MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA

FÁBIO LUIZ JUNIOR
VINÍCIUS HESSEL BENEDITO DE SOUSA

IMPLEMENTAÇÃO DE UM ATAQUE DE NEGAÇÃO DE SERVIÇO NO AMBIENTE *NETWORK SIMULATOR*.

Rio de Janeiro
2011

INSTITUTO MILITAR DE ENGENHARIA

FÁBIO LUIZ JUNIOR

VINÍCIUS HESSEL BENEDITO DE SOUSA

IMPLEMENTAÇÃO DE UM ATAQUE DE NEGAÇÃO DE SERVIÇO NO AMBIENTE *NETWORK SIMULATOR*.

Projeto de fim de curso apresentado ao Curso de Graduação de Engenharia de Computação como requisito parcial para a obtenção do título de Engenheiro.

Orientador: Maj Sergio dos Santos Cardoso Silva - M.Sc.

Rio de Janeiro

2011

INSTITUTO MILITAR DE ENGENHARIA

Praça General Tibúrcio, 80 – Praia Vermelha

Rio de Janeiro – RJ CEP: 22290-270

004.62 Junior, Fábio Luiz.
L953i Implementação de um ataque de negação de serviço no ambiente *Network Simulator* / Fábio Luiz Junior, Vinícius Hessel Benedito de Sousa – Rio de Janeiro: Instituto Militar de Engenharia, 2011.

49p : il.

Projeto de fim de curso **–** Instituto Militar de Engenharia – Rio de Janeiro, 2011.

1.Network Simulator 2 Redes-Simulação I Sousa, Vinícius Hessel Benedito de II Título III Instituto Militar de Engenharia

CDD 004.62

INSTITUTO MILITAR DE ENGENHARIA

FÁBIO LUIZ JUNIOR

VINÍCIUS HESSEL BENEDITO DE SOUSA

IMPLEMENTAÇÃO DE UM ATAQUE DE NEGAÇÃO DE SERVIÇO NO AMBIENTE *NETWORK SIMULATOR*.

Projeto de fim de curso apresentado ao Curso de Graduação de Engenharia de Computação como requisito parcial para a obtenção do título de Engenheiro.

Orientador: Maj Sergio dos Santos Cardoso Silva - M.Sc.

Aprovada em 17 de junho de 2011 pela seguinte Banca Examinadora:

_____

Maj Sergio dos Santos Cardoso Silva - M.Sc.

_____

Maj Anderson Fernandes P. dos Santos - D.Sc.

_____

Cap Júlio César Duarte – D.Sc.

Rio de Janeiro

2011

AGRADECIMENTOS

Dedicamos o referido trabalho às nossas famílias que sempre nos auxiliaram e deram incentivos para prosseguirmos em nosso projeto. Aos nossos colegas por compartilharem de bons e maus momentos, lado a lado. E, principalmente, ao Maj Cardoso, nosso orientador, por fornecer a nós o devido incentivo e oportunidade para execução de trabalhos, fornecer a ajuda necessária para vencer obstáculos e, por fim, por mostrar os primeiros passos a serem dados no caminho da pesquisa.

Sumário

LISTA DE FIGURAS

LISTA DE SIGLAS

API	*Application Programming Interface*
IRC	*Internet Relay Chat*
ASCII	*American Standard Code for Information Interchange*
DDoS	*Distributed Denial of Service*
DoS	*Denial of Service*
IDS	*Intrusion Detection System*
IP	*Internet Protocol*
MAC	*Media Access Control*
NS	*Network Simulator*
PCAP	Packet Capture
RNG	*Random Number Generator*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*
ICMP	*Internet Control Message Protocol*
PPP	*Point-To-Point Protocol*

RESUMO

Este trabalho versa sobre a implementação de um ataque de negação de serviço no ambiente Network Simulator 3 (NS-3). Para tanto, utilizou-se a versão estável 3.9 do simulador, a qual, no início do trabalho, era a mais recente.

O trabalho inicia-se com uma breve introdução sobre ataques de negação de serviço e um levantamento de outros simuladores de rede para uma comparação com o NS-3.

Estudou-se sua estrutura de classes, como o sistema de *tracing*, o sistema de roteamento, suas classes *Helpers*, suas classes *Net Devices*, a pilha de protocolos internet disponível, suas limitações na simulação entre outros assuntos.

Com o conhecimento do funcionamento do NS3, passou-se para a implementação de uma pequena topologia com um roteador, um servidor alvo do ataque e cinco hosts atacantes. O ataque era feito através do envio maciço de pacotes UDP, TCP e ICMP pelos atacantes.

Através dessa pequena topologia, pode-se construir outra com maior escala. Para tanto, fez-se a junção de várias dessas pequenas topologias originando uma maior. Para dar mais realidade à simulação, utilizou-se valores aleatórios para o número de atacantes a cada simulação, reprodução da proporção dos tipos de enlaces encontrada nas redes reais, também, da inserção de tráfego de fundo.

O trabalho finaliza-se com a execução e verificação da simulação, somado com a geração de tracing do ataque.

Palavras-chaves: *Network Simulator* 3, DDoS, redes e Simulação.

ABSTRACT

This paper describes the implementation of a denial of service attack on the environment Network Simulator 3 (NS). For this, we used the stable version 3.9 of the simulator, which, at the outset, was the latest.

The work begins with a brief introduction about denial of service and a survey of other network simulator for a comparison with the NS.

We studied its components, such as tracing system, the routing system, Helpers their classes, their classes Net Devices, the Internet protocol stack available, their limitations in the simulation among other issues.

With the knowledge of the functioning of NS3, we started to implement a small topology with a router, a server targeted for attack and hosts five attackers. The attack was done by sending massive UDP, TCP and ICMP by attackers.

Through this small topology, was build one with a larger scale. As such, there is a junction of several of these small topologies leading to a higher. To give more reality to the simulation, we used the random such as number of attackers for each simulation, the reproduction rate of the types of links, also the insertion of background traffic.

The work ends with the implementation and verification of simulation, combined with the generation of tracing the attack.

Keywords: Network Simulator 3, DDoS, network and Simulation.

1 INTRODUÇÃO

Simulação computacional é a tentativa de imitar um modelo do mundo real através de um computador de tal forma que seu comportamento possa ser estudado. Este modelo é criado tentando-se replicar certo número de detalhes do sistema real. Quanto maior esse número, mais perfeita e confiável será a simulação. Para que a simulação produza resultados confiáveis, o modelo deve ser validado por técnicas especificas para esse fim.

É importante ressaltar que nem sempre é aconselhável realizar simulações para análise de sistemas. Isso ocorre quando o modelo puder ser resolvido simplesmente por métodos analíticos, ou seja, por equações matemáticas fechadas. No entanto, como sistemas de redes de computadores envolvem um grande número de variáveis, é inviável, em algumas situações, a solução por métodos analíticos e utiliza-se, portanto, simulação para seu estudo.

O emprego de simuladores de redes de computadores é fundamental para muitos fins como, por exemplo, ensino, aprendizagem e projetos de redes. O software Network Simulator 3 (conhecido simplesmente como NS3) é um dos simuladores de redes mais empregados e que será estudado no decorrer deste trabalho. Possui como característica fundamental o fato do tempo, no decorrer da simulação, ser considerado discreto. Ou seja, o tempo flui em pequenos saltos (Δt). Isso restringe o universo de processos que podem ser simulados. Os processos aceitos por esse tipo de simulador são os de estados discretos ou os de estados contínuos que podem ser discretizados de maneira que não modifique significativamente a natureza deste.

Sobre a estrutura deste trabalho, ela está dividida em 7 capítulos, organizados da seguinte maneira:

- Na introdução, é apresentada uma definição de simulador e os objetivos do trabalho.
- No Capítulo 2, é realizada uma explanação sobre ataques DDoS (Distributed Denial of Service)
- No Capítulo 3, é apresentado um breve levantamento de outros simuladores de redes existentes.

- No Capítulo 4, tem-se uma pesquisa sobre simulações já existentes de ataques DDoS no NS3.
- No Capítulo 5, tem-se a estrutura básica do Network Simulator, na sua versão 3.9, apresentando conceitos do projeto do simulador essenciais ao desenvolvedor dos scripts de simulação. Ainda nesse capítulo, são apresentadas e implementadas duas topologias de rede, uma inicial e outra principal, para a simulação de um ataque DDoS. A topologia inicial serviu de base para a construção da topologia principal, o qual é maior e mais complexa.
- No Capítulo 6, Resultados Alcançados, tem-se uma síntese dos resultados obtidos ao longo deste trabalho.
- Na conclusão, é exposto o resultado final do trabalho e apresentados os comentários sobre o cumprimento ou não dos objetivos.

1.1 Importância deste trabalho

Considerando-se as características de redes de computadores atuais, usuários, empresas e instituições que utilizam tais redes, como a Internet, devem estar atentos aos princípios de segurança. As típicas ferramentas de segurança, como Sistemas de Detecção de Intrusão (IDS – *Intrusion Detection System*), *Firewalls*, Antivírus, entre outros, utilizam bases de dados de ataques passados como assinaturas das ameaças virtuais. No entanto, devido ao constante surgimento de novos ataques e à evolução dos existentes, novas bases de dados devem ser produzidas constantemente. Com a análise dessas bases de dados, pode-se medir a eficácia das ferramentas e assim facilitar a escolha delas de maneira mais confiável. Um interessante *site* com várias informações sobre o assunto é o *Panda Security* (1).

1.2 Objetivos.

Este projeto tem por objetivo utilizar o software *Network Simulator* 3 para a criação de classes e scripts que auxiliem na realização de uma simulação de ataque distribuído de negação de serviço (DDoS - *Distributed Denial of Service*).

Através de uma simulação, será gerada e validada uma base de dados com o *tracing* de um ataque. Esta base poderá ser usada futuramente para testes de ferramentas de defesa existentes ou análise de novas soluções de defesa.

2 INTRODUÇÃO À TEORIA DE ATAQUES DDOS

Ataques DoS visam tornar um sistema computacional propositalmente lento ou até completamente paralisado, para seus usuários legítimos, utilizando para isso o consumo indiscriminado de recursos (memória, poder de processamento, largura da banda, etc) da máquina vítima ou através da exploração de vulnerabilidades do sistema alvo.

Nos primórdios da Internet, ataques DoS eram lançados de um único computador. Esses ataques apoiavam-se, principalmente, na diferença de largura de banda existente entre o atacante e a vítima. Um atacante, possuindo maior largura de banda, poderia esgotar facilmente os recursos da vítima.

Com a rápida evolução da rede, protocolos e aplicações, este tipo de ataque tornou-se ineficaz. A abordagem moderna consiste em subverter computadores pessoais para os propósitos de ataque, formando, assim, o que se denomina *botnet*.

O termo *botnet* refere-se a um conjunto de computadores infectados (zumbis ou escravos) com algum software malicioso que permite a um atacante (controlador) executar comandos arbitrários nessas máquinas. Essa comunicação entre controlador e escravos pode ocorrer de forma direta, através de conexões TCP (*Transmission Control Protocol*) diretas ou datagramas UDP (*User Datagram Protocol*), ou indireta, através de canais IRC (*Internet Relay Chat*) ou *Proxy*. Além disso, o controlador pode usar níveis de máquinas intermediárias (mestres) para aumentar o grau de anonimato e diminuir o tráfego necessário para o envio de comandos aos escravos. Essas máquinas mestres não realizam, necessariamente, ataques aos alvos sendo que, sua principal função é a de retransmitir a comunicação do controlador aos escravos. A FIG. 2.1 apresenta uma estrutura típica de um ataque distribuído de negação de serviço.

Assim, o atacante dispõe de um conjunto de agentes capazes de lançar um ataque DDoS sincronizado ou seguindo uma estratégia qualquer.

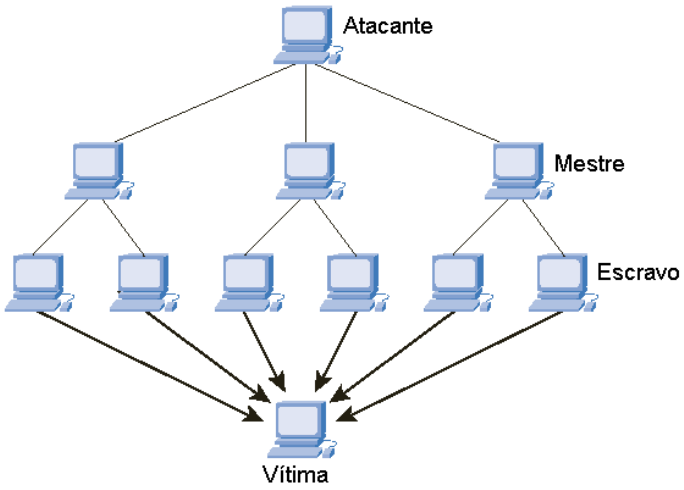

FIG. 2.1 Um tipo típico de ataque distribuído de negação de serviço.

Quando o atacante dispõe de uma *botnet* que julga adequada, a ofensiva pode ser iniciada. O atacante realiza o procedimento, particular do código utilizado, de disparo do “gatilho” que pode consistir em envio de comandos, parâmetros ou arquivos de configurações. Disparada a investida, as máquinas escravas passam a executar códigos que implementam os métodos de ataque. Esses códigos podem incluir métodos que colaboram para o objetivo do ataque de diversas formas: consumindo largura de banda, poder de processamento ou memória, ou exaurindo recursos associados a serviços.

A principal estratégia utilizada, atualmente, para conseguir esses efeitos, e que não depende diretamente de nenhuma vulnerabilidade existente no sistema da vítima, é a inundação de pacotes. Ataques dessa categoria consistem no envio, em grande quantidade, de pacotes para a vítima causando uma sobrecarga do sistema ou dos canais de comunicação. A conseqüência dessa sobrecarga pode ser uma lentidão ou uma total paralisação de serviços do alvo. Este resultado dependerá, a princípio, de quantos computadores foram empregados como escravos e do quão intenso é o poder computacional da vítima.

3 PESQUISA

3.1 LEVANTAMENTO DE OUTROS SIMULADORES.

Seguindo o cronograma deste trabalho, foi feito um levantamento e estudo das características básicas de outros simuladores de redes disponíveis. Nesse levantamento, foram encontrados diversos softwares de simulação, dos quais se destacaram os seguintes: o GloMoSim, o OPNET, o NCTUns e o IMUNES. A seguir, será descrito as principais vantagens e desvantagens de cada um desses e uma breve justificativa do porque não foram escolhidos como software para cumprir o objetivo principal deste trabalho.

O GloMoSim é um simulador de alto desempenho e moderno, porém possui enfoque em redes *wireless* e móveis não se adequando a este trabalho. Tal fato se justifica, pois este trabalho tem como objetivo principal simular ataques DDoS em redes cabeadas. Mais informações sobre o GloMoSim pode ser obtida no site About GloMoSim (2).

O OPNET é altamente utilizado no ambiente corporativo para simulação de redes de telecomunicações. Porém, não é um software livre, possuindo um alto custo nas aquisições de licenças. Sendo assim, inviável para uso acadêmico e de pesquisa. Informações adicionais no site do proprietário do software (3).

O NCTUns é didático, de fácil utilização e moderno. Porém é limitado em sua escalabilidade, pois o número máximo de nós na simulação é baixo. Tal fato inviabiliza a utilização deste simulador tendo em vista que as *botnet*s possuem uma grande quantidade de nós.

Por último, tem-se o IMUNES que foi um forte candidato a substituição do NS3 neste trabalho. Para este, será aberta uma breve subseção para descrever suas características mais detalhadamente.

3.1.1 IMUNES (Integrated Multiprotocol Network Emulator / Simulator).

O *IMUNES* é um sistema de emulação/simulação realístico de redes baseados no sistema operacional *freeBSD*. O IMUNES foi projetado para se integrar com o *kernel* do *freeBSD* e através disso cria nós virtuais utilizando-se de várias instâncias

da pilha de protocolos de internet disponível no sistema operacional. É por causa dessa característica que ele é conhecido como emulador/simulador de redes.

Tal característica permite gerar uma emulação totalmente realística de uma rede, possibilitando que cada nó execute aplicativos, de forma independente, em modo usuário. Isso permite que aplicativos geradores de tráfego, analisadores de rede, servidores de Web entre outras ferramentas de rede rodem nos nós virtuais como se fossem máquinas reais. Além disso, o *IMUNES* possui um ambiente gráfico para a criação e gestão de cenários de simulação como a imagem FIG. 3.1 ilustra. A interface gráfica pode rodar nos principais sistemas operacionais do mercado através da instalação da ferramenta Tcl/Tk. No entanto, a emulação em si só pode ser executada em ambientes *freeBSD* com privilégios de super usuário.

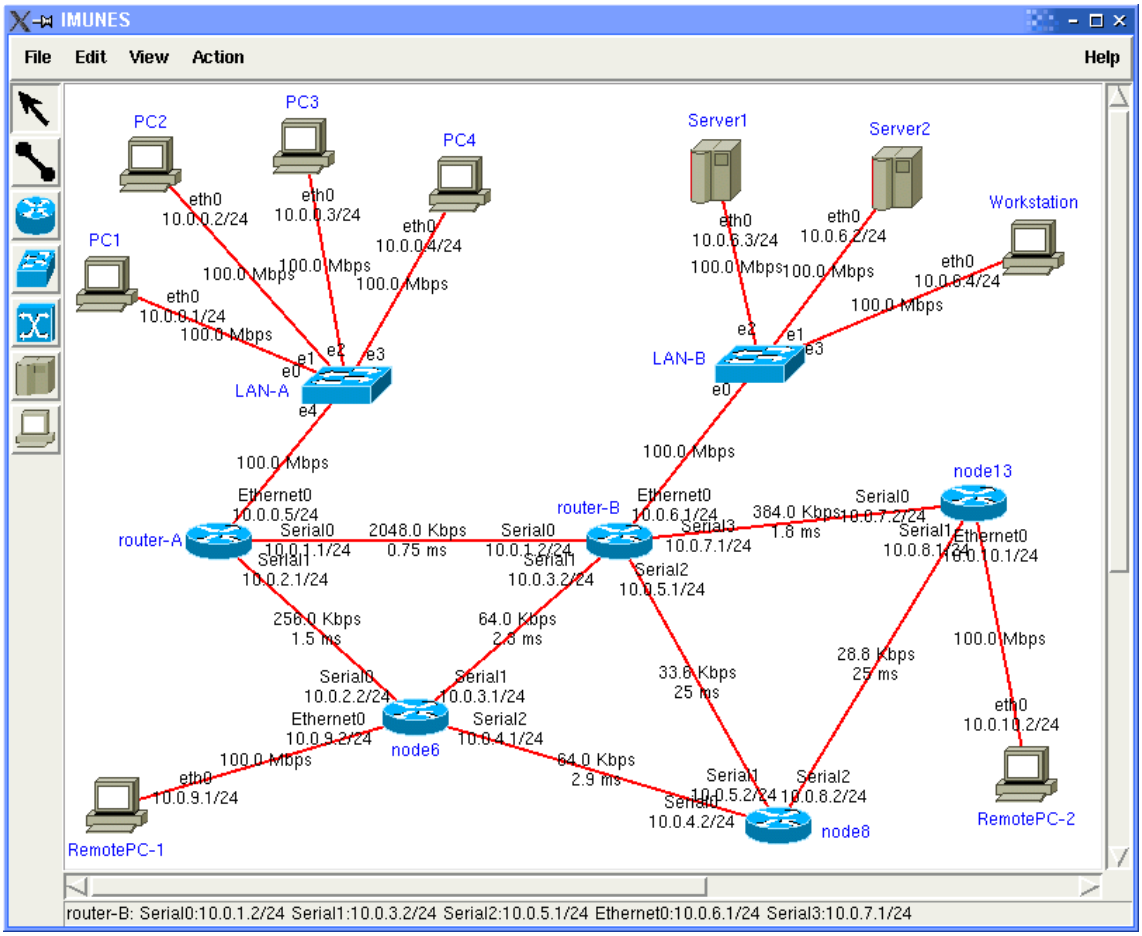

FIG. 3.1 Imagem da criação de uma topologia através da interface gráfica do IMUNES.

Contudo, o *IMUNES* apresenta algumas desvantagens se comparadas com o NS3. A primeira é a deficiência na documentação. A documentação do *IMUNES* é escassa e difícil de encontrar. Em contra partida, o NS3 possui uma vasta documentação de sua arquitetura, de sua estrutura de classes, possuindo exemplos de topologias já implementadas entre outras informações.

A segunda desvantagem é que a montagem da topologia fica restrita ao modo gráfico. Isso faz com que a construção e manipulação das *botnet*s com imensa quantidade de nós fique prejudicada e limitada ao que a interface disponibiliza.

Terceira e última desvantagem observada em comparação ao NS3 é que este último já possui uma grande utilização pela área acadêmica e está com uma base bem consolidada. Inclusive, ele é utilizado para validar outros simuladores.

Visto isso, acredita-se que o NS3 seja uma boa escolha para testar e simular ataques DDoS.

Essas e outras informações podem ser encontradas no site oficial do IMUNES (4) e no seu manual de utilização (5).

3.2 PESQUISA BIBLIOGRÁFICA

Foi realizada a pesquisa bibliográfica de teoria de ataques de negação de serviço e implementações disponíveis para o NS-3.

A primeira fase da pesquisa, referente a teoria de ataques, foi realizada no trabalho *Estudo de ataques distribuídos de negação de serviço* (6).

Na segunda fase, visando-se reunir classes, bibliotecas ou *frameworks* úteis à implementação do ataque de negação de serviço no NS3, realizou-se uma pesquisa na literatura.

Atualmente, o projeto do NS3 não prevê um *framework* de segurança e, inclusive, apresenta algumas limitações na simulação de recursos pertinentes ao tema. Por isso as classes presentes, freqüentemente, necessitam de adaptações para serem utilizadas em ataques.

O uso do NS3 já se encontra bastante difundido e em muitos trabalhos ele é utilizado como forma de validação de teorias. No entanto, não se conseguiu encontrar os códigos desenvolvidos ou discussões detalhadas sobre estas simulações de validação.

Dessa forma, ao final da pesquisa, não foi possível reunir classes úteis ao projeto e a implementação da simulação iniciou-se baseada nas classes presentes no NS3, sendo que futuras necessidades do projeto deverão ser sanadas com classes desenvolvidas a partir dessas.

Apesar de não fornecer *frameworks* ou bibliotecas, é importante citar o grupo de discussões ns-3-users (7), que utiliza os serviços do Google Groups (8). Trata-se de um grupo bastante ativo com discussões de caráter geral sobre *bugs* e desenvolvimento no NS3 contendo trechos de código e abordagens para desenvolvimento de funcionalidades não implementadas no projeto oficial.

4 *NETWORK SIMULATOR*.

O NS3 é um sistema de simulação de eventos discretos para ambientes *unix-like* com foco, principalmente, em simulações de sistemas baseados na arquitetura Internet e voltado para pesquisa e educação. Apesar de ser reconhecido como sucessor do simulador NS2, ele foi totalmente re-projetado e reescrito aproveitando-se, no entanto, do conhecimento obtido com a experiência do desenvolvimento e uso do NS2. Toda implementação foi realizada com o uso da linguagem C++, no entanto oferece uma API (*Application Programming Interface*), ainda incompleta, para desenvolvimento de *scripts* em Python. Diferentemente da arquitetura pouco escalável de seu antecessor, o NS3 faz uso intensivo de padrões de projeto e técnicas visando um alto grau de flexibilidade e possui uma arquitetura modular, o que lhe garante alta escalabilidade.

Essa modularidade cria, naturalmente, famílias de classes que cooperam para objetivos comuns. Dessa forma, conforme consta no manual do NS3 (5), a arquitetura pode ser dividida em camadas de crescente nível de abstração, conforme a FIG. 4.1:

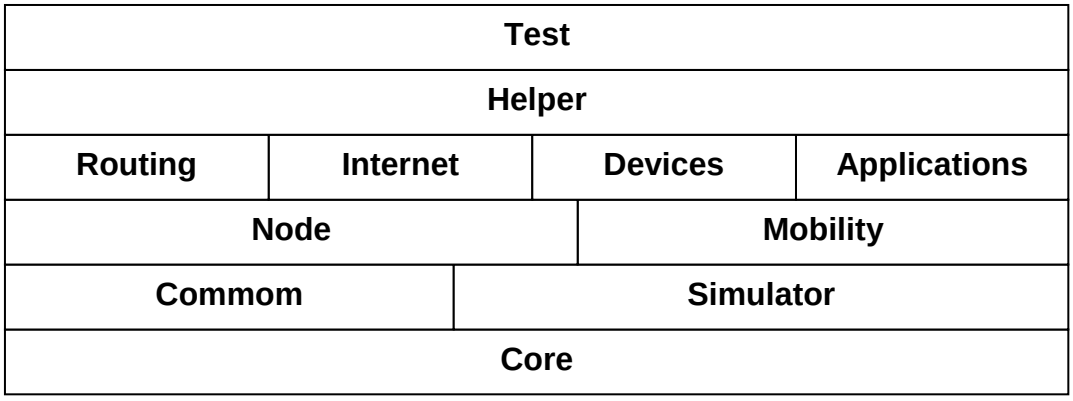

FIG. 4.1 Arquitetura de camadas do NS3.

Abaixo serão descritas brevemente as funções de cada uma dessas camadas:

- Core: Compõe o núcleo do simulador, implementando os sistema de log, variáveis aleatórias, ponteiros inteligentes, entre outras funcionalidades.
- Commom: Implementa os pacotes e as escritas em arquivos PCAP e ASCII.
- Simulador: Coordena o agendamento de eventos, aritmética do tempo, sincronização do relógio para simulações reais, entre outras funções.

- Node: Implementa a classe Node, os netDevices, os diferentes tipos de endereços (IPv4, MAC, etc), entre outras funfionalidades.
- Mobility: Implementa a mobilidade física de nós em redes móveis.
- Routing: Coordena o mecanismo de roteamento.
- Internet: Implementa a pilha de protocolos internet.
- Devices: Implementa os diferentes tipos de enlaces como, por exemplo, ponto-a-ponto e *wifi*.
- Applications: Contém os vários aplicativos que podem ser executados no nós.
- Helper: Implementa uma interface que facilita a programação dos scripts de simulação.
- Test: Conjunto de scripts que implementam casos de teste para verificar o funcionamento correto do simulador e o desempenho do computador.

Quanto às atuais funcionalidades implementadas, o NS3 compreende:

- Construção de redes virtuais e suporte a diversos itens de auxílio a simulações de redes.
- Suporte a emulação de rede, permitindo interação com redes reais.
- Simulação distribuída, possibilitando simulações de grande porte.
- Suporte a animação ainda em fase de desenvolvimento.
- Suporte *a tracing, logs* e estatísticas de componentes da simulação.

Atualmente, o NS3 encontra-se em fase de intenso desenvolvimento sendo que cada ciclo de lançamento de nova versão tem o período médio de quatro meses.

Devido aos rigorosos requisitos de flexibilidade e escalabilidade adotados no projeto, o NS3 apresenta uma estrutura de funcionamento que exige do usuário relativo grau de conhecimento dos conceitos adotados. Esses conceitos variam em grau de abstração. Enquanto conceitos de menor nível de abstração são responsáveis pela estruturação e funcionamento da simulação, conceitos de mais alto nível auxiliam o usuário na criação de *script* tornando transparente grande parte da complexidade interna do simulador.

A seguir, serão apresentados conceitos importantes, tanto para o desenvolvimento de *scripts*, quanto para a customização da estrutura do NS3 tornando possível a criação de novas classes e simulações mais poderosas.

4.1 Estrutura do Network Simulator

O *Network Simulator* utiliza, frequentemente, em sua estrutura padrões de projetos e conceitos de relativa complexidade como *callbacks* e *reflections*. Essa engenharia de software torna o simulador bastante flexível e complexo e, por isso, uma fase de estudo, tanto do ponto de vista do código como conceitual, torna-se importante e imprescindível.

Durante o estudo do NS3 é importante observar que, eventualmente, certos termos podem ser usados de maneira não usual. Esse é o caso da expressão "*Script* de Simulação", largamente utilizada em toda a documentação oficial.

Apesar de possuir uma API para desenvolvimento de simulações em Python, que se trata de uma linguagem de *script*, a documentação utiliza o mesmo termo para o desenvolvimento de simulações em C++. Linguagens de *script* normalmente referem-se às linguagens interpretadas dentro de aplicações e que servem para estender sua funcionalidade. Esse não é o caso da API para C++ fornecida pelo NS3 que permite o desenvolvimento de simulações com todos os recursos da linguagem, incluindo orientação a objetos, ponteiros, entre outros.

No restante deste trabalho, será utilizada a expressão "*Script* de Simulação" com o mesmo significado presente na documentação oficial.

4.1.1 Variáveis Aleatórias.

Variáveis aleatórias são largamente utilizadas em diversos tipos de simulações e, por isso, a qualidade final da simulação depende, em grande parte, das propriedades dessas, como, por exemplo, a independência entre variáveis.

O NS3 utiliza o gerador MRG32k3a de Pierre L'Ecuyer (9), capaz de gerar sequências de números aleatórios, denominadas RNG (*Random Number Generator*), com período de aproximadamente 3.1×10^{57}. Partições de uma RNG formam subseqüências de números aleatórios não correlacionados entre si. Dessa forma, o NS3 produz, de maneira eficiente, conjuntos de variáveis aleatórias independentes.

Por padrão, o NS3 utiliza a mesma semente na geração de RNG para toda execução da simulação. Isso implica um determinismo dos resultados, nas diversas

execuções. A fim de inserir um fator aleatório entre execuções de uma simulação, o NS3 oferece duas abordagens:

- Abordagem pela variação da semente.

Essa abordagem utiliza o método *SeedManager::SetSeed(int)* para configurar as sementes. Apesar de gerar resultados diferentes entre execuções, ela não garante independência entre os RNG gerados.

- Abordagem pela variação do número de execução.

Essa abordagem utiliza o método *SeedManager::SetRun(int)* sendo a mais recomendada. Ela consiste na variação de um parâmetro denominado Número de Execução. O Número de Execução determina o conjunto de subseqüências de RNG utilizadas garantindo, assim, independência entre execuções.

No NS3 são oferecidas as seguintes classes de variáveis aleatórias:

- *UniformVariable*
- *ConstantVariable*
- *SequentialVariable*
- *ExponentialVariable*
- *ParetoVariable*
- *WeibullVariable*
- *NormalVariable*
- *EmpiricalVariable*
- *IntEmpiricalVariable*
- *DeterministicVariable*
- *LogNormalVariable*
- *TriangularVariable*
- *GammaVariable*
- *ErlangVariable*
- *ZipfVariable*

4.1.2 *Callback*.

Devido à estrutura fortemente modular do NS3, inevitavelmente surgirão referências entre módulos. Com o intuito de reduzir a dependência entre classes e aumentar a escalabilidade e flexibilidade do sistema, é fornecida uma API para chamadas de *callback* que possibilitam a referência a métodos de outras classes através de uma camada de ponteiros de funções.

A API de *Callback* oferece dois serviços básicos:

- Declaração de tipo de *callback*, utilizada para definição da assinatura de *callback*
- Instanciação de *callback*, utilizado para a chamada de um *callback*, previamente declarado.

4.1.3 Gerenciamento de Memória.

O controle de memória é realizado através da técnica de contagem de referência. Todo objeto com contagem de referência mantêm um contador interno que sinaliza o número de ponteiros que se refere a ele. Quando um objeto cria um novo ponteiro que o referencia, seu contador é incrementado. Por outro lado, quando um ponteiro deixa de apontar para ele, o contador é decrementado. Se a contagem chega a zero, o objeto é deletado com segurança.

O NS3 tem uma classe de ponteiros inteligentes, Ptr<type>, que provê métodos com implementação transparente de contagem de referência. Além disso, essa classe se comporta de forma semelhante a ponteiros comuns.

4.1.4 Modelos de Objetos.

Com o objetivo de expandir os recursos de orientação a objetos providos pelo C++, o NS3 define classes modelos das quais serão derivadas a maioria das classes especializadas do simulador. Essas classes modelos oferecem os recursos de gerenciamento de memória e um sistema de agregação através de ponteiros

inteligentes e um sistema de agregação de objetos. As três classes base definidas são:

- *Object*
- *ObjectBase*
- *SimpleRefCount*

A classe *Object* oferece os recursos de gerência de memória e sistema de agregação enquanto a classe *ObjectBase* oferece apenas o sistema de agregação e a classe *SimpleRefCount*, apenas a gerência de memória.

4.1.5 Agregação de Objetos.

Caso toda a necessidade de objetos especializados fosse sanada com o uso de herança de classes, um grande número de classes especializadas seria necessário aumentando a complexidade da hierarquia de classes e tornando o sistema pouco escalável. Com o objetivo de evitar o uso de herança, as classes modelos oferecem um sistema de agregação que permite a criação de referências a outros objetos de forma flexível. Além disso, a API desse sistema de agregação resolve o problema de *downcasts* através do método Object::GetObject(type) que retorna um ponteiro para o objeto da agregação do tipo *type*.

4.1.6 *Tracing*.

Em última instância, o objetivo de uma simulação é obter a estatística do comportamento do sistema simulado. Com o intuito de permitir a criação dessas estatísticas de forma flexível, o NS3 oferece o sistema de *Tracing*.

Este sistema trabalha com os conceitos de fonte de *tracing* e sorvedouro de *tracing*. Fontes de tracing são entidades capazes de sinalizar um determinado evento e fornecer informações sobre esse evento. Cada objeto capaz de gerar alguma forma de resultado em *tracing* deve definir um conjunto de eventos utilizados como fonte de *tracing*.

Por outro lado, sorvedouros de *tracing* são entidades capazes de, mediante a informação provida pela fonte, realizar um conjunto de ações úteis. Dessa forma, diversas fontes de *tracing* podem ser espalhadas pela simulação sem nenhuma conseqüência e sorvedouros de *tracing* podem ser conectados às fontes convenientemente e sob demanda. Além disso, diversos sorvedouros de *tracing*, que realizam diversas ações, podem ser conectados na mesma fonte de *tracing*, tornando esse sistema bastante flexível. Os sorvedouros oferecem simples classes de *helper* que possibilitam a geração de *tracing*s em formatos específicos, como ASCII (*American Standard Code for Information Interchange*) ou PCAP (Packet Capture). Como exemplo, pode-se citar os sorvedouros de *tracings* existentes na implementação dos enlaces ponto-a-ponto. Esses sorvedouros não têm influência alguma na simulação, a não ser quando eles são conectados às fontes de *tracings*, através do registro de *callback*, via comando EnablePcapAll da classe PointToPointHelper.

4.1.7 Sistema de configuração do NS3.

Na análise de um problema, geralmente leva-se em conta os resultados de diversas instâncias da simulação com variados parâmetros. A alteração desses parâmetros diretamente na variável responsável, no *script* de simulação, implicaria em uma nova recompilação para cada execução da simulação. Para evitar esse procedimento repetitivo, o NS3 oferece um sistema que permite a associação de variáveis a parâmetros da linha de comando. Essa associação permite a execução de diversas instâncias da simulação sem recompilação de código, apenas variando parâmetros da linha de comando possibilitando, ainda, a automatização através de Shell *Script*.

Seja, por exemplo, uma variável *bandWidth*, do tipo inteiro, que deverá ser alterada em cada execução da simulação. Alterar seu valor no *script* de simulação, antes de cada recompilação, seria uma tarefa bastante dispendiosa. Utilizando-se os recursos do Sistema de Configuração do NS3, pode-se associar a variável a um parâmetro de linha de comando *bandWidthParm*, por exemplo.

static integerDefaultValue<int> bandWidth (“bandWidthParm”, “Mensagem de ajuda”, 5);

Dessa forma, pode-se passar o valor de configuração através do parâmetro bandWidthParm.

4.1.8 Componentes do NS3.

A seguir, será discorrido sobre os principais componentes do NS3 conforme descrito na sua documentação (10).

4.1.8.1 Classe Nó.

A classe nó é uma das principais classes base do simulador. Ela faz uso do sistema de agregação para evitar especializações desnecessárias, dando um alto grau de flexibilidade ao usuário. Seus atributos são *ClasseID*, *SystemID* (utilizado em simulações distribuídas), uma lista de Aplicações, uma lista de *NetDevices*. Uma visão básica da classe nó está disponível na FIG. 4.2.

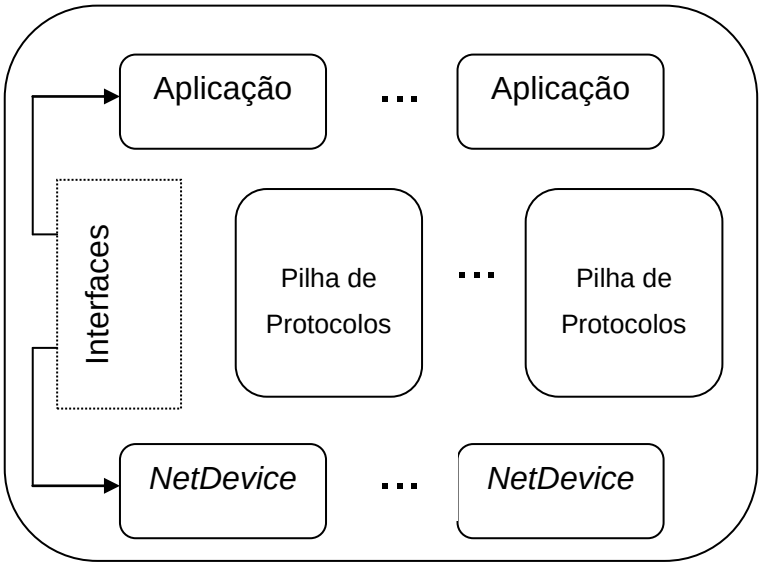

FIG. 4.2: Visão de alto nível de abstração da classe Nó.

4.1.8.2 Pacotes.

Em uma rede real, a comunicação entre os nós é feita através de pacotes que são, essencialmente, conjuntos de dados de aplicações sucessivamente encapsuladas em camadas de diversos protocolos.

O NS3 oferece uma classe de comportamento semelhante a pacotes de redes reais, mas com alguns atributos adicionais. Um objeto Pacote, do simulador, é composto por um *buffer* de bytes e uma lista de *tags* de bytes. Os sucessivos cabeçalhos e *trailers* são serializados no *buffer* de bytes sendo que existe uma correspondência bit-a-bit com um pacote de rede real implementando os mesmos protocolos. Para esse formato de objeto, fragmentação e desfragmentação são operações simples sobre uma sequência de bytes.

Diferentemente das redes reais, em que pacotes são bem definidos pelo conjunto de protocolos que a rede utiliza, os usuários do simulador normalmente precisarão agregar algum tipo de informação adicional aos pacotes. Com esse objetivo, os pacotes do simulador oferecem a possibilidade de inserção de informações arbitrárias na forma de uma lista de *tags* que são pequenos blocos de 16 bytes. Os pacotes podem ter qualquer quantidade de *tags*.

4.1.8.3 Helper.

Helper é uma API de alto nível que possui como função facilitar a programação de *scripts* de simulação. Cada método da API *Helper* chama um ou vários outros métodos de baixo nível de tal modo que seja realizada uma ação desejada. Assim, basta para o programador chamar um método da API *Helper* em vez de chamar vários métodos de baixo nível para executar uma tarefa. Isso faz com que o código se torne menos extenso e tedioso para o programador. No entanto, a API *Helper* não substitui a programação em baixo nível. Visto que pode haver a necessidade de implementar funcionalidades básicas que não estejam na nesta API.

A API *Helper* não possui nenhuma dependência com o resto do código do software podendo, portanto, ser totalmente ignorada sem que se altere o funcionamento da simulação.

Para o melhor entendimento do que vem a ser essa interface de programação, será exposto um exemplo ilustrativo.

Suponha que seja necessário criar uma rede TCP/IP contendo 2000 nós. Para isso, utilizando a API *Helper*, primeiro deve-se instanciar um objeto do tipo *NodeContainer* que será o contêiner dos nós. Chamamos esse objeto de *nodes*. No objeto, chama-se o método *create()* passando como parâmetro o número de nós que se deseja criar. Após isso, instancia-se um outro objeto *stack* do tipo *InternetStackHelper.* Em seguida, chama-se o método *install()* passando o objeto *nodes* como parâmetro. Isso fará com que sejam associados os protocolos de pilha da Internet (TCP, UDP, IP, etc.) em cada um dos nós no contêiner.

```
//Criação de um objeto do tipo contêiner de nós.
NodeContainer nodes;
//Criação de dois mil nós dentro do contêiner.
nodes.Create (2000);
//Criação de um objeto do tipo InternetStackHelper (Classe presente na API Helper).
InternetStackHelper stack;
/*Instalação dos protocolos de pilha da Internet (TCP, UDP, IP, etc.) em cada um dos nós do contêiner */
stack.Install (nodes);
```

É possível perceber que é relativamente fácil, utilizando a interface Helper, criar os 2000 nós da rede e associá-los aos protocolos usuais da internet. Caso não fosse possível utilizar essa API, seria necessário associar os protocolos aos nós um a um individualmente.

Além da classe *InternetStackHelper*, existem várias outras como: *CsmaHelper*, I*pv4RoutingHelper, DeviceEnergyModelHelper, InterferenceHelper*, etc.

4.1.8.4 Net Device.

No mundo real, para conectar um computador a uma rede é necessário um dispositivo de hardware especifico para a conexão. Como exemplo, pode-se citar uma rede *wireless* que, para se conectar a ela, é necessária uma placa de hardware

que suporte tal funcionalidade. Além disso, é necessário um software específico de controle do hardware conhecido como *Device Driver*. No ambiente de simulação do NS3, tanto o hardware como o *Device Driver* são modelados por classes conhecidas como *Net Devices*. Essas classes contêm métodos que gerenciam a conexão dos nós.

Para um melhor entendimento, tem-se as seguintes classes como exemplo:

CsmaNetDevice – Classe que modela uma conexão que utiliza o protocolo de acesso ao canal CSMA.
PointToPointNetDevice – Classe que modela uma rede ponto a ponto.
WifiNetNevice – Classe que modela uma rede wireless.
SubscriberStationNetDevice e *BaseStationNetDevice* -- Essas duas classes modelam uma rede Wimax

Além dessas classes, temos outras duas consideradas especiais. Elas são responsáveis pela comunicação da simulação com o mundo externo.

A classe *TapBridge* é usada quando se deseja que um computador real ou uma máquina virtual, que suporte *devices* Tun/Tap, interaja com a simulação. Um objeto do tipo *TapBridge* cria um *net device* dentro da simulação para este *host* real/virtual. Isso faz com que seja possível a passagem de pacotes nos dois sentidos entre este *host* e a simulação.

A classe *EmuNetDevice* é usada quando se deseja que um nó da simulação envie ou receba pacotes para uma rede real externa. É parecida com a *TapBridge*, porém, nesse caso, temos uma rede externa à simulação. No envio de pacotes para o exterior da simulação, utiliza-se uma técnica chamada de MAC (*Media Access Control*) *Spoofing.* Os endereços físicos de saída são da forma 00:00:00:XX:XX:XX, onde X representa um algarismo hexadecimal qualquer. Os 24 primeiros bits do endereço MAC, que representam o fabricante do dispositivo de rede, são zerados de tal forma que não haja conflito de endereços com a rede externa. Na entrada de pacotes externos na simulação, o dispositivo de hardware que os conecta deve estar necessariamente configurado em modo promíscuo. Isso se justifica já que deve capturar todos os pacotes e não somente os que possuem o seu endereço MAC.

Abaixo, temos alguns métodos da classe *PointToPointDevice* que são exposto para fins ilustrativos .

void ns3::PointToPointNetDevice::SetDataRate(DataRate bps). – Através desse método, é possível configurar a velocidade do dispositivo de rede simulado.

virtual bool ns3::PointToPointNetDevice::Send(Ptr<Packet> packet, const Address &dest, uint16_t protocolNumber) – Método utilizado para enviar um pacote de dados a um certo destinatário.

void ns3::PointToPointNetDevice::Receive(Ptr<Packet> p) – Recebe um pacote de dados

Todas as classes que modelam dispositivos de rede devem ser conectadas, obrigatoriamente, a classes que modelam os canais com as mesmas propriedades. A saber, a classe *WifiNetNevice* deve se conectar ao canal modelado por *WifiChannel*. Isso se justifica, pois elas devem implementar a mesma interface.

A seguir será dado um exemplo da implementação de um caso simples:

```
//Cria o contêiner de nós e instancia 10 nós.
NodeContainer csmaNodes;
csmaNodes.Create (10);
//Instancia um objeto da API Helper.
CsmaHelper csma;
//Seta a velocidade máxima do canal.
csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
//Seta o delay do canal.
csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));
//Configura o tamanho máximo do Frame.
csma.SetDeviceAttribute ("FrameSize", UintegerValue (2000));
//Instala o Net Device em cada um dos nós do contêiner.
NetDeviceContainer csmaDevices = csma.Install (csmaNodes);
```

4.1.8.5 Internet-Stack.

Uma classe Nó, por si só, não é útil a uma simulação. Como visto anteriormente, essa classe possui um conjunto de atributos que define suas funcionalidades. Para que um pacote consistente com o canal utilizado seja produzido, uma série de protocolos deve encapsular os dados das aplicações. Essa série de protocolos é modelada, no NS3, através da classe *Internet-Stack* que implementa os principais protocolos utilizados no ambiente de Internet. É importante observar que um *InternetNode*, que implementa a pilha de protocolos *Internet-Stack*, não é um objeto de classe derivada da classe *Node*, mas estende o comportamento dessa última através da agregação.

Para agregar uma pilha de protocolos de Internet a um nó, pode-se fazer uso da API Helper:

void ns3::InternetStackHelper::Install (NodeContainer c) const – Agrega uma pilha de protocolos a cada nó do *container* de nós.

void ns3::InternetStackHelper::Install (Ptr< *Node* > *node*) const – Agrega uma pilha de protocolos ao ponteiro inteligente *node.*

4.1.8.6 Roteamento.

Roteamento é o processo de escolha do melhor caminho por onde os pacotes irão trafegar. No simulador NS3, utiliza-se uma abordagem centralizada quanto ao cálculo das rotas. Um objeto *Singleton* do tipo *GlobalRouteManager* calcula sozinho as melhores rotas e depois distribui para os nós.

Por *default*, todos os nós criados pela API Helper que são associados aos protocolos Internet Stack com o uso da *InternetStackHelper* possuem a capacidade de roteamento. As rotas calculadas são adicionadas às tabelas de roteamento de cada nó no inicio da simulação. Essas rotas possuem menor prioridade que as rotas estáticas.

A API Helper pública para a manipulação do roteamento global é simples, contendo apenas dois métodos. Existem classes que implementam outros tipos de

algoritmo de roteamento, porem estas não foram utilizados nesse trabalho. Abaixo temos uns exemplos de manipulação do mecanismo de roteamento global:

static void ns3::Ipv4GlobalRoutingHelper::PopulateRoutingTables(void) – Método para popular a tabela de roteamento dos nós. Esse método é chamado automaticamente no início da simulação.

static void ns3::Ipv4GlobalRoutingHelper::RecomputeRoutingTables(void) – Método para recalcular as tabelas de roteamento.

Caso seja necessário recalcular as tabelas de roteamento periodicamente, pode-se recorrer ao artifício abaixo.

Simulator::Schedule (Seconds (5), &Ipv4GlobalRoutingHelper ::Recompute Routing Tables) – Recalcula a tabela de roteamento a cada 5 segundos.

4.2 Simulação

Nessa subseção, serão discutidas algumas características da simulação construída no NS3 como suas limitações, modelo utilizado, script entre outras.

4.2.1 Limitação das Simulações

No atual ponto de desenvolvimento, o NS3 apresenta algumas limitações para simulação de algumas características de modelos de rede, em particular para simulação de ataques de negação de serviço.

A implementação base do NS3 não fornece uma estrutura para a mensuração de processamento e memória consumidos em um ataque de negação de serviço. Assim, para simular a falha de serviços como, por exemplo, um servidor WEB, é necessário o desenvolvimento de classes adicionais específicas, sobre a plataforma NS3 para esse fim.

Dessa forma, os principais efeitos que se pretende observar é o consumo de banda, que leva ao descarte de pacotes nos nós roteadores ou alvo, causado por ataques de inundação de pacotes.

4.2.2 Modelo de Simulação

Para a criação do modelo de simulação, optou-se por utilizar um modelo incremental em que a complexidade é aumentada a cada nova etapa. Esta forma de desenvolvimento permite a realização de testes de forma facilitada uma vez que cada etapa de incremento pode gerar um modelo completo para ser simulado. Assim, iniciou-se construindo primeiramente uma topologia básica a qual serviu de base para uma mais complexa, essas duas topologias serão vistas a seguir.

4.2.2.1 Topologia inicial.

A topologia escolhida para o modelo inicial consiste em um conjunto de nós, que representam *hosts* maliciosos, cada um ligado a um único roteador central através de enlaces Dial-Up, DSL ou de fibra ótica que são distribuídos segundo sugestão em (11). Nessa distribuição, são adotadas as proporções de 0,3; 0,6 e 0,1 respectivamente para esses enlaces. No entanto, essa distribuição ainda é ponderada pela quantidade de horas que, em média, uma máquina com esses enlaces ficam conectadas à internet. Essas horas, também seguindo sugestão, são respectivamente 2, 6 e 24 horas por dia. Essa métrica visa representar a disponibilidade de utilização dos *hosts* infectados pela máquina que coordena o ataque. Além disso, o roteador central também é ligado a um nó alvo através de um *link* de fibra óptica. Na FIG. 4.3, tem-se um exemplo da topologia para uma quantidade de 5 nós.

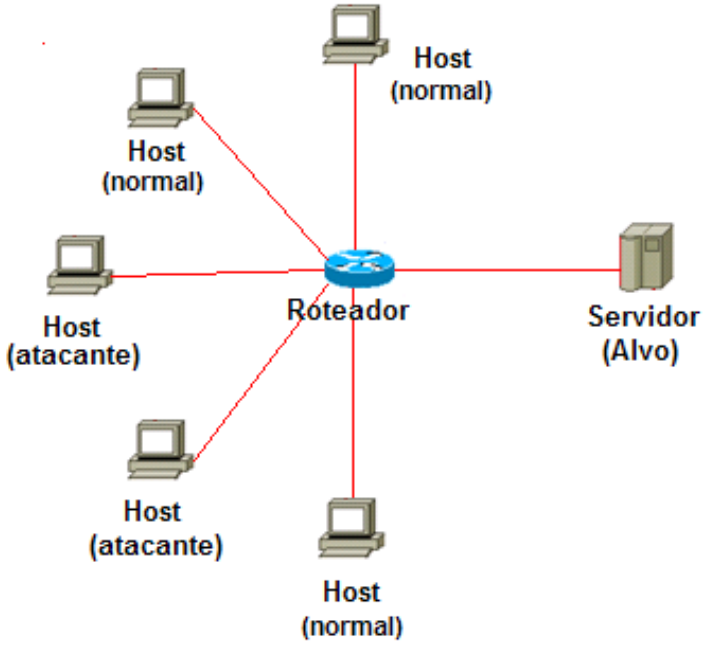

FIG. 4.3 Exemplo com 5 nós, sendo 2 atacantes, da topologia construída para a simulação no NS3.

Na implementação dessa topologia, a quantidade de *hosts* que a compõe está parametrizada, podendo variar de 1 a 127. Além disso, as máquinas que farão o ataque são aleatoriamente selecionadas seguindo um parâmetro passado para a simulação. Para a geração do ataque deste trabalho, foi-se utilizado 40% das máquinas como sendo maliciosas.

Os nós maliciosos enviam um fluxo de pacotes UDP, ICMP (*Internet Control Message Protocol*) e TCP simulando típicas ferramentas de ataque somando-se a isso um tráfego legitimo de fundo. Nesta etapa, a aplicação envia pacotes à taxa constante e possui um tempo aleatório para início de operação e fixo para o fim do ataque.

Para o envio de pacotes TCP, foi configurada uma aplicação responsável pela realização da conexão em três vias do protocolo TCP e posterior envio de pacotes de dados. Já o envio de pacotes UDP e ICMP, é feito através de aplicações com intervalo entre envio de pacotes convenientemente configurado e sem a realização de *spoofing* de endereço IP. No entanto, caso seja necessário, é possível utilizar esta técnica no NS3.

O nó roteador é caracterizado por um nó que possuí uma interface de rede para cada subrede dos nós maliciosos e uma interface de rede para a subrede do nó alvo. Como o protocolo de roteamento não é um fator crítico para a simulação do

ataque, optou-se por uma tabela de roteamento estática, criada antes do início da simulação.

Além disso, as interfaces de rede do nó roteador estão configuradas com o algoritmo Tail Drop (12), de gerenciamento de fila, para uso do buffer de pacotes. Nesse algoritmo, comumente utilizado em roteadores reais, todos os pacotes são tratados sem distinção no momento do enfileiramento, que ocorre até que toda a memória, ou a capacidade máxima configurada, seja preenchida. Estando a fila cheia, os pacotes que chegam à interface são automaticamente descartados.

O nó alvo está configurado com uma aplicação para escuta de porta TCP. Essa aplicação basicamente implementa o estabelecimento da conexão em três vias do protocolo TCP e, após a realização bem sucedida da conexão, comporta-se como um sorvedouro de pacotes TCP.

Conforme previsto pelo padrão TCP/IP, o *host* responde normalmente aos pacotes ICMP *Echo Request*, dos nós maliciosos, com pacotes ICMP *Echo Reponse.*

4.2.2.2 Topologia principal.

Visando simular um ataque de negação de serviço o mais realístico possível, construiu-se, a partir da conexão de topologias iniciais, uma nova topologia maior e mais complexa. Essa topologia é composta de três níveis de roteadores, no qual o servidor alvo está ligado ao roteador de nível três. Os hosts atacantes estão separados em seis subredes nas quais cada uma contém 127 hosts. Tal topologia está ilustrada na FIG. 4.4.

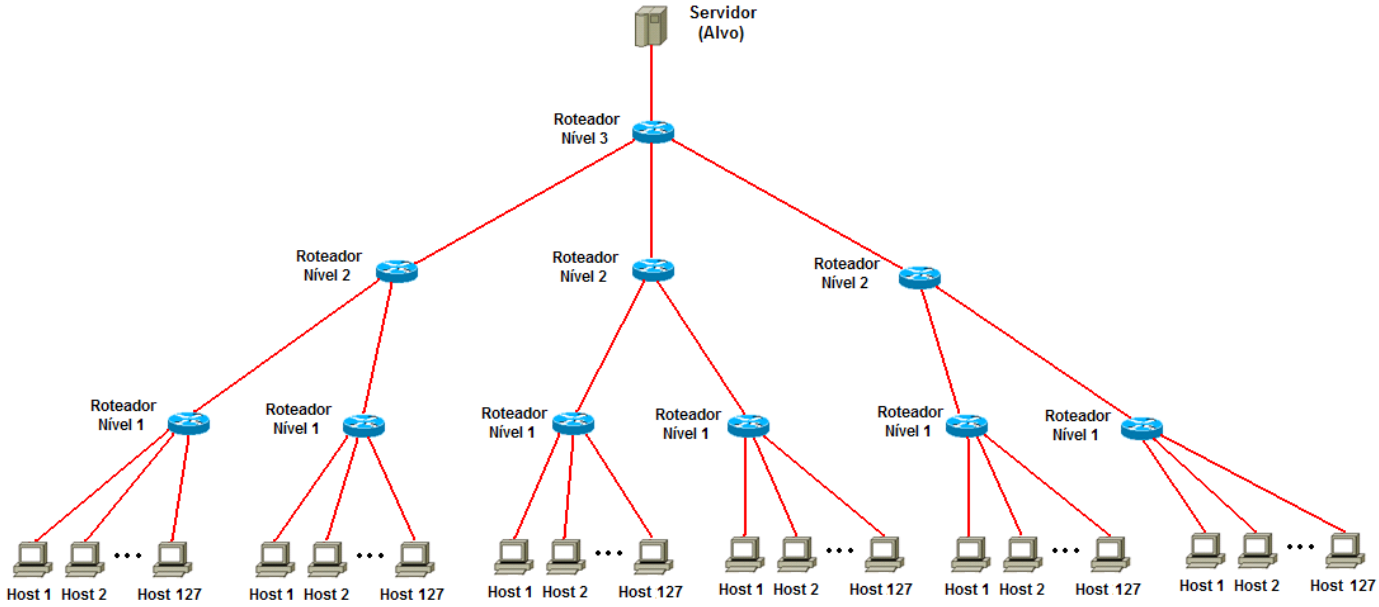

FIG. 4.4: Esquema da topologia principal da ataque.

Dentro de cada subrede, 40% dos hosts são selecionados aleatoriamente em cada simulação para participarem do ataque. Além disso, as conexão (host-roteador nível 1) dentro de cada subrede continua sendo distribuída entre ADSL, Dial-Up e fibra ótica conforme a topologia inicial. Seguindo aproximadamente uma proporção das larguras de banda da Rede Rio (13), as conexões entre os roteadores nível 1 e nível 2 são de 100Mbps e entre os de nível 2 e nível 3 são de 1Gbps ambas de fibra ótica. A conexão entre o roteador nível 3 e o servidor alvo possui largura de banda de 100Mbps, sendo também de fibra ótica. Somando-se a isso, configuraram-se adequadamente as latências de acordo com o tipo de cada *link*. Essas características de diferentes enlaces visam reproduzir a heterogeneidade das redes reais existente.

Equivalentemente à simulação feita com a topologia inicial, está será feita através da inundação do servidor por pacotes TCP, UDP e ICMP. Além desses pacotes de ataque, foi implementado também trafego de fundo.

Neste caso, tem-se uma escala bem maior que a da topologia inicial, contendo um número de atacantes de aproximadamente 305, 40% dos 762 hosts da topologia.

É importante notar que grande parte das características da topologia está parametrizada. Assim, limitações como capacidade de processamento e

armazenamento do *tracing* fazem com que sejam escolhidos valores modestos para os parâmetros. No entanto, nada impede que esses valores sejam extrapolados no script da simulação, sendo necessários apenas recursos computacionais suficientes.

4.2.3 *Script* da Simulação

A criação de uma topologia de rede, no simulador NS3, em geral, segue um processo que se inicia com a criação dos nós de rede que farão parte da simulação. Após a criação dos nós, cria-se a pilha de protocolos, normalmente utilizando-se a pilha de protocolos Internet. A seguir, criam-se os *links* entre os nós, podendo-se utilizar enlaces ponto-a-ponto, barramentos Ethernet, entre outros. A criação e configuração desses *links* variam ligeiramente segundo o tipo escolhido, mas todos fornecem uma classe Helper que facilita sua criação e configuração. Com os objetos de enlace prontos criam-se, a partir deles, as interfaces de rede para os nós. Nesta etapa, pode-se utilizar especializações da classe de interface de rede, como as interfaces de rede próprias para o uso de IPv4.

Para melhor organização dos objetos que representam os nós de rede, foram criados três objetos *containers* para os nós maliciosos, roteador e alvo. O uso de *containers* para objetos trás diversos benefícios para o desenvolvimento do *script* de simulação uma vez que reduz a necessidade de uso de diversos nomes para um conjunto de objetos além de permitir o acesso a esse conjunto de objetos de forma unificada. Um quarto container foi criado com o único objetivo de permitir o acesso a todos os nós da simulação de forma unificada.

A pilha de protocolos da Internet pode ser agregada a todos os nós através de sua classe *helper InternetStackHelper* e do container contendo todos os nós.

A criação dos enlaces entre os nós é feita através da classe *helper PointToPointHelper* que implementa um canal *full duplex* e uma versão simplificada do protocolo PPP (*Point-To-Point Protocol*). Como cada par (nó malicioso - roteador) e (alvo - roteador) estão em subredes diferentes, optou-se por criar um objeto da classe Vector, disponível na biblioteca STL do C++, para armazenamento de *containers* de interfaces de rede, *NetDeviceContainer*, contendo as interfaces dos pares. Uma vez que os *containers* de interface de rede, com os *links* ponto-a-ponto criados, estejam prontos, pode-se associar os endereços IP às interfaces através do

helper Ipv4AddressHelper, que permite a configuração dos endereços IP e máscara da subrede, e da classe especializada *Ipv4InterfaceContainer* que armazena as interfaces de rede já configuradas com o IPv4, a partir da sua superclasse não especializada.

Até aqui, esses procedimentos foram utilizados para a implementação do módulo básico. Para a construção da topologia principal, criou-se um método de acesso ao roteador, da topologia inicial, com o nome de getRouter(). Através desse método é possível ligar os módulos básicos em estrutura maiores utilizando-se das classes de criação de enlace do NS3.

4.2.3.1 Aplicações

As aplicações são objetos agregados aos objetos da classe *Node,* mas diferentemente das aplicações reais, que são acessadas através do nó em que estão sendo executadas, o acesso às aplicações da simulação é simplificado através de *containers ApplicationContainer*, que agrupam aplicações similares. Nos modelos de simulação, existem três *containers* de aplicação, a saber: Container para a aplicação de UDP *Flood*, para aplicação de ICMP *Flood* e para aplicação de TCP *Flood*.

Uma característica crítica, que torna as simulações mais próximas da realidade, é o uso de variáveis aleatórias para a criação de atributos não determinísticos. Uma série de atributos podem ser aleatorizados como latência de enlaces, intervalo de resposta de aplicações, falhas de comunicação, entre outros.

Nesse primeiro modelo, foi inserido um fator aleatório no tempo de início de operação de cada aplicação através de uma variável aleatória de distribuição uniforme.

A seguir, será apresentado o detalhamento da implementação dos objetos de aplicação presentes na simulação.

a) Implementação do UDP *flood*

As aplicações responsáveis por criar o fluxo de pacotes UDP, das máquinas maliciosas para o alvo, vem das classes *UdpServer* e *UdpClient*, presentes na implementação oficial do simulador.

A classe *UdpClient* faz a função de cliente, enviando ao servidor os pacotes UDP. Através do *helper UdpClientHelper*, pode-se configurar facilmente os três principais atributos da aplicação cliente: número máximo de pacotes enviados, tamanho de cada pacote e intervalo entre envios. O endereço IP e a porta do *host* destino, nesse caso o alvo, são passados como parâmetros no método construtor da classe *UdpClientHelper*.

A classe *UdpServer* faz a função de servidor, recebendo e processando os pacotes UDP vindo dos clientes. Essa aplicação é instalada na máquina alvo, ao qual simula a execução de um serviço UDP. Assim como a classe *UdpClient*, a *UdpServer* também possui alguns parâmetros como, por exemplo, o número da porta ao qual irá responder. Na ausência dessa aplicação no servidor, os clientes receberiam pacotes ICMP de resposta dizendo que não há serviço prestado, assim como ocorre nas redes reais.

b) Implementação do ICMP *flood*.

Para a implementação do envio de pacotes ICMP, fez-se uso da classe V4PingHelper que compõe o sistema NS3. Esta classe é apenas uma facilitadora na utilização da classe V4Ping. A classe V4PingHelper cria uma aplicação *ping* e a associa a um nó da rede. Para que isso seja feito, ela cria várias instâncias da classe V4Ping, configura-as de acordo com os parâmetros passados e instala-as nos respectivos nós desejados.

Alguns parâmetros que podem ser passados para a classe V4PingHelper são:

Intervalo de tempo entre envios consecutivos
Tamanho do pacote ICMP
Endereço IP de destino.

Além disso, pode-se também configurar o tempo de início e término da aplicação Ping.

c) Implementação do TCP *flood*.

O simulador NS3 possui um conjunto de classes que implementa uma simples aplicação de envio e recepção de pacotes TCP. Duas classes foram utilizadas para implementar esta funcionalidade, são elas: *PacketSinkHelper* e *OnOffHelper*. Estas classes são apenas facilitadoras e formam um nível acima ao das classes *PacketSink* e a *OnOffApplication* que realmente implementam as aplicações.

A classe *PacketSinkHelper* cria uma aplicação responsável por receber e consumir tráfego gerado por um nó qualquer. Nesse caso, foi utilizado para consumo de tráfego TCP mas também pode ser utilizada para consumo de outros protocolos de internet básicos. Nessa classe, podem-se configurar alguns parâmetros como, por exemplo, os endereços IP aos quais a aplicação responderá e a porta TCP na qual ficará escutando.

A classe *OnOffHelper* cria uma aplicação responsável por gerar o tráfego. Em outras palavras, ela funciona como um cliente. Nesse caso, foi utilizada para gerar tráfego TCP. A escolha do protocolo é passada como parâmetro para o construtor da classe.

Esta aplicação possui como característica gerar tráfego a uma taxa constante de bits e passar entre os estados ligado “On” e desligado “Off”. No estado ligado, a aplicação gera tráfego. No estado desligado, a aplicação não gera tráfego. O tempo de duração de cada estado pode ser configurado através de parâmetros que recebam valores constantes ou variáveis aleatórias. Além disso, existem outros parâmetros que podem ser configurados como os abaixos:

- Taxa de bits de envio dos pacotes.
- Tamanho dos pacotes
- Endereço IP de destino.
- Quantidade total de bytes enviados.
- Número da porta.

Além disso, pode-se também configurar o tempo de início e término dessas aplicações através de chamadas de métodos específicos.

d) Implementação do Tráfego de Fundo.

O tráfego de fundo é a transferência de dados comuns, isto é, considerada não-maliciosa, entre o *host* e o servidor. A taxa de envio de dados pelo *host* é modelada probabilisticamente segundo diversas distribuições como descrito em (14). Por isso, foi criada uma classe chamada de *LegitimateTraffic,* para a geração de tráfego legítimo, que oferece a possibilidade de escolha do tipo de variável aleatória utilizada no envio de dados.

Assim, a classe oferece a possibilidade de modelagem tanto no tamanho como no número dos pacotes enviados através de variável aleatória escolhida, podendo inclusive que estas variáveis sigam distribuições distintas. Assim, seguindo (14), as distribuições utilizadas foram a de Poisson para o número de pacotes e a de Paretto para o tamanho dos pacotes.

Para o envio de dados, pode-se optar pelo uso de *Socket* TCP ou *Socket* UDP, mas o modelo implementado utiliza *Socket* TCP para a geração de tráfego de fundo através de requisições a porta 80.

Seguindo o modelo de desenvolvimento utilizado no NS3 criou-se, também, uma classe *helper*, a *LegitimateTrafficHelper*, para configuração e instalação dessa aplicação nos nós. Através dessa classe *helper*, configura-se o endereço do nó servidor, define-se o tipo das variáveis aleatórias além dos valores de tempo para início e término de execução da aplicação de geração de tráfego de fundo.

4.2.3.2 Verificação da simulação.

A verificação da simulação foi feita em cima da topologia inicial. Foram feitas algumas simulações com essa topologia alterando alguns parâmetros básicos como tempo de execução, distribuição dos endereços IP nos hosts e número de pacotes máximos enviados. Após isto, utilizou-se o software Wireshark (15) para abrir os *tracings* (.PCAP) de cada máquina da rede inicial. Os *tracings* ASCII, que possuem maior quantidade de informação, foram abertos com um simples editor de texto.

Assim, foi feita uma verificação visual dos pacotes. Alguns itens conferidos foram:

- Se o sistema de roteamento estava funcionando corretamente;

- A correspondência dos tempos de transito dos pacotes com as latências dos enlaces;
- Se o número de pacotes estava condizente com a largura de banda disponível;
- Se ocorria descarte de pacotes;
- O número de sequência dos pacotes TCP, entre outros itens.

Todos os quesitos verificados conferiram satisfatoriamente com os valores teóricos calculados, mostrando, assim que a topologia inicial estava corretamente implementada. No entanto, parece haver dois problemas na implementação do NS3.

O primeiro é que os *checksums* dos pacotes IP criados pelo simulador não confere com os teóricos. Isso parece ser feito propositalmente para diminuir o custo computacional das simulações (16). Para forçar o cálculo do *checksum*, deve-se passar parâmetros para a simulação via linha de comando.

O segundo problema encontrado foi na implementação do protocolo TCP. O mecanismo de *timeout* parece não funcionar corretamente, pois caso uma das partes envolvidas na conexão fique indisponível, a outra parte fica enviando indefinidamente pacotes TCP tentando restabelecer a comunicação.

A verificação da topologia principal não foi executada tão detalhadamente como a da topologia inicial. Isto ocorreu por dois motivos principais. O primeiro, pois como a topologia principal é composta de vários módulos da inicial, acredita-se que a probabilidade de ocorrência de erros na principal seja menor. O segundo, como a topologia principal é bem maior em escala que a inicial, a verificação visual de todos os itens abordados na verificação da primeira demandaria muito tempo na segunda.

Apesar de não ter havido uma verificação tão minuciosa da topologia principal, ela está respondendo corretamente a todos os casos de teste de simulação.

4.2.4 *Tracings*

O modelo implementado gera dois tipos de *tracing*: o *tracing* pcap e o *tracing* ascii.

O *tracing* pcap é gerado na forma de arquivos para cada interface de rede em cada nó. Esse *tracing* pode ser visualizado em diversas aplicações que suportam o padrão pcap como, por exemplo o Wireshark (15), e fornecem informações sobre estado da rede em diversos aspectos e momentos distintos. Em última instância, esse conjunto de arquivos é o resultado esperado por uma simulação.

O *tracing* ascii, por sua vez, é gerado na forma de um único arquivo para o modelo de simulação. Nele estão contidas informações relacionadas aos eventos do simulador, como empilhamentos nas filas de interfaces de rede ou recepção de pacotes pela a camada de enlace da pilha de protocolos. Seu uso é mais adequado para a depuração da simulação, funcionando como um arquivo de *log* do simulador.

5 RESULTADOS ALCANÇADOS.

Nesta seção será feita uma síntese dos principais resultados alcançados ao longo deste trabalho. Abaixo, serão expostos, em ordem cronológica, os resultados alcançados dando uma breve explanação sobre o mesmo.

a) **Criação de um módulo de topologia básica para a simulação:** Foi criada, após o estudo básico do NS3, uma topologia básica o qual serviu de base para a criação da topologia principal.
b) **Verificação do funcionamento da topologia básica:** Depois de criada a topologia básica, foram feitas algumas análises para verificar seu funcionamento. Essas análises constataram, satisfatoriamente, a equivalência entre a teoria de redes com os resultados obtidos na simulação.
c) **Geração de uma base de dados pela topologia básica:** Gerou-se, ainda que somente para testes, um *tracing* simplificado de um ataque de negação de serviço.
d) **Criação de classes reutilizáveis para geração de tráfego de fundo:** Como o NS3 não disponibilizava nenhuma API para geração de trafego de fundo, fez-se necessário a criação de uma classe para esse fim, assim como, a sua classe *helper*.
e) **Criação de uma topologia principal:** Tomando-se por base os módulos da topologia básica, pode-se construir uma topologia maior e mais complexa conectando os módulos básicos. A topologia principal resultante foi a base para a simulação pretendida.
f) **Geração da base de dados para a topologia principal:** Chegando-se ao principal objetivo deste trabalho, conseguiu-se gerar satisfatoriamente uma base de dados para uma simulação de ataque distribuído de negação de serviço.

6 CONCLUSÃO.

Pode-se concluir que o *Network Simulator* é um software bastante complexo devido, principalmente, ao grande número de classes que possui. Para estruturar essas inúmeras classes, tornar o código do NS3 mais legível e também escalável, utilizou-se intensamente, na sua implementação, padrões de projeto. Assim, em outras palavras, ele pode ser facilmente evoluído e adaptável às necessidades especificas, característica que não ocorria com o NS2.

Apesar de ser utilizado principalmente para a simulação de redes baseadas nos protocolos da internet, o NS3 também pode simular outros tipos de redes que utilizam pacotes para a transmissão de dados. Além disso, pode comunicar-se com redes físicas externas, máquinas reais e virtuais possibilitando, assim, uma grande flexibilidade de utilização. É possível, também, fazer com que simulações em diversas máquinas possam se interagir.

É importante notar que embora o NS3 seja bastante utilizado para validação de teorias, ele ainda possui a carência de um *framework* de segurança. Este *framework* possivelmente facilitaria e agilizaria testes de ferramentas novas e atuais que trabalham com redes. Acredita-se que esse trabalho possa ser uns primeiros passos para gerar código possibilitando, assim, compor um futuro *framework*.

Sobre o cumprimento dos objetivos, foram satisfatoriamente atingidos com a criação de uma topologia de ataque resultando na geração de *tracing*. No entanto, por mais que tentou-se copiar as características das redes reais e inserir aleatorização, ainda podem ser feitas muitas melhorias como por exemplo na estrutura da topologia principal que se encontra muito simétrica.

Finalizando este documento, deixam-se algumas sugestões para trabalhos futuros que tomam este como base. São elas: A criação de uma organização que simule sistemas autônomos; a implementação de um módulo *switch* nível 2; a submissão dos *tracings* gerados a uma ferramenta IDS para verificação do lançamento ou não de alertas e a inserção de protocolos de roteamento dinâmicos na topologia criada neste trabalho.

BIBLIOGRAFIA

1. Information about Viruses, Spyware, Trojans and Internet threats - Panda Security. *Panda Security.* [Online] [Cited: Maio 15, 2011.] http://www.pandasecurity.com/homeusers/security-info/.

2. About GloMoSim. *About GloMoSim.* [Online] [Cited: Maio 27, 2011.] http://pcl.cs.ucla.edu/projects/glomosim/.

3. Application and Network Performance with OPNET | Monitoring, Troubleshooting, Auditing, and Prediction. *OPNET.* [Online] [Cited: Maio 20, 2011.] http://www.opnet.com/.

4. IMUNES - An Integrated Multiprotocol Network Emulator / Simulator. *Department of Telecommunications.* [Online] [Cited: Março 15, 2011.] http://imunes.tel.fer.hr/imunes/.

5. IMUNES Manual. [Online] [Cited: Março 13, 2011.] http:// imunes.tel.fer.hr/imunes/dl/ imunes_ug_20100915.pdf.

6. **Junior, Fábio Luiz and De Sousa, Vinícius Hessel Benedito.** *Estudo de ataques distribuídos de negação de serviço.* Instituto Militar de Engenharia. Rio de Janeiro : s.n., 2010. Trabalho de iniciação à pesquisa.

7. Discussions. *NS-3-USERS.* [Online] [Citado em: 14 de Janeiro de 2011.] http://groups.google.com/group/ns-3-users?pli=1.

8. Grupos do Google. *Google.* [Online] [Cited: Abril 15, 2011.] http://groups.google.com/.

9. ns-3 Manual Release ns-3.11. [Online] [Cited: Março 20, 2011.] p. 6. http://www.nsnam.org/docs/release/ns-3.11/tutorial/ns-3-tutorial.pdf.

10. NS-3 Architecture. *NS-3 Poject.* [Online] [Cited: Setembro 24, 2010.] http://www.nsnam.org/docs/ architecture.pdf.

11. **Dagon, David, et al.** *A Taxonomy of Botnet Structures.* Georgia Institute of Technology.

12. Cisco IOS Release 12.0 Quality of Service Solutions Configuration Guide - Congestion Avoidance Overview. *Cisco.* [Online] [Cited: Maio 28, 2011.] http://www.cisco.com/en/US/docs/ios/12_0/qos/configuration/guide/qcconavd.html.

13. Rede Rio de Computadores. [Online] [Cited: Maio 29, 2011.] http://www.rederio.br/topologia.php#.

14. **Mirkovic, Jelena, Wilson, Brett and Hussain, Alefyia.** *Automating DDoS Experiment.*

15. Wireshark - Go deep. *Wireshark.* [Online] [Cited: Maio 20, 2011.] http://www.wireshark.org/.

16. NS3 Tutoria:Checksum error in IP packet. . *ns-3-users.* [Online] [Cited: Maio 20, 2011.] http://groups.google.com/group/ns-3-users/browse_thread/thread/ab8e0c5e4cded2e5?pli=1.