Um Framework para a Simulação de Redes Móveis Ad hoc

Alexandre Mele & Markus Endler

Departamento de Informática
PUC-Rio
{mele, endler}@inf.puc-rio.br

PUC-RioInf.MCC30/03
September, 2003

Abstract

In this document we present MobiCS2, a flexible framework for prototyping and simulation of protocols for mobile ad hoc networks. After describing its structure and main hot spots, we present an instantiation of this framework for a simplified version of the DSR routing protocol using IEEE 802.11.

Keywords

Mobile Ad Hoc Networks, Framework, Simulation, Protocol Prototyping, Routing

Resumo

Neste documento apresentamos MobiCS2, um framework flexível para a prototipação e simulação de protocolos para redes móveis ad hoc. Depois de descrevermos a estrutura e os principais pontos de flexibilização do framework, apresentaremos uma instanciação do mesmo para (uma versão simplificada) do protocolo de roteamento DSR sobre o IEEE 802.11.

Palavras-chave

Redes Móveis Ad hoc, Framework, Simulação, Roteamento, Prototipação de Protocolos

Um Framework para a Simulação de Redes Móveis Ad hoc

Alexandre Mele e Markus Endler
Departamento de Informática
PUC-Rio
Marquês de São Vicente 225
22453-900 Rio de Janeiro
{mele,endler}@inf.puc-rio.br

1. Introdução

Com o recente amadurecimento e proliferação das tecnologias para comunicação sem fio e de dispositivos móveis com diferentes capacidades de processamento e comunicação de dados, aumentou também o interesse por protocolos, serviços e aplicações para redes móveis ad hoc.
Estas redes tendem a ser muito dinâmicas e instáveis, demandando o projeto de novos protocolos adequados para estes tipos de redes. Estes protocolos podem se prestar a uma série de diferentes tarefas, tais como, encaminhamento de pacotes, anúncio e descoberta de recursos remotos, difusão e gerenciamento das informações dinâmicas sobre os recursos e serviços disponíveis na rede, entre outros. Esta demanda pelo desenvolvimento de novos protocolos específicos para redes ad hoc por sua vez requer a existência de ferramentas adequadas para a prototipação, a depuração e a análise de desempenho e escalabilidade destes protocolos em um ambiente simulado. Devido à diversidade de tecnologias de comunicação sem fio e dos dispositivos móveis por um lado, e dos protocolos a serem desenvolvidos por outro lado, torna-se imperativo que as ferramentas sejam suficientemente flexíveis para que possam ser adaptadas às necessidades e ao grau de abstração desejado por cada usuário. Além disto, o uso de tal ferramenta deve ser simples e de rápido aprendizado, de forma que a mesma possa ser utilizada também para fins didáticos.

Este trabalho descreve o projeto e implementação do MobiCS2, um framework em Java para a prototipação e simulação de protocolos para redes móveis ad hoc. Neste trabalho seguiu-se a filosofia de projeto de outro framework, MobiCS [RE00, Mob02] para redes móveis infra-estruturadas, que tem sido bastante usado para o ensino e pesquisa de protocolos para redes móveis. Além de prover um arcabouço estruturado e extensível para a criação de ambientes de simulação de protocolos para redes ad hoc, esta nova versão do framework também fornece implementações de alguns *hot-spots* (pontos de flexibilização) que servem de base para futuras extensões e customizações do framework.

Na próxima seção apresentamos simuladores e frameworks relacionados ao nosso trabalho. Na seção 3 descreveremos a estrutura do framework e os modelos de simulação. Em seguida, na seção 4, fazemos uma breve revisão do padrão IEEE 802.11 e do DSR, explicaremos como protocolos são programados no framework, e descrevemos detalhes da instanciação para 802.11 e DSR. Na seção 5 descrevemos um cenário de simulação realizado e na seção 6 fazemos umas conclusões.

2. Trabalhos Relacionados

Nesta seção serão apresentados os principais ambientes de prototipação e simulação para redes móveis, bem como alguns frameworks, incluindo o framework MobiCS, que serviu de motivação para este trabalho.

O ns-2 [NS2] é um simulador flexível de protocolos de rede que suporta simulações de protocolos para redes móveis ad hoc, organizados em camadas. O ns-2 também pode ser usado como uma emulador de rede, o qual é capaz de interagir com uma rede real. O ns-2 inclui um gerador de cenários, o qual permite a geração automática de diferentes topologias de redes, padrões de tráfego e de falhas.
O simulador tem um modelo de programação em dois níveis: o núcleo do simulador é implementado em C++ que interage com módulos programados em OTcl, uma extensão orientada a objeto do Tcl. Enquanto o núcleo implementa a lógica e as primitivas da máquina de simulação (*simulation engine*), o OTcl é usado para desenvolver os scripts de simulação e descrever as funcionalidades dos elementos de rede simulados. Ambas as linguagens podem ser utilizadas para implementar protocolos de rede a serem simulados. Devido este modelo de

programação, o ns-2 é um simulador de protocolo de propósito geral, flexível, e extensível.
O ns-2 implementa quatro protocolos de roteamento para redes *ad hoc*: DSDV, DSR, AODV e TORA. Para a subcamada MAC existem duas opções: IEEE 802.11 ou TDMA, enquanto que para a camada de transporte estão disponíveis os protocolos TCP e UDP. É possível definir ainda qual é o tipo de antena utilizada (p.ex.. omnidirecional), qual é o consumo de energia, e o alcance do sinal de rádio de cada nó móvel. Os nós móveis são dispostos, através de coordenadas cartesianas, em um espaço tridimensional delimitado e gradeado, sendo que a resolução da grade pode ser ajustada. Para movimentar um nó móvel neste espaço são informadas as coordenadas de seu destino e a sua velocidade até este destino.
As principais diferenças do MobiCS2 com relação ao ns-2 são o uso de uma única linguagem (Java), tanto para a extensão do simulador e dos modelos de simulação, como para a programação dos protocolos. Além disto, por ser um framework orientado a objeto (e não uma biblioteca de módulos), o MobiCS2 pode ser customizado de uma forma mais ampla.

O GloMoSim (*Global Mobile Information Systems Simulation Library*) [ZBG98] é um simulador de redes sem fio baseado em bibliotecas, desenvolvido na UCLA. O GloMoSim é composto por uma coleção de módulos de biblioteca implementados em PARSEC (*PARallel Simulation Environment for Complex systems*) [Bag98], uma linguagem baseada em C para descrição de simulações seqüenciais e paralelas.
O GloMoSim implementa um conjunto de protocolos de rede para comunicação sem fio, que são estão organizados em uma arquitetura em camadas. Novos protocolos e módulos implementados em PARSEC podem ser facilmente adicionados à biblioteca do GloMoSim para compor diferentes simulações. O núcleo do PARSEC implementa uma máquina de simulação de alto desempenho, que permite ao GloMoSim simular de redes de larga escala com total transparência de simulação, tanto para o programador do protocolo quanto para o usuário do simulador. No modelo de programação do PARSEC, os protocolos são implementados em módulos monolíticos chamados *entities* (entidades), permitindo somente a composição vertical de protocolos (protocolos em camadas). As primitivas do PARSEC são insuficientes para a programação de novas abstrações, tais como padrões de mobilidade, de conectividade, de consumo de energia, etc.
Enquanto o enfoque do GloMoSim é ser um ambiente para simulação eficiente de redes em larga escala, no MobiCS2 os principais objetivos são simplicidade de uso e extensibilidade.

O MobiCS (*Mobile Computing Simulator*) [RE00, Mob02] é um framework de um simulador para a prototipagem, teste e avaliação de protocolos distribuídos para computação. A versão atual do MobiCS contempla apenas redes móveis infra-estruturas, e para estas implementa abstrações como: host móvel (MH – *Mobile Host*), estação de suporte à mobilidade (MSS – *Mobility Support Station*), host fixo (FH – *Fixed Host*), célula, enlaces com fio e sem fio, entre outras.
Uma das principais contribuições do MobiCS é definir uma arquitetura de software que permite a troca do modo de simulação sem afetar o protocolo sendo prototipado, ou seja, o mesmo protocolo pode ser testado em diferentes modos de simulação sem a necessidade de alteração do seu código. O MobiCS implementa dois modos de simulação: determinístico e estocástico. No modo determinístico o usuário define um script de simulação descrevendo um cenário particular, que é executado passo a passo, permitindo ao usuário observar o estado do protocolo e/ou elemento de rede a cada momento da simulação. No modo estocástico o usuário atribui padrões de comportamento probabilístico aos elementos simulados (MH, MSS, enlaces), relativos à sua mobilidade, conectividade, propensão a falhas, etc. O modo determinístico é bastante útil para a depuração de protocolos, enquanto que o modo estocástico é tipicamente usado para avaliar um protocolo com relação a sua complexidade de mensagens (sua escalabilidade) e sua sensibilidade à variações na razão requisição- mobilidade (*call-to-mobility ratio*).

O framework MobiCS é disponibilizado através de uma biblioteca de classes Java, que oferece um ambiente flexível e extensível (capacidade de criar novas abstrações) para a criação rápida de simuladores com modelos de rede baseados em abstrações adequadas para o protocolo a ser prototipado (e simulado). Portanto, ao contrário do que ocorre em ns-2 e GloMoSim, o usuário do MobiCS utiliza uma única linguagem de programação, Java - largamente difundida e disponível em qualquer computador - tanto para desenvolver o protocolo quanto para definir a

configuração da rede e estender os modelos de simulação.
Além disto, o MobiCS provê ao usuário um modelo de programação baseado em micro-protocolos [Bha98], através do qual é possível descrever protocolos altamente modularizados e estruturados em componentes funcionais, que interagem através de eventos. No MobiCS estas componentes são *Wired*, *Wireless* e *Hand-off*, relativos, respectivamente, à comunicação do protocolo na rede fixa, pelos enlaces sem fio, e à comunicação relacionada a uma migração de um MH entre célula.

No entanto, por ser específico para redes infra-estruturadas, MobiCS contém certas limitações e simplificações relativas à comunicação sem fio (p.ex. cada MH interage com no máximo uma MSS e não há interferências entre os MHs em uma célula) que tornam impossível o seu uso para redes móveis *ad hoc*. Estas limitações do framework original motivaram o presente trabalho.
Existem também outros conhecidos frameworks para a prototipação e simulação de protocolos, tais como o x-kernel/x-sim [HP91], o NEST [DSY90] ou o *Scalable Simulation Framework* [Cow99], mas nenhum deles contempla redes móveis.

3. O Framework MobiCS2

O framework **MobiCS2** foi desenvolvido com o objetivo de ser uma ferramenta de fácil aprendizado, flexível e extensível para a prototipação e simulação de protocolos para redes móveis ad hoc. Os principais requisitos que nortearam o trabalho foram os seguintes:

- Criar abstrações que tivessem uma analogia direta com o domínio das redes móveis *ad hoc*;
- Permitir que um nó móvel pudesse executar mais de um protocolo, e possibilitar a estruturação de protocolos em camadas (pilhas de protocolos);
- Criar as abstrações *de modelo de mobilidade* e *modelo de energia* (para os nós da rede) que pudessem ser estendidos e detalhados independente do conceito de nó;
- Disponibilizar no framework algumas implementações dos pontos de flexibilização (*hot-spots*), principalmente relativos às camadas física e de enlace, de forma a facilitar o uso para camadas de protocolos de rede e superiores;
- Permitir a troca do algoritmo para escalonamento de eventos usado no núcleo do simulador.
- Criar abstrações *para modelos de conectividade* entre os nós com variados graus de refinamento, desde uma simples topologia de interconexão, até a inferência do grau de conectividade a partir de variáveis tais como posição dos nós e alcance do sinal de rádio.

O framework MobiCS2 é composto de quatro pacotes principais: **`nodes`**, **`commsys`**, **`simulation`** e **`util`**, mostrados na figura 1.

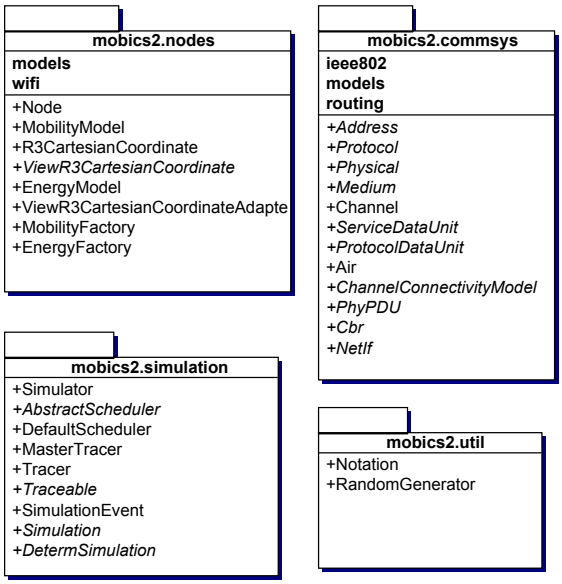

Fig. 1: Pacotes do Framework MobiCS2

O pacote **`nodes`** contém as classes e interfaces que estão diretamente relacionadas com o nó de rede, como por exemplo, o seu modelo de mobilidade e de energia.
O pacote **`commsys`** (do inglês *Communication System*) dá suporte ao sistema de comunicação, e possui abstrações para representar os meios físicos de transmissão, os canais de comunicação, endereços, interfaces de rede, e principalmente os protocolos.
O pacote **`simulation`** contém as classes usadas na máquina de simulação e que lidam com o escalonamento e entrega de quaisquer eventos, tais como a chegada de um pacote em um protocolo, a ocorrência de um timeout, ou o término de energia em um nó da rede.

O pacote **util** contém classes utilitárias do framework, como por exemplo, uma classe para geração de números pseudo-aleatórios.
A seguir, descreveremos as principais classes de cada pacote.

3.1 O Pacote nodes

A principal classe do pacote nodes é a classe **node**, que representa o nó da rede móvel ad hoc. As suas principais características são: possuir uma posição no espaço, que só pode ser alterada pelo seu modelo de mobilidade (**MobilityModel**); possuir recursos de energia, representado por **EnergyModel**; e possuir um conjunto de protocolos, podendo formar uma pilha de protocolos, possibilitando o acesso ao sistema de comunicação.
MobilityFactory e **EnergyFactory** são responsáveis respectivamente pela criação destes modelos e são utilizadas pelo nó da rede.
Por fim, a classe **R3CartesianCoordinate** representa um sistema de coordenadas cartesianas ortogonais no espaço R^3, e a interface **ViewR3CartesianCoordinate** contém um subconjunto dos métodos da primeira, que permite apenas consultar a posição, mas não alterá-la. Tal interface pode ser usada, por exemplo, por protocolos (de roteamento) que se baseiam na posição física de um nó, como por exemplo o protocolo LAR [KV98].

3.2 O Pacote commsys

A classe abstrata **Protocol** representa uma abstração do protocolo de comunicação. Esta classe serve de base para todos os protocolos a serem simulados na rede móvel ad hoc. Todo protocolo tem dois atributos de identificação, que são o nome e a versão do protocolo. A utilização da classe **Protocol** para a programação de protocolos será discutida na seção 4.
As classes abstratas **ServiceDataUnit** e **ProtocolDataUnit** representam, respectivamente, a Unidade de Dados do Serviço (SDU) e a Unidade de Dados do Protocolo (PDU), ambas baseadas no modelo de referência OSI da ISO.
A classe abstrata **Address** representa uma abstração de um endereço, que pode ser de três tipos: *unicast*, *broadcast* ou *multicast*. Esta classe deve ser estendida para representar endereços da camada de rede, da subcamada MAC, etc. Um protocolo pode possuir vários endereços. Por exemplo, um protocolo da sub-camada MAC tem o seu endereço MAC, mas pode também pertencer a um grupo *multicast*, possuindo assim um segundo endereço.
A classe abstrata **Physical** define uma abstração do nível físico do modelo de referência OSI. Esta classe é responsável pela conexão do nó de rede ao meio físico para transmissão e recepção de dados entre os nós. Esta classe deve ser especializada para refletir as características específicas do modelo pretendido, como por exemplo, as propriedades de nível físico do padrão IEEE 802.11.
A classe abstrata **Medium** define uma abstração de um meio de transmissão (um par trançado, fibra óptica, ar, etc.) usado para comunicação entre os nós da rede. A classe **Physical** de um nó deve estar associada um meio para que este nó possa transmitir para outros nós da rede, o que é feito através do método **attach()**.
A classe **Channel** representa um canal de comunicação em um meio de transmissão. No modelo, **Medium** pode possuir vários canais, como por exemplo, um canal para transmissão de dados e o outro canal para controle.
A classe abstrata **ChannelConnectivityModel** é um modelo abstrato da conectividade associado a cada canal. Este modelo determina a cada momento (instante simulado) quais outros nós da rede estão ao alcance de transmissão de cada nó. Este modelo define também qual é o tempo de propagação desta transmissão, podendo-se também alterar o dado entregue ao destinatário, por exemplo para simular erros durante a transmissão.
Quase todas as classes são pontos de flexibilização do framework. A única exceção é a classe **Air** (uma implementação da classe **Medium**) que representa o meio "ar", por onde trafegam as ondas de rádio.

3.3 O Pacote simulation

A classe **Simulator** é a principal classe do *núcleo do simulador* no framework. É através desta classe que se comanda o início e o término de uma simulação, e que se configura o ambiente de simulação. Por exemplo, através do método **setRegion()** é especificado o espaço para movimentação dos nós, e que é usado modelo de mobilidade dos nós móveis.
A classe abstrata **AbstractScheduler** define as abstrações de um escalonador e, juntamente com a classe **Simulator**, permite modificar o algoritmo utilizado pelo núcleo de simulador para escalonar e entregar os eventos, incluindo assim mais um ponto de flexibilização do framework. A classe

DefaultScheduler estende a classe **AbstractScheduler**, implementando um escalonador default (simulação seqüencial de eventos discretos) de **Simulator**.
A classe abstrata **Simulation** foi criada apenas para facilitar a criação de simulações. Extensões desta classe devem conter o código para a criação e configuração da rede móvel, como o número de nós móveis, região de movimentação, etc.
A classe abstrata **DetermSimulation** estende a classe **Simulation**, e serve para criar simulações controladas, executadas passo-a-passo. Este tipo de simulação é interessante quando se deseja ter total controle sobre os eventos da simulação, facilitando a depuração de um protocolo. Neste caso, define-se também um script de simulação com a sequência de eventos a ser simulada.

3.4 Modelos de Simulação

Os modelos de simulação são pontos de flexibilização do framework através dos quais o usuário descreve as propriedades dos nós da rede e do meio físico de comunicação. O framework MobiCS2 prevê três modelos de simulação: o *modelo de mobilidade*, o *modelo de energia* e o *modelo de conectividade do canal*, que, segundo a nossa convicção, são suficientes para caracterizar a grande maioria das redes móveis ad hoc. O framework contém versões básicas destes três modelos, que podem ser especializados de acordo com as necessidades do usuário.

3.4.1 Modelo de Mobilidade

Apesar do conceito de mobilidade estar ligado diretamente ao conceito de nó nóvel, em vez de representar a mobilidade diretamente na classe **Node**, foi criada uma classe separada para representar o modelo de mobilidade (classe **MobilityModel**), que é responsável por definir a mobilidade dos nós da rede, isto é, por atualizar o posição de cada nó. Sua implementação consiste de um vetor velocidade, que a cada instante simulado pode ser consultado ou alterado, atualizando assim a posição do nó.

O MobiCS2 disponibiliza uma implementação de modelo de mobilidade (classe **RandomWayPoint**) que é uma implementação um pouco modificada do modelo *Random Way-Point* [JM96], um modelo bastante utilizado em simulações de redes móveis ad hoc.

3.4.2 Modelo de Energia

O modelo de energia (classe **EnergyModel**) é responsável por definir o nível de energia do nó da rede, representado pelo atributo **level**, e que varia entre 0 e 1. Quando o nível de energia é igual a 0 isto significa que o nó está sem energia, fazendo com que todo processamento dos protocolos no nó seja interrompido e o nó passe para o estado (*power off*).

A classe **EnergyModel** possui o método `consume()` para reduzir o nível de energia. Através deste método, pode-se especificar a quantidade de energia que deve ser consumida em determinada tarefa, como por exemplo na transmissão e recepção de pacotes**.**
Um segundo método é o **setLevel()**, permite setar diretamente um novo valor para o nível de energia. Desta forma pode-se criar extensões de **EnergyModel** que possam também aumentar o nível de energia.
O MobiCS2 apresenta uma implementação deste ponto de flexibilização através da classe **LinearConsumeBattery**, que modela uma bateria (recurso de energia limitado), no qual a energia do nó vai sendo consumida linearmente ao longo do tempo simulado.

3.4.3 Modelo de Conectividade do Canal

Todo canal de comunicação (classe **Channel**) tem que ter associado a ele *um modelo de conectividade* para poder ser usado para a transmissão. No entanto, mas um mesmo modelo de conectividade pode ser compartilhado por vários canal.
O modelo define três características determinantes da conectividade entre dois nós durante uma transmissão. A primeira característica é se a transmissão do nó de origem alcança o nó de destino ou não. A segunda característica define o tempo de propagação da transmissão, e a terceira característica define a qualidade da transmissão, ou seja, se o dado que foi transmitido chega ao nó destino com ou sem qualquer alteração. Com este modelo (representado pela classe abstrata **ChannelConnectivityModel**) é possível controlar vários aspectos relacionados com a transmissão no canal de comunicação.
A classe apresenta apenas o método **transmit(),** que recebe como parâmetros os nós de origem e de destino (representados por seus protocolos da camada física) e o dado sendo transmitido, e implementa o algoritmo para transmissão de dados no canal de comunicação. Para isto, chama métodos abstratos da

própria classe, e que devem ser implementados por subclasses de **`ChannelConnectivityModel`**.
Os três métodos abstratos são **`isReachable`**, **`processData`** e **`getPropagationTime`**, sendo que o primeiro retorna um booleano que informa se a transmissão do nó de origem alcança o nó de destino, o segundo processa o dado sendo entregue ao destino, podendo alterar os seus atributos e o terceiro retorna o tempo de propagação da transmissão em segundos.

O MobiCS2 apresenta duas implementações deste modelo: A classe **`Topology`** representa um modelo simples para definir a topologia da rede, sem levar em consideração características físicas do ambiente simulado, como por exemplo, as posições dos nós da rede e suas velocidades, com o objetivo de oferecer mais controle sobre a simulação.
A outra implementação do modelo de conectividade do canal é a classe **`SimpleRadioPropagation`**, que representa um modelo de propagação de sinal de rádio bem simplificado. Este modelo leva em consideração apenas as posições dos nós de origem e de destino no instante da transmissão, e o raio de alcance do sinal de rádio do nó de origem.

4 Implementação de 802.11 e DSR no MobiCS2

A fim de validar o framework, fizemos uma instanciação do mesmo para o protocolo de roteamento para redes *ad hoc* Dynamic Source Routing (DSR) utilizando o padrão IEEE 802.11 como camada física/enlace.
Antes de descrevermos os detalhes desta instanciação, faremos uma breve revisão do 802.11 e do DSR. Em seguida descreveremos como o MobiCS2 dá suporte à programação de pilhas de protocolos, e como são as classes abstratas relacionadas à camada física. Finalmente, entraremos nos detalhes da instanciação.

4.1 Uma breve revisão do IEEE 802.11

O padrão IEEE 802.11 [Cro97] (também conhecido como **WiFi**) inclui uma família de especificações de camadas física e acesso ao meio para WLANs (Wireless Local Area Network) e define a mesma interface dos demais padrões IEEE 802 entre a camada MAC e as camadas superiores de protocolos.

O mecanismo básico de acesso do IEEE 802.11 em redes ad hoc é o DCF (*Distributed Coordination Function*), que essencialmente é o método de acesso CSMA/CA (*Carrier Sense Multiple Access with Collision Avoidance*).
Para lidar com o problema de terminais escondidos, o padrão define um mecanismo usando dois quadros de controle, RequestToSend (RTS) e ClearToSend (CTS), enviados, pelo emissor e o receptor, respectivamente.
O pacote RTS inclui o tempo previsto para transmissão dos dados, isto é, o tempo necessário para transmitir o quadro de dados integralmente mais o quadro ACK, a ser enviado pelo receptor. Toda a estação que receber o sinal RTS tem que fixar um tempo de backoff, NAV (Net Allocation Vector) de acordo com a duração prevista no RTS.
Se o receptor recebe o RTS, ele responde com um CTS (depois de esperar por um período SIFS). O sinal CTS contém também o tempo previsto para transmissão da mensagem propriamente dita, para que todas as estações que recebam o CTS possam ajustar os seus NAV. Assim, todas as estações dentro do raio de ação do emissor e do receptor saberão quando poderão novamente tentar acessar o meio. Portanto, com este mecanismo reserva-se o meio para um único emissor. Por isto, a utilização de RTS/CTS pode resultar em baixa eficiência de transmissão (grande perda de banda passante) e delay de transmissão não-previsível

4.2 Uma breve revisão do DSR

O *Dynamic Source Routing* (DSR) [JM96,JM99] é um protocolo de roteamento do tipo *source routing* sob demanda. Um nó mantém um cache contendo as rotas (a partir da origem) que necessita. As atualizações no cache são feitas sempre que o nó descobre novas rotas.
As duas principais fases do protocolo são: descobrimento de rotas e manutenção de rotas. *Descobrimento:* Quando o nó origem deseja enviar um pacote para um destino, este procura no seu cache de rotas para determinar se ele já possui uma rota para o destino. Se o nó origem não tem a rota, ou se a validade no cache já expirou, então o nó origem inicia o processo de descobrimento de rotas, fazendo a difusão (broadcasting) de um pacote de requisição de rota que contém o endereço da origem e do destino, e um número de identificação único (que impede difusões repetidas). Cada nó intermediário verifica se conhece uma rota para o destino e se não conhecer acrescenta o seu endereço no registro de rota do pacote e o encaminha para seus vizinhos. Uma resposta da rota é gerada quando o pacote de requisição de rota é recebido pelo destino ou por um

nó intermediário que possua uma rota para o destino. Ao chegar em tal nó, um pacote de requisição de rota já contém no seu registro de rota a seqüência de *hops* que leva da origem até este nó.
Manutenção: Quando um nó encontra um problema de transmissão fatal na sua camada de enlace de dados, este gera um pacote de erro de rota. Quando um nó recebe um pacote de erro de rota, este remove o *hop* em erro do seu cache de rotas, e todas as rotas que contém o *hop* com erro são podadas neste ponto. Pacotes de *Acknowledgment* são usados para verificar o funcionamento correto dos enlaces da rota, em que o nó ouve (em modo promíscuo) se o próximo *hop* do encaminhamento do pacote através da rota foi bem sucedido.

4.3 Protocolos no MobiCS2

Para o MobiCS2 um protocolo é um centro de serviço, com apenas um servidor não-preemptivo e com três filas de entrada, todas com disciplina *First Come First Serve*. Cada protocolo presta serviços para outros protocolos cliente ou fornecedor, numa relação que pode ser representada como uma estrutura em grafo em cada nó da rede.
Ao programar protocolos no MobiCS2 deve-se estender as classes abstratas **`Protocol`**, **`ServiceDataUnit`**, **`ProtocolDataUnit`** e **`Address`**.
As unidades de dados recebidas por protocolos clientes são processadas pelo método abstrato **`whenSDU`**. Para processar as unidades de dados dos protocolos fornecedores precisa-se implementar métodos com a assinatura **`when`**`<DataUnit>`, onde `<DataUnit>` é o nome da classe da unidade de dados que deve ser tratada. A classe **`Protocol`** também oferece um suporte para tratamento de eventos de tempo.

4.4 A Camada Física

A camada física é representada pelas classes abstratas **`Physical`** e **`PhyPDU`**, que são extensões das classes **`Protocol`** e **`ProtocolDataUnit`**, respectivamente. A classe **`Physical`** possui um atributo `id`, que é um identificador único em uma simulação e que pode servir de base para a geração do endereço MAC. A Classe **`Physical`** efetua a conexão com o meio físico (**`Medium`**) e aloca canais de comunicação.
A classe **`PhyPDU`** possui alguns atributos referentes à transmissão no meio físico, tais como `dataRate`, que representa o valor da taxa de transmissão em bits por segundo, e atributos relacionados com o tempo: **`beginTransmission`**, **`beginReception`** e **`endReception`**, onde o primeiro é o tempo simulado em que o **`Physical`** começa a transmissão no meio físico, e o segundo e terceiro são, respectivamente, os tempos de início e fim de recepção da **`PhyPDU`** pela camada física do nó de destino.

4.5 Implementação do IEEE 802.11

A camada física instanciada no framework implementa uma versão bem simplificada da camada física do padrão IEEE 802.11, operando no seu modo DCF (Distributed Coordination Function).

A interface de rede para o 802.11 é representada pela classe **`NetIfIEEE802`** (derivada da classe abstrata **`NetIf`**), que é utilizada pelas classes **`WiFiPhy`** e **`WiFiMAC`**.
A classe **`NetIf`** armazena e disponibiliza uma série de atributos sobre a interface de rede, como por exemplo, o seu endereço físico e IP (classes **`MacAddress`** e **`IpAddress`**, derivadas da classe **`Address`**)
A fim de manter a instanciação do 802.11/DSR simples, decidimos fazer com que a classe **`NetIfIEEE802`** gerasse automaticamente o seu endereço IP, único em toda a rede simulada.

A classe **`Node`** disponibiliza dois métodos para o acesso às interfaces de rede de um nó: **`getNumNetIf()`** retorna o número de interfaces de rede que o nó possui, e o método **`getNetIf()`** retorna o objeto interface de rede (**`NetIf`**) de determinado índice Assim, para descobrir o endereço IP da primeira interface de rede de um nó chamado "`n1`", pode-se chamar **`n1.getNetIf(0).getIpAddress()`**.

Em nossa instanciação foi modelada uma versão simplificada da subcamada MAC do padrão IEEE 802.11 DCF, com ênfase no mecanismo CSMA/CA com RTS/CTS.
As classes que representam a subcamada MAC são: **`WiFiMAC`**, **`MacAddress`**, **`WiFiData`** para quadros de Dados (derivada da classe **`ProtocolDataUnit`**). Para os quadros de Controle de acesso ao meio implementamos as Classes

WiFiRTS, **WiFiCTS** e **WiFiACK**, derivadas da classe **ServiceDataUnit**.

Também foi disponibilizada a classe **WiFiNode**, derivada da classe **Node**, que já instancia automaticamente e interconecta todos os protocolos de baixo nível necessários para a comunicação: **WiFiPhy**, **WiFiMAC**, interface de rede **NetIfIEEE802** e o protocolo **ARP**, necessário para a tradução de endereços IP para MAC. Como o protocolo **WiFiPhy** já se conecta (**attach()**) automaticamente à mídia **Air**, e também já aloca um canal nesta mídia, basta instanciar o **WiFiNode** para ter um nó de rede com acesso ao sistema de comunicação.
Como qualquer outro protocolo, ao instanciar um protocolo de roteamento este já é automaticamente incluído no nó de rede informado em seu construtor. Basta então chamar o método **connectRoutingProtocol()** da classe **WiFiNode** para que o protocolo de roteamento seja conectado ao restante da pilha de protocolos do **WiFiNode**, como é mostrado na Figura 2.

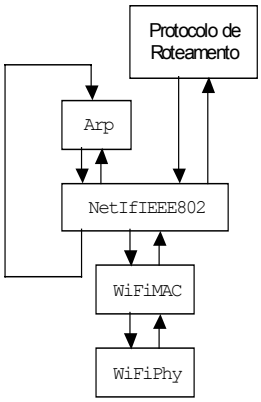

Fig. 2: Interconexão de Protocolos no WiFiNode

4.6 Implementação do DSR

Para implementação do DSR foi criada a classe **Dsr** (derivada da classe **Protocol**) e as seguintes classes que representam os diferentes tipos de pacotes do protocolo, todas implementando a interface **NetIfSDU**, para que a interface de rede possa processá-las.

- **DsrRouteRequest** com os atributos: source, destination, identifier, target e routeRecord
- **DsrRouteReply** com os atributos: source, destination, e routeRecord
- **DsrData** com os atributos source, destination, e routeRecord
- **DsrRouteError** com os atributos source, destination, unreachable e routeRecord

Onde identifier, unreachable e routeRecord, representam, respectivamente, o número de identificação único da requisição de rota, o endereço IP do nó que não foi alcançado, e o registro de rota, que é implementado por uma lista encadeada de endereços IP.

Como o **Dsr** executa em cima do 802.11, pode-se assumir que os enlaces são bidirecionais, e o **Dsr** pode usar a rota reversa da rota contida no pacote **DsrRouteRequest** para encaminhar o pacote **DsrRouteReply**,. Além disto, como o 802.11 implementa entrega com confirmação (**WiFiACK**), o **Dsr** pode utilizar os "*link-layer acknowledgements*" na fase de Manutenção de Rotas, conforme especificado pelo DSR [JM99].

Seguindo um modelo de estruturação modular de protocolos de roteamento (para redes ad hoc) nas componentes genéricas *RouteManager RouteAccess* e *Forwarding* [Mele02], foram criadas as seguintes interfaces e classes específicas para o DSR:

- Interface **DsrRouteManager** define métodos para descobrir, manter e remover rotas;
- Interface **DsrForwarding** define métodos que tratam do encaminhamento de pacotes com rota conhecida;
- Classe **DsrRouteAccess** implementa uma interface para acessar as rotas armazenadas no **DsrRouteData**, sem expor a estrutura do cache.
- Classe **DsrQueue** implementa uma fila para armazenar os pacotes que estão pendentes para encaminhamento, aguardando a resposta (**DsrRouteReply**) de um descobrimento de rota;
- Classe **DsrRouteData** implementa um cache de rotas para armazenamento das rotas Esta classe é instanciada pela classe **DsrRouteAccess;**

O framework disponibiliza ainda a interface **RoutingSDU**, que define a interface da SDU de qualquer protocolo de roteamento, padronizando

assim a interface entre estes protocolos e os protocolos de níveis superiores, como por exemplo, um protocolo de transporte.

5. Resultados de uma simulação

Os principais parâmetros da nossa simulação foram baseados no trabalho de [Bro98], onde foi feita uma comparação de desempenho de quatro protocolos de roteamento para redes móveis ad hoc (DSDV, TORA, DSR e AODV).

O cenário da nossa simulação é uma rede móvel ad hoc constituída de 50 nós móveis, que se movem em uma região retangular plana (1500m x 300m), durante 900 segundos de tempo simulado.

Todos os nós executam o protocolo de roteamento DSR (classe **`Dsr`**) em cima dos protocolos do padrão IEEE 802.11 (classe **`WiFiNode`**).

Na simulação existem 10 nós móveis (nós "0" - "9") que geram tráfego com taxa de bits constante (CBR) para outros 10 nós da rede (nós "10" – "19"), que consomem este tráfego, formando assim 10 pares de nós com comunicação *peer-to-peer*. Os nós que geram tráfego têm uma taxa de envio (**`sendingRate`**) de quatro pacotes por segundo, e enviam pacotes de tamanho de 64 bytes.

O início da geração de tráfego, para cada nó, é determinado por uma variável aleatória, com distribuição uniforme entre os tempos simulados 0 e 180 segundos.

Todos os nós da rede se movimentam segundo o modelo de mobilidade *Random Way-Point* (conforme seção 3.4.1). A velocidade máxima dos nós da rede é de 20 m/s (72 Km/h), o que dá uma velocidade média de 10 m/s. O grau de mobilidade da simulação é caracterizado pelo tempo de pausa. As simulações foram executadas para cinco tempos de pausa diferentes: 0, 120, 300, 600 e 900 segundos. Enquanto um tempo de pausa de 0 segundo corresponde a um movimento contínuo (sem pausa), o de 900 segundos (a duração da simulação) corresponde a uma simulação onde todos os nós estão parados.

Nesta simulação o nível de energia do nó não foi levado em consideração e a energia foi mantida no nível máximo durante toda a simulação

O modelo de conectividade do canal utilizado na simulação foi o **`SimpleRadioPropagation`**, com o alcance do sinal de rádio (atributo **`radioRange`** da classe **`WiFiPhy`**) ajustado para o valor de 250 m para todos os nós, enquanto que a taxa de transmissão do IEEE 802.11 foi configurada para 2Mb/s.

Foram executadas 10 simulações para cada parâmetro de tempo de pausa, totalizando em 50 execuções da simulação.

O protocolo de roteamento DSR foi avaliado quanto à razão de entrega de pacotes, isto é, a razão entre o número total de pacotes originados pelos nós geradores de tráfego, e o número total de pacotes efetivamente recebidos pelos nós consumidores. Esta medição é importante porque descreve a taxa de perda que é vista pelos protocolos de transporte, representando a vazão máxima que a rede pode suportar.

O resultado das simulações pode ser visto no gráfico da Figura 3. Comparando o resultado obtido em nossa simulação com o resultado obtido em [Bro98], vemos que em termos de valores absolutos os resultados apresentam algumas diferenças, mas os gráficos essencialmente apresentam o mesmo comportamento. A diferença entre os valores absolutos é causado pelo grau diferente de detalhe da implementação dos protocolos no MobiCS2 e no ns-2. No MobiCS2 foi implementada uma versão básica do protocolo DSR, sem nenhuma otimização ou extensão (opcionais). É provável que estas otimizações do DSR sejam responsáveis pela ligeira melhora no desempenho apresentado no trabalho do [Bro98].

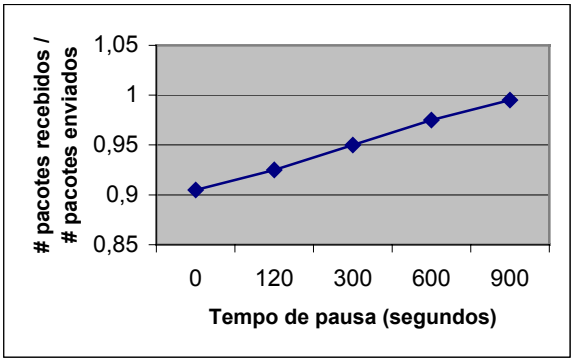

Figura 3 – Razão de entrega de pacotes em função do tempo de pausa

6. Conclusão

Neste artigo descrevemos a estrutura e as principais abstrações do framework MobiCS2, bem como uma instanciação do mesmo para uma versão simplificada do protocolo de roteamento Dynamic Source Routing (DSR) sobre o IEEE 802.11 DCF.

Note-se que neste trabalho não tivemos como objetivo criar uma implementação detalhada (e eficiente) do padrão 802.11 e do protocolo DSR, e que o objetivo também não foi o de comparar os resultados obtidos

na simulação com os de outras simulações mais do DSR. Em vez disto, a instanciação teve como objetivo validar a usabilidade e a flexibilidade do framework, e acreditamos que neste sentido a experiência foi bastante positiva.
No futuro, pretendemos implementar algumas otimizações sugeridas para o DSR e também outros protocolos de roteamento, além de trabalhar na criação de modelos de mobilidade, conectividade e energia mais sofisticados.

Referências

[Bag98] Bagrodia, R. et al Parsec: A Parallel Simulation Environment for Complex Systems, **Computer**, Vol. 31(10), October 1998, pp. 77-85

[Bha98] Bhatti, N.T. et al. Coyote: a system for constructing fine-grain configurable communication services. **ACM Transactions on Computer Systems**, 16(4): 321-366, November 1998.

[Bro98] Broch, J. et al. A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols. **Proc. of the 4th Intern. Conference on Mobile Computing and Networking (MobiCom'98),** ACM, Dallas, TX, October 1998.

[Cow99] Cowie et al.. Towards Realistic Million-Node Internet Simulations. **Proc. of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)**, July 1, 1999.

[Cro97] Crow, B.P. et al. IEEE 802.11 Wireless Local Area Networks. In **IEEE Communications Magazine**, 116-126. September 1997.

[DSY90] Dupuy, A.; Schwartz, J.; Yemini, Y. Bacon, D. NEST: A network simulation and prototyping testbed, **Comm. ACM**, 33(10),: 63-74, October 1990.

[HP91] Hutchinson, N.C.; Peterson, L.L; An Architecture for Implementing Network Protocols, **IEEE Trans. on Software Engineering**, 17(1):64-76, January 1991.

[JM99] JOHNSON, D.B.; MALTZ, D.A. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks, oct 1999. **IETF Draft**, 49 pages. Disponível em http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-03.txt>. Acesso em 1 nov 2001.

[JM96] JOHNSON, D.B.; MALTZ, D.A. Dynamic Source Routing in Ad Hoc Networks. **Mobile Computing**, T. Imielinski and H. Korth, Eds., Kulwer, 1996, pp. 152-81.

[KV98] KO, Y.-B.; VAIDYA, N.H. Location-aided routing (LAR) in mobile ad hoc networks. **ACM MOBICOM**, November 1998.

[Mele02] Mele, A.; Um Framework para Simulação de Redes Móveis Ad hoc, **Dissertação de Mestrado**, PUC-Rio, December 2002.

[Mob02] MobiCS: Simulador para Computação Móvel. Disponível em <http://www.inf.puc-rio.br/~endler/MobiCS/>
Acesso em 26 jun.2003.

[NS2] The VINT Project. **The ns Manual,** 31 jan 2002. Disponível em <http://www.isi.edu/nsnam/ns/ns-documentation.html> . Acesso em 26 jun 2002.

[RE00] Rocha, R.A.; Endler, M. Flexible Simulation of Distributed Protocols for Mobile Computing. **Proc. of the 3rd Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM)**, 123-126, Boston, August 11th, 2000.

[ZBG98] Zeng, X.; Bagrodia, R.; Gerla, M. GloMoSim: a Library for Parallel Simulation of Large-scale Wireless Networks, **Proc. of the 12th Workshop on Parallel and Distributed Simulations -- PADS '98**, May 26-29, 1998