicações distribuídas úteis é que surgiu a necessidade de projetar algoritmos, equipamentos e protocolos de rede. Grande parte de uma aplicação de rede abrange o tratamento do seu protocolo de comunicação. Dentro do conceito da Internet e, mais especificamente, das redes TCP/IP estes protocolos são os chamados protocolos da camada de aplicação.

2.2 – O Network Simulator

Simulação de redes de computadores é uma técnica muito utilizada para avaliação de desempenho de sistemas em geral. Quando a rede a ser avaliado não está disponível, caso comum em muitas situações, uma simulação é o caminho mais fácil de prever o comportamento ou comparar soluções alternativas.

Esta sessão apresenta o simulador de redes ns-2 (VINT 2005), o *Network Simulator* versão 2, que está sendo desenvolvido dentro do projeto VINT, por algumas universidades e por centros de pesquisa norte americanos. O ns-2 possui funcionalidades específicas para simular redes de computadores com diversos protocolos e serviços comuns na internet, o que faz dele uma poderosa ferramenta para configurar simulações complexas e também para comparação de resultados de pesquisas.

Dada a dificuldade da obtenção e configuração de equipamentos adequados para o experimento de pesquisadores da área de redes, o ns-2 torna-se uma importante ferramenta para validar estudos neste segmento. O ns-2 também provê avaliações sucintas de topologias de redes comerciais. Neste sentido, consultores e projetistas de redes poderão usufruir do ns-2 em seus diagnósticos. Da mesma forma, os investimentos em redes corporativas poderão ser direcionados de forma mais segura e eficaz.

3 – Uma Abordagem para Utilização Amigável do ns-2

Este capítulo tem como objetivo descrever a problemática do trabalho, incluindo a descrição do problema atacado e a justificativa da solução proposta. Além disso, é apresentado o ambiente utilizado para o desenvolvimento da solução.

3.1 – Definição do Problema

A utilização de uma interface adequada deve ser uma das principais preocupações no desenvolvimento de qualquer software que interaja com algum usuário. Da mesma forma, simuladores de redes com interfaces inadequadas podem gerar problemas relacionados a falhas no modelo simulado e até ao tempo de desenvolvimento de uma simulação.

A forma com que um software apresenta seus resultados é a sua interface de saída e a esta se aplicam requisitos semelhantes à interface de entrada. A interface de saída do ns-2 deve apresentar seus resultados de forma a satisfazer as necessidades e expectativas do usuário. Ao realizar uma simulação, o ns-2 possibilita a criação de uma animação a ser apresentada pelo Nam. Esta animação apresenta a movimentação dos pacotes sobre os enlaces e nodos da topologia, entretanto ignora a medição estatística de características da simulação. Apenas com as ferramentas disponíveis pelo ns-2, não é possível realizar análises de quesitos de QoS, como vazão, índice de perdas, atraso e variação de atraso (*jitter*).

Outra forma com que o ns-2 mostra seus resultados é através do arquivo *trace*, que contém o registro de todos os eventos ocorridos durante a simulação. Este arquivo de *trace*, entretanto, não possui nenhuma informação tratada, apenas dados relacionados a eventos como: a chegada de um pacote em um nodo, a saída de um pacotes de uma fila, etc. O ns-2 permite configurar *traces* específicos para uma fila ou para a simulação como um todo. Em ambas as situações, cada linha do arquivo representa um evento e segue o seguinte formato:

```
AÇÃO TEMPO ORG DST TIPO TAM FLAGS FID END_ORG END_DST SEQ PID
```

Listagem 3.1: Sintaxe do arquivo *trace* gerado pelo ns-2

13. A listagem 3.1 mostra em seqüência a denominação das colunas de cada linha no arquivo *trace*.

Percebe-se que existe uma frustração do usuário final ao realizar uma simulação e ter com resposta um arquivo de milhares de linhas contendo diversos símbolos que só fazem sentido no ns-2. Este arquivo de *trace*, mesmo em pequenas simulações, costuma ter dezenas de *kilobytes*, chegando a *gibabytes* em simulações de grande porte.

As deficiências nas interfaces de entrada e saída do simulador são ainda mais perceptíveis quando da realização de simulações de grande porte. Nestes casos os scripts OTcl que descrevem a simulação chegam a ter mais de 25.000 (vinte e cinco mil) linhas, sendo que a partir de 300 linhas os scripts começam a se tornar complexos e de difícil manutenção. Outra limitação encontrada no ns-2 quando em simulações de grande porte é a capacidade de processamento. Como descrito na sessão 3.4 sobre comparação de simuladores, o ns-2 ocupa muito espaço na memória e quando não há mais memória disponível a simulação não pode ser realizada, sendo abortada no momento em que a memória se esgota.

3.2 – A solução proposta

Os problemas apresentados na sessão anterior, sobretudo com relação à usabilidade do ns-2, motivaram a criação de uma ferramenta que minimize ou elimine:

- A dificuldade de aprendizado e utilização do simulador;
- A complexidade de elaborar simulações de grande porte;
- A falta de clareza e objetividade dos resultados.

O presente trabalho propõe a criação de uma ferramenta que torne mais amigável a utilização do ns-2, especificamente as tarefas acima relacionadas.

Muitos projetos de pesquisa e desenvolvimento se iniciam dispostos a implementar novos módulos e funcionalidades ao simulador, mesmo sem ter realmente uma visão sistêmica dos processos que ocorrem no ns-2. Tendo este conhecimento, o presente trabalho atua sobre uma camada independente da estrutura interna do simulador, tornando a ferramenta proposta mais portável em relação às diferentes versões e releases do *Network Simulator*.

Devido à dificuldade no aprendizado da linguagem OTcl, a solução proposta atua como um gerador de scripts, tornando transparente a API do simulador. Para realizar esta tarefa, a ferramenta proposta deve possuir alguns mecanismos simples de configuração da simulação. A idéia é que a descrição de toda a simulação possa ser feita por de forma simples e objetiva. A descrição da simulação, portanto, pode ser divida em 3 (três) etapas:

- Definição da topologia física: descreve o cenário da simulação, abrangendo os nodos, enlaces e suas configurações, entre elas a largura de banda, a política de enfileiramento e o atraso inserido pelo enlace;

- Definição dos fluxos de dados: relação de todos os fluxos de dados que serão inseridos na simulação, incluindo detalhes como o protocolo do fluxo de dado, o tipo de distribuição da amostragem, o tamanho dos pacotes, a vazão inicial dos fluxos e endereços de origem e destino;

- Configurações gerais: configurações referentes à simulação em geral e à própria ferramenta proposta, como o protocolo de roteamento utilizado, o tempo de simulação, o nível de log do gerador, alguns tratamentos especiais e quais parâmetros se desejam analisar na simulação.

Com a proposta bem especificada e os riscos avaliados, pode-se iniciar o desenvolvimento da ferramenta. Para um melhor embasamento sobre como gerenciar o projeto de desenvolvimento do software, diversas metodologias foram estudas e algumas de suas práticas adotadas. O estudo e a escolha das práticas estão descritos na próxima sessão.

3.3 – A metodologia de desenvolvimento adotada

Recentemente, surgiu um forte movimento de apoio às metodologias de desenvolvimento ágil (Martin 2002). Estas metodologias procuram encontrar maneiras melhores de desenvolver software atacando, principalmente, a falta de agilidade das metodologias de desenvolvimento tradicionais. Neste sentido, as metodologias ágeis têm como lema valorizar:

- Indivíduos e interação mais que processos e ferramentas;

- Software em funcionamento mais que documentação abrangente;

- Colaboração com o cliente mais que negociação de contratos;

- Responder a mudanças mais que seguir um plano.

Dentre as metodologias ágeis, XP (*Extreme Programming*) é uma metodologia ágil para equipes pequenas e médias desenvolvendo software com requisitos vagos e em constante mudança. XP é baseada em mínimo de documentação e máximo de comunicação. Seu principal objetivo é reduzir o custo de modificações no software ao longo do ciclo de vida do projeto, possibilitando respostas rápidas ao usuário. Respostas rápidas significam que novas funcionalidades devem estar em funcionamento o mais rápido possível.

Os requisitos do presente projeto mudam na medida em que novas necessidades surgem ou novas possibilidades são visualizadas, não sendo possível realizar estimativas em longo prazo. Por este motivo, foi descartada a utilização integral de qualquer metodologia de desenvolvimento tradicional, com base em longas etapas de especificação. Foram adotadas, portanto, di-

versas práticas da metodologia XP (a listagem de práticas abaixo se refere a um suposto cliente, pois o presente projeto foi desenvolvido como parte de uma tese de doutorado - vide capítulo 4), entre elas:

- Iterações curtas: o software é entregue em pequenos *releases*. Com esta prática, o cliente obtém retorno o mais cedo possível e os riscos de migração são minimizados. Esta é a prática que melhor exprime o sentimento de XP sobre dar respostas rápidas às mudanças de requisitos;

- Cliente sempre disponível: o cliente está sempre disponível, não somente para auxiliar, mas para fazer parte da equipe resolvendo dúvidas, alterando o escopo de uma iteração, redefinindo prioridades e detectando possíveis erros o mais cedo possível;

- Jogo de planejamento: cada iteração é como se fosse uma etapa de um jogo. O objetivo deste jogo é colocar em produção as funcionalidades de maior valor/prioridade possível em menos tempo;

- Testes de aceitação: são critérios definidos pelo cliente para aceitação de software. Ao final de cada iteração um novo release, antes de ser lançado, deve passar por estes testes;

- Implementação simples: o software é implementado da forma mais simples possível para satisfazer as necessidades do cliente;

- Metáfora: o desenvolvedor se comunica com outros membros do projeto através de metáforas. A Metáfora é uma poderosa ferramenta de comunicação que elimina rapidamente a complexidade da idéia e constrói a informação para o interlocutor;

- *Stand Up Meeting*: reuniões rápidas no início de cada dia de trabalho;

- Otimizações por último: Tarefas de otimização de funcionalidades são realizadas apenas quando sobra tempo, a não ser que façam parte dos requisitos dos testes de aceitação.

A cada início de iteração, diversas tarefas são selecionadas e ordenadas por prioridade. As iterações não devem passar de duas semanas e deve-se ter o controle sobre o tempo estimado e o realizado de cada tarefa. Ao final de uma iteração, as práticas de XP possibilitam ter

novas funcionalidades prontas para o uso. Está resposta rápida à demanda do usuário ocorre sem grandes perdas de tempo em planejamento ou documentação de software - que nem sempre é bem aproveitada.

Para um bom andamento do projeto, a adoção de práticas de metodologias de desenvolvimento deve ser complementada com a utilização de ferramentas que automatizam tarefas rotineiras no desenvolvimento de software. Neste sentido, a questão do ambiente de desenvolvimento é abordada na sessão seguinte.

4 – NS4D. Network Simulator for Dummies

A proposta descrita no capítulo anterior tornou-se realidade com o desenvolvimento do *Network Simulator for Dummies*, o NS4D. O presente capítulo tem como objetivo descrever as funcionalidades e a forma como cada solução foi projetada e implementada nesta ferramenta.

A figura 4.1 demonstra a estrutura interna do NS4D:

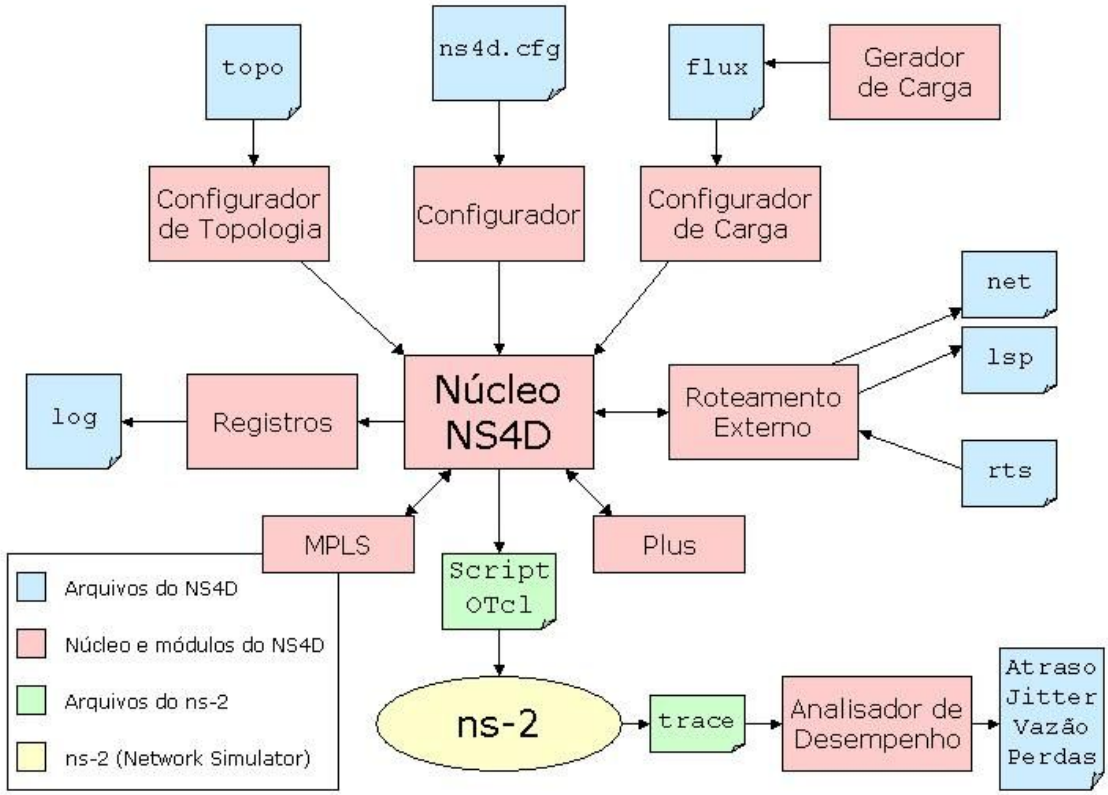

Figura 4.1: Estrutura interna do NS4D, incluindo núcleo, módulos e arquivos utilizados

Para gerar os scripts OTcl, alguns comandos dos que sempre se repetem foram automatizados pelo NS4D. Dentre eles a instanciação do objeto gerente do ns-2 e o comando de iniciar do escalonador. Em outros comandos OTcl variam apenas os valores informados, como a definição do algoritmo de roteamento, a habilitação dos arquivos de *trace* do ns-2 e configuração do tempo de simulação.

A tarefa de gravar informações sobre o comportamento da execução de um programa é a melhor forma para o usuário entender o seu funcionamento. Um dos grandes problemas no ns-2 é a falta de feedback ao usuário justamente quanto ao andamento da execução. Ao realizar uma simulação, em caso de sucesso o ns-2 não apresenta ao usuário nenhum tipo de informação e, em caso de erro, nem sempre as mensagens são significativas.

O módulo de registro de log do NS4D registra quais nodos estão sendo configurados, quais enlaces entre são criados e quais fluxos são injetados na rede. Também é fornecida ao usuário, como forma de controle, todas as configurações extraídas pelo NS4D do arquivo de configuração. Em simulações complexas ou computadores antigos, as simulações podem ser lentas e, nestes casos, o acompanhamento do arquivo de log é útil como *feedback* ao usuário sobre a execução da ferramenta. Erros de configuração podem ser rapidamente identificados através de registros no arquivo de log, por exemplo:

Nenhuma rede de computador é simulada sem ter um cenário definido. Nenhum fluxo de dados pode trafegar através de uma rede sem esta rede possuir nodos e enlaces. Como não poderia ser diferente, as simulações realizadas pelo ns-2 devem obrigatoriamente definir estes elementos da simulação.

```
set primeiro_nodo [$ns node]
set segundo_nodo  [$ns node]
set terceiro_nodo [$ns node]
```

Listagem 4.1: Trecho de script OTcl contendo a declaração de nodos

A listagem 4.1 mostra como os nodos são definidos na API OTcl do ns-2. Nela são definidos 3 (três) nodos com os nomes “primeiro_nodo”, “segundo_nodo” e “terceiro_nodo”. Pode-se dizer que, cada um deles, é instanciado através da classe node.

Analogamente a uma rede real de computadores, os nodos, ou roteadores, devem ser conectados através de enlaces. No ns-2, os enlaces são encarregados de traçar uma rota entre um nodos origem e um nodo destino seguindo alguns parâmetros:

```
$ns duplex-link  $primeiro_nodo $segundo_nodo   500Kb    1s  DropTail
$ns simplex-link $segundo_nodo  $terceiro_nodo    2Mb   50ms SFQ
$ns simplex-link $terceiro_nodo $primeiro_nodo   10Mb    2ms RED
```

Listagem 4.2: Trecho de script OTcl contendo a declaração de enlaces

Observando a listagem 4.2 podemos identificar alguns novos conceitos. Cada linha representa um enlace inserido na simulação. Algumas variações foram propositalmente inseridas para podermos analisar algumas possibilidades de configuração de enlaces. Para cada enlace há definições de: tipo (bidirecional ou unidirecional), nodos de origem e destino, largura de banda do enlace, atraso do enlace e tipo de enfileiramento.

Enlaces unidirecionais (*simplex*) conectam os nodos apenas do sentido origem/destino, enquanto, enlaces bidirecionais (*full-duplex*) podem ser comparados a dois enlaces unidirecionais em sentidos opostos: origem/destino e destino/origem. As definições de largura de banda e de atraso do enlace seguem o padrão de e valor concatenado à métrica. A largura de banda aceita valores com as métricas Kb (o mesmo que Kbps), Mb (Mbps) e Gb (Gbps). O atraso dos enlaces aceita valores com as métricas s (segundo), ms (milisegundo) e us (microsegundo).

A simulação de redes sem a injeção de fluxos de dados é tão inexpressiva quanto à realização de medições em uma rede real com todos os seus computadores desligados. Um fluxo de dados é um conjunto seqüencial de pacotes que possuem a mesma origem e o mesmo destino e pertencem a uma mesma conexão da camada de transporte na arquitetura TCP/IP.

O aumento da demanda por recursos de redes intensifica a expansão das redes de computadores e gera cenários cada vez maiores a serem simulados. Na medida que a largura de banda dos enlaces e os nodos da rede aumentam, a quantidade de fluxos de dados necessária para saturá-la aumenta exponencialmente.

No intuito de facilitar ainda mais a criação de grandes quantidades de fluxos de dados, o NS4D possui um gerador de carga de trabalho. Este gerador é distribuído como uma ferramenta separada do núcleo do NS4D, portanto deve ser executada manualmente. O gerador de carga de trabalho, a partir de um arquivo de configurações, gera de forma aleatoriamente controlada um arquivo “flux” com os fluxos de dados.

A geração de carga de trabalho completa uma lacuna extremamente importante do NS4D. A definição de estratégias automatizadas para esta geração serve como base para demonstrar que as simulações realizadas pelos usuários não possuem uma carga de trabalho tendenciosa. Mesmo em pequenas simulações, com por exemplo 30 fluxos de dados, o tempo gasto para gerá-los pode inviabilizar o projeto. Este argumento é reforçado por normalmente ser necessário realizar seguidas simulações no mesmo cenário com diversos conjuntos diferentes de fluxos de dados para validá-las. Este módulo, portanto, é indispensável para qualquer ferramenta de suporte a simulação de redes.

Neste sentido, para satisfazer as expectativas do usuário quanto a uma ferramenta de suporte a simulação de redes, esta deve ser capaz de medir quantitativamente os resultados da sua simulação. Neste cenário, surgem os scripts AWK do módulo de análise, descrito na sessão seguinte.

Nas metodologias de desenvolvimento ágeis, muito se fala sobre satisfazer a expectativa do usuário quando novas versões de software são lançadas. Concomitantemente, a teoria da usabilidade prega que os resultados da execução do software devem ser os esperados pelo usuário. De nada adianta ao usuário possuir ferramentas que detalham como os dados devem ser inseridos no sistema se, após a execução, os resultados apresentados não o servem de nada.

No ambiente de simulação composto pelo ns-2 e pelo NS4D, o mesmo conceito se aplica. Existe uma ferramenta de suporte à geração de carga de trabalho, módulos de configuração e logs de registro, entretanto até a presente sessão, as únicas formas com que os resultados da simulação são comunicados ao usuário são através de dados no arquivo *trace* e de animação no Nam.

Para análise dos quatro principais parâmetros de desempenho (vazão, índice de perdas, atraso e *jitter*), três arquivos com scripts AWK foram criados:

- `delay_jitter.awk`: este script implementa o algoritmo para calcular o atraso e o *jitter* fim a fim de um fluxo de dados;

- `thput.awk`: este script implementa o algoritmo para cálculo da vazão de algum fluxo de dados. O nome “`thput`” vem da expressão *throughput*, que significa vazão em inglês;

- `indice.awk`: este script implementa o algoritmo que calcula a porcentagem de perdas na transmissão de algum fluxo de dados.

Todos os scripts extraem seus resultados a partir do arquivo de *trace* da simulação. Para o cálculo do atraso, o script AWK varre o arquivo *trace* a procura do tempo inicial e final de transmissão de um pacote na simulação. Ao obter o atraso de todos os pacotes de um fluxo faz-se uma média aritmética. O *jitter* é calculado a partir do atraso de cada pacote subtraído do atraso do pacote anterior. Novamente, o resultado é a média aritmética do *jitter* de todos os pacotes. Para o cálculo da vazão, o tamanho de todos os pacotes que chegam no seu destino é acumulado para cada segundo de simulação. O resultado é a transformação deste valor acumulado de bytes para bits, transformando a medida em bytes por segundo. O índice de perdas é taxa de descarte de pacotes em relação à vazão inicial do fluxo. O tamanho de todos os pacotes descartados é acumulado e confrontado com o tamanho de todos os pacotes que foram enviados pelo nodo de origem.

O módulo de análise de desempenho é a principal forma com que o NS4D ataca o quesito "satisfação subjetiva" da usabilidade de software. O preenchimento desta extensa lacuna deixada pelo ns-2 eleva o potencial do ambiente de simulação. Os usuários que antes se preocupavam em colher estas informações agora podem investir seu tempo propondo e validando modelos cada vez melhores.

Este módulo de análise de desempenho pode ser utilizado em qualquer simulação *wired* no ns-2, entretanto outros módulos implementam funcionalidades mais específicas. A sessão seguinte descreve o módulo que permite simular cenários com MPLS.

3.3 – Distribuição do NS4D

Portabilidade de software e implantação efetiva de sistemas são aspectos cruciais de práticas modernas de engenharia de software. É inaceitável iniciar hoje um projeto de software que tem como expectativa rodar em apenas uma plataforma. Como prova da importância da portabilidade há o recente crescimento da procura por linguagens que rodam sobre máquinas virtuais ou interpretadores, como Perl, PHP e principalmente Java. Restrições de hardware podem mudar a escolha da plataforma, novos usuários podem surgir com um novo tipo de plataforma ou uma nova versão de um sistema operacional proprietário pode trazer novidades incompatíveis. Além disso, software que possuem um mecanismo de implantação fácil e com menos chances de erro são valiosos.

Pensando desta forma, o NS4D adotou a mesma forma de distribuição que a comunidade de software livre: utilizando as ferramentas *Autoconf* e *Automake* (Vaugham 2000). Estes pacotes têm o intuito de deixar um software mais portável e simplificar a compilação e implantação dele.

Autoconf é uma ferramenta que torna softwares mais portáveis através da realização de testes para descobrir características do sistema antes do software ser compilado. O código fonte do software pode, então, se adaptar a diferentes características.

Automake é uma ferramenta para gerar *Makefile's* (arquivos que descrevem o que compilar) dentro de diversos padrões. *Automake* simplifica o processo de descrever a organização de um software e realiza funções como a checagem e ligação de dependências entre os arquivos de código fonte.

A utilização destas duas ferramentas reduz o esforço em duas etapas do projeto: no desenvolvimento e na instalação do software. Ao desenvolvedor não é mais necessário criar scripts para compilação e implantação da ferramenta. Já ao usuário final, basta ter acesso a um arquivo compactado via internet ou qualquer outra fonte, em seguida descompactá-lo, e rodar os scripts `configure` e `Makefile` – com isto, o NS4D estará instalado.

Como prova de sucesso no desenvolvimento do NS4D, o capítulo seguinte narra um estudo de caso que valida o emprego desta ferramenta como uma ferramenta amigável de suporte a simulação de redes com o ns-2.

5 – Estudo de Caso

Como forma de verificar a eficiência do NS4D conforme proposto, este capítulo apresenta os resultados de um experimento que empregou a ferramenta. Este experimento é parte das atividades realizadas durante a elaboração da tese de doutorado do professor Roberto Alexandre Dias (Dias 2004b)

O objetivo do experimento foi testar o desempenho de um algoritmo de roteamento global baseado em heurísticas e confrontá-lo com o desempenho de algoritmos de roteamento convencionais. Também foi acoplado ao módulo de roteamento um pacote de programação matemática – o Xpress-MP – no intuito de contrastar a capacidade do algoritmo de roteamento em estudo em encontrar a solução de mais alta qualidade.

Os experimento foram realizados utilizando um computador Pentium IV de 1.2 GHz e 256 MBytes de memória RAM. A exemplo do ambiente de utilizado para o desenvolvimento do NS4D, foi instalada a distribuição Mandrake (versão 8.2) do sistema operacional Linux.

A topologia de rede utilizada no experimento é similar à topologia utilizada por (Banerjee 2002), possui complexidade média, com 17 roteadores e 31 enlaces. Dentre os roteadores, apenas oito são geradores ou receptores de tráfego. Os enlaces foram divididos em duas categorias: enlaces de acesso, com 2kbps de largura de banda e atraso de 10ms; e (2) enlaces de núcleo, com 8kbps de largura de banda e atraso de 5ms. Esta topologia está apresentada na figura 5.1:

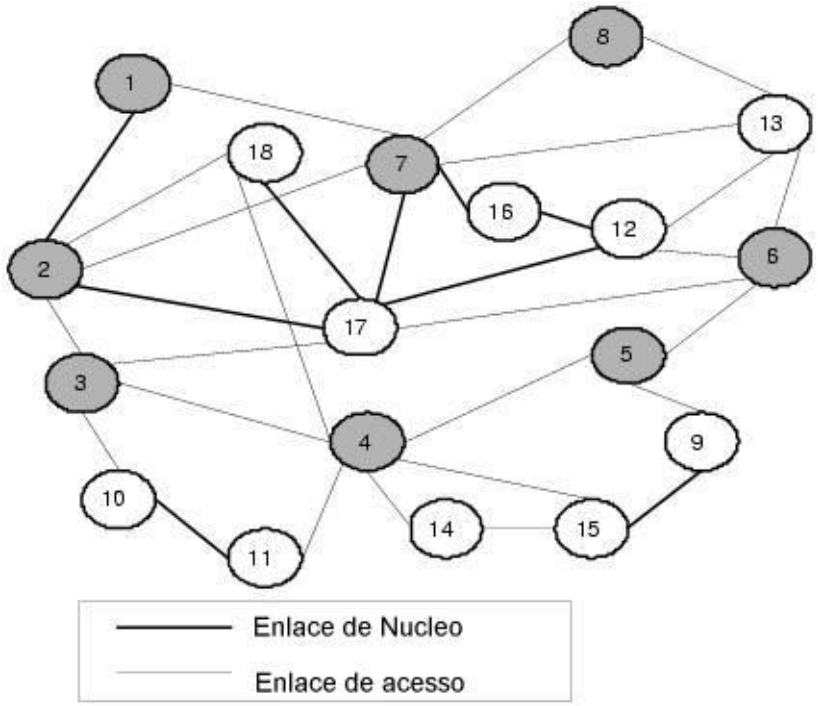

Figura 5.1: A primeira topologia de rede

Com os arquivo “topo” e “flux” prontos, o NS4D foi rodado para gerar o script OTcl. Pode-se verificar a eficácia da ferramenta quando se analisa a quantidade de linhas de código que o pesquisador não precisou escrever. Para o primeiro experimento, o mais simples, o script OTcl gerado pelo NS4D ficou com 10.750 (dez mil setecentos e cinqüenta) linhas.

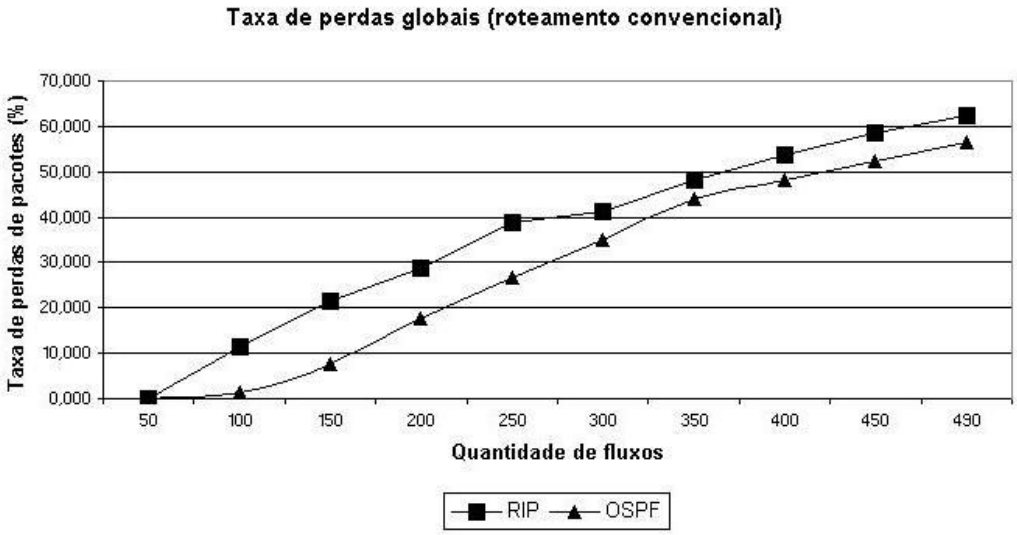

Figura 5.2: Gráfico de índice de perdas para simulações com RIP e OSPF

Na figura 5.2, é apresentado o gráfico de índice de perdas. As duas linhas do gráfico se referem a medições em duas simulações: uma com RIP e outra com OSPF. No gráfico, o índice de perdas que cresce na medida que mais fluxos são injetados nas simulações, mantendo sempre o melhor desempenho do protocolo OSPF.

Este estudo de caso serviu como teste de aceitação para o NS4D. Este, por sua vez, alcançou o resultado esperado, tanto na melhoria da usabilidade do ambiente de simulações quanto na execução eficaz das tarefas a que se propôs. Resumidamente, o NS4D alcançou seu objetivo, pois ampliou os horizontes e possibilitou ao trabalho que o empregou alcançar seus objetivos.

6 – Conclusões

O objetivo de criar uma ferramenta que facilite a definição de caminhos para fluxos de dados na rede foi alcançado com o módulo de configuração MPLS. Com ele, o usuário tem total controle sobre o comportamento da carga de trabalho, já que os fluxos podem ter suas rotas explicitamente configuradas.

O módulo Plus do NS4D tornou possível estender o ns-2, atacando de forma transparente para o usuário funcionalidades não previstas na sua arquitetura. Este módulo permite inserir atraso de processamento nos nodos de uma rede, fazer reserva de recursos para fluxos específicos e agregações de fluxos com características semelhantes.

O NS4D também permite o acoplamento do ns-2 com implementações isoladas de algoritmos de roteamento. Esta tarefa é realizada pelo módulo de roteamento. Com ele é possível configurar as rotas do simulador de acordo com o que é definido por uma implementação externa e isolada de um algoritmo de roteamento.

Anexo C – Código Fonte

C.1 – hepta.c

```
/*********************************************************
    Este script gera código para o ns a partir de 2 arquivos com
tabelas de entrada


 -Formato dos arquivos de entrada;
   1. Exemplo de arquivo de topologia física:

Orig  Dest  BW          Delay       Fila        Tipo
0     1     2000kb      10ms        DropTail    duplex-link
0     6     2000kb      10ms        DropTail    duplex-link
1     2     2000kb      10ms        DropTail    duplex-link
1     6     2000kb      10ms        DropTail    duplex-link

   2. Exemplo de arquivo de topologia lógica ( definição dos fluxos ):

ID    Fonte Destino     Cliente     T_start     T_stop      TamPkt
      Atraso      Setup Holding     N[1]  N[2]  N[3]  N[4]  N[5]  N[6]
      N[7]  Agreg
1     7     0     12    2213  2724  210   124   1     2     1     3
      8     15    24    34    48    0
2     2     5     11    4185  4659  210   132   1     2     2     11
      25    45    70    101   139   0
3     1     7     20    2822  2843  210   140   2     4     2     11
      25    45    70    101   139   0
4     3     6     24    6489  6957  210   104   1     2     1     3
      7     12    20    29    40    0
5     0     4     4     6132  8423  210   50    5     10    16    65
      147   262   409   589   803   0

    ps.: A primeira linha do arquivo será ignorada pois supõe-se que
exista um cabeçário para as colunas.


 -Como executar o arquivo:
   Após a compilação com gcc utilize a seguinte linha de comando:

   ./hepta <id> <plus1> <plus2> [tempo de simulação em segundos]

    onde plusX pode ser 'expl' ou 'xy', que serao executados na ordem
definida
   exemplo:
```

```
   ./hepta 1 xy 0 3

   Caso o tempo da simulação não for informado o valor default será de
5s.

*******************************************************/
#include "../include/hepta.h"

int main(int argc, char *argv[]) {

  unsigned long int tot_fluxos = 0;
  double   tempo = 0;
  int      total_nodos;
  int      nulos;
  int      i;

  // Testando argumentos de entrada
  if (argc != 2) {
    usage();
  }

  // Abrindo arquivo de log da execucao (modo append)
  util.log = fopen("log", "a");
                                                                  fprintf
(util.log,"\n###############################################\n");

  bzero(util.arquivo, sizeof(util.arquivo));
  sprintf(util.arquivo, "hepta%s.cfg", argv[1]);
  if (CfgCarga(util.arquivo))
    exit(-1);

  strcpy(util.id, argv[1]);

  /* Mapeando tarefas e fluxo da execucao */
  mapeiaPlus();
  mapeiaFluxoExecucao();

  // Lendo arquivos de entrada e inicializando biblioteca de fluxos
  carregaTopo(argv[1]);
  assert(!carregaFlux(argv[1]));

  // Define troncos (traffic trunks)
   if  (  (cfg.plus.agregacao  !=  SEM_AGREGACAO)  ||  (cfg.alg.ativo  !=
PHP_ONLINE) )
    defineTroncos();

  // Habilitando impressao TCL
  assert(!iniciaTCL(argv[1]));

  // source para rtProto ISIS
  if (cfg.alg.ativo == ISIS) {
    imprimeCfgISIS();
  }

  imprimeInicio(atoi(argv[1]));
  imprimeCores();
  if (imprimeRtproto()) exit(-1);

  // Configuracoes para ISIS
  if (cfg.alg.ativo == ISIS) {
    imprimeISIS();
```

```
  }

  // Organizando nodos
  extraiNodos();
  filtraRepeticao(cfg.topo.n_links*2, &total_nodos);
  ordenaNodos(total_nodos);

  // Criando nodos
  total_nodos = imprimeNodos(total_nodos);

  imprimeLinks();
  formataTopo();

  // Configurando MNS (MPLS for NS)
  if ((cfg.alg.ativo == PHP) || (cfg.alg.ativo == PHP_ONLINE))
    imprimeLDP();

  // Diretorio para plotagem de dados estatistica
  system("mkdir -p results/");

  switch (cfg.alg.ativo) {
    case Static:
    case DV:
    case LS:
    case ISIS:
      util.momento = cfg.duracao.simulacao;
      imprimeFluxosTRONCO(util.tot_troncos);
      imprimeFimFluxoTRONCO(util.tot_troncos);
      break;

    case PHP:
      util.momento = cfg.duracao.simulacao;

      // Alocando memoria para fluxos
      rt = (st_Rota *)malloc(sizeof(st_Rota)*util.tot_troncos+1);
      bzero(rt, sizeof(rt));

      fprintf(util.log, "Algoritmo PHP\n");

        assert((tot_fluxos = geraArqLspPHP(argv[1], util.momento)) >=
0);
      assert(!geraArqNet(argv[1], total_nodos));
      assert(!executaPHP(argv[1]));
      assert((tempo = carregaRotas(argv[1])) >= 0);

       fprintf(util.log, "Tempo de processamento do algoritmo: %f\n",
tempo);
      fflush(util.log);

      imprimeFluxosTRONCO(tot_fluxos);
      imprimeRotasTRONCO(tot_fluxos);
      imprimeFimFluxoTRONCO(tot_fluxos);
      break;

    case PHP_ONLINE:
      switch (cfg.plus.agregacao) {
        case SEM_AGREGACAO:
          // Alocando memoria para fluxos
          niveis = (st_Niveis *)malloc(sizeof(st_Niveis)+1);
                          rt   =   (st_Rota   *)malloc(sizeof(st_Rota)
*cfg.carga.n_fluxos+1);
```

```
          bzero(niveis, sizeof(niveis));
          bzero(rt, sizeof(rt));

          // Alocando memoria para estatistica de niveis
           niveis->cont_alta  = (unsigned long int *)malloc(sizeof(long
int) * cfg.carga.n_niveis+1);
           niveis->cont_baixa = (unsigned long int *)malloc(sizeof(long
int) * cfg.carga.n_niveis+1);
        for (i = 0; i <= cfg.carga.n_niveis; i++)
        {
            unsigned long int *pAlta  = niveis->cont_alta;
            unsigned long int *pBaixa = niveis->cont_baixa;
            *(pAlta+i) = 0;
            *(pBaixa+i) = 0;
        }

          util.nrodadas=0;
          if (cfg.alg.intervalo_minimo < cfg.duracao.escala)
            cfg.alg.intervalo_minimo = cfg.duracao.escala;
          else if (cfg.alg.intervalo_minimo < 50)
            cfg.alg.intervalo_minimo = 50;

           fprintf(util.log, "Algoritmo PHP_ONLINE. Intervalo_minimo=%
s\n", formataNumero(cfg.alg.intervalo_minimo));
                  util.momento               =  (unsigned  long  int)
cfg.alg.intervalo_minimo + INICIO;
          util.id_rota        = 0;
          util.reroteamentos  = 0;
          util.renivelamentos = 0;
          util.interrupcoes   = 0;
          util.alta_reroteamentos  = 0;
          util.alta_renivelamentos = 0;
          util.alta_interrupcoes   = 0;
          util.porcentagem    = -1;
          util.fim_carencia   = 0;
          util.mudanca        = 0;
          util.andamento      = 0;
          util.ultima_rodada  = 0;

          while (util.momento <= cfg.duracao.simulacao)
          {
            double   tempo = 0;
            char     buff[sizeof(util.arquivo)];

            bzero(buff, sizeof(buff));
            sprintf(buff, "%s.%012lu", argv[1], util.momento);

               assert((tot_fluxos = geraArqLsp(buff, util.momento)) >=
0);
            if (tot_fluxos && util.mudanca)    {
              util.mudanca       = 0;

                   fprintf(util.log, "\n********************RODADA(%d)
********************\n", ++util.nrodadas);
                    fprintf(util.log, "INICIO DA RODADA: %s\n", strip
(formataNumero(util.momento)));
              fflush(util.log);

              assert(!geraArqNet(buff, total_nodos));
              assert(!executaPHP(buff));
              assert((tempo = carregaRotas(buff)) >= 0);
```

```
            // Incrementa momento
            util.momento += (unsigned long int)(tempo * ESCALA);
            // Incrementa momento
            if ((tempo * ESCALA) < cfg.alg.intervalo_minimo)
                              util.momento  +=  (unsigned  long  int)
cfg.alg.intervalo_minimo - (tempo * ESCALA);

            calculaCarencia(tot_fluxos);
            imprimeFluxos(tot_fluxos);
          if (util.fim_carencia == 0)
              util.fim_carencia = util.momento;
            imprimeRotas(tot_fluxos);
            imprimeFimFluxo(tot_fluxos);
              plotaMudancas(util.reniv, util.alta_reniv, util.rerot,
util.alta_rerot, util.inter, util.alta_inter);

             fprintf(util.log, "Tempo de processamento do algoritmo:
%f\n", tempo);
                     fprintf(util.log,  "FIM  DA  RODADA:  %s\n",  strip
(formataNumero(util.momento)));
            fflush(util.log);


          }
          else {
            util.carencia = 0;
                              util.momento  +=  (unsigned  long  int)
cfg.alg.intervalo_minimo;
            imprimeFimFluxo(tot_fluxos);
          }
          atualizaPorcentagem();
        }
        fprintf(stdout, "\n");
        fflush(stdout);
        break;

    case SRC_DST:
    case CLI_DST:
        // Alocando memoria para fluxos
        niveis = (st_Niveis *)malloc(sizeof(st_Niveis)+1);
                          rt   =   (st_Rota   *)malloc(sizeof(st_Rota)
*cfg.carga.n_fluxos+1);
        bzero(niveis, sizeof(niveis));
        bzero(rt, sizeof(rt));

        // Alocando memoria para estatistica de niveis
        niveis->cont_alta  = (unsigned long int *)malloc(sizeof(long
int)*cfg.carga.n_niveis+1);
        niveis->cont_baixa = (unsigned long int *)malloc(sizeof(long
int)*cfg.carga.n_niveis+1);
        for (i = 0; i <= cfg.carga.n_niveis; i++)
        {
          unsigned long int *pAlta  = niveis->cont_alta;
          unsigned long int *pBaixa = niveis->cont_baixa;
          *(pAlta+i) = 0;
          *(pBaixa+i) = 0;
        }

        util.nrodadas=0;
        if (cfg.alg.intervalo_minimo < cfg.duracao.escala)
```

```
            cfg.alg.intervalo_minimo = cfg.duracao.escala;
          else if (cfg.alg.intervalo_minimo < 50)
            cfg.alg.intervalo_minimo = 50;

           fprintf(util.log, "Algoritmo PHP_ONLINE. Intervalo_minimo=%
s\n", formataNumero(cfg.alg.intervalo_minimo));
                    util.momento               =  (unsigned  long  int)
cfg.alg.intervalo_minimo + INICIO;
          util.id_rota        = 0;
          util.reroteamentos  = 0;
          util.renivelamentos = 0;
          util.interrupcoes   = 0;
          util.alta_reroteamentos  = 0;
          util.alta_renivelamentos = 0;
          util.alta_interrupcoes   = 0;
          util.porcentagem    = -1;
          util.fim_carencia   = 0;
          util.mudanca        = 0;
          util.andamento      = 0;

          while (util.momento <= cfg.duracao.simulacao)
          {
            double   tempo = 0;
            char     buff[sizeof(util.arquivo)];

            bzero(buff, sizeof(buff));
            sprintf(buff, "%s.%012lu", argv[1], util.momento);

            calculaDemanda();
             assert((tot_fluxos = geraArqLspTRONCO(buff, util.momento))
>= 0);
            if (tot_fluxos && util.mudanca)
          {
              util.mudanca       = 0;

                   fprintf(util.log, "\n*******************RODADA(%d)
*******************\n", ++util.nrodadas);
                    fprintf(util.log, "INICIO DA RODADA: %s\n", strip
(formataNumero(util.momento)));
              fflush(util.log);

              assert(!geraArqNet(buff, total_nodos));
              assert(!executaPHP(buff));
              assert((tempo = carregaRotas(buff)) >= 0);

              // Incrementa momento
              if ((tempo * ESCALA) > cfg.alg.intervalo_minimo)
                util.momento += (unsigned long int)(tempo * ESCALA);
              else
                                util.momento  +=  (unsigned  long  int)
cfg.alg.intervalo_minimo;

              calculaBandaEfetiva(NAO_ATUALIZA);
              imprimeNovaBanda();
              calculaBandaEfetiva(ATUALIZA);

            calculaCarenciaTRONCO(tot_fluxos);
              imprimeFluxosTRONCO(tot_fluxos);
            if (util.fim_carencia == 0)
                util.fim_carencia = util.momento;
              imprimeRotasTRONCO(tot_fluxos);
```

```
              imprimeFimFluxoTRONCO(tot_fluxos);
                plotaMudancas(util.reniv, util.alta_reniv, util.rerot,
util.alta_rerot, util.inter, util.alta_inter);

               fprintf(util.log, "Tempo de processamento do algoritmo:
%f\n", tempo);
                     fprintf(util.log, "FIM DA RODADA: %s\n", strip
(formataNumero(util.momento)));
              fflush(util.log);

            }
            else {
              util.carencia = 0;
                                util.momento += (unsigned long int)
cfg.alg.intervalo_minimo;
            calculaBandaEfetiva(NAO_ATUALIZA);
              imprimeNovaBanda();
              imprimeFimFluxoTRONCO(tot_fluxos);
            }
            atualizaPorcentagem();
          }

          fprintf(stdout, "\n");
          fflush(stdout);
          break;
      }
      imprimeFinalizacao(tot_fluxos);
      break;
    default:
          fprintf(util.log, "ERRO: Algoritmo \"%d\" desconhecido.",
cfg.alg.ativo);
      exit(-1);
  }
  free(rt);

  // Estatistica: id | start_previsto | start_efetivo
  if (cfg.alg.ativo == PHP_ONLINE) {
    plotaTempoEspera(argv[1]);
    plotaMudancasFinal();
    plotaNiveis();
    plotaReroteamentos();
    plotaRenivelamentos();
    plotaFreqEspera();
  }

  // Procedimento finish
  imprimeFinish(argv[1]);

  // Anotações/rodapeh
  imprimeRodapeh(argv[1]);

  // Rodando a simulação
  imprimeRun();

  // Fechando arquivos
  finalizaTCL();
  fclose(util.log);
  return 0;
}

void usage()
```

```
{
  perror("./hepta <id>\n");
  exit( -1 );
}
```

C.2 – cfgcarga.c

```
#include "../include/hepta.h"

int CfgCarga(char arq[])
{
  spConfigNode      *rootNode;
  spConfigAttribute *AuxAttrib;

  /* Aloca e inicializa estrutura */
  if ((rootNode=spConfigNodeNew())==NULL)
  {
    fprintf(stderr,"ERRO: falha na spConfigNodeNew()\n");
    return(-1);
  }
  /* Executa carga no arquivo */
  if (spConfigLoad(rootNode,arq))
  {
    fprintf(stderr,"ERRO: falha na carga do arquivo \"%s\"\n",arq);
    return(-2);
  }

  /* Seta o nome do nodo, sobreescrevendo o anterior, se existir */
  spConfigNodeName(rootNode,"rootNode");

  /* Preenchimento da estrutura Cfg */

  /* <CFG_GERAIS> */
  /***** <REGISTROS> *****/
            if      ((AuxAttrib      =      spConfigAttributeFindPath
(rootNode,"/CFG_GERAIS/REGISTROS/arquivo_log"))!=NULL) {
    if (AuxAttrib->type==SP_CONFIG_VALUE_INT) {
      cfg.log.arquivo_log = AuxAttrib->value.i;
    } else {
      cfg.log.arquivo_log = atoi(AuxAttrib->value.string);
    }
  } else {
              fprintf(stderr,"/CFG_GERAIS/REGISTROS/arquivo_log    nao
encontrado.\n");
    return -1;
  }

            if      ((AuxAttrib      =      spConfigAttributeFindPath
(rootNode,"/CFG_GERAIS/REGISTROS/arquivo_net"))!=NULL) {
    if (AuxAttrib->type==SP_CONFIG_VALUE_INT) {
      cfg.log.arquivo_net = AuxAttrib->value.i;
    } else {
      cfg.log.arquivo_net = atoi(AuxAttrib->value.string);
```

```
    }
  } else {
              fprintf(stderr,"/CFG_GERAIS/REGISTROS/arquivo_net    nao
encontrado.\n");
    return -1;
  }

            if      ((AuxAttrib      =      spConfigAttributeFindPath
(rootNode,"/CFG_GERAIS/REGISTROS/arquivo_lsp"))!=NULL) {
    if (AuxAttrib->type==SP_CONFIG_VALUE_INT) {
      cfg.log.arquivo_lsp = AuxAttrib->value.i;
    } else {
      cfg.log.arquivo_lsp = atoi(AuxAttrib->value.string);
    }
  } else {
              fprintf(stderr,"/CFG_GERAIS/REGISTROS/arquivo_lsp    nao
encontrado.\n");
    return -1;
  }

            if      ((AuxAttrib      =      spConfigAttributeFindPath
(rootNode,"/CFG_GERAIS/REGISTROS/arquivo_rts"))!=NULL) {
    if (AuxAttrib->type==SP_CONFIG_VALUE_INT) {
      cfg.log.arquivo_rts = AuxAttrib->value.i;
    } else {
      cfg.log.arquivo_rts = atoi(AuxAttrib->value.string);
    }
  } else {
              fprintf(stderr,"/CFG_GERAIS/REGISTROS/arquivo_rts    nao
encontrado.\n");
    return -1;
  }

  /***** <ALGORITMO> *****/
            if      ((AuxAttrib      =      spConfigAttributeFindPath
(rootNode,"/CFG_GERAIS/ALGORITMO/ativo"))!=NULL) {
    if (AuxAttrib->type==SP_CONFIG_VALUE_INT) {
      cfg.alg.ativo = AuxAttrib->value.i;
    } else {
      cfg.alg.ativo = atoi(AuxAttrib->value.string);
    }
  } else {
    fprintf(stderr,"/CFG_GERAIS/ALGORITMO/ativo nao encontrado.\n");
    return -1;
  }

  if (cfg.alg.ativo == PHP_ONLINE)
  {
                if    ((AuxAttrib    =    spConfigAttributeFindPath
(rootNode,"/CFG_GERAIS/ALGORITMO/intervalo_minimo"))!=NULL) {
      cfg.alg.intervalo_minimo = atol(AuxAttrib->value.string);
    } else {
       fprintf(stderr,"ATENCAO: /CFG_GERAIS/ALGORITMO/intervalo_minimo
nao encontrado. Padrao \"1000\"\n");
      cfg.alg.intervalo_minimo=1000;
    }
  }

  /***** <PLUS> *****/
  if ((cfg.alg.ativo == PHP_ONLINE) || (cfg.alg.ativo == PHP))
  {
```

```
                  if    ((AuxAttrib    =    spConfigAttributeFindPath
(rootNode,"/CFG_GERAIS/PLUS/tipo_execussao"))!=NULL) {
      if (AuxAttrib->type==SP_CONFIG_VALUE_INT) {
        cfg.plus.tipo_execussao = AuxAttrib->value.i;
      } else {
        cfg.plus.tipo_execussao = atoi(AuxAttrib->value.string);
      }
    } else {
         fprintf(stderr,"ATENCAO:  /CFG_GERAIS/PLUS/tipo_execussao  nao
encontrado.\n");
      cfg.plus.tipo_execussao=NORMAL;
    }

    if ((cfg.plus.tipo_execussao == XY) || (cfg.plus.tipo_execussao ==
XY_EXPL) || (cfg.plus.tipo_execussao == EXPL_XY))
    {
                      if    ((AuxAttrib    =    spConfigAttributeFindPath
(rootNode,"/CFG_GERAIS/PLUS/fator_xy"))!=NULL) {
        if (AuxAttrib->type==SP_CONFIG_VALUE_INT) {
          cfg.plus.fator = AuxAttrib->value.i;
        } else {
          cfg.plus.fator = atoi(AuxAttrib->value.string);
        }
      } else {
               fprintf(stderr,"ATENCAO:  /CFG_GERAIS/PLUS/fator_xy  nao
encontrado. Padrao = \"5\".\n");
        cfg.plus.fator=5;
      }
      if (cfg.plus.fator==0) {
         fprintf(stderr,"ATENCAO: Confira valores cd tipo_execussao e
fator. Impossivel gerar divisao xy com fator 0.\n");
        return -1;
      }
    }
  }

            if      ((AuxAttrib      =      spConfigAttributeFindPath
(rootNode,"/CFG_GERAIS/PLUS/agregacao"))!=NULL) {
    if (AuxAttrib->type==SP_CONFIG_VALUE_INT) {
      cfg.plus.agregacao = AuxAttrib->value.i;
    } else {
      cfg.plus.agregacao = atoi(AuxAttrib->value.string);
    }
  } else {
           fprintf(stderr,"ATENCAO:   /CFG_GERAIS/PLUS/agregacao   nao
encontrado.\n");
    cfg.plus.agregacao=SEM_AGREGACAO;
  }

            if      ((AuxAttrib      =      spConfigAttributeFindPath
(rootNode,"/CFG_GERAIS/PLUS/setup_prio_media"))!=NULL) {
    if (AuxAttrib->type==SP_CONFIG_VALUE_INT) {
      cfg.plus.setup_prio_media = AuxAttrib->value.i;
    } else {
      cfg.plus.setup_prio_media = atoi(AuxAttrib->value.string);
    }
  } else {
       fprintf(stderr,"ATENCAO:  /CFG_GERAIS/PLUS/setup_prio_media  nao
encontrado. Padrao = 5\n");
    cfg.plus.setup_prio_media = 5;
  }
```

```
  if (cfg.plus.agregacao) {
                  if    ((AuxAttrib    =    spConfigAttributeFindPath
(rootNode,"/CFG_GERAIS/PLUS/trata_atraso"))!=NULL) {
      if (AuxAttrib->type==SP_CONFIG_VALUE_INT) {
        cfg.plus.trata_atraso = AuxAttrib->value.i;
      } else {
        cfg.plus.trata_atraso = atoi(AuxAttrib->value.string);
      }
    } else {
           fprintf(stderr,"ATENCAO:  /CFG_GERAIS/PLUS/trata_atraso  nao
encontrado. Padrao = MINIMO\n");
      cfg.plus.trata_atraso = MINIMO;
    }

                  if    ((AuxAttrib    =    spConfigAttributeFindPath
(rootNode,"/CFG_GERAIS/PLUS/trata_prioridade"))!=NULL) {
      if (AuxAttrib->type==SP_CONFIG_VALUE_INT) {
        cfg.plus.trata_prioridade = AuxAttrib->value.i;
      } else {
        cfg.plus.trata_prioridade = atoi(AuxAttrib->value.string);
      }
    } else {
        fprintf(stderr,"ATENCAO: /CFG_GERAIS/PLUS/trata_prioridade nao
encontrado.\n");
      cfg.plus.trata_prioridade = MAXIMO;
    }
  }

  /***** <DURACAO> *****/
            if      ((AuxAttrib      =      spConfigAttributeFindPath
(rootNode,"/CFG_GERAIS/DURACAO/timeout_rejeicoes"))!=NULL) {
    if (AuxAttrib->type==SP_CONFIG_VALUE_STRING) {
           cfg.duracao.timeout_rejeicoes  =  (unsigned  long  int)atol
(AuxAttrib->value.string);
    } else if (AuxAttrib->type==SP_CONFIG_VALUE_INT) {
        cfg.duracao.timeout_rejeicoes = (unsigned long int)AuxAttrib-
>value.i;
    }
  } else {
      fprintf(stderr,"ERRO:  /CFG_GERAIS/DURACAO/timeout_rejeicoes  nao
encontrado.\n");
    return -1;
  }

            if      ((AuxAttrib      =      spConfigAttributeFindPath
(rootNode,"/CFG_GERAIS/DURACAO/tempo_simulacao"))!=NULL) {
    if (AuxAttrib->type==SP_CONFIG_VALUE_STRING) {
         cfg.duracao.simulacao  =  (unsigned  long  int)atol(AuxAttrib-
>value.string);
    } else if (AuxAttrib->type==SP_CONFIG_VALUE_INT) {
      cfg.duracao.simulacao = (unsigned long int)AuxAttrib->value.i;
    }
  } else {
       fprintf(stderr,"ERRO:  /CFG_GERAIS/DURACAO/tempo_simulacao  nao
encontrado.\n");
    return -1;
  }

            if      ((AuxAttrib      =      spConfigAttributeFindPath
(rootNode,"/CFG_GERAIS/DURACAO/escala"))!=NULL) {
```

```
    if (AuxAttrib->type==SP_CONFIG_VALUE_STRING) {
            cfg.duracao.escala  =  (unsigned  long  int)atol(AuxAttrib-
>value.string);
    } else if (AuxAttrib->type==SP_CONFIG_VALUE_INT) {
      cfg.duracao.escala = (unsigned long int)AuxAttrib->value.i;
    }
  } else {
             fprintf(stderr,"ERRO:   /CFG_GERAIS/DURACAO/escala   nao
encontrado.\n");
    return -1;
    //cfg.duracao.escala=1000;
  }
  cfg.duracao.simulacao *= (unsigned long int)cfg.duracao.escala;
  cfg.duracao.simulacao *= (unsigned long int)ESCALA;
  cfg.duracao.simulacao += (unsigned long int)INICIO;

  /***** <TOPOLOGIA> *****/
            if      ((AuxAttrib      =      spConfigAttributeFindPath
(rootNode,"/CFG_GERAIS/TOPOLOGIA/n_links"))!=NULL) {
    if (AuxAttrib->type==SP_CONFIG_VALUE_INT) {
      cfg.topo.n_links = AuxAttrib->value.i;
    } else {
      cfg.topo.n_links = atoi(AuxAttrib->value.string);
    }
  } else {
           fprintf(stderr,"ERRO:   /CFG_GERAIS/TOPOLOGIA/n_links   nao
encontrado.\n");
    return -1;
  }

  /***** <CARGA_DE_TRABALHO> *****/
            if      ((AuxAttrib      =      spConfigAttributeFindPath
(rootNode,"/CFG_GERAIS/CARGA_DE_TRABALHO/n_niveis"))!=NULL) {
    if (AuxAttrib->type==SP_CONFIG_VALUE_INT) {
      cfg.carga.n_niveis = AuxAttrib->value.i;
    } else {
      cfg.carga.n_niveis = atoi(AuxAttrib->value.string);
    }
  } else {
     fprintf(stderr,"ERRO: /CFG_GERAIS/CARGA_DE_TRABALHO/n_niveis nao
encontrado.\n");
    return -1;
  }

            if      ((AuxAttrib      =      spConfigAttributeFindPath
(rootNode,"/CFG_GERAIS/CARGA_DE_TRABALHO/n_fluxos"))!=NULL) {
    if (AuxAttrib->type==SP_CONFIG_VALUE_INT)
      cfg.carga.n_fluxos = (long int)AuxAttrib->value.i;
    else {
      cfg.carga.n_fluxos = atol(AuxAttrib->value.string);
    }
  } else {
     fprintf(stderr,"ERRO: /CFG_GERAIS/CARGA_DE_TRABALHO/n_fluxos nao
encontrado.\n");
    return -1;
  }

   /* Desaloca a memoria alocada pelo item. No caso, a estrutura
inteira. */
  spConfigNodeFree(rootNode);
```

```
  return 0;
}
```

C.3 – fluxo.awk

```
# Como executar o script:
# awk -f fluxo.awk -v FID=(id do fluxo) (arquivo de entrada) > (arquivo de saída)
#
# exemplo:
# awk -f fluxo.awk -v FID=1 out.tr > fluxo.tr

BEGIN {
        highest_packet_id = 0;
        delay_total = 0;
}

{
    # Executa este segmento em cada linha do arquivo de entrada

    # Recebe parâmetros
        action = $1;
        time = $2;
        flow_id = $8;
        packet_id = $12;

        # Discrimina monitoramento por fluxo
        if ( flow_id == FID ) {
                if ( packet_id > highest_packet_id ) highest_packet_id = packet_id;
                fid[packet_id] = flow_id;
                service[packet_id] = $5;

                if ( start_time[packet_id] == 0 )  {
                        start_time[packet_id] = time;
                }

                # Ignora pacotes dropados
                if ( action == "d" ) {
                end_time[packet_id] = -1;

                # Inicializa principais vetores
                } else {
```

```
                if ( action == "r" ) {
                        hop[packet_id] = (hop[packet_id] + 1);
                        end_time[packet_id] = time;
                        size[packet_id] = $6;
                   }
                 }
        }
}
END {
        for ( packet_id = 0; packet_id <= highest_packet_id; packet_id++ ) {
                start = start_time[packet_id];
           end = end_time[packet_id];
                if ( start < end ) {

                        # Resolve número de saltos
                        hops = (hop[packet_id] - 1);

                        delay_total = delay_total + delay;

                        # Resolve delay e jitter
                        old_delay = delay;
                        delay = 0;
                        jitter = 0;
                        delay = end - start;
                        if ( delay > old_delay ) {
                                jitter = delay - old_delay;
                        } else {
                                jitter = old_delay - delay;
                        }

                        # Colunas impressas no arquivo de saída
                        printf("%f %f %f %f %s %i %i %i %i\n", end, delay, jitter, delay_total,
start, service[packet_id], size[packet_id], fid[packet_id], hops, packet_id);
                }
        }
}
```

Bibliografia

Ahn, G. e Chun, W. (2002), Architecture of MPLS Network Simulator (MNS), Relatório Técnico, Chungman University of Korea. Coréia do Sul.

Bajaj, S. et. al. (1999), Improving Simulation for Network Research, Relatório Técnico, University of Southern Califórnia. EUA.

Banerjee, G. e Sidhu, D. (2002), Comparative analysis of path computation techniques for MPLS traffic engineering, *Computer Networks: The Int. Journal os Computerand Telecommunications Networking* 40(1): 149-165.

Boehm, B. (2000), Spiral Development: Experience, Principles, and Refinements, Workshop on Spiral Development, ESC-SR-00-08

Callon, R., Rosen, E. e Viswanathan, C. (2001), RFC 3031. Multiprotocol label switching architecture.

David M. (2002), Comparison of Network Simulators Revisited, Nicol & Dartmouth College.

Deitel, H., Deitel, P., Nieto, T., Lin, T. e Sadhu, P. (2001), XML How to Program, Ed. Prentice Hall. EUA.

Dias, R., Camponogara, E., Farines, J.-M., Willrich, R. e Campestrini, A. (2003a), Implementing traffic engineering in MPLS-based IP networks with Lagrangean relaxation, IEEE ISCC 2003, p. 373-378.

Dias, R., Camponogara, E., Farines, J.-M., Willrich, R. e Campestrini, A. (2003b), Otimização Lagrangeana em Engenharia de Tráfego para Redes IP sobre MPLS, XXI Simpósio Brasileiro de Redes de Computadores, Natal, Brasil, p. 475-490.

Dias, R., Camponogara, E., Farines, J.-M., Willrich, R. e Campestrini, A. (2003c), Using Lagrangean Relaxation to Improve Performance on IP Networks over MPLS, Gestion de Reseaux de Services – GRES'2003, Fortaleza. Brasil, p.27-37.

Dias, R., Camponogara, E., Farines, J.-M., Willrich, R. e Campestrini, A. (2004a), Engenharia de Tráfego em Redes IP sobre Tecnologia MPLS: Otimização Baseada em Heurísticas, XXII Simpósio Brasileiro de Redes de Computadores, Gramado. Brasil.

Dias, R. (2004b), Engenharia de Tráfego em Redes IP sobre Tecnologia MPLS: Otimização Baseada em Heurísticas, Florianópolis. Brasil.

IEEE (2004), Information Technology, Portable Operating System Interface (POSIX) – Part 1: System Application Program Interface (API)[C Language], IEEE Std. 1003.1:1990, Edição 2004, Piscataway. EUA.

INDEX Group (2005), JavaSim, http://www.j-sim.org, Univertity of Illinois. EUA.

Kurose, J. e Ross, K. (2003), Redes de Computadores e a Internet: Uma Nova Abordagem, 1ª edição, Ed. Addison Wesley, São Paulo. Brasil.

Martin, R. (2002), Agile Software Development, Principles, Patterns, and Practices, Ed. Pearson

Medina, A., Lakhina, A., Matta, I. E Byers, J. (2001), Brite: Universal topology generation from a user's perspective, International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems – MASCOTS'01.

Mitchell, M., Oldham J. e Samuel, A. (2001), Advanced Linux Programming, Ed. New Riders, Indianapolis. EUA.

Nielsen, J. (1993), Usability Engeneering, Ed. Academic Press, New York, EUA.

Nielsen, J. (1997), Usability Testing, Ed. John Wiley & Sons, New York, EUA.

Perlman, R. (2000), Interconnections, Second Edition: Bridges, Routers, Switches, and Internetworking Protocols, Ed. Addison-Wesley, Massachusetts. EUA.

Schildt, H. (1997), C Completo e Total, 3ª edição, Ed. Makron Books.

SourceForge (2005), http://sourceforge.net.

Tanenbaum, A. (1996), Computer Networks, 3ª edição, Ed. Prentice Hall, New York. EUA.

The SSF Research Network (2005), SSFNet, http://www.ssfnet.org/, EUA.

Vaugham, G. (2000), GNU Automake, Autoconf and Libtool, Ed. New Riders. EUA.

VINT (2005), Network Simulator, http://www.isi.edu/nsnam/ns.

Xiao, X., e Ni, M. N. (1999), Internet QoS: A Big Picture, IEEE Network