Simmcast: uma ferramenta de simulação para avaliação de protocolos multicast

HISHAM HASHEM MUHAMMAD
MARINHO PILLA BARCELLOS

UNISINOS - Universidade do Vale do Rio dos Sinos
PIPCA - Programa Interdisciplinar de Pós-Graduação em Computação Aplicada
C6 - Centro de Ciências Exatas e Tecnológicas
Av. Unisinos, 950 - São Leopoldo, RS - CEP 93022-000
{hisham,marinho}@exatas.unisinos.br

Resumo

Este artigo descreve o Simmcast, um *framework de simulação* de eventos discretos baseado em processos que auxilia no projeto e avaliação de protocolos multicast. O Simmcast, baseado em Java, permite que os protocolos simulados sejam especificados utilizando um framework *multi-thread* orientado a objetos; blocos básicos são combinados a fim de construir novas simulações. Um protocolo multicast simples é demonstrado como um exemplo. Para realizar a sua validação, dois modelos de protocolos de multicast confiável usados em um conhecido estudo analítico foram simulados, e os resultados foram comparados com os valores originais.

No Simmcast, apresenta-se aos usuários um modelo de simulação de protocolos abstrato e de fácil entendimento, e então, sobre este, pode-se gradualmente ampliar o nível de detalhe da simulação. Ao contrário de outras ferramentas existentes, o Simmcast provê um modelo flexível que é dedicado ao projeto e avaliação de *protocolos multicast.*

Palavras-chave: protocolos multicast, avaliação de desempenho, simulação.

1 Introdução

O uso de simulação tem sido uma ferramenta poderosa no processo de projetar e avaliar protocolos de comunicação. Outros meios de obter conhecimento sobre um protocolo são a avaliação analítica e experimentos. Estas três formas são complementares, e podem representar diferentes partes do mesmo processo de desenvolvimento. A avaliação analítica é muito útil para demonstrar o impacto geral dos argumentos de entrada de um protocolo (como o tamanho da janela), do número de participantes, ou de mudanças em condições específicas da rede (como a taxa de perda). Entretanto, ela requer modelos simplificados da realidade, limitando o número de variáveis consideradas e as fontes de não-determinismo, já que os resultados são obtidos alterando argumentos em fórmulas. Avaliação analítica é usada principalmente para determinar tendências gerais no comportamento de modelos abstratos de protocolos, ou para comprovar propriedades de um protocolo.

Em contraste, experimentos práticos são baseados em um conjunto de execuções de um protocolo sobre uma rede real, e portanto produzem os resultados mais precisos. Apesar disso, devido à grande influência da topologia e dos sistemas e configurações envolvidos, os resultados coletados são dependentes de um cenário específico, e são difíceis de serem reproduzidos em outro. Isto se aplica particularmente a protocolos de multicast escaláveis, já que é impraticável realizar um experimento de larga escala em uma rede de longo alcance. Outra limitação considerável é que esta abordagem requer que o protocolo já esteja implementado antes que seu funcionamento geral possa ser testado.

A simulação se encontra no nível intermediário, entre a avaliação analítica e os experimentos práticos. Ela permite que o projetista do protocolo ajuste o nível de detalhe, atendo-se apenas aos recursos desejados ([Jain, 1991]). Além

disso, simulação possui o potencial de permitir o aumento gradual de detalhamento, para que os processo resulte em um protocolo rodando sobre uma rede emulada, pronto para ser movido para uma rede real.

Este artigo descreve o Simmcast, ou Simulation of Multicast. Simmcast é um framework de simulação que permite que protocolos sejam facilmente definidos por uma combinação de blocos básicos. Ele é orientado a objetos e suporta a especificação de protocolos multicast multi-thread. A principal diferença entre protocolos unicast e multicast é o conceito de grupo. Agentes devem assinar ou deixar grupos, e mandar pacotes para um ou mais endereços unicast e multicast. Este nível de suporte está embutido no simulador, afetando seus argumento de entradas, modelo de filas, métricas de saída e arquivos de traço.

O artigo inicia com uma visão geral do Simmcast na Seção 2. A Seção 3 ilustra o seu uso e valida seus resultados. A Seção 4 discute trabalhos similares, enquanto a Seção 5 conclui o artigo.

2 Visão Geral

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 agentes comunicam-se sobre múltiplos grupos multicast. As próximas sub-seções darão 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 ([Little, 1999]). 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 re-implementar esta (por exemplo, envio de mensagens unicast e multicast).

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

2.2 Blocos básicos

No Simmcast uma simulação é descrita combinando-se um conjunto de blocos básicos, 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 são apresentados; para cada bloco básico, existe uma classe correspondente.

Nós (`Class Node`) são as entidades fundamentais de interação, e podem ser identificados por um valor inteiro único. Dependendo do nível de abstração desejado, nós podem representar um agente de protocolo em um host, um roteador, ou uma das muitas entidades que interagem em um host ou roteador. Os nós são conectados por **caminhos virtuais de pacotes** (`Class Path`). As propriedades do caminho são banda, probabilidade de perda de pacotes e delay de propagação (fixo ou tirado de uma distribuição aleatória). Cada caminho tem associado a si uma fila ($LQ_{x,y}$) 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 [Barcellos, 1998]), 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

nós, 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 nós e caminhos. A conectividade de rede é descrita através de uma matriz quadrada de tamanho $N \times N$, onde cada elemento corresponde a um caminho unidirecional do nodo x ao y. A arquitetura em camadas do Simmcast não impõe nenhum esquema de roteamento específico, para que diferentes tipos de esquemas possam ser livremente implementados estendendo a classe `Node`. **Pacotes** (`Class Packet`) são a unidade de transmissão de dados entre dois ou mais nós. `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 nó x possui uma fila de envio por caminho de saída (denotada como $SQ_{x,y}$, para uma fila de envio de x a um dado nó y), e uma única fila de recebimento, denotada como RQ_x, à qual todos os pacotes enviados a x são adicionados. Cada nó x possui também uma fila de tempo, TQ_x, para armazenar eventos assíncronos.

A Figura 1 descreve a estrutura interna do nó e o fluxo de pacotes por suas filas com uma configuração exemplo constituída de três nós: x, y e z. Quando um dado nó y envia um pacote ao nó z ($y \neq z$), o pacote é enfileirado em $SQ_{y,z}$, caso o espaço permita (caso contrário, o pacote é descartado). O nó y permanece bloqueado por T_{send} unidades de tempo, antes que o controle retorne a ele. Pacotes deslocam-se de $SQ_{y,z}$ para $LQ_{y,z}$ de acordo com a ocupação de $SQ_{y,z}$, o tamanho dos pacotes e a banda associada ao caminho. Note que $LQ_{y,z}$ tem tamanho infinito. Pacotes são então mantidos em $LQ_{y,z}$ por T_{prop} unidades de tempo, e estão sujeitos à perda de acordo com uma probabilidade de perda associada ao caminho. Quando um pacote chega a z, ele é movido da $LQ_{y,z}$ à RQ_z, novamente, caso o espaço permita. Os pacotes permanecem na RQ_z por um tempo arbitrário, e são removidos desta de acordo com as operações `receive` e `tryReceive` geradas pela lógica do protocolo no nó z.

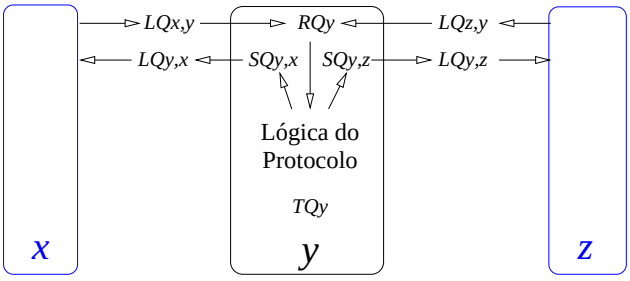

Figura 1: Ilustração da estrutura do nó: x, y, e z são nós, mas apenas y está representado por completo.

Para completar a especificação do nó, 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). No Simmcast cada processo mapeia-se a uma thread. Eventos assíncronos podem ser implementados utilizando **Timers**. 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 nó detecte que ocorreu um rompimento na rede. Timers também podem ser usados para implementar comportamento periódico em protocolos, como no TCP[1].

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 e custo de rede. Throughput é usualmente definido como a quantidade efetiva de dados transmitidos sobre o tempo necessário para transmiti-los (potencialmente de maneira confiável[2]).

[1] Implementações do TCP lidam com eventos periodicamente a cada intervalo de 200 e 500ms.

[2] nest caso, também chamado de *goodput.*

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. A primeira abordagem envia o mesmo pacote várias vezes, mas somente àqueles interessados em receber uma cópia desse pacote, enquanto a seguinte envia uma cópia única a todos os receptores, independentemente do seu interesse no pacote. Indiferentemente à unidade escolhida (pacotes, bytes ou bits), existem pelo menos três maneiras diferentes de avaliar o custo de rede.

A primeira forma é contar a quantidade de pacotes *transmitidos* pelos transmissores (dados e controle) e receptores (controle). Neste modelo, com N receptores, uma transmissão unicast custa o mesmo que uma multicast; isto é claramente irreal para qualquer $N > 1$, considerando a banda extra (através de $N-1$ caminhos) e os tempos de processamento demandados (em $N-1$ receptores). A segunda maneira distingue o custo entre unicast e multicast contando o número de pacotes recebidos. Neste modelo, um pacote enviado via multicast irá custar N (recepções), enquanto cada pacote unicast custará apenas 1 (recepção). Logo, o uso desnecessário de multicast será penalizado, como desejado. Todavia, este modelo não considera o custo de processamento necessário para enviar as múltiplas cópias do mesmo pacote: enviar N cópias de um pacote representa N recepções, da mesma forma que mandar uma única cópia via multicast. Isto, claramente, não é desejado.

As abordagens acima são limitadas porque elas desconsideram a topologia da rede. A terceira forma de determinar o custo é somar todos os pacotes, bits ou bytes transportados através de todos os caminhos (dados e controle) que constituem a topologia. Um protocolo que envia todos os pacotes via multicast será contabilizado por isto, assim como um que somente envie via múltiplos unicasts. O Simmcast provê formas para o usuário de coletar estes dados e consolidá-los. Finalmente, outras métricas podem ser de interesse, mas isto depende 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.

Além de métricas de saída, arquivos de traço são um recurso essencial de simuladores de protocolo. Eles 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, LQ, RQ e TQ).

2.4 Executando uma simulação

Para executar uma simulação, é necessário especificar três ítens: (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. O padrão para o Simmcast é o nível mais abstrato: a rede é um conjunto de conexões diretas (caminhos) ponto-a-ponto entre os nós. A topologia é uma árvore multicast de um nível. Um maior nível de detalhe pode ser obtido fazendo parte dos nós agirem como roteadores.

3 Exemplos de Modelos de Protocolos

Esta seção ilustra o uso do Simmcast descrevendo dois modelos de protocolos de multicast confiável bastante abstratos usados em um estudo analítico de escalabilidade ([Towsley, 1997]). Os resultados deste estudo são então comparados aos valores obtidos pelo simulador. Para os experimentos de simulação, as premissas correspondem àquelas listadas em [Towsley, 1997]: (a) ACKs e NACKs nunca são perdidos; (b) ambos transmissor e receptor possuem buffer infinitos; (c) não há controle de congestionamento; (d) não há controle de fluxo; (e) não há controle de sessão.

As premissas acima são válidas a medida que [Towsley, 1997] busca determinar os limites máximos de throughput

para protocolos *iniciados por transmissor* e *iniciados por receptor*, encontrando a teórica taxa *máxima* de processamento de pacotes possível no transmissor e nos receptores. Logo, os protocolos usados eram apenas modelos, ao invés de protocolos completos. Por exemplo, todos os pacotes são sempre re-enviados por multicast para o grupo inteiro após a detecção de uma perda. Além disso, diversos importantes pressupostos foram assumidos: não há tempo de propagação (o que implica em banda infinita); mais além, assume-se que pacotes de reconhecimento nunca são perdidos, e que todos os buffers das partes envolvidas são infinitos. Esta seção mostra modelos equivalentes aos protocolos A e N1, construídos usando o Simmcast. Os símbolos foram obtidos de [Towsley, 1997]: $E[X_p]$, $E[X_a]$, $E[X_n]$, $E[X_t]$ significando, respectivamente, os tempos médios para transmissão de pacotes, gerenciamento de ACKs, gerenciamento de NACKs e timeout de retransmissão.

O protocolo A, cujo código é mostrado na Figura 2, utiliza retransmissão seletiva. Um pacote é transmitido por vez, e retransmitido até que pelo menos um ACK tenha sido obtido de cada receptor. Pacotes de dados são identificados unicamente por um número de sequência. Já que não há nem tempo de propagação nem atraso nos receptores, o *round-trip-time* sempre iguala-se ao tempo para transmitir um pacote. Portanto, o timeout de retransmissão é sempre o melhor possível. Em outras palavras, o transmissor pode verificar se todos os ACKs chegaram logo após completada a transmissão.

```
TRANSMISSOR
enquanto ha pacotes para serem transmitidos
   envia pacote de sequencia seq
   espera tempo E[Xp], tempo de envio de pacote
   enquanto existem acks prontos para serem recebidos
      recebe ack
      espera tempo E[Xa], tempo de recebimento e tratamento de Ack
      adiciona ack a lista de acks se ja nao presente
      se lista de acks esta completa (todos receptores)
         incrementa seq
         limpa lista de acks
      caso contrario
         espera E[Xt], o tempo de tratamento de timeout

RECEPTOR
faça sempre
      recebe pacote de dados de sequencia seq
      envia reconhecimento positivo para sequencia seq
```

Figura 2: A - Protocolo baseado em ACK

Para o protocolo N1 (e muitos outros protocolos baseados em receptor), a detecção de perda é feita nos receptores, através de uma lacuna na seqüência de numeração dos pacotes. Já que não é possível determinar quantos pacotes seguidos serão perdidos por qualquer receptor, o transmissor deve mandar todos os seus pacotes, ao invés de fazê-lo um a um como ocorre com o protocolo A. Ter múltiplos pacotes e recuperação pendentes ocorrendo tornaria o modelo desnecessariamente complexo; ao invés disso, perdas são simuladas *no lado do receptor*. No caso de uma perda (determinada probabilisticamente), um pacote NACK poderá precisar ser transmitido (não o será se outra cópia do pacote já tiver sido recebida com sucesso). O transmissor envia um pacote, e, como no protocolo A, pode imediatamente após checar se existe algum NACK disponível. A seguir, qualquer NACK existente é recebido; um NACK é suficiente para fazer com que o transmissor re-envie o pacote. O pseudo-código para o protocolo N1 é exibido na Figura 3.

Os resultados para os modelos simulados com o Simmcast são mostrados na Figura 4, junto com aqueles obtidos em [Towsley, 1997]. Os resultados foram obtidos aplicando as fórmulas (4) e (10) para deste estudo: probabilidade de perda $p = 0.05$ (perda de 5%), tempo de transmissão de pacote de 1ms, tempo de gerenciamento de ACK e NACK de 0.5ms, e processamento de timeout de 0.024ms. Em relação ao gráfico, note-se que ambos modelos possuem escalabilidade bastante limitada; note-se também que as curvas representam tempos máximos de processamento (portanto sob as melhores condições) de protocolos baseados em transmissor e receptor, e não as performances típicas.

A utilização de diferentes técnicas, como avaliação analítica e simulação, é uma prática positiva, pois são partes

TRANSMISSOR

```
enquanto ha pacotes para serem transmitidos
   envia pacote de sequencia seq
   espera tempo E[Xp], tempo de envio de pacote
   enquanto ha nacks prontos para serem recebidos
      recebe nack
      espera tempo E[Xn], tempo de recebimento e processamento de Nack
      se nenhum nack foi recebido
         incrementa seq, avancando para proximo pacote
```

RECEPTOR

```
faça sempre
   recebe pacote de dados de sequencia seq
   se sorteio de perda de pacote
      se recepcao do pacote seq nao foi bem sucedida
         envia reconhecimento negativo em relacao ao pacote seq
```

Figura 3: N1 - Protocolo baseado em NACK

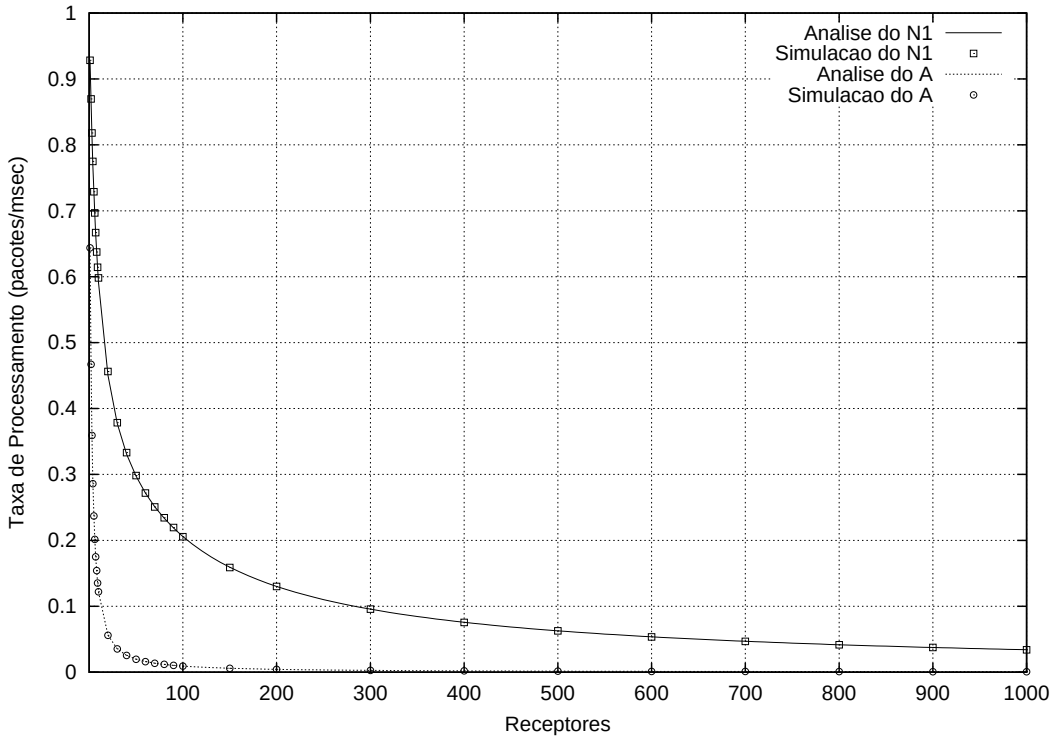

Figura 4: Comparação entre resultados analíticos e modelos simulados para A e N1.

complementares do processo científico do estudo de protocolos. Por um lado, modelos analíticos são usualmente mais simples e não exigem poder computacional; por outro, modelos de simulação podem empregar cenários mais complexos e gerar resultados mais precisos.

4 Trabalhos Relacionados

Existe uma gama considerável de trabalho em simulação de redes de computadores, resultando em diversas ferramentas interessantes. Estas podem ser divididas em *simuladores* e *ambientes de teste.* Ambientes de teste permitem que certas condições sejam emuladas filtrando, atrasando ou duplicando pacotes. Alguns exemplos destes ambientes são o Delayline ([Ingham, 1994]), Dummynet ([Rizzo, 1997]), ComFIRM ([Leite, 2000]), e o x-sim ([Brakmo, 1996]). Estas ferramentas não permitem que *todas* as condições sejam testadas, porque elas são limitadas por condições *existentes*. Alterando condições existentes, os ambientes de teste de redes modificam sistemas reais de modo que protocolos possam ser avaliados/testados em cenários particulares. Devido à existência de muitas fontes de não-determinismo em sistemas reais, resultados de um dado experimento não necessariamente poderão ser obtidos todas as vezes.

Simuladores, em contraste, devem gerar resultados reprodutíveis. Além disso, simuladores permitem que qualquer condição específica conhecida seja testada; em geral, o desenvolvedor tem muito mais controle sobre o experimento com simuladores do que com ambientes de teste. Entretanto, o modelo a ser simulado representa uma *visão abstrata* do protocolo real e da rede sob este (com sua topologia, conectividade, correlação de perda de pacote, padrões de tráfego, etc.). Construir um modelo de simulação envolve partir de premissas que simplifiquem o cenário de modo a focalizar-se nos aspectos relevantes do estudo. Sem estas simplificações, o modelo seria tão complexo quanto o sistema que ele pretende simular. De qualquer forma, a precisão dos resultados da simulação depende de quão válidas as premissas iniciais eram ([Little, 1999]).

O exemplo mais conhecido de simuladores de rede é o VINT ns ([Breslau, 2000]). O ns tem sido usado principalmente na pesquisa da performance do TCP, mas também em pesquisas de dinâmica quanto à interação de múltiplos protocolos, políticas de enfileiramento em roteadores, protocolos multimídia, protocolos de multicast confiável e controle de congestionamento. O ns tem ainda a vantagem de já possuir suporte a uma grande quantidade de tecnologias e protocolos reais. Ao contrário do Simmcast, ns utiliza duas linguagens: protocolos são escritos em C++, e scripts de simulação em OTcl (um dialeto de Tcl). Por possuir muitos recursos e ser dual, entretanto, ele tem o código desnecessariamente complexo: o simulador tem mais de 200.000 linhas de código; além disso, em algumas partes C++ e OTcl se misturam. Para desenvolver um novo protocolo, o desenvolvedor precisa compreender o funcionamento interno do ns, modificá-lo, e recompilar o ns com o suporte ao novo protocolo. Por exemplo, para criar novos tipos de pacote, arquivos do tipo cabeçalho do ns precisam ser alterados; claramente, esta não é uma boa abordagem de API. Agentes devem ser conectados através do script de simulação OTcl, o que dificulta a transmissão de um agente ns para múltiplos endereços unicast e multicast. Além disso, ao contrário do Simmcast, ns não suporta a *abordagem baseada em processos*: logo, ele *não* tem suporte para protocolos multi-thread. Estes fatores possuem um impacto negativo no desenvolvimento de protocolos mais complexos.

Simmcast foi construído visando desde o início protocolos multicast (como visto na seção 2), apesar de ele poder também ser usado para o projeto de protocolos unicast. Ele suporta multi-threading, permitindo a especificação de protocolos que são construídos a partir de componentes simples síncronos, que simplificam enormemente o projeto do protocolo. O Simmcast utiliza uma única linguagem, Java: os experimentos são especificados em um arquivo de configuração que invoca métodos Java de classes do simulador ou do usuário, através de *carga dinâmica de classes*, um poderoso recurso suportado pelo Java, e fundamental para o projeto do simulador. Simmcast é também um framework: novos protocolos simulados podem ser rapidamente desenvolvidos combinando-se blocos básicos e especializando código já presente.

5 Conclusão

A contribuição deste artigo é apresentar o Simmcast, um framework de simulação discreta baseado em processos para o projeto e avaliação de protocolos multicast. Como tal, o Simmcast tem um grande suporte interno para comunicação multicast. Seu framework possui uma estrutura de blocos básicos que permite a criação de experimentos com diferentes

níveis de abstração. O simulador é de fácil utilização, pelas seguintes razões. Em primeiro lugar, a interface de programação orientada a objetos é baseada em um conjunto conciso de primitivas, e isola o usuário do funcionamento interno do simulador. Além disso, os usuários especificam um novo protocolo combinando blocos básicos disponíveis e inserindo neles o código para a lógica do protocolo, sem a necessidade de recompilar o próprio Simmcast. Finalmente, novos experimentos são criados modificando parâmetros de entrada em um arquivo de descrição, novamente sem ter que recompilar qualquer código.

Para ilustrar o uso do Simmcast, uma comparação entre uma avaliação analítica de protocolos multicast e a simulação correspondente foi realizada. Os resultados obtidos mostraram que modelos abstratos podem ser expressados precisamente com o Simmcast. A partir daí, modelos abstratos podem ser complementados para representar diferentes níveis de detalhe, provendo informações preciosas sobre a performance real de protocolos multicast.

Referências

[Barcellos, 1998] M. Barcellos & P. Ezhilchelvan, "A Reliable Multicast Protocol using Polling for Scaleability". In **IEEE INFOCOM'98** (The Conference on Computer Communications), San Francisco, 29 March-2nd, April 1998, pp.1180-87.

[Brakmo, 1996] L. S. Brakmo & L. L. Peterson, "Experiences with network simulation", In **ACM SIGMETRICS**, Philadelphia, May 23-26, May 1996, pp. 80-90.

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

[Ingham, 1994] D. B. Ingham & G. D. Parrington, "Delayline: A Wide-Area Network Emulation Tool", **Computing Systems**, v.7, n. 3, pp.313-332, 1994.

[Jain, 1991] R. Jain, "The Art of Computer Systems Performance Analysis: techniques for experimental design, measurement, simulation, and modeling", **John Wisley & Sons**, New York, 1991, 685p.

[Leite, 2000] F. O. Leite, "ComFIRM - Injeção de Falhas de Comunicação Através da Alteração de Recursos do Sistema Operacional", **Dissertação de Mestrado**, CPGCC-UFRGS, Porto Alegre, Dez. 2000

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

[Rizzo, 1997] L. Rizzo, "Dummynet: a simple approach to the evaluation of network protocols", **ACM Computer Communication Review**, v. 27, n.1, Jan. 1997.

[Towsley, 1997] D. Towsley, J. Kurose, and S. Pingali, "A Comparison of Sender-Initiated and Receiver-Initiated Reliable Multicast Protocols", **IEEE Journal of Selected Areas in Communications**, v.15, n.3, pp.398-406, 1997.