3
Ferramenta de Simulação

Para definir a ferramenta de simulação a ser utilizada para implementação do protocolo HIP e para coleta dos resultados de simulação com uso desse protocolo, realizou-se um estudo comparativo [8] entre as ferramentas OMNeT++ [21] e Network Simulator 2, NS-2 [4].

Os itens avaliados nessas duas ferramentas foram: o tipo de arquitetura, a linguagem de definição do cenário de simulação e de topologia de rede, a biblioteca de objetos e a documentação disponível, entre outros. Após esse estudo, optou-se pela ferramenta OMNeT++.

O OMNeT++ é um simulador de eventos discretos orientado a objetos. Graças a sua arquitetura modular, é possível, sempre que necessário, acoplar novos módulos a ele, para que novas funções sejam disponibilizadas como, por exemplo, a simulação de redes de sensores, do protocolo IPv6 e do protocolo MIPv6.

Ele também possui algumas denominações próprias, uma delas é o *modelo*. Essa nomenclatura representa a implementação do sistema ou protocolo que se deseja modelar. O modelo é a combinação de módulos simples e/ou compostos aninhados hierarquicamente. Dessa forma, o desenvolvedor de um modelo pode refletir a estrutura lógica de camadas independentes que possuem um canal de comunicação bem definido, como a arquitetura apresentada pelo modelo OSI e pelo modelo TCP/IP, no framework de simulação. Isso também facilita a extensão dos módulos existentes e o acréscimo de novos módulos. Por exemplo, o módulo NetworkLayer6 pode ser implementado como o módulo IPv6 unido ao módulo ICMPv6 e portas, estruturas de acesso aos módulos direcionais, de saída e entrada para a camada de transporte e camada de enlace / física, conforme apresentado na figura 10 a seguir:

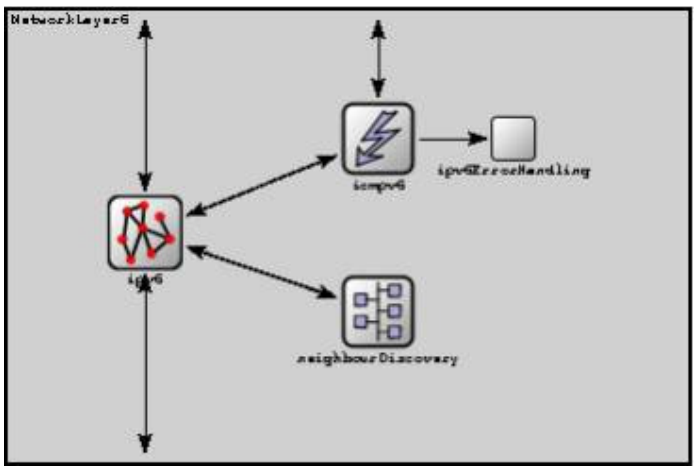

Figura 10: Módulo Composto NetworkLayer6: módulos IPv6 e ICMPv6

Os módulos que compõem um modelo se comunicam através da troca de mensagens, representação de quadros ou pacotes numa rede de computadores, trabalhos ou clientes numa fila ou qualquer outro tipo de entidade móvel. As mensagens podem conter estruturas complexas de dados de forma a garantir que as informações necessárias às camadas adjacentes sejam preenchidas corretamente. Cada módulo pode enviar mensagens diretamente para o destinatário final ou para um caminho pré-definido, através dos pontos de comunicação dos módulos.

O OMNeT++ é muito utilizado na simulação de protocolos e redes de telecomunicações. Por essa razão, ele possui uma estrutura chamada de *controlInfo* que permite que informações pertinentes à camada superior ou inferior sejam inseridas à mensagem original. Por exemplo, na implementação do TCP, quando um pacote é passado da camada TCP para a camada IP, o endereço IP de destino deve ser informado assim como possivelmente outros parâmetros. Da mesma forma que quando o IP passa um pacote para o TCP após o desencapsulamento do cabeçalho IP, ele deverá informar ao TCP pelo menos o endereço de origem do IP. Essas informações adicionais seriam encontradas nessa estrutura controlInfo.

Cada módulo possui pontos de conexão para outros módulos, conhecidos como portas. Elas representam os pontos de entrada e saída das mensagens entre módulos, que são sempre *simplex*, ou seja, sempre são de uma única direção.

As portas dos módulos são conectadas através de canais de comunicação. O canal especifica o tipo de conexão que haverá entre dois ou mais módulos bem como suas características, tais como taxa de transmissão, delay e taxa de erro do

meio de transmissão. Estes atributos são opcionais. Os canais podem ter um nome com o objetivo de permitir o reuso do mesmo em outros cenários de simulação.

Além da comunicação entre módulos, também é possível um módulo enviar mensagens para si mesmo com o objetivo de representar eventos internos. A esse tipo de mensagem chama-se auto-mensagens. Quando um módulo deseja esperar um determinado tempo, ele usa o recurso das auto-mensagens, enviando para ele mesmo mensagens enquanto o tempo estipulado não é atingido. Dessa forma, através dessa ação, o módulo naturalmente possui um mecanismo que representa um temporizador.

Existem dois tipos de módulos: compostos e simples. Os módulos que contém sub-módulos são denominados módulos compostos; já os módulos simples são aqueles que representam o nível mais baixo na hierarquia de módulos. Enquanto o conteúdo dos módulos compostos é a composição de sub-módulos definida através da linguagem de descrição de topologias NED (NEtwork Description) o conteúdo do módulo simples é implementado através da linguagem de programação C++, usando as classes da biblioteca de simulação do OMNeT++.

Tanto módulos simples como os compostos são instâncias de tipos de módulos básicos do framework ou de outros módulos gerados. Durante a descrição de um modelo, o desenvolvedor define os tipos de módulos e faz a instanciação dos módulos desenvolvidos a partir desses tipos.

O simulador também permite que os módulos possam ser armazenados em diretórios diferentes daquele onde se encontra o cenário de simulação, permitindo o agrupamento dos mesmos como melhor lhe convier. Por exemplo, é possível criar diretórios que representem cada camada do modelo TCP/IP e classificar os módulos simples e compostos obedecendo a essa arquitetura. Isso facilita na pesquisa e na extensão do framework além de permitir a criação de bibliotecas próprias.

Existem diversos frameworks disponíveis na página oficial do OMNeT++. Para esse estudo, os frameworks mais interessantes são:

- *IPv6SuiteWithINET* [6]: Implementação dos protocolos IPv4 e IPv6 com algumas de suas implementações de mobilidade (MIPv6, HMIPv6). Padrão 802.11 implementado apenas para arquitetura Infra-Estruturada;

- *INET Framework* [7]: Implementação dos protocolos IPv4 e IPv6 sem mobilidade a nível de rede. Padrão 802.11 implementado para arquiteturas Ad-hoc e Infra-estruturada;
- *Mobility Framework* [15]: Simulação de redes wireless e redes móveis no OMNeT++, por exemplo: redes Ad-hocs, rede de sensores.

Optou-se por utilizar o módulo *INETFramework* ao invés do *IPv6SuiteWithINET* para implementar o protocolo HIP por orientação dos próprios desenvolvedores da ferramenta, já que o segundo framework foi descontinuado e o suporte a possíveis problemas decorrentes de sua utilização deixou de ser assegurado.

O novo framework, apesar de também possuir o protocolo IPv6, não dispunha de qualquer estrutura de mobilidade IPv6 implementada. Por essa razão além da implementação do protocolo HIP, houve a necessidade de implementação de todos os mecanismos básicos de mobilidade para o protocolo IPv6: uso do protocolo Neighbour Discovery e atualização das tabelas de endereço IPv6 e de roteamento dos dispositivos móveis.

O *INETFramework* contém implementações dos protocolos IPv4, IPv6, TCP e UDP além de vários modelos de aplicação. Ele também inclui um modelo de redes MPLS [23] com sinalização RSVP-TE [3] e LDP [1]. Modelos do nível de enlace tais como PPP, Ethernet e 802.11 também estão presentes nesse framework. Rotas estáticas podem ser definidas usando módulos autoconfiguradores de rede ou podem ser utilizadas implementações de protocolos de roteamento, tais como OSPFv2 e RIP.

Todos os frameworks citados anteriormente são construídos sobre o OMNeT++ e usam o mesmo conceito de comunicação via troca de mensagens e definição de módulos e protocolos através do aninhamento de módulos simples e compostos. Nesse simulador, normalmente, protocolos são representados por módulos simples que possuem uma interface externa composta por portas, conectores opcionais e parâmetros descritos em arquivos NEDs e a implementação em classes C++ de mesmo nome. Estes módulos podem ser combinados livremente para formar tipos de dispositivos diferentes.

As interfaces de rede, por exemplo, a Ethernet e a IEEE 802.11, são comumente módulos compostos por uma fila, um endereço MAC e módulos específicos do modelo de interface, conforme se pode observar na figura 11.

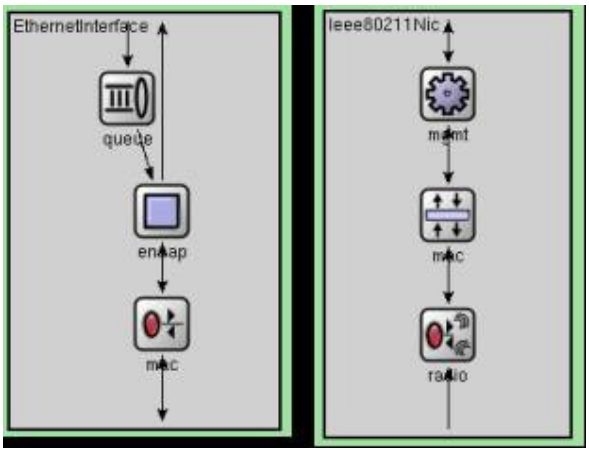

Figura 11: Módulos que compõem as interfaces Ethernet e IEEE 802.11

Nem todos os módulos implementam protocolos; existem módulos que:

- armazenam dados, como o módulo *RoutingTable* / *RoutingTable6*;
- facilitam a comunicação entre módulos, como o módulo *NotificationBoard*;
- executam a auto-configuração de todos os dispositivos que participam da simulação em uma única rede, como o módulo *FlatNetworkConfigurator* / *FlatNetworkConfigurator6*;
- controlam a direção e a velocidade da movimentação do nó móvel no espaço definido na simulação, como o módulo *ConstSpeedMobility*;
- executam a gerência e manutenção associadas a canais de rádio em simulações wireless, o módulo *ChannelControl*, entre outros.

No *INETframework*, quando um protocolo de camada superior quer enviar pacotes de dados para um protocolo de camada inferior, ele cria um cabeçalho de onde insere as informações necessárias para o tratamento da mensagem na camada inferior, cria o objeto mensagem com os dados que deseja transmitir, adiciona o cabeçalho à mensagem, e a envia à camada inferior, a qual também realizará o mesmo procedimento até alcançar a camada física. O processo inverso é implementado quando um protocolo de camadas inferiores recebe um pacote e o envia para cima após desencapsulá-lo. O cabeçalho externo dos dados é implementado com o uso do objeto *controlInfo*.

Os diretórios do framework INET serão apresentados a seguir com um pequeno descritivo:

- Applications: Aplicações disponíveis. As aplicações estão organizadas em sub-pastas que identificam o seu tipo: Ethernet, Generic, PingApp, TCPApp, UDPApp;
- Base: Implementação de funções básicas como filas, acesso aos módulos, mapeamento de protocolos, módulo NotificationBoard;
- Bin: Localização do arquivo compilado INET que será usado nas simulações;
- Documentation: API do framework detalhada;
- Etc: Localização de plugins, scripts, esquemas XML e arquivos default.ini (definição de valores padrão para parâmetros não definidos no arquivo ini da simulação) e netconf2.dtd (definição de estruturas que são compreendidas pelo parse de XML do framework);
- Examples: Alguns exemplos de simulações de diversos protocolos. Estão subdivididos nos diretórios: Adhoc, Ethernet, INET (uso de IPv4), IPv6, MPLS, OSPFv2, Quagga, RTP, Wireless;
- Mobility: Definição de tipo de movimento a ser aplicado no nó móvel durante a simulação. Exemplo: movimentos circulares, retangulares, lineares, aleatórios, entre outros;
- Network: Implementação dos protocolos que atual na camada de rede. Estão subdivididos em: ARP, AutoRouting, Contract (funções básicas para IPv6 e IPv4 – importantes: IPAddressResolver, IPProtocolId.msg), ICMPv6 (importante: IPv6NeighbourDiscovery), IPv4, IPv6, LDP, MPLS, OSPFv2, Quagga, RSVP_TE, TED;
- NetworkInterfaces: Implementação das interfaces de rede disponíveis no framework. Estão subdividos em: Contract (funções comuns), Ethernet, EthernetSwitch, Ieee80211, MF80211, MFCore, PPP, Radio;
- Nodes: Definição dos módulos compostos que serão utilizados diretamente nas simulações. Estão organizados nas seguintes pastas: Adhoc, INET (IPv4), IPv6, MPLS, Wireless (apenas para uso de IPv4);

- Obsoletes: Códigos antigos não mais utilizados;
- Tests: Testes disponíveis para verificação de funcionamento do framework;
- Util: Implementação de funções úteis ao funcionamento do framework, tais como parser para XML;
- World: Estrutura de controle para simulação de redes móveis / wireless.

Dentre as classes, estruturas e funções disponíveis nesse framework, vale destacar as seguintes na implementação do protocolo HIP:

- InterfaceEntry: Estrutura que armazena as informações de uma interface, tais como endereço IPv6, índice da porta do módulo, endereço MAC;
- InterfaceTable: Armazena e gerencia todas as informações no formato da estrutura InterfaceEntry. Cada dispositivo da simulação possui sua própria tabela;
- IPv6InterfaceData: Estrutura que contém os detalhes do endereçamento IPv6: validade do endereço, endereço preferencial, endereços nos diversos escopos: local, global;
- IPAddressResolver: Classe utilitária para encontrar informação IPv6 ou IPv4 de um dispositivo;
- RoutingTable6: Representa a tabela de roteamento IPv6 de cada dispositivo IP presente na simulação;
- IPv6, ICMPv6 e Neighbour Discovery: Módulos responsáveis por todo tratamento de camada de rede e fornecimento de dados para as estruturas de dados já apresentadas InterfaceTable e RoutingTable6.