Simulação de Roteamento na Avaliação de Protocolos Multicast e Sistemas Distribuídos de Grupo

Hisham H. Muhammad, Marinho P. Barcellos, Rosana Casais

[1]PIPCA- Programa de Pós-Graduação em Computação Aplicada
Centro de Ciências Exatas e Tecnológicas
Unisinos - Universidade do Vale do Rio dos Sinos
Av. Unisinos, 950 - São Leopoldo, RS - CEP93022-000

{hisham,marinho,rosana}@exatas.unisinos.br

Abstract. *Simmcast, a process-based, discrete-event simulator aims to promote performance evaluation and to allow studying of multicast protocols in different modalities, layers, and levels of abstraction. It allows the specification of multicast protocols and group-based distributed systems using an object-oriented multi-threaded framework. Previously, we developed the basic components of the framework; now, such components are combined so that to allow the modelling of routing in Simmcast. Hence, there is a range of experiments that became possible through the use of arbitrary topologies. The present paper discusses routing simulation models and how these are integrated into the infrastructure of Simmcast. We analyzed models without any routing, with static routing, and with dynamic routing.*

Resumo. *Simmcast, um simulador de eventos discretos baseado em processos, visa auxiliar a avaliação de desempenho e o estudo de protocolos multicast em diferentes modalidades, camadas e níveis de abstração. Ele permite que protocolos multicast e sistemas distribuídos baseados em grupos sejam especificados utilizando um framework multi-threaded orientado a objetos. Anteriormente, foram desenvolvidos os componentes básicos do framework; neste artigo, os componentes são combinados de forma a permitir a modelagem de roteamento no Simmcast. Com isso, abre-se um espectro de diferentes experimentos que tornam-se possíveis através do uso de topologias arbitrárias. Este artigo discute modelos de simulação de roteamento e como estes são integrados à infra-estrutura do Simmcast. Foram analisados modelos sem roteamento, com roteamento estático e com roteamento dinâmico.*

1. Introdução

Simmcast é um projeto de pesquisa cujo objetivo é investigar técnicas de simulação de protocolos multicast e sistemas distribuídos baseados na comunicação em grupo. Como ferramenta, o Simmcast permite o estudo de protocolos e sistemas (através da análise das mensagens trocadas) e a avaliação de desempenho. Diferentemente de simuladores de

rede tradicionais, como o VINT ns-2 ([3, 7])[1], o Simmcast está centrado no processo de prototipação e desenvolvimento de protocolos (tal como [17]). Por essa razão, o Simmcast (ferramenta) é um simulador de eventos discretos baseado em processos, que permite que os protocolos simulados sejam especificados (em Java) utilizando um framework *multi-threaded* orientado a objetos. Blocos básicos de construção[2] estão disponíveis ao usuário e podem ser combinados a fim de construir novas simulações.

O Simmcast lida com multicast e comunicação em grupo em diferentes camadas (rede, transporte, aplicação) e níveis de abstração (com ou sem topologia). Exemplos de categorias que podem ser simuladas são os protocolos para transmissão multicast confiável um-para-vários ([10, 15, 18]), e protocolos de comunicação em grupo vários-para-vários ([6, 12]).

Em trabalhos anteriores ([1, 13]), foram desenvolvidos os componentes básicos do framework, projetados de forma a descrever os elementos fundamentais de uma simulação de um protocolo de comunicação em grupo. Neste trabalho, estes componentes são combinados de forma a permitir a modelagem de roteamento no Simmcast. Com isso, abre-se um espectro de diferentes experimentos que tornam-se possíveis através do uso de topologias arbitrárias compostas de três elementos: *hosts*, *roteadores* e *links*. Além disso, é viabilizado o estudo e avaliação de protocolos de roteamento multicast em redes arbitrárias compostas apenas de roteadores e links.

Com roteamento multicast, pacotes que são enviados por um host são distribuídos através de uma árvore multicast cujos elementos folha são os hosts atuais membros do grupo destinatário, e os nodos internos da árvore representam os roteadores. A composição dessa árvore multicast é determinada de acordo com o protocolo de roteamento em questão. Exemplos são Distance Vector Multicast Routing Protocol (DVMRP), Multicast Open Shortest Path First (MOSPF), Protocol Independent Multicasting (PIM) e Core Based Trees (CBT). O serviço desejado de um protocolo de roteamento multicast é a distribuição eficiente de um pacote aos hosts pertencentes a um grupo, variando a maneira com que os protocolos implementam o serviço. No contexto do ambiente de simulação, varia o grau de detalhe de abstração com que esse serviço pode ser modelado.

Este artigo discute modelos de simulação de roteamento e como estes são integrados à infra-estrutura da ferramenta Simmcast. O artigo está organizado da seguinte maneira: a Seção 2. revisa características do simulador relevantes a esse trabalho. A Seção 3. discute modelos de simulação sem roteamento, enquanto a Seção 4. considera diferentes abordagens e apresenta o modelo de roteamento adicionado ao Simmcast. A Seção 5. tece conclusões e lista trabalhos futuros.

2. Simmcast

Esta seção resume os princípios básicos do Simmcast, concentrando nos aspectos que são necessários ao entendimento do que segue; uma descrição mais completa pode ser encontrada em [13].

[1] O Simmcast difere fundamentalmente em termos de filosofia e projeto do simulador ns; para uma comparação, vide [13].

[2] *building blocks.*

A arquitetura do Simmcast visa adaptar-se a todos os tipos de protocolos multicast: protocolos de roteamento, protocolos de nível de transporte confiáveis e semi-confiáveis, bem como aplicações distribuídas onde "processos de aplicação" comunicam-se sobre múltiplos grupos multicast. As próximas sub-seções fornecem uma visão geral do Simmcast, iniciando-se por sua API (Application Program Interface).

2.1. API - Application Program Interface

Como já foi mencionado, o Simmcast é um framework orientado a objetos de simulação discreta baseado na linguagem de programação Java e na biblioteca JavaSim ([9]). A principal idéia por trás do framework proposto é prover uma API com as operações típicas de comunicação e controle de tempo, bem como uma arquitetura de software concorrente para o projeto e avaliação de protocolos multicast. Para construir uma simulação, o usuário deve adicionar ou estender as classes do framework de acordo com o protocolo específico e configuração avaliada, mas ainda utiliza a maior parte da funcionalidade do framework sem ter que reimplementar esta (por exemplo, envio de mensagens unicast e multicast).

Simplicidade de uso é um dos princípios de projeto do Simmcast. O uso de módulos de processamento (no caso, threads) síncronos é encorajado pelo modelo de simulação baseado em processo. Toda a arquitetura se baseia em operações primitivas disponibilizadas ao usuário, em cinco categorias: transmissão (`send`), recepção bloqueante (`receive`) e não-bloqueante (`tryReceive`), gerenciamento de grupos multicast (`join`, `leave`), controle de eventos assíncronos (`setTimer`, `cancelTimer`, `onTimer`) e controle de processos (`sleep`, `wakeUp`).

2.2. Blocos Básicos de Construção

No Simmcast uma simulação é descrita combinando-se um conjunto de blocos básicos de construção, provendo código adicional quando necessário. Estes são constituídos de dois componentes principais: *processos* e *filas*. Processos são objetos ativos que correspondem às threads de execução. Eles adicionam e removem objetos das filas. Filas são usadas principalmente para modelar pacotes em trânsito. A seguir, os blocos básicos de construção são apresentados; para cada bloco, existe uma classe correspondente.

Nodos (`class Node`) são as entidades fundamentais de interação. Um nodo é identificado por um valor inteiro único. Dependendo do nível de abstração desejado, nodos podem representar um processo de aplicação, um agente de protocolo em um host, um roteador, ou uma das muitas entidades que interagem em um host ou roteador. Os nodos são conectados por *packet paths*, ou **caminhos de pacotes** (`class Path`). As propriedades do caminho são banda, probabilidade de perda de pacotes e atraso de propagação (fixo ou tirado de uma distribuição aleatória). Cada caminho tem associado a si uma fila *path queue* ($pq_{x,y}$) para armazenar pacotes sendo propagados de x para y. Caminhos no Simmcast podem representar um link físico, uma conexão que combine propriedades de rede de diversos links físicos (como em [2]), ou uma fila de mensagens local.

O conceito de **grupo** (`class Group`) é fundamental para protocolos multicast. A participação em um grupo pode ser definida pelo usuário através do arquivo de descrição de simulação que descreve o experimento. Alternativamente, os nodos, através do código de protocolo associado, podem assinar ou deixar grupos dinamicamente, de acordo com a lógica do protocolo.

Uma **rede** (`class Network`) é uma combinação arbitrária de nodos e caminhos. A arquitetura do Simmcast não impõe nenhum esquema de roteamento específico, para que diferentes tipos de esquemas possam ser livremente implementados estendendo a classe `Node`. Esta extensibilidade é explorada na Seção 4., onde são descritos modelos de roteamento para o Simmcast. **Pacotes** (`class Packet`) são a unidade de transmissão de dados entre dois ou mais nodos. `Packet` contém os atributos mínimos necessários a um pacote no Simmcast, e para novos protocolos, herança é usada para definir novos tipos de pacotes.

Cada nodo x mantém uma série de filas:

- *send queue:* uma fila de envio para cada caminho de saída, denotada como $sq_{x,y}$, para envios de x para y;
- *receive queue*: uma única fila de recebimento, denotada como rq_x, onde todos os pacotes que chegam a x são adicionados;
- *timer queue*: uma fila de temporizador, denotada como tq_x, para registrar futuros eventos assíncronos (não representada na Figura 1).

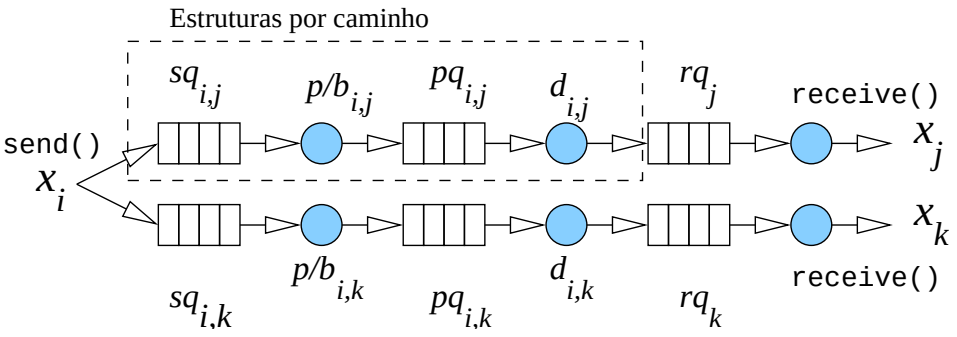


Figura 1: Ilustração do fluxo de pacotes através das filas e seus tempos de serviço, para um exemplo de transmissão multicast de um nodo x_i para os nodos x_j e x_k.

A Figura 1 ilustra as filas e tempos associados que um pacote deve passar enquanto é transmitido entre dois nodos diretamente conectados, x_i e x_j. O nodo x_i solicita o envio de um pacote multicast para um grupo a que pertencem x_j e x_k; uma cópia do pacote é enfileirada em $sq_{i,j}$ e $sq_{i,k}$, existindo espaço em cada uma destas filas. O nodo x_i é bloqueado por um período igual a t_{send} , quando $t_{send} > 0$; o propósito de t_{send} é permitir ao usuário "transparentemente" limitar as taxas de transmissão. Um pacote em $sq_{i,j}$ mais cedo ou mais tarde chega ao início da fila[3], e então espera por um tempo igual a $p/b_{i,j}$, com p igual ao tamanho do pacote e $b_{i,j}$ à largura de banda do caminho de x_i para x_j. Após isso, o pacote deixa $sq_{i,j}$ e se junta à $pq_{i,j}$. O pacote fica na $pq_{i,j}$ por tempo $d_{i,j}$, onde $d_{i,j}$ é a latência de atraso do caminho de x_i para x_j. Mais tarde, o pacote é enfileirado em rq_j, caso exista espaço nesta fila. Após um tempo, o pacote chega ao início da fila rq_j, e a partir de então, no próximo `receive()` ou `tryReceive()` de uma thread em x_j, o pacote parte de rq_j. O tempo de serviço é ditado pela lógica do protocolo, mas não pode ser inferior a t_{recv}.

[3] a descrição dos eventos associados a x_k serão omitidos por uma questão de espaço, considerando sua equivalência com x_j.

Para completar a especificação do nodo, existem dois blocos básicos que usualmente correspondem a recursos do sistema operacional: *threads* e *timers*. **Threads** simplificam um protocolo porque elas permitem que o desenvolvedor modele a arquitetura como um conjunto de entidades síncronas mais simples que interagem entre si; por outro lado, threads podem ser disparadas dinamicamente para gerenciar mensagens de entrada (modelo com uma thread por mensagem). Eventos assíncronos podem ser implementados utilizando **Timers** (temporizadores). Existem muitos casos de eventos assíncronos em software de protocolos, sendo timeouts os exemplos mais comuns. Timeouts são usados, por exemplo, para detectar perda de pacotes. Além disso, timeouts permanentes e consecutivos permitem que um nodo detecte que ocorreu um particionamento na rede. Timers também podem ser usados para implementar comportamento periódico em protocolos, como no TCP.

2.3. Saída (métricas)

O uso de simulação permite que protocolos sejam avaliados de acordo com um conjunto de métricas. A título de ilustração, são considerados abaixo os critérios típicos para avaliação de protocolos multicast confiáveis, throughput efetivo e custo de rede. Throughput efetivo é usualmente definido como a quantidade efetiva de dados transmitidos sobre o tempo necessário para transmiti-los de maneira confiável.

Custo de rede pode ser definido como a quantidade de banda necessária para completar (de forma confiável) a transmissão de todos os dados. Na comunicação unicast, esta é uma definição simples. Entretanto, é ligeiramente mais complexo avaliar o custo de rede quando se utiliza multicast, devido à replicação de pacotes através da árvore de distribuição. Note que um pacote pode ser transmitido, em geral, ou via uma ou mais transmissões unicast, ou via uma única transmissão multicast, dependendo se o pacote é destinado ao grupo inteiro ou a um subconjunto. Para determinar o custo de transmissão, soma-se todos os pacotes, bits ou bytes transportados através de todos os caminhos (dados e controle) que constituem a topologia.

Outras métricas podem ser de interesse, mas dependerá do protocolo ou aplicação considerada. Alguns exemplos são o número de perdas por congestionamento obtidas em um roteador quando um mecanismo de controle de congestionamento esteja ativo, ou o tempo médio até que um receptor possa se recuperar de uma perda de pacote durante a recepção de tráfego em tempo real. O Simmcast permite que métricas como estas sejam inferidas a partir de uma API que fornece uma visão somente-leitura da rede simulada em paralelo à execução do experimento. Isto possibilita que os valores sejam gerados durante a simulação, sem onerar o processamento com a geração de traços.

Alternativamente, é possível através desta mesma API gerar arquivos de traço que permitem que o comportamento de um protocolo seja investigado de forma similar ao que ocorre com um "sniffer" em uma rede real. Como o Simmcast é baseado em simulação discreta, ele permite que estes eventos sejam registrados em arquivos de traço. Simmcast considera a existência de apenas dois tipos gerais de eventos: inclusão e remoção de filas (sq, pq, rq e tq).

2.4. Executando uma Simulação

Para executar uma simulação, é necessário especificar três itens: (1) o protocolo simulado, combinando blocos básicos existentes e especializando as classes necessárias; (2)

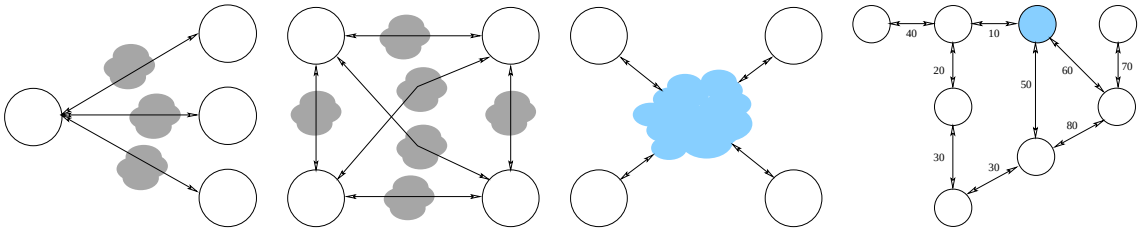
a topologia de rede, criando a sua conectividade; e (3) a topologia do protocolo: como especificar a alocação de agentes na rede previamente definida. Note que os dois últimos passos são feitos preparando um arquivo de descrição de simulação, sem a necessidade de recompilação.

Diferentes níveis de representação da rede são possíveis. No nível mais abstrato, a rede é um conjunto de conexões diretas (caminhos) ponto-a-ponto entre os nodos. Vários destes modelos de topologia são considerados na seção seguinte. Um maior nível de detalhe pode ser obtido fazendo parte dos nodos Simmcast agirem como roteadores, conforme discutido na Seção 4..

3. Modelos sem Roteamento

A arquitetura acima exposta permite o uso de topologias arbitrárias em simulações, mas não impõe qualquer lógica de roteamento. De fato, variando o significado atribuído a um nodo e suas filas, simulações no Simmcast podem abranger um largo espectro de abstração. Em um extremo, uma simulação pode contar com um conjunto de processos de aplicação totalmente conectados através dos caminhos lógicos (vide Seção 2.). No outro extremo, uma simulação pode ser detalhada a ponto de representar interações entre camadas de protocolos em múltiplos elementos de rede (como hosts, roteadores, comutadores).

Existe uma gama de experimentos de simulação que podem ser realizados utilizando modelos sem roteamento. Alguns exemplos de topologias são ilustrados na Figura 2 e comentados a seguir.



(a) 1-N (b) N-N (c) nuvem (d) grafo

Figure 2: Exemplos de modelos sem roteamento

O caso **(a)** na Figura 2 representa uma comunicação multicast um-para-vários onde existe um caminho lógico[4] bi-direcional um-para-um entre o transmissor e cada um dos receptores. Uma variação deste modelo é apresentada no caso **(b)**, que ilustra uma comunicação multicast vários-para-vários. Em ambos os casos, os caminhos lógicos são responsáveis por modelar probabilisticamente atrasos e perdas de pacotes. A completa abstração do nível de rede provida por estes modelos se adequa bem ao estudo de protocolos de nível mais alto, como por exemplo, controle de composição de grupo ([6, 12]).

Os caminhos lógicos acima são independentes entre si, e portanto não modelam compartilhamento do meio. Um modelo de topologia que mantém a abstração de caminhos lógicos e é capaz de representar este compartilhamento é ilustrado no caso **(c)**. Tal

[4]note que existe uma "nuvem" de rede em cada caminho.

é obtido através de um nodo central que age como nuvem. Nele, uma matriz descreve as propriedades de comunicação entre cada um dos pares de nodos e é responsável pela implementação de atrasos aleatórios na distribuição de pacotes. Para cada nodo, existe na nuvem uma fila de saída associada, por onde passam todos os pacotes destinados ao nodo em questão. Esta fila tem o propósito de adicionar atrasos de enfileiramento e perdas devido a estouro de *buffer*. Com este modelo, torna-se possível o mapeamento de uma rede apenas em função de atrasos e perdas fim-a-fim.

Os casos (a), (b) e (c) utilizam modelos abstratos de topologia. Em determinadas situações, será necessário precisá-la, seja em função de regras lógicas ditadas pela aplicação, ou em função da interconexão física entre os nodos. O caso **(d)** ilustra estas situações, onde todos os nodos são processos de aplicação ou agentes de transporte, mas há restrições na interação entre os mesmos (especificadas na figura através de um grafo). Um exemplo é a simulação de sistemas cliente-servidor tolerantes a falhas baseados em grupos de servidores com replicação ativa; outro é a implementação de algoritmos distribuídos, tais como os algoritmos para eleição ou formação de árvore descritos em [11].

4. Roteamento

Em [1, 13], descrevemos casos baseados nos modelos apresentados na seção anterior. Neste trabalho, consideramos a inclusão de roteamento em simulações com rede arbitrária. Ao considerar simulações em tais topologias, um simulador deve apresentar as seguintes características:

- independência de roteamento, ou seja, a lógica dos protocolos ou sistemas que são objeto de estudo não são alterados em função do roteamento, permitindo usar abordagem incremental;
- separação clara entre a camada (protocolo ou sistema) sendo simulada e a abstração de rede subjacente;
- utilização de topologias arbitrárias, incluindo anel uni/bi-direcional, estrela, árvores, barramento, ou a combinação destas em uma outra topologia;
- geração de traços em diferentes níveis de abstração, permitindo visualizar interações entre elementos do protocolo ou sistema (por exemplo, visualizar *com ou sem* a rede, com ou sem pacotes de roteamento, etc.).

O suporte a roteamento expande as capacidades do ambiente de simulação, permitindo:

- importação de topologias descritas em arquivos e criadas por geradores automáticos de topologias, tal como o gt-itm ([8]);
- dispor de diferentes lógicas de roteamento multicast intra- e inter-domínio, como árvore baseada em fonte ou compartilhada, e protocolos correspondentes;
- utilização de topologias dinâmicas: modelagem de falhas (temporárias ou permanentes) de hosts, roteadores e links, bem como simulação de redes ad hoc ou computação móvel ([17]), com inclusão, remoção e relocação dinâmicas de hosts/links.

Em função desses requisitos, explorou-se a extensibilidade da arquitetura do Simmcast, criando-se blocos derivados a partir dos blocos básicos de construção providos pelo framework. Topologias são compostas de hosts, roteadores e links. Refletindo

isto, blocos de construção do tipo *path* assumem o papel de links, surgindo dois novos blocos de construção, derivados de *node:* ***hosts*** e ***routers***.

Como esperado, instâncias do bloco *host* detêm a lógica do protocolo ou aplicação sendo estudada, enquanto instâncias do bloco de construção *router* são responsáveis pelo encaminhamento e distribuição de pacotes. Um bloco de construção *host* é um nodo que possuirá apenas uma conexão bi-direcional (isto é, dois *paths*, um de entrada e um de saída, conectados com um mesmo *router*). Um nodo *router* poderá possuir um número arbitrário de conexões de entrada e de saída.

É possível mapear todos os tipos básicos de topologia (incluindo anel, estrela, árvore e barramento) e a combinação destes utilizando *hosts*, *routers* e *paths*. Por exemplo, em uma topologia em árvore, a raiz e as folhas (fonte e receptores) são representadas por *hosts*, enquanto os nodos internos correspondem a *routers*. Um exemplo dessa topologia é ilustrado na Figura 3. A árvore indicada na figura é utilizada na distribuição de pacotes quando "F" transmite um pacote a um grupo contendo os receptores 1 a 7. Os roteadores 12 e 13 não estão marcados porque não tomam parte dessa árvore de caminho mais curto enraizada em "F". Similarmente, estão marcados apenas os caminhos que fazem parte da árvore.

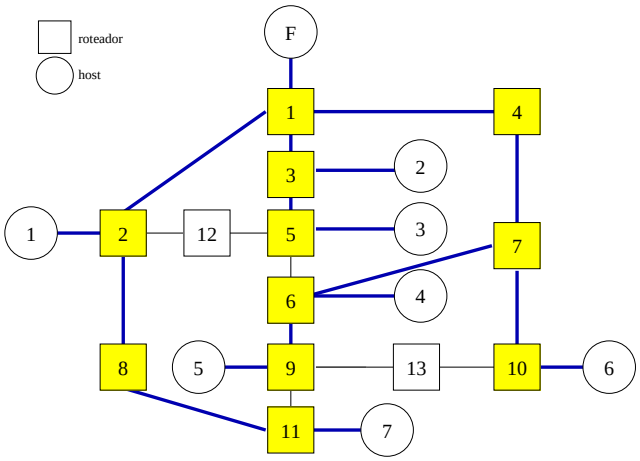


Figura 3: exemplo de simulação em topologia arbitrária, com árvore multicast baseada na fonte.

No framework, os blocos de construção *host* e *router* correspondem às classes `HostNode` e `RouterNode`, subclasses de `Node`. Refletindo as características apontadas anteriormente, no `HostNode` o método `send` passa a encaminhar toda mensagem enviada para o *router* associado, independemente do destinatário informado. O `RouterNode` estende o `Node` adicionando múltiplas filas de recebimento (rq), uma por link de entrada. Objetos `NodeThread` são usados para representar no `HostNode` a lógica de aplicação e no `RouterNode` o protocolo de roteamento.

Pacotes são encaminhados por *routers* de acordo com uma tabela de roteamento. Esta tabela pode ser preenchida de forma estática (no início da simulação) ou dinâmica (através de um protocolo de roteamento). Estas duas abordagens serão consideradas a seguir.

4.1. Roteamento Estático

Roteamento estático pode ser utilizado em simulações onde as interações entre os roteadores para execução dos algoritmos de roteamento podem ser abstraídas. Neste modelo, uma tabela de roteamento acessível por todos os nodos é estabelecida através de um algoritmo de composição de árvore, que pode ser definido pelo usuário. O Simmcast implementa como padrão o algoritmo baseado em SPT-*Shortest Path Tree* (vide [14]), mas permite que outros esquemas também sejam utilizados.

O uso de uma tabela global que congrega o conjunto necessário de árvores traz uma série de implicações. Em primeiro lugar, o tamanho da tabela pode ser um fator limitante, pois depedende do número de nodos, do número de grupos existentes e do número de fontes (em caso de árvore baseada na fonte; tal problema já foi identificado em [16]). O tempo de convergência (isto é, o tempo gasto até que todos os nodos possuam suas tabelas de roteamento acertadas) é zero, uma vez que a tabela é montada antes do início do experimento. A ocorrência de falhas de links e nodos na simulação é refletida imediatamente na tabela, e, desta forma, percebida por todos os nodos ao mesmo tempo.

4.2. Roteamento Dinâmico

O roteamento estático descrito na seção anterior é suficiente para grande parcela das simulações, onde os detalhes do funcionamento dos protocolos de roteamento são ignorados. No entanto, em certos casos, particularmente na investigação de protocolos e algoritmos de roteamento multicast, o objeto de estudo é a camada de rede, e portanto passa a ser necessário executar sobre a rede simulada, além do protocolo de aplicação, o algoritmo distribuído de roteamento.

O roteamento dinâmico permite uma visão mais real e detalhada da rede. Ele pode ser utilizado tanto para execução de simulações com menor nível de abstração, como para estudo e análise de protocolos de roteamento multicast. Nestas situações, as métricas tipicamente utilizadas são tempo e custo até a convergência, atraso médio entre fonte e receptores e custo global em pacotes para uma transmissão multicast ([4, 5]).

O suporte do Simmcast a estas métricas se dá no arquivo de traço através da identificação de classes de pacotes e eventos relevantes, como por exemplo a alteração de propriedades no link. A informação gerada no traço é extensível pelo usuário; pode-se adicionar ao traço dados específicos do protocolo desenvolvido.

No modelo de roteamento dinâmico, uma subclasse de `RouterNode`, `DynamicRouterNode`, contém uma ou mais threads (objetos `NodeThread`) responsáveis por executar o algoritmo de roteamento. Cada `DynamicRouterNode` contém sua própria tabela de roteamento, e é responsável por mantê-la atualizada. Isto é feito através de troca de mensagens de controle entre os nodos roteadores.

O uso deste modelo, mais realístico que o roteamento estático, impacta na aplicação ou protocolo de transporte através do acréscimo de tráfego na rede, causado tanto pelas mensagens de controle como pelo encaminhamento indevido de mensagens. Este último ocorre devido ao tempo necessário até a convergência da rede. Por exemplo, quando um *host* deixa um grupo, pacotes continuam sendo encaminhados a ele até que os devidos roteadores sejam notificados. Isto dependerá da topologia da árvore e da composição do grupo.

4.3. Inclusão Incremental de Roteamento

Os modelos de roteamento desenvolvidos para o Simmcast e apresentados nas seções anteriores foram concebidos de acordo com a filosofia incremental (evolução do abstrato para o detalhado) que rege o simulador. Por exemplo, em simulações de sistemas distribuídos baseados em comunicação em grupo e protocolos de multicast confiável em nível de transporte, pode-se tirar proveito da compatibilidade entre as diferentes abordagens providas pelo modelo de blocos *host* e *router*: sem roteamento, com roteamento estático e com roteamento dinâmico. Isto permite uma evolução gradual do experimento, exceto quando o mesmo envolve a investigação de protocolos de roteamento: neste caso, a lógica do objeto de estudo da simulação reside no próprio `RouterNode`, ao invés de em `HostNode`.

A abordagem incremental é possível uma vez que as mesmas threads usadas pelos objetos `Node` podem ser embutidas em um objeto `HostNode`, de modo a iniciar a modelagem do protocolo utilizando um maior grau de abstração. Posteriormente, este mesmo protocolo poderá então ser submetido a uma rede com roteamento estático ou dinâmico, utilizando as mesmas threads em objetos `HostNode` que poderão interagir com objetos `StaticRouterNode` ou `DynamicRouterNode`. Desta forma, o experimento pode utilizar tanto um dos modelos abstratos descritos na Seção 3. como uma topologia arbitrária.

5. Conclusão

Em [1, 13] foram descritos aspectos fundamentais do Simmcast como seu framework e blocos básicos de construção. O presente artigo discute o projeto de facilidades de roteamento multicast (necessárias em simulações mais detalhadas) baseado na combinação dos blocos. A extensibilidade do framework nos permitiu manter a arquitetura inalterada ao agregar roteamento ao Simmcast via nodos *router*. Os benefícios em termos de ferramenta são dois: (a) permitir trabalhar com protocolos multicast e sistemas distribuídos baseados em comunicação em grupo usando topologias arbitrárias, com distribuição de pacotes via árvore multicast; e (b) auxiliar o desenvolvimento (através do estudo, análise e comparação) de protocolos de roteamento multicast.

A ferramenta Simmcast é ao mesmo tempo o resultado prático e a *prova-de-conceito* de um projeto de pesquisa cujo objetivo é investigar a simulação de protocolos multicast e sistemas distribuídos baseados em grupo. Acreditamos que os princípios utilizados na elaboração do Simmcast, através de seu framework de simulação e blocos de construção, podem ser aplicados em outros processos de avaliação de desempenho.

Entretanto, o Simmcast é um projeto em andamento, ainda possuindo aspectos de *design* e implementação em aberto. Além disso, a ferramenta pode ser aumentada com uma série de facilidades, incluindo interface gráfica e novos protocolos de roteamento multicast. Trabalhos futuros estão relacionados às possibilidades apontadas na Seção 4., como implementação de diferentes lógicas de roteamento multicast e importação de topologias.

Agradecimentos

Este trabalho foi realizado com auxílio financeiro do CNPq e da FAPERGS. Gostaríamos de agradecer também aos avaliadores pelos comentários e sugestões feitas no processo de revisão do artigo.

Referências

[1] Barcellos, M., Muhammad, H., Detsch, A. (2001) "Simmcast: a Simulation Tool for Multicast Protocol Evaluation", In: XIX Simpósio Brasileiro de Redes de Computadores (SBRC 2001), SBC (Porto Alegre), Florianópolis, 21-25 maio 2001.

[2] Barcellos, M. and Ezhilchelvan, P. (1998) "A Reliable Multicast Protocol using Polling for Scaleability", In: IEEE INFOCOM'98- The Conference on Computer Communications, Proceedings, IEEE (New York), San Francisco, 29 March-2nd. April 1998.

[3] Breslau, L. et alli, (2000) "Advances in Network Simulation", In: IEEE Computer, v.33, n.5, pp. 59-67, May 2000.

[4] Câmara, D. and Loureiro, A. (2000) "Um Novo Protocolo de Roteamento para Redes Móveis Ad hoc", In: XVIII Simpósio Brasileiro de Redes de Computadores (SBRC 2000), SBC (Porto Alegre), Florianópolis, 23-26 maio 2000.

[5] Luis H. Costa, Serge Fdida, Otto Duarte, (2001) "Hop by Hop Multicast Routing Protocol". In: ACM SIGCOMM 2001, San Diego, California, USA, Aug. 2001.

[6] Ezhilchelvan, P., Macedo, R. and Shrivastava, S. (1995) "Newtop: A Fault-Tolerant Group Communication Protocol". In: IEEE 15th Intl. Conf. Distributed Computing Systems, pp.296-306, May 1995.

[7] Fall K. and Varadhan, K. (2001) "The ns Manual", The VINT Project, UC Berkeley, LBL, USC/ISI, Xerox PARC, `http://www.isi.edu/nsnam/ns/ns-documentation.html`, November 2001.

[8] Georgia Tech Internetwork Topology Models – gt-itm (2001), http://www.cc.gatech.edu/projects/gtitm/, November 2001.

[9] Little, M. C. (1999) "JavaSim Users Guide", Public Release 0.3, Version 1.0, `http://javasim.ncl.ac.uk`

[10] Liu, C., Ezhilchelvan, P. and Barcellos, M. (1999) "A Multicast Transport Protocol for Reliable Group Applications", In: First International Workshop on Networked Group Communication - NGC'99. Springer-Verlag, Pisa, 17-20 November, 1999, pp.170-187.

[11] Lynch, N. "Distributed Algorithms", Morgan Kaufmann, San Francisco, 872p., 1996.

[12] Mishra, S., Fei, L., Lin, X., Xing, G. (2001) "On Group Communication Support for CORBA", In: IEEE Transactions on Parallel and Distributed Systems, v.12, n.2, Feb. 2001.

[13] Muhammad, H., and Barcellos, M. (2002) "Simulating Group Communication Protocols Through an Object-Oriented Framework", In: 35th Annual Simulation Symposium (SS2002), IEEE (New York), San Diego, 14-18 April 2002.

[14] Paul, S. “Multicasting on the Internet and Its Applications”, Kluwer Academic Publishers, 421p., 1998.

[15] Radoslavov, P., Papadopoulos, C., Govindan, R., Estrin, D. (2001) “A Comparison of Application-level and Router-assisted Hierarchical Schemes for Reliable Multicast”, INFOCOM 2001, Alaska, 26-29 April 2001. IEEE (New York), April 2001.

[16] Riley, G., Ammar, M., Fujimoto, R. (2000) “Stateless Routing in Network Simulations”, In 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecom munication Systems (Mascots 2000), August 2000, San Francisco, CA.

[17] Rocha, R., Endler, M. (2001) “MobiCS: An environment for prototyping and simulating distributed protocols for mobile networks”, In: 3rd. IEEE International Conference in Mobile and Wireless Communication Networks (MWCN´2001), Recife, Brazil, 44-51, August 2001.

[18] Xiao, Z., Birman, K. (2001) “A Randomized Error Recovery Algorithm for Reliable Multicast”, INFOCOM 2001, Alaska, 26-29 April 2001. IEEE (New York), April 2001.