Desenvolvimento e comparação de dois protocolos para multicast atômico em computação móvel

Mateus de Freitas Ribeiro

Dissertação apresentada ao
Instituto de Matemática e Estatística da
Universidade de São Paulo
para obtenção do grau de
Mestre em Ciência da Computação.

Área de Concentração: **Computação Móvel**

Orientador: **Prof. Dr. Markus Endler**

Durante parte do desenvolvimento deste trabalho, o autor recebeu apoio financeiro do CNPq

– São Paulo, SP – Setembro de 2002 –

Desenvolvimento e Comparação de Dois Protocolos para Multicast Atômico em Computação Móvel

Este exemplar corresponde à redação final
da dissertação devidamente corrigida
e defendida por Mateus de Freitas Ribeiro
e aprovada pela comissão julgadora.

São Paulo, 09 de setembro de 2002.

Banca Examinadora:

- Prof. Dr. Markus Endler (orientador) — DI-PUC-Rio
- Prof. PhD. Fábio Kon — DCC-IME-USP
- Prof. PhD. Antônio Alfredo Ferreira Loureiro — DCC-UFMG

À minha noiva Déa.

Agradecimentos

A Deus, que me deu forças, discernimento, equilíbrio e confiança para mudar para São Paulo diante de tantos novos desafios, estudar e atingir meus objetivos.

A todos aqueles que contribuíram diretamente no meu trabalho:

Ao Markus, pela confiança em mim e pela ótima orientação, mesmo distante fisicamente, sempre com muita dedicação, paciência, empolgação, estímulo, exigência e persistência para tornar o trabalho cada vez melhor.

Ao Fábio, que me aceitou como seu co-orientando (mesmo que informalmente) quando Markus se transferiu para o Rio, contribuindo muito no trabalho, especialmente até a qualificação.

Ao Antônio Loureiro, com suas críticas sempre construtivas que foram muito importantes para aumentar a qualidade da dissertação.

Ao Ricardo da Rocha, que mesmo já tendo defendido sua dissertação, trabalhando, não mediu esforços para implementar os *time-outs* no Mobi*CS*, além de tirar, sempre que possível, minhas dúvidas durante a implementação dos protocolos.

A todos aqueles que contribuíram indiretamente no meu trabalho:

Ao Weslley, a Bianka, o Leandro, o Alessandro, a Rosianni, o Daniel, o Marcelo, o Francisco, a Vera e todos os outros que sempre me ajudaram no que foi preciso e possível, além de toda amizade que construímos durante este período.

Ao Alfredo, o Marcelo Finger, o Alan, a Cris e todos os demais docentes que ajudaram seja de forma específica no trabalho, seja com a ampliação dos meus conhecimentos na área, ou incentivando-me sempre que possível.

Ao Denis Montini e ao Luca Lavorante, que foram grandes amigos, apoiando-me e ajudando-me para que eu apresentasse bem no dia da defesa. Valeu à pena simular tanto a apresentação. Obrigado mesmo (mas não pensem que eu esqueci as cinco perguntas durante a defesa - brincadeira). Também agradeço a Maria Alice, que sempre me apoiou para que eu pudesse concluir o trabalho.

Aos meus tios Josenildo e Cecília, e primos Marcelo e Mariana, que me acolheram como sendo parte da família, durante dois anos e meio. Meu agradecimento a vocês é especial, pois tornaram este sonho possível. Muito obrigado!

A Déa, que sempre acreditou em mim, no nosso amor, e mesmo ficando longe (fisicamente) de mim durante tanto tempo, sempre foi minha companheira, sempre me ajudou e me deu forças para que eu fosse em frente. Déa, este trabalho também é seu!

A minha família: meu pai, minha mãe, Lucinha, Kiko e Isis, que sempre me apoiaram e ajudaram no que foi preciso. Amo vocês! A meus avós e parentes que torceram e torcem por mim.

Ao meu tio Walter e a meu avô Pedro que, tenho certeza, sempre estiveram presentes, de uma forma ou de outra.

Resumo

Aplicações que demandam uma sincronização entre usuários móveis requerem um mecanismo para a difusão (multicast) de mensagens entre dispositivos móveis. Em alguns casos, o multicast deve ser atômico, isto é, ou todos os elementos do grupo processam a mensagem difundida, ou nenhum deles. O AM^2C é um protocolo para multicast atômico em Computação Móvel. Entretanto, o principal problema do AM^2C é sua falta de escalabilidade, uma vez que este protocolo faz uma difusão para todas as estações-base na rede fixa.

Esta dissertação descreve o projeto, implementação, simulação e avaliação dos protocolos AM^2C e iAM^2C, onde o segundo protocolo é uma variante escalável do primeiro. As implementações e simulações foram feitas usando o ambiente *MobiCS- Mobile Computing Simulator*, que é uma ferramenta para o teste e a análise de desempenho de protocolos distribuídos em redes móveis. A dissertação apresenta uma comparação detalhada dos protocolos para diferentes configurações de rede e diversos padrões de migração dos *hosts* móveis.

Abstract

Applications that demand some synchronization among mobile users require a mechanism for reliable delivery of multicast messages to a group of mobile hosts. In some cases, the multicast must be atomic, i.e., either all or none of the mobile hosts accept each message. The AM^2C is a protocol for atomic multicast in Mobile Computing. However, the main problem of AM^2C is its lack of scalability, since it is based on broadcasts to all base stations in the fixed network.

This work describes the project, implementation, simulation and evaluation of the protocols AM^2C and iAM^2C, where the second protocol is a scalable variant of the first. The implementations and simulations were performed using *MobiCS- Mobile Computing Simulator*, that is a tool for testing and doing performance analysis of distributed protocols in mobile networks. This work presents a detailed comparison of the protocols for different network configurations and various patterns of mobile hosts migrations.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Com o grande crescimento ocorrido nas áreas de telefonia celular, redes locais sem fio [4, 16] e serviços via satélite, em um futuro próximo será possível que informações e recursos sejam acessíveis a seus usuários a qualquer momento, independente do lugar onde estejam situados. Cada vez mais computadores portáteis e assistentes pessoais digitais (*PDA's*) estão disponíveis no mercado e cada vez mais esses equipamentos deverão ter a capacidade de se comunicar com outros computadores em rede e computadores móveis. Esse ambiente computacional é chamado de Computação Móvel ou Computação Nômade [23].

A computação móvel constitui um novo paradigma computacional que está revolucionando o desenvolvimento e uso de sistemas distribuídos [15]. Nesse novo ambiente, usuários têm acesso a serviços independentemente de suas localizações durante suas movimentações. Dessa forma, computação móvel amplia o conceito de computação distribuída. Na Figura 1.1 é apresentado o modelo de rede móvel estruturada de um sistema distribuído com computadores móveis. Este modelo consiste de duas partes: uma formada por uma infra-estrutura fixa de comunicação, com componentes convencionais como *hosts*, roteadores, *bridges*, etc (rede fixa); e outra, interligada à primeira, representada por um conjunto de áreas geográficas (células) onde é possível haver comunicação sem fio entre os dispositivos móveis (computadores móveis localizados na célula). Cada célula é atendida por transmissores de rádio-freqüência (estações-bases ou *Mss*s, de *Mobility Support Station*). A comunicação entre um dispositivo móvel e qualquer outro dispositivo é intermediada por uma estação-base. Neste trabalho estamos assumindo que toda a comunicação na rede fixa e entre *Mss*s é confiável, isto é, que não há perdas e/ou corrupção de mensagens.

A computação móvel introduz uma série de problemas inexistentes em sistemas distribuídos

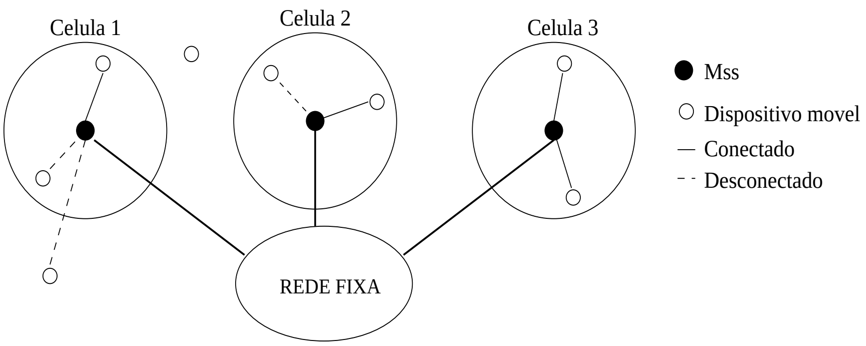

Figura 1.1: Rede móvel estruturada

tradicionais, que são causados, principalmente, pelos seguintes fatores:

- **Comunicação sem fio:** a conexão sem fio é muito mais instável do que a conexão com fio, pois está sujeita às interferências na transmissão de sinais, regiões não cobertas, ecos, etc. Além disto, conexões sem fio apresentam problemas como largura de banda menor, alta taxa de erros (cerca de cinco a dez vezes a ordem de grandeza da taxa de erros na comunicação com fio), desconexões freqüentes (sejam voluntárias ou não), maior risco à segurança (mais facilidade na interceptação de mensagens e no rastreamento de computadores móveis), entre outros.

- **Mobilidade:** a capacidade de um computador móvel mudar de localização enquanto está se comunicando com a rede provoca diferenças na maneira como algumas informações são vistas. Certos dados de configuração estáticos na computação tradicional assumem uma natureza dinâmica na computação móvel. Por exemplo, um computador com endereço fixo pode ser configurado estaticamente para acessar o servidor mais próximo (ou adequado), enquanto que um computador móvel necessita de um mecanismo para selecionar dinamicamente o servidor a ser usado. Além disto, precisa-se tratar de determinar o endereço atual de um dispositivo móvel a fim de que a comunicação com tal dispositivo torne-se viável.

  A mobilidade dos computadores móveis também gera problemas relacionados à gerência de localização de tais elementos (como o custo de pesquisa da localidade de um determinado dispositivo), à heterogeneidade da computação móvel (como, por exemplo, a carga dinâmica das estações-base), entre outros problemas.

- **Portabilidade:** em contraste com computadores tradicionais, dispositivos móveis são menores e mais leves a fim de facilitar a portabilidade. Por isso, recursos como energia[1], capacidade de processamento, memória principal, existência de memória secundária e interface com o usuário são limitados e precisam ser levados em conta no projeto de aplicações, serviços e protocolos para computação móvel.

1.1 Motivação

Apesar dos problemas mencionados anteriormente, é provável que em um futuro próximo seja possível usufruir de uma série de aplicações a partir de computadores móveis. Algumas dessas aplicações podem requerer alguma forma de coordenação ou sincronização entre grupos de clientes móveis. Exemplos de tais aplicações são planejamento e execução estratégica de missões, otimização de rotas de veículos, trabalho corporativo, e outros. Para dar suporte à coordenação entre usuários, um dispositivo móvel deve ser capaz de difundir uma mensagem (*multicast*) para um grupo fixo de computadores móveis de forma confiável. Algumas aplicações exigem ainda que o multicast seja atômico (com uniformidade dinâmica [3]), ou seja, que a mensagem de multicast seja processada ou por **todos** ou por **nenhum** dos membros do grupo, garantindo assim uma visão consistente de um estado global distribuído. Em outras palavras, aplicações que utilizam multicast baseado em uma semântica de entrega atômica de mensagens adotam o conceito de *validade de um multicast*, que estipula o seguinte:

- Se algum elemento destinatário de um multicast M não confirmar o recebimento do mesmo ou não for capaz de processar a mensagem, então M não será processado por nenhum outro membro do grupo;

- Se todos os destinatários de um multicast M confirmarem seu recebimento e forem capazes de processá-lo, então M é confirmado e todos os destinatários que estiverem ativos irão processar o multicast.

Desta forma, é relevante a disponibilização de serviços de multicast confiável que garantam a entrega de mensagens *todos-ou-nenhum* em uma rede móvel, onde os dispositivos podem mudar de ponto de acesso e/ou ficarem desconectados freqüentemente.

[1]Uma vez que a tecnologia de construção de baterias não tem acompanhado a evolução de outros segmentos da informática, o fator "energia" é considerado o maior problema no uso de computadores móveis.

1.2 Objetivos e Resultados

O trabalho de pesquisa aqui proposto diz respeito ao desenvolvimento, implementação, avaliação e comparação de dois protocolos para multicast atômico em computação móvel (o AM^2C [13] e o iAM^2C [10]). Os protocolos foram concebidos para redes móveis estruturadas, como exemplificado na Figura 1.1. Simulamos o AM^2C e o iAM^2C e obtivemos dados que permitiram comparar os dois protocolos, identificando as vantagens e desvantagens de cada um. Os resultados das simulações também demonstraram que ambos os protocolos garantem a atomicidade de entrega de mensagens requerida, realizando multicast de forma confiável.

1.3 Principais Contribuições

A principal contribuição deste trabalho consiste no projeto e implementação de dois protocolos que realizam multicast confiável e atômico entre grupos de usuários móveis. Os protocolos podem ser úteis para aplicações que necessitem dar suporte à ações coordenadas entre dispositivos móveis. O trabalho compara (através de simulações) os protocolos em diversas configurações de rede e padrões de mobilidade dos *Mhs*, identificando em quais cenários um ou outro protocolo é mais adequado.

1.4 Organização do Texto

Esta dissertação está organizada da seguinte forma: o próximo capítulo apresenta os resumos de alguns trabalhos relacionados com os protocolos aqui propostos. Os Capítulos 3 e 4 descrevem, respectivamente, os protocolos AM^2C e iAM^2C em detalhes. O Capítulo 5 apresenta a forma como os protocolos foram implementados e os testes realizados com as simulações, juntamente com suas interpretações. No Capítulo 6 é relatada a experiência adquirida com as implementações e simulações dos protocolos, bem como as principais dificuldades encontradas durante este processo. Finalmente, o Capítulo 7 resume os princiapis pontos do trabalho, concluindo o mesmo e apresentando os trabalhos futuros.

Capítulo 2

Trabalhos Relacionados

Este capítulo descreve sucintamente os principais trabalhos relacionados aos protocolos AM^2C e iAM^2C. Desta maneira, a seção 2.1 apresenta protocolos e algoritmos relacionados com multicast de mensagens. Já a seção 2.2 aborda algumas das técnicas utilizadas para o gerenciamento da localização de unidades móveis que, como veremos nos capítulos seguintes, serão utilizadas pelo protocolo iAM^2C.

2.1 Trabalhos Relacionados com Multicast

Existem vários protocolos e/ou algoritmos relacionados com a entrega confiável de mensagens multicast para grupos de *hosts* móveis. Em outros trabalhos são implementados protocolos que realizam multicast de forma não confiável. Porém, com exceção dos protocolos analisados e implementados neste trabalho, desconhecemos outros protocolos que realizem multicast atômico, ou seja, que implementem uma semântica de entrega *todos-ou-nenhum*.

O protocolo *MCAST* [1], de Acharya e Badrinath, garante a entrega confiável de mensagens a seus respectivos destinatários. Neste caso, confiabilidade significa que eventualmente a mensagem de multicast será entregue "exatamente uma vez" a todos os *hosts* móveis destinatários alcançáveis. A política de entrega de mensagens no *MCAST* é garantida mesmo considerando que um *Mh* pode se conectar a diferentes *Mss*s e em diferentes momentos, e que cópias de uma mensagem podem chegar nos *Mss*s em momentos diferentes devido à latência de comunicação na rede. O protocolo utiliza a estratégia de difundir cópias de uma mensagem para todos os *Mss*s, os quais se encarregam de retransmitir a mensagem para todo *Mh* local e que seja um elemento destinatário.

O modelo de sistema do *MCAST* para interconexão móvel é semelhante ao utilizado pelo protocolo AM^2C (como veremos no capítulo seguinte), o qual é baseado em estações de suporte à mobilidade na rede fixa. O *MCAST* assume que a rede fixa garante a entrega confiável e seqüencial de mensagens entre dois *Mss*s e que a rede sem fio garante a entrega de mensagens em ordem FIFO para os *hosts* móveis.

Uma implementação de um mecanismo de retransmissão de pacotes para lidar com as características dinâmicas de um grupo multicast é vista em [11]. Este mecanismo foi implementado no protocolo *RM2* [27] (*Reliable Mobile Multicast*), que é um protocolo adaptativo para multicast confiável projetado tanto para ambientes fixos como para ambientes móveis.

O *RM2* provê a entrega em seqüência e sem perda de pacotes para usuários móveis, utilizando uma abordagem hierárquica para o multicast. Além dos *hosts* móveis, também fazem parte do modelo de sistema do protocolo os *RS*s (servidores de retransmissão) e os *MR*s (roteadores de multicast).

O protocolo *RM2* se utiliza dos resultados de uma análise entre a carga na rede gerada quando pacotes são retransmitidos via multicast ou unicast, e a carga excedente gerada pela quantidade de pacotes duplicados observados na rede. A partir desta análise, chega-se a um algoritmo eficiente que leva em consideração essas duas medidas e que faz o melhor uso possível dos recursos da rede.

Em Raynal *et. al.* [26] é apresentado um algoritmo eficiente para a entrega de mensagens em ordem causal, onde é utilizado um vetor de *timestamps*. Para garantir esta semântica de entrega, uma mensagem (seja na rede fixa ou no enlace sem fio) precisa carregar consigo somente a informação das mensagens predecessoras diretas em relação a cada elemento destinatário. Assim, se duas mensagens *M1* e *M2* possuem o mesmo elemento destinatário e se o envio de *M1* precede causalmente o envio de *M2*, então a entrega de *M1* será feita antes da entrega de *M2*. Ainda, o algoritmo leva em consideração a informação extra que é mantida nos algoritmos existentes (como a dependência transitiva entre as mensagens) e como essa sobrecarga pode ser eliminada sem comprometer a corretude do algoritmo.

Alagar e Venkatesan [2] apresentam três algoritmos (que são uma extensão do algoritmo de Schiper, Raynal e Toueg [21]) para entrega causal e confiável de mensagens em sistemas de computação móvel. O primeiro algoritmo lida com restrições de recursos dos elementos móveis, mas não é facilmente escalável em relação ao número de *Mhs* no sistema. Já o segundo, elimina a desvantagem do primeiro pelo custo de restringir alguns tipos de mensagens. O terceiro algoritmo é uma combinação dos dois anteriores. Nestes algoritmos, diferentemente do que ocorre em [26], a maioria das tarefas de ordenação e filtro de mensagens são processadas nos *Mss*s, que atuam como representantes dos seus *Mhs* locais.

Outros trabalhos [20, 18, 31] tratam de extensões do protocolo de multicast IP para *hosts* móveis, que não provê quaisquer garantias de entrega dos multicasts. A versão atual do *Mobile IP* [24] possui dois métodos para multicast entre *hosts* móveis, chamados de *assinatura remota* e *tunelamento bi-direcional* (*remote subscription e bi-directional tunneling*). Os dois métodos são baseados no uso de *home agent* e *foreing agent*, como no *Mobile IP* original.

Harrison *et al.* propõe um mecanismo chamado *MoM* [18] para serviços de multicast. Tal mecanismo resolve problemas de ajuste de convergência e duplicação de multicast definindo um Provedor Específico de Multicast (*Designated Multicast Service Provider - DMSP*) dentre todos os *home agents* que têm um *host* móvel conectado em um link diferente.

A idéia básica do *MoM* é usar a funcionalidade do *home agent* do *IETF Mobile IP* [24] para entregar pacotes multicast aos dispositivos móveis de forma eficiente. O modelo mantém a escalabilidade do mecanismo através de uma otimização do *DMSP* por grupo multicast e do uso de tunelamentos dinâmicos de multicast para *foreing networks.*

Em [30] muitas alternativas na escolha do DMSP são consideradas e seus desempenhos são avaliados usando um simulador de eventos discretos. Ainda, Kanodia *et al.* em [19] propõem um mecanismo de adaptação utilizando um algoritmo competitivo para chavear dinamicamente entre o protocolo de *tunneling* e *subscription.*

Anastasi *et. al.* [17] propõem um protocolo para multicast confiável entre *hosts* móveis onde o remetente de cada mensagem pode selecionar a semântica de entrega da mensagem: FIFO, causal ou total. Neste protocolo, o grupo de remetentes e destinatários pode ser dinâmico, ou seja, um *Mh* pode entrar ou sair do grupo dependendo da necessidade da aplicação. Ao contrário de outros protocolos, esta proposta não possui uma componente de *hand-off*, de forma que nenhuma mensagem neste sentido é trocada entre *Mss*s durante a migração de um *Mh*. Isto torna o protocolo eficiente em cenários com muitas migrações e com um número grande de elementos móveis. Por outro lado, o protocolo requer que os *Mss*s periodicamente re-difundam (*re-broadcasting*) multicasts pendentes e que os *Mhs* requisitem explicitamente retransmissões de mensagens "perdidas" através de *Negative Acks* (*NACKS*).

Maffeis *et al.* descrevem em [22] um serviço genérico e confiável na camada de transporte, que oferece multicast preservando-se a ordem de chegada das mensagens para diferentes protocolos de comunicação. Este serviço é implementado como um conjunto chamado de *servidores GTS* (*Generic Multicast Transport Service*) que se comunicam com outros servidores *GTS* ou com os clientes.

Alguns dos serviços oferecidos pelo *GTS* são: tolerância a falhas, preservação da ordem de entrega de mensagens multicast, comunicação cifrada, reconfiguração e operações em modo desconectado.

2.2 Localização de Dispositivos Móveis

Em Computação Móvel o gerenciamento da localização de unidades móveis se torna essencial para a comunicação com estes tipos de *hosts*.

O gerenciamento de localização essencialmente consiste de buscas (*lookups*) pela atual localização de um dispositivo móvel quando uma mensagem é enviada ou uma chamada é gerada por este, e de atualizações de localização (*updates*), quando o dispositivo se move para uma nova região.

Pitoura e Samaras [25] apresentam técnicas de gerenciamento de estruturas de dados relacionadas à localização (registros de localização, ou *Location Databases - LDs*), ou seja, registros usados para armazenar a informação sobre a localização de usuários móveis. Neste sentido, são consideradas duas arquiteturas básicas de *LDs*: a arquitetura de duas camadas (*two-tier architecture*) e a arquitetura hierárquica, que é o alvo desta discussão. Esses modelos são baseados em três níveis, que são a rede fixa (*fixed network*), a rede de acesso (*access network*) e a rede inteligente (*intelligent network*) [29]. A primeira é a rede com fio convencional; a segunda é a interface entre os usuários móveis e a rede fixa; já a terceira é a rede que conecta qualquer registro de localização.

A arquitetura hierárquica é uma extensão do esquema de duas camadas através de uma hierarquia dos registros de localização conectados por meio da rede inteligente. Neste esquema, um registro em um nível mais alto contém informações de localização de usuários mapeados nos registros dos níveis abaixo. Usualmente, esta arquitetura é baseada em uma estrutura de árvore. Assim, um registro de uma folha serve a uma única célula e contém informações sobre a localização dos usuários localizados em tal célula. Um registro de um nó intermediário mantém informações sobre a localização dos usuários registrados nas células da sub-árvore correspondente. Esta informação pode ser um ponteiro para um registro de um nível mais baixo ou o endereço (identificador da célula) da localização do usuário.

Algumas técnicas são utilizadas pelo modelo hierárquico a fim de reduzir o custo dos *lookups* e *updates*. Estaremos nos concentrando no primeiro caso. A partir de agora, adotamos a mesma notação utilizada em [25], onde $LCA(i,j)$ denota o ancestral comum mais próximo entre as células i e j.

Uma das técnicas usadas na arquitetura hierárquica é o *caching*. Quando uma chamada é feita (ou mensagem é enviada) da célula i para um usuário x, localizado na célula j, o procedimento de busca percorre a árvore "subindo" de i até o *LCA(i,j)*, e depois "descendo"até j. Da mesma forma, deve-se considerar um *acknowledgment* que retorna de j até i. O *caching* efetivamente ocorre durante este retorno, quando um par de ponteiros (*bypass pointers*), chamados de *forward bypass* e *reverse bypass*, é criado. O primeiro é uma entrada de um ancestral de i, digamos s, que aponta para um ancestral de j, digamos t. O segundo é um ponteiro de t para s. Dessa forma, na próxima chamada da célula i para o usuário x, a mensagem de busca "sobe" na árvore até encontrar s, e então segue direto para o registro t. De forma similar, o *acknowledgment* é processado no sentido inverso.

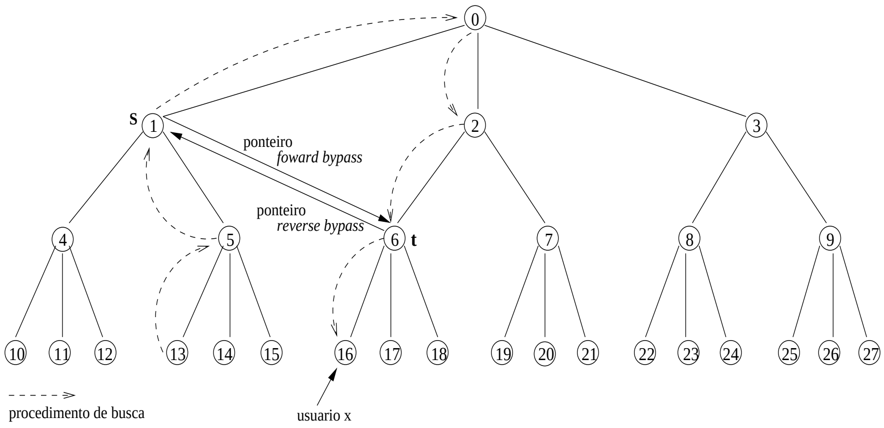

Figura 2.1: Esquema Hierárquico Utilizando Técnicas de Caching

A figura 2.1 exemplifica o esquema hierárquico com a utilização de técnicas de *caching*, onde uma chamada é feita da célula 13 para o usuário x localizado na célula 16. Um ponteiro *forward bypass* é inserido na árvore apontando do nó 1 para o nó 6, enquanto que um ponteiro *reverse bypass* por sua vez, aponta do nó 6 para o nó 1. Desta maneira, na próxima chamada da célula 13 para o usuário x o procedimento de busca percorre a árvore do nó 13 até o nó 1 e, então, pode seguir até o nó 6 ou via *LCA(1,6)*, ou seja, o nó 0, ou através do ponteiro *forward bypass* (caminho mais curto). Em ambos os casos, nenhuma busca é efetuada nos registros dos nós 0 e 2.

O nível dos nós s e t onde o par de *bypass pointers* é criado pode variar. No modelo *simple caching*, s e t são folhas. Já no modelo *level caching*, s e t são nós de qualquer nível e, geralmente, possuem níveis diferentes. Estes mesmos modelos são utilizados em esquemas hierárquicos que mantém a própria localização do usuário nos registros, ao invés de ponteiros.

Uma outra técnica utilizada para reduzir o custo dos *lookups* é a replicação. Nesta técnica, a informação sobre a localização de um usuário é replicada (de acordo com certo algoritmo) em diversos registros, de forma a encurtar o processo de localização do usuário/dispositivo destinatário de uma chamada (ou mensagem). No entanto, a cada vez que o elemento móvel migrar, o seu endereço em todas as réplicas terá que ser atualizado. Por isto, a replicação só é usada quando o benefício desta é maior que seu custo.

Veremos que um dos protocolos discutidos neste trabalho (iAM^2C) adota um gerenciamento de localização em duas camadas, que se utiliza de uma forma específica de *caching* e replicação. A principal diferença consiste no fato de que no iAM^2C o objetivo principal é reduzir o custo da difusão de mensagens pela rede fixa, e não o de localizar um usuário destinatário de uma mensagem. Ainda assim, este protocolo faz uso de uma representação da área de cobertura em dois níveis hierárquicos (domínio e célula), e usa replicação para paralelizar o acesso e a difusão de multicasts.

Capítulo 3

O Protocolo AM^2C

3.1 Motivação

Um serviço de multicast confiável para redes com computadores (*hosts*) móveis é essencial para o desenvolvimento de aplicações baseadas na coordenação e sincronização de ações entre usuários móveis. Aplicações que além disto precisam garantir que as cópias de algum dado nos *hosts* móveis sejam mantidas perfeitamente sincronizadas demandam um protocolo para multicast *atômico*, ou seja, em que ou *todos* ou *nenhum* dos *hosts* do grupo processem cada mensagem.

Um exemplo de tal aplicação poderia ser um sistema de controle de estoque para vendedores ambulantes, que necessitam manter uma visão consistente dos dados armazenados nos *hosts* móveis de todos os membros do grupo. Suponha que o computador móvel de cada vendedor contém uma lista de produtos (e quantidades de itens) que podem ser vendidos diretamente sem que se faça necessária uma consulta a outros vendedores ambulantes, e que a quantidade total de itens de produtos disponíveis para venda é a soma das quantidades de itens registrados em cada um dos computadores dos vendedores. Assim, caso um vendedor precise vender mais itens do que a quantidade indicada em seu computador, este terá que requisitar/reservar também itens do produto originalmente alocados a outro(s) vendedor(es). Esta informação sobre a *reserva de itens* deve estar disponível para todos os vendedores, a fim de garantir uma visão consistente do conjunto total de itens disponíveis e evitar vendas não sincronizadas do mesmo lote de itens. Em particular, a *reserva* não pode ser confirmada se um dos vendedores não estiver "conectado" à rede (*off-line*), pois este vendedor pode estar executando a venda de algum lote de itens a ele alocado e que esteja sendo também reservado por outro vendedor.

Protocolos *Two Phase Commit* (2PC) [14] representam uma possibilidade interessante para a implementação de serviços de multicast atômico, devido à sua simplicidade. No entanto, é importante que o protocolo 2PC seja executado pelos *hosts* da rede fixa, que apresentam alta confiabilidade e disponibilidade, quando comparados com os *hosts* móveis. Com isto, e o uso de *time-outs* para a transmissão sem fio, diminui-se sensivelmente o risco de bloqueios, que são o principal problema de protocolos 2PC.

De forma a dar suporte à aplicações em redes móveis que necessitem de um multicast atômico, foi projetado um protocolo para multicast atômico entre *hosts* móveis, chamado de AM^2C [13]. Essencialmente, o AM^2C consiste de um *Two Phase Commit* com uma fase adicional para "coleta de lixo". Neste protocolo, uma mensagem de multicast só é confirmada se todos os destinatários estiverem disponíveis (alcançáveis e dispostos a processar a mensagem) durante a primeira fase do protocolo. Neste caso o multicast é *confirmado*, caso contrário é *abortado*. A segunda fase garante que todos os *hosts* móveis serão eventualmente informados sobre o estado final do multicast, mesmo que isso requeira várias retransmissões deste estado para os *Mhs*. Finalmente, a última fase é executada a fim de remover a informação sobre o estado final do multicast. A seção 3.3 discute o AM^2C de forma mais detalhada.

3.2 Modelo de Sistema e Suposições

O modelo de sistema é composto por duas categorias de máquinas: as estações de suporte à mobilidade (*Mobility Support Stations*, ou *Mss*s) e os *hosts* móveis (*Mh*s).

Um *Mss* é um *host* da rede fixa que possui uma interface para comunicação sem fio e que está interligado aos demais *Mss*s pela rede convencional. Cada *Mss* define uma região geográfica (célula) onde é possível enviar e receber mensagens para/de todos os *Mhs* que estejam localizados na célula (estes constituem o conjunto `LocalMhs`). Neste modelo, assumimos que *Mss*s não falham e que a comunicação pela rede fixa é confiável.

Os *hosts* móveis são elementos que possuem um identificador único no sistema e podem estar em dois possíveis estados: *ativados* ou *desativados*. Quando está desativado um *Mh* não é capaz de receber ou enviar nenhuma mensagem. Para entrar no sistema, um *Mh* envia a mensagem `Join` para o *Mss* responsável pela célula na qual se encontra. Uma vez que o *Mh* se torna membro do sistema, tal *Mss* passa a ser o *Mss* responsável pelo *Mh* (denominado de Mss_{Mh}). De maneira análoga, antes de sair do sistema um *Mh* deve enviar a mensagem `Leave` para seu Mss_{Mh}.

Hosts móveis são capazes de migrar de uma célula para outra. Ao entrar na região de uma nova célula um *Mh* envia a mensagem `Greet(`Mss_{Mh}`)` para o *Mss* da nova célula (que chamaremos genericamente de Mss_n), indicando o *Mss* da célula anterior, Mss_{Mh}. A partir daí é iniciado o protocolo de *hand-off* entre o novo e o antigo *Mss*. Um *hand-off* pode ser entendido como o processo de transição de um *Mh* de uma célula para outra, onde uma série de ações devem ser tomadas a fim de efetuar o registro do *Mh* na nova célula e de transferir alguns dados considerados relevantes para a aplicação. A seção 3.5 descreve o protocolo de *hand-off* em detalhes.

Toda vez que um *Mh* volta ao estado ativo na mesma célula na qual se encontrava antes da sua desativação, a mensagem `Greet` também é enviada. Neste caso específico, o protocolo de *hand-off* não é iniciado pelo Mss_{Mh}, uma vez que não houve troca de célula. Neste modelo, abstraímos dos detalhes de como ocorre a "percepção" da entrada de um *Mh* em uma nova célula, por entender que isto pode variar dependendo da tecnologia de comunicação sem fio sendo utilizada e por não ser relevante do ponto de vista dos protocolos sendo estudados.

A interface de comunicação sem fio, além de possuir uma menor largura de banda passante, também apresenta uma taxa maior de erros transientes, ou seja, que podem ocorrer esporadicamente e que geralmente têm curta duração. Por isto, qualquer protocolo envolvendo uma transmissão de dados por um meio sem fio deve contemplar repetidas tentativas de transmissão por este meio. Na descrição do protocolo AM^2C (assim como do iAM^2C) de certa forma também abstraímos deste detalhe, simplesmente indicando uma única transmissão pelo meio sem fio e um tempo máximo de espera pela confirmação de um *Mh*. Ou seja, no caso do uso de uma tecnologia de rede sem fio que faça repetidas tentativas de transmissão, então o envio de uma mensagem em nosso modelo deve ser interpretado como o período compreendido entre o início de uma série de retransmissões até o recebimento da mensagem pelo seu elemento destino.

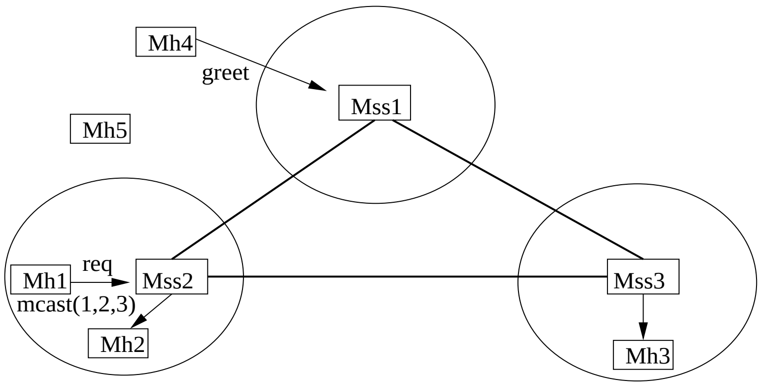

Figura 3.1: Modelo de Sistema no Protocolo AM^2C.

A Figura 3.1 exemplifica um sistema composto por três *Mss*s e cinco *Mhs*, onde o *Mh1* faz uma requisição de multicast de uma mensagem para *Mh2* e *Mh3*.

Resumindo, as principais suposições do modelo são:

1. *Mss*s não falham e todos os *Mss*s estão conectados. É assumido que os *hosts* fixos também possuem recursos suficientes para o armazenamento e o processamento de dados e mensagens;

2. A comunicação na rede fixa é confiável, isto é, as mensagens em algum momento chegarão a seu destino. A entrega das mensagens na rede fixa é feita em ordem causal (esta semântica de entrega pode ser facilmente implementada utilizando-se um vetor de *timestamps*, como proposto por Raynal *et. al.* [26] - vide seção 2.1);

3. A comunicação sem fio não é confiável, mas a latência máxima de comunicação no meio sem fio é fixa e conhecida, ou seja, é possível determinar uma falha na transmissão sem fio;

4. Em cada instante, todo *Mh* do sistema está associado a um único *Mss* (Mss_{Mh});

5. Cada *Mh* possui uma identificação única no sistema que independe de sua localização. Antes de deixar o sistema um *Mh* envia uma mensagem `Leave` para seu Mss_{Mh}. Se o mesmo dispositivo eventualmente retornar ao sistema o fará através da mensagem `Join`, e receberá uma nova identificação;

6. Se um *Mh* está ativo e se encontra na célula de um Mss_{Mh}, então responde a qualquer mensagem enviada por Mss_{Mh}. Caso contrário, ele não responde a nenhuma mensagem;

7. Depois de ter se anunciado a um novo *Mss* (Mss_n), um *Mh* não interage com nenhum outro *Mss* que não seja Mss_n.

3.3 Descrição do Protocolo

O protocolo de multicast atômico AM^2C é uma extensão do protocolo *MCAST* [1] de Acharya e Badrinath. Este último implementa uma semântica de entrega *exatamente uma vez*, garantindo assim que todos os destinatários alcançáveis de uma certa mensagem vão recebê-la uma única vez. Todavia, o *MCAST* não provê nenhuma confirmação para os destinatários sobre o fato de a mensagem ter sido ou não aceita pelos mesmos. Já o AM^2C implementa uma semântica de entrega *todos ou nenhum*, ou seja, realiza um multicast **atômico**. Atomicidade significa que uma mensagem só será processada pelos destinatários se todos estiverem disponíveis no momento da transmissão do multicast e confirmarem o recebimento do mesmo. Além disso, o AM^2C informa ao grupo de destinatários de uma mensagem se esta foi ou não aceita por todos os membros do grupo.

O protocolo de multicast atômico AM^2C é executado em 3 fases. A seguir, serão descritos os principais eventos em cada uma das fases:

- **FASE I**

  Um *host* móvel, dito Mh_{sender}, envia uma requisição de multicast de uma mensagem M para o seu *Mss* responsável, que neste caso passa a iniciar o processo de difusão na rede fixa e, por isso, será denominado de Mss_{init}. Juntamente com a mensagem segue uma lista dos destinatários do multicast (*M.Dest*). Mss_{init} difunde então a nova requisição para todos os *Mss*s (assim como ocorre no *MCAST* [1]) e fica aguardando uma resposta dos mesmos. Cada *Mss* por sua vez, envia M para todos os *Mh*s que estão sob sua responsabilidade (`LocalMhs`) e que fazem parte de *M.Dest*, e espera T1 unidades de tempo por uma resposta destes. Caso não haja nenhum *Mh* local pertencente à lista de destinatários, ou se todos os *Mh*s locais respondem com um $\texttt{Ok}_M$ antes de T1 se esgotar, então o *Mss* envia uma mensagem $\texttt{Ok}_M$ para Mss_{init}. Caso contrário, é enviada a mensagem $\texttt{NOk}_M$. Esta mensagem portanto sintetiza as respostas de todos os *Mhs* locais que são destinatários. O mesmo é feito pelo Mss_{init}, sendo a única diferença que Mh_{sender} não é incluído na lista de destinatários locais.

  Quando um *Mh* $\in$ *M.Dest* recebe M de seu Mss_{Mh}, ele armazena a mensagem e responde imediatamente[1] com um $\texttt{Ok}_M$ ou $\texttt{NOk}_M$, dependendo do caso se a mensagem pode ou não ser processada pela aplicação local executada no *Mh* (por exemplo, pode não haver energia suficiente para processar a mensagem). Caso M chegue a um *Mss* durante um *hand-off* envolvendo o *Mss* e um *Mh* $\in$ *M.Dest*, então dependendo do estágio do *hand-off* (isto será discutido em mais detalhes na seção 3.5.2), eventualmente o tempo de espera no *Mss* da célula para a qual o *Mh* está migrando é estendido por mais T1 unidades de tempo.

- **FASE II**

  Depois de receber respostas de todos os *Mss*s, Mss_{init} verifica se todas as respostas dos *Mss*s são do tipo $\texttt{Ok}_M$ ou se alguma delas é do tipo $\texttt{NOk}_M$. No primeiro caso, o estado final (*status*)

[1] Assume-se também que o *Mh* possui a capacidade de responder a uma mensagem em um espaço de tempo previsível e conhecido.

do multicast é dito *Confirmado (Commited)*, caso contrário, *Abortado (Aborted)*. Mss_{init} então difunde o estado final ($Comm_M$ ou $Abor_M$) para todos os *Mss*s.

Quando recebe uma mensagem $Comm_M/Abor_M$ de Mss_{init}, cada *Mss* a armazena e encaminha a mensagem para o conjunto de *Mh*s locais que pertencem a *M.Dest*. O *Mss* aguarda um período de T2 unidades de tempo por uma confirmação dos referidos *Mh*s locais. Seja `C` o conjunto de *Mh*s locais que confirmaram o recebimento desta mensagem de *status*. Então o *Mss* envia a mensagem `AckSet{C}` de volta a Mss_{init}.

Caso um *Mss* se torne responsável por um novo *Mh* que tenha migrado para sua célula, digamos Mh_{new}, o *Mss* verificará se para cada mensagem armazenada *M*, Mh_{new} pertence a *M.Dest*. Em caso positivo, o *Mss* transmite o estado final de *M* para Mh_{new} e aguarda até T2 unidades de tempo pela confirmação do recebimento. Se esta confirmação chega a tempo, o *Mss* envia uma mensagem `AckSet{Mh`$_{new}$`}` para Mss_{init}. Caso contrário, nenhuma mensagem é enviada.

- **FASE III**

  Toda vez que Mss_{init} recebe uma mensagem `AckSet(C)` de um *Mss*, ele adiciona os elementos contidos em `C` numa lista chamada *M.Confirmados*. Quando *M.Confirmados* for igual a *M.Dest*, então Mss_{init} remove a entrada referente à mensagem *M* armazenada localmente e difunde uma mensagem (Del_M) a todos os *Mss*s para que estes apaguem o estado final de *M* armazenado em seus *buffers* locais. Ao receber tal mensagem, cada *Mss* executa a operação solicitada.

3.4 Exemplo de Funcionamento do AM^2C

Nesta seção apresentamos um cenário para demonstrar o funcionamento do protocolo AM^2C.

Na Figura 3.2 são ilustrados os tipos de mensagens transmitidas pelo protocolo AM^2C em um cenário composto por três *Mh*s e três *Mss*s, onde inicialmente Mh_i encontra-se na célula de Mss_i, para $1 \leq i \leq 3$. Mh_1 requisita um multicast da mensagem *M* com $M.Dest = \{Mh_1, Mh_2, Mh_3\}$ (a inclusão de Mh_1 como destinatário tem como objetivo possibilitar que o mesmo receba o estado final do multicast) para Mss_1. Fazendo o papel de Mss_{init} para esta requisição, Mss_1 então difunde *M* para todos os outros *Mss*s, no caso Mss_2 e Mss_3. Estes, por sua vez, enviam a mensagem para seus *Mh*s locais que pertencem ao conjunto de destinatários de *M* (Mh_2 e Mh_3). Tanto Mh_2 como Mh_3

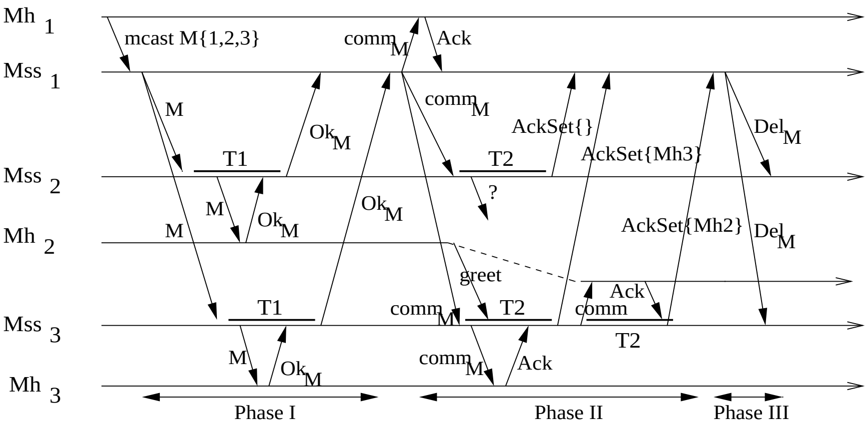

Figura 3.2: Funcionamento do protocolo AM^2C

respondem a seus respectivos Mss_{Mh} antes de T1 se esgotar. Mss_2 e Mss_3 enviam então um $\texttt{Ok}_M$ para Mss_{init}, que então estabelece o estado final de M como *confirmado (Commited)* e envia uma mensagem $Comm_M$ para Mh_1 (solicitante), Mss_2 e Mss_3. Ao enviar $Comm_M$ para Mh_2, Mss_2 não obtém resposta, pois Mh_2 está migrando para a célula de *Mss*$_3$. Com isto, Mss_2 envia uma mensagem `AckSet{}` vazia para Mss_{init}. Já Mss_3 consegue enviar a mensagem $Comm_M$ para Mh_3, o qual confirma o recebimento da mesma, fazendo com que Mss_3 envie a mensagem `AckSet{`Mh_3`}` a Mss_{init}.

Quando termina o processo de *Handoff* e Mh_2 passa a pertencer ao conjunto de *Mh*s locais de Mss_3, este transmite então o estado final do multicast ($Comm_M$) para Mh_2, que por sua vez confirma seu recebimento. Em seguida, Mss_3 envia a mensagem `AckSet{`Mh_2`}` para Mss_{init}. Finalmente, após ter recebido as confirmações de todos os *Mh*s pertencentes ao conjunto de destinatários de *M*, Mss_{init} envia a mensagem Del_M para todos os *Mss*s, a fim de que estes apaguem os dados relativos à mensagem M armazenados localmente.

Vale notar que todos os *Mss*s guardam o resultado (estado final) do multicast até que todos os *Mh*s confirmem o recebimento do mesmo. Assim, todos os *Mh*s $\in$ *M.Dest* que eventualmente se tornem ativos depois de um certo período de inatividade sempre serão informados sobre o estado final de cada multicast. A fim de evitar que registros de estados finais permaneçam no sistema devido um *Mh* $\in$ *M.Dest* que tenha deixado o sistema, a mensagem `Leave` faz com o Mss_{Mh} desconsidere o *acknowledgment* do *Mh*, causando assim a difusão da mensagem `Delete`.

Por motivos de simplificação, no exemplo da figura 3.2 apresentamos apenas as mensagens relevantes para o entendimento da situação, bem como abreviamos seus nomes. A tabela 3.1 mostra a equivalência das mensagens do protocolo AM^2C e as mencionadas no exemplo.

No Exemplo	**Na Implementação**
`mcastM`	`MulticastRequested`
`M`	$\texttt{FwdMsgFromMss}_{init}$ ou $\texttt{FwdMsgFromMss}_n$
$\texttt{Ok}_M$	`RespFromMh` ou `RespFromMss`
$\texttt{comm}_M$	$\texttt{StatusFromMss}_{init}$ ou $\texttt{StatusFromMss}_n$
`Ack`	`AckMh`
`AckSet`	`AckMss`
$\texttt{Del}_M$	`Delete`

Hand-off	`Greet`
	`Dereg`
	`DeregAck`

Entrada/Saída do Sistema	`Join` `Leave`

Tabela 3.1: Mensagens do protocolo AM^2C.

3.5 O protocolo de Hand-off

A componente de *hand-off* do AM^2C define as interações entre dois *Mss*s quando um *Mh* migra da célula de Mss_o para a célula de Mss_n. Durante este processo, é essencial que os dois *Mss*s troquem informações sobre o *Mh* migratório consideradas relevantes para o protocolo. Inicialmente descreveremos a estrutura de dados utilizada para a troca de informações entre o Mss_o e o Mss_n. Em seguida, explicaremos em detalhes o funcionamento do *hand-off*, também através de um exemplo.

3.5.1 A Estrutura *h_RECD* no AM^2C

No AM^2C, quando um *Mh* (digamos Mh_{new}) migra, o *Mss* da antiga célula (Mss_o) envia para o *Mss* da nova célula (Mss_n) uma estrutura de dados que está associada ao *Mh*, e que chamamos de *h_RECD*. Esta estrutura é um *array* semelhante ao descrito em [1], que é fundamental para que o Mss_n conheça as mensagens que já foram recebidas por Mh_{new}, ou melhor, que já foram tratadas por algum *Mss*.

Cada elemento *h_RECD*[i] de um *Mh* contém o maior número de seqüência de um multicast iniciado por Mss_i e cuja resposta (ou ausência dela) tenha sido tratada por um dos *Mss*s do sistema. Por exemplo, considere uma mensagem de multicast *M*, cujo *Mss* inicializador seja Mss_i e cujo número de seqüência seja *j*. Quando *M* chega em um *Mss*, para todo *Mh* local e destinatário da mensagem, o *Mss* verifica a condição *h_RECD[*Mss_i*]* $< j$. Se esta for satisfeita, então a mensagem é transmitida para o *Mh*. Caso contrário, significa que o *Mh* já recebeu *M* ou algum outro *Mss* já tratou tal mensagem. Se o *Mh* não for destinatário da mensagem, o *Mss* deve incrementar *h_RECD[*Mss_i*]* para sinalizar aos demais *Mss*s (caso o *Mh* migre) que esta mensagem já foi tratada e que não precisa ser retransmitida para o *Mh*.

Quando um Mss_{Mh} recebe uma resposta de um *Mh* referente à mensagem *M*, este deve incrementar o *h_RECD* na posição correspondente, indicando assim que a mensagem já foi recebida pelo *Mh*. Este mesmo incremento também deve ser realizado para todo *Mh* que não responda com um $\mathtt{Ok}_M/\mathtt{NOk}_M$ dentro do tempo T1 (primeira fase do protocolo), ou quando o *Mss* da nova célula opta por não retransmitir *M* para o *Mh* que tenha acabado de migrar para a sua célula. Este caso será discutido mais adiante.

Notemos que os incrementos na estrutura *h_RECD* são cruciais para que os *Mss*s considerem apenas as mensagens que realmente não foram recebidas e/ou tratadas em relação aos *Mhs* do sistema. Todavia, a estrutura *h_RECD* deve estar sempre sincronizada com as mensagens recebidas pelos *Mhs*, de forma a evitar que um *Mh* destinatário de uma mensagem deixe de recebê-la e que a mensagem seja retransmitida diversas vezes desnecessariamente para o *Mh*. Assim, como a decisão sobre a transmissão de uma mensagem de um *Mss* para um *Mh* é baseada em um número de seqüência presente em *h_RECD*, este não pode ser incrementado mais de uma vez para uma mesma mensagem.

Vale ressaltar ainda que uma implementação da estrutura *h_RECD* da forma como a descrita acima, só garante a entrega de todas as mensagens aos *Mhs* porque as mensagens de um mesmo Mss_{init} são entregues em ordem FIFO para cada *Mss*. Caso isso não fosse garantido, um *Mh* destinatário de um multicast iniciado por um dado *Mss* (digamos Mss_i) poderia deixar de recebê-lo, pois o número de seqüência do multicast não corresponderia necessariamente a *h_RECD*[i] + 1. Por exemplo, suponha que Mss_i difunda duas mensagens de multicast (`M1` e `M2`, nesta ordem), sendo o *Mh* destinatário apenas de `M1`. Se `M2` fosse entregue ao *Mss* da célula onde se encontra o *Mh* antes de `M1`, o *h_RECD* relacionado ao *Mh* seria incrementado (pois $Mh \notin$ `M2.Dest`) e conteria então o valor "1" (supondo que o valor inicial seja "0"). Dessa forma, quando o *Mss* fosse testar se o *Mh* já recebera ou não a mensagem `M1` (verificando se $h\_RECD[\mathrm{i}] < 1$), o *Mss* concluiria que a mensagem teria sido recebida e, consequentemente, não enviaria `M1` para o *Mh*.

3.5.2 O Protocolo de Hand-off no AM^2C

A seguir, são descritos os eventos e ações que definem esta componente do protocolo quando um *Mh* (seja Mh_{new}) migra da célula de Mss_o para a célula de Mss_n. A tabela 3.2 mostra as possíveis seqüências de eventos mais relevantes que podem ocorrer em Mss_o (linhas o_1 a o_4) e Mss_n (linhas n_1 a n_4) durante o processamento concorrente da primeira fase do AM^2C e do *hand-off*. Nesta tabela, o símbolo "$\prec$" denota o predicado *ocorreu antes de* (em tempo real), e as correspondentes ações são descritas no decorrer desta seção.

Linha	Seqüência de Eventos
o_1	`Dereg` $\prec$ `M`
o_2	`M` $\prec$ `Dereg` $\prec$ `M + T1`
o_3	`M` $\prec$ $\texttt{Ok}_M$ $\prec$ `Dereg`
o_4	`M` $\prec$ $\texttt{NOk}_M$ ou `M + T1` $\prec$ `Dereg`
n_1	`Greet` $\prec$ `DeregAck(j)` $\prec$ `M`
n_2	`M` $\prec$ `M + T1` $\prec$ `Greet`
n_3	`Greet` $\prec$ `DeregAck(j)` $\prec$ `M + T1`
n_4	`Greet` $\prec$ `M + T1` $\prec$ `DeregAck(j)`

Tabela 3.2: Seqüência de eventos mais relevantes ocorridas em Mss_o e Mss_n.

1. Depois que Mh_{new} se anuncia a Mss_n (através da mensagem `Greet`), este envia a mensagem `Dereg` para Mss_o, solicitando o desvinculamento de Mh_{new} da célula de Mss_o.

2. Ao receber a mensagem `Dereg`, Mss_o imediatamente remove Mh_{new} do grupo `LocalMhs` para então enviar uma confirmação para Mss_n (mensagem `DeregAck`). Neste momento, quatro situações distintas podem ocorrer. Para isto, seja M uma mensagem de multicast difundida por um determinado *Mss* inicializador (Mss_{init}), cujo número de seqüência seja i e, por simplificação, que tenha Mh_{new} como único destinatário. Considere também que a última mensagem recebida por Mh_{new} e iniciada por Mss_{init} foi i - 1.

   (a) Caso a chegada de M em Mss_o ocorra depois da chegada da mensagem `Dereg` (linha o_1 da tabela), então Mss_o não envia M para Mh_{new} (note que este não pertence mais ao grupo `LocalMhs`), responde com um $\texttt{Ok}_M$ para Mss_{init} e envia a mensagem `DeregAck(i-1)` para Mss_n confirmando o desvinculamento de Mh_{new} de sua antiga célula. O parâmetro i - 1 indica, neste caso, que este é o número de seqüência da última mensagem recebida por Mh_{new} (e iniciada por Mss_{init}).

(b) Se `Dereg` chega depois de M mas antes do *time-out* desta ter ocorrido no *Mss* (linha o_2), então, caso M já tenha sido enviada para Mh_{new} (situação em que Mh_{new} migra sem responder), Mss_o responde com um $\texttt{NOk}_M$ para Mss_{init} e envia a mensagem `DeregAck(i)` para Mss_n. Ou seja, diante da incerteza se Mss_n vai retransmitir M para Mh_{new}, Mss_o opta por indiretamente abortar o multicast (isto é, Mss_o trata a chegada de `Dereg` como se fosse um $\texttt{NOk}_M$ de Mh_{new}).

(c) Quando a mensagem `Dereg` chega em Mss_o depois da chegada de M, bem como depois da resposta de Mh_{new} (linha o_3), Mss_o simplesmente responde com um $\texttt{Ok}_M$ para Mss_{init} (considerando que a resposta de Mh_{new} foi $\texttt{Ok}_M$) e envia `DeregAck(i)` para Mss_n. Neste caso, a resposta do *Mh* causa o incremento de *h_RECD[*Mss_{init} *]*.

(d) Caso Mh_{new} tenha respondido com um $\texttt{NOk}_M$ ou caso `Dereg` chegue em Mss_o após o *time-out* de M(linha o_4), então Mss_o responde a Mss_{init} com um $\texttt{NOk}_M$ e envia a mensagem `DeregAck(i)` para Mss_n. O incremento de *h_RECD[*Mss_{init} *]* neste caso ocorre ou devido à resposta de Mh_{new}, ou devido à ocorrência do *time-out*, como descrito na subseção anterior.

3. Quando a mensagem `DeregAck` chega em Mss_n, a responsabilidade pelo novo *Mh* é oficialmente passada para Mss_n, o qual acrescenta Mh_{new} em seu grupo `LocalMhs`. Baseado no número de seqüência contido em *h_RECD*, Mss_n decide então o que fazer em relação às mensagens multicast que têm Mh_{new} como destinatário e que ainda não foram confirmadas. Existem quatro possibilidades. Considere M uma tal mensagem como a descrita no item 2. Vamos supor que o valor presente em *h_RECD[*Mss_{init} *]* seja j (proveniente da mensagem `DeregAck(j)`).

   (a) Caso a mensagem M chegue em Mss_n após a chegada das mensagens `Greet` e `DeregAck(j)` (linha n_1 da tabela) então, se $j < i$, isto é, Mh_{new} ainda não recebeu a mensagem, Mss_n envia M para Mh_{new} e responde com um $\texttt{Ok}_M/\texttt{NOk}_M$ para Mss_{init} (a depender da resposta de Mh_{new}). Note que esta é uma situação onde há um atraso na chegada da mensagem M em Mss_n.

(b) Se a mensagem `Greet` chega em Mss_n após o *time-out* de M (linha n_2), nesta altura Mss_n já enviou sua resposta a Mss_{init}. Assim, Mss_n não envia M para Mh_{new} (e não tomará nenhuma ação até a chegada da mensagem `DeregAck`).

(c) Quando a mensagem `DeregAck(j)` chega em Mss_n antes do *time-out* T1 de M ocorrer (linha n_3), então se $j < i$, a mensagem é retransmitida para Mh_{new} e o *time-out* é prorrogado. A depender da resposta de Mh_{new}, Mss_n responde com um $\mathtt{Ok}_M/\mathtt{NOk}_M$ para Mss_{init}.

(d) Caso a mensagem `DeregAck(j)` chegue após o *time-out* T1 de M, então, se $j < i$ (Mh_{new} ainda não recebeu M), Mss_n envia um $\mathtt{NOk}_M$ para Mss_{init}, abortando indiretamente o multicast. Nesta caso, o Mss_n trata a mensagem para Mh_{new} (incremento de h_RECD), considerando sua resposta como `NOk`. Nas situações (a) e (c), o h_RECD é incrementado no momento da resposta de Mh_{new}.

Portanto, a componente *hand-off* do AM^2C estipula que enquanto o *hand-off* não é completado, Mss_o permanece responsável pelo envio de respostas (para o Mss_{init} correspondente) das mensagens já recebidas. Como Mss_o não pode garantir que Mss_n irá receber a mensagem `DeregAck` a tempo de permitir uma retransmissão da mensagem para Mh_{new}, então é mais seguro abortar indiretamente o multicast (através do envio de um $\mathtt{NOk}_M$ para Mss_{init}). Por outro lado, Mss_n não exerce nenhuma influência sobre o estado final de um multicast para qualquer mensagem que já tenha sido tratada por Mss_o.

É importante observar que assim o protocolo *hand-off* garante que a propriedade de atomicidade requerida pelo protocolo AM^2C é sempre satisfeita. Um multicast só pode ser confirmado garantidamente se todos os destinatários receberem a mensagem. Por isso, se há alguma incerteza quanto ao recebimento da mensagem por alguns desses elementos, então é preferível abortar o multicast.

Na Figura 3.3 temos um exemplo do funcionamento da componente *hand-off* do AM^2C quando um Mh (Mh_{new}) migra sem responder a Mss_o e a mensagem `Dereg` chega neste antes de M+T1 (linha o_2 da tabela 3.2), onde M corresponde a uma mensagem de multicast difundida por um dado Mss inicializador (Mss_{init}). Neste caso, Mss_o responde com um `NOk` para Mss_{init} e envia a mensagem `DeregAck` para Mss_n informando-o, via h_RECD, que já tratou M para Mh_{new}. Em Mss_n a mensagem `Greet` chega antes de M+T1. Mss_n envia então a mensagem `Dereg` para Mss_o e fica aguardando pela chegada da mensagem `DeregAck` ou pela ocorrência do *time-out* T1, para então decidir o que fazer. Quando `DeregAck` chega em Mss_n (antes de M+T1), então através de

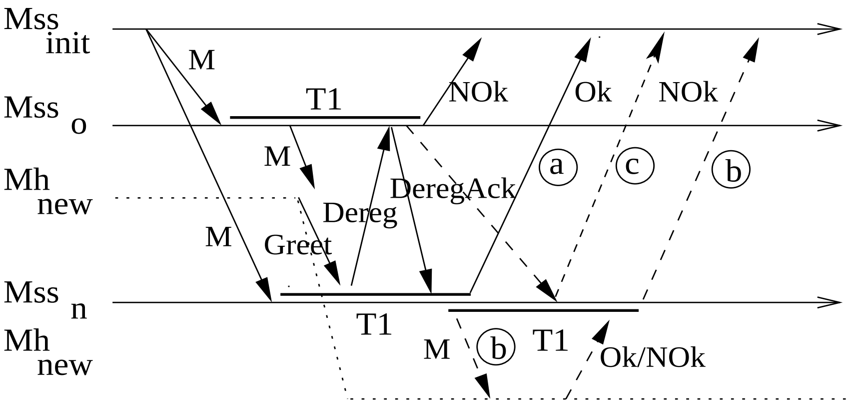

Figura 3.3: Exemplo de Hand-off no AM^2C

h_RECD Mss_n percebe que Mss_o já enviou uma resposta para Mss_{init}. Neste caso (linhas sólidas), após o *time-out* Mss_n responde para Mss_{init} sem considerar Mh_{new} (caso a). Entretanto, caso Mss_o não tivesse tratado a mensagem M (linha pontilhada), então Mss_n retransmitiria M para Mh_{new}, prorrogaria o *time-out* T1, e aguardaria a resposta do *Mh*. Após o *time-out*, Mss_n responderia a Mss_{init} baseado nesta resposta (caso b). Esta situação, assim como o caso (a), correspondem à linha n_3 da tabela, onde j=i e j<i, respectivamente (sendo i o número de seqüência de M). Se a mensagem `DeregAck` chegasse em Mss_n depois de ocorrido o *time-out*, então Mss_n enviaria um `NOk` para Mss_{init} (caso (c) e linha n_4 da tabela).

3.6 Limitações do AM^2C

O principal problema do protocolo AM^2C é sua falta de escalabilidade em relação ao número de *Mss*s no sistema. Não apenas mensagens de multicast e de estado de um multicast (além da mensagem `Delete`) são transmitidas do *Mss* inicializador (Mss_{init}) para todos os *Mss*s do sistema, mas também todos os *Mss*s enviam respostas para Mss_{init} nas duas primeiras fases do protocolo. Dessa forma, o número de mensagens trocadas na rede fixa pode ser grande e tornar o protocolo inviável para uma rede com um número grande de *Mss*s. Uma estimativa do número de mensagens transmitidas pelo AM^2C é apresentada na seção 4.7.

Além disto, o AM^2C causa um certo disperdício de memória, pois durante a primeira fase do protocolo todos os *Mss*s precisam manter armazenada a mensagem de multicast, mesmo que não haja *Mh* na célula do *Mss* que pertença ao grupo de destinatários da mensagem.

A fim de solucionar os problemas mencionados acima, projetamos uma variante do protocolo AM^2C, que é descrito no capítulo seguinte.

Capítulo 4

Uma Variante: o iAM^2C

4.1 Introdução

Este capítulo descreve o *Indirect* AM^2C (ou iAM^2C [10]), um protocolo para multicast atômico que é uma versão com endereçamento indireto do protocolo AM^2C [13] e que utiliza um gerenciamento hierárquico de localização de unidades móveis [25] para difundir mensagens na rede fixa. Ao contrário do protocolo original, o iAM^2C apresenta boa escalabilidade com relação ao número de *Mss*s.

4.2 Motivação

A principal motivação para o desenvolvimento do iAM^2C foi obter um protocolo que realize multicast atômico sem as limitações intrínsecas ao AM^2C discutidas na seção 3.6.

4.3 Modelo de Sistema e Suposições

O modelo de sistema consiste das máquinas descritas na seção 3.2, incluindo um terceiro elemento denominado servidor intermediário de mensagens multicast (*Intermediate Multicast Server*, ou *IMS*s).

Neste novo modelo, cada *Mss* está associado a um único *IMS*, e o conjunto de *Mss*s associados a um *IMS* é denominado o *domínio* do *IMS*. Assumimos que a intersecção dos domínios dos *IMSs* é vazia, e que a união dos domínios corresponde ao conjunto composto por todos os *Mss*s do sistema.

Um *IMS* é um servidor de mensagens que pode executar em qualquer *host* fixo da rede e que armazena as mensagens multicast durante o processo de difusão. Além disto, cada *IMS* mantém a informação sobre o conjunto `LocalMhs` para cada um dos *Mss*s de seu domínio, que chamamos de mapa de localização de *Mhs* nos *Mss*s do domínio (ou simplesmente, `LocMap`). Assim, para cada destinatário de um multicast, um *IMS* pode deduzir para quais *Mss*s deve transmitir uma mensagem de multicast.

A idéia central do protocolo iAM^2C é reduzir o número de elementos que armazenam as mensagens multicast a serem transmitidas para os *Mhs*. Ao invés de usar todos os *Mss*s como repositórios das mensagens multicast, o iAM^2C utiliza os *IMSs* para este propósito.

Os *IMSs* desempenham um papel semelhante ao dos *Mss*s no protocolo AM^2C. A principal diferença está na forma como os *IMSs* difundem uma mensagem na rede. Seja M uma mensagem para um grupo de *Mhs* destinatários. Em vez de difundir M para todos os *Mss*s, difunde-se M para todos os *IMSs* (que formam um conjunto supostamente menor do que o dos *Mss*s). Os *IMSs* por sua vez irão retransmitir M apenas para aqueles *Mss*s de seu domínio que possuam algum $Mh \in (M.Dest \cap Mss.$`LocalMhs`).

A figura 4.1 compara a maneira utilizada pelos protocolos AM^2C e iAM^2C para difundir uma mensagem. Considere um sistema composto por dezessete *Mss*s, e que um dado *Mh* solicitou um multicast de uma mensagem M para seu *Mss* responsável (Mss_0). Suponha ainda que apenas quatro do conjunto de todos os *Mss*s possuem algum *Mh* destinatário. Desta forma, no iAM^2C M é enviada apenas para os *Mss*s 3, 6, 12 e 15 (através do *IMS*), enquanto que no AM^2C todos os *Mss*s recebem a mensagem. Nitidamente, uma conseqüência destes fatos é o tráfego gerado na rede fixa em função da forma de difusão usada por cada protocolo. É importante ressaltar ainda que esse mesmo número de mensagens seria gerado também na segunda e na terceira fases do protocolo AM^2C (com exceção da última fase, onde não há respostas dos *Mss*s).

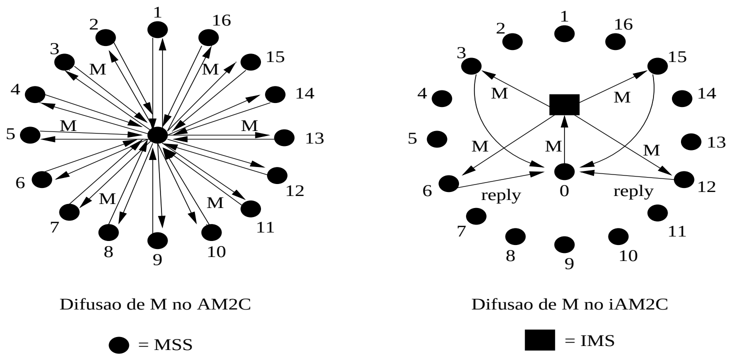

Figura 4.1: Comparação entre os protocolos AM^2C e iAM^2C.

A informação sobre a atual localização de um dado *Mh* é atualizada pelos *Mss*s (através da mensagem `LocationUpdate`) a cada vez que um *hand-off* é completado (vide seção 4.6). Esta mesma mensagem faz também o papel de requisição de retransmissão de todas as mensagens de multicast "pendentes", e que tenham um determinado *Mh* migrado na lista dos destinatários.

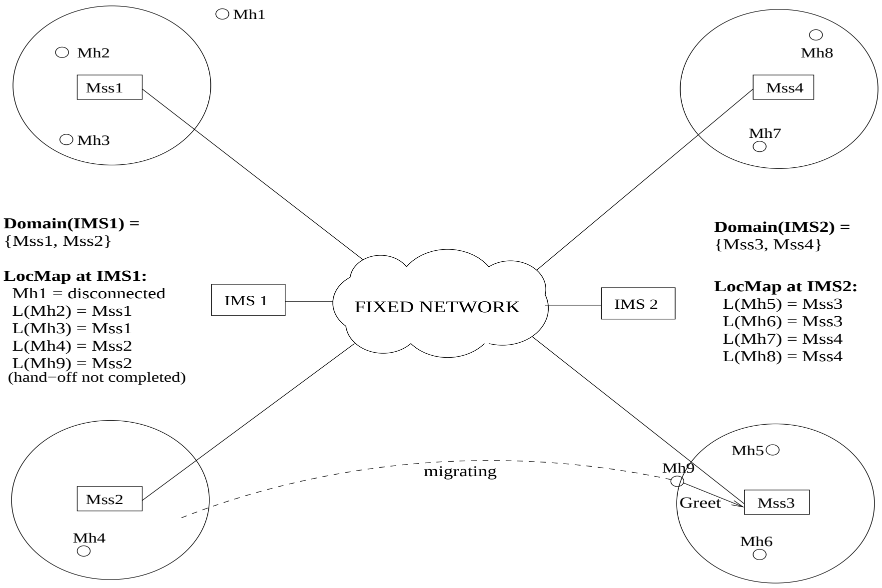

Figura 4.2: Modelo de Sistema no Protocolo iAM^2C.

A Figura 4.2 exemplifica um sistema composto por quatro *Mss*s, nove *Mhs* e dois *IMSs*, onde *Mss1* e *Mss2* estão no domínio de *IMS1*, e *Mss3* e *Mss4* estão no domínio de *IMS2*. Como mostrado, cada *IMS* mantém um `LocMap` dos *Mhs* presentes nas células dos *Mss*s que pertencem ao seu domínio.

As suposições do modelo do iAM^2C são basicamente as mesmas do modelo do protocolo original. Dessa forma, as suposições já comentadas anteriormente[1] serão apenas citadas e indicadas com um (*). As demais referem-se especificamente ao protocolo iAM^2C. São elas:

1. *Mss*s e *IMSs* não falham e todos os *hosts* da rede fixa estão conectados;

2. Cada *Mss* está associado a um único *IMS*;

3. Seja S o conjunto de todos os *Mss*s do sistema. Seja D_i o conjunto de *Mss*s pertencentes ao domínio do *IMSi* ($1 \leq i \leq n$, onde n = número de *IMSs* no sistema). Temos que ($D_1 \cup D_2 \cup$ ... $\cup D_n) = S$, e ($D_1 \cap D_2 \cap$ ... $\cap D_n) = \emptyset$.

[1] Para maiores detalhes consultar a seção 3.2 do Capítulo 3.

4. É assumido que os *hosts* fixos possuem recursos físicos suficientes para armazenamento e processamento (*);

5. A comunicação na rede fixa é confiável(*);

6. A comunicação sem fio não é confiável, mas é possível detectar uma perda de mensagem (*);

7. Cada Mh possui uma identificação única no sistema que independe de sua localização. Antes de deixar o sistema um Mh envia uma mensagem `Leave` para seu Mss_{Mh} (*);

8. Em um dado momento um Mh está associado a um único Mss (*);

9. Se um Mh está ativo, este responde a qualquer mensagem enviada por Mss_{Mh}. Caso contrário, ele não responde a nenhuma mensagem(*);

10. Depois de ter se anunciado a um novo Mss (Mss_n) um Mh não interage com nenhum outro Mss que não seja Mss_n(*);

4.4 Descrição do Protocolo

Assim como no protocolo AM^2C, o iAM^2C é também um protocolo 2PC com uma etapa adicional de coleta de lixo. A seguir, serão descritos os principais eventos e ações em cada uma das fases:

- **FASE I**:

  Um *host* móvel, Mh_S, envia para o seu Mss_{Mh_S} uma requisição de multicast de uma mensagem M para os destinatários mencionados em $M.Dest$. O Mss_{Mh_S}, que doravante chamaremos de Mss_{init}, faz portanto o papel de inicializador do 2PC para esta requisição.

  Mss_{init} difunde a nova requisição para `todos` os $IMSs$ do sistema. Cada IMS por sua vez, armazena M em um *buffer* local e a retransmite para todos $Msss$ de seu domínio que tenham algum Mh local pertencente a $M.Dest$. Cada Mss que recebe M a retransmite para todo $Mh \in$ ($M.Dest \cap Mss$.`LocalMhs`) e espera por T1 unidades de tempo por respostas dos Mhs. O Mss acrescenta em seu *log* a identificação da mensagem, indicando que a mesma já foi repassada para os Mhs.

  Quando ocorre o *time-out* T1, o Mss envia uma mensagem `OkSet` para Mss_{init}, a qual contém a informação sobre o conjunto de Mhs que responderam à mensagem M (com suas respectivas respostas). Ou seja, a mensagem `OkSet` é um conjunto de pares (Mh_i, $resp_i$), onde $resp_i =$

$\{\mathtt{Ok}_M, \mathtt{NOk}_M\}$. Se um *Mh* não responder à mensagem antes do *time-out* T1, o *Mss* considera sua resposta como $\mathtt{NOk}_M$.

Quando um $Mh \in M.Dest$ recebe M do seu Mss_{Mh}, este armazena a mensagem em uma área de trabalho temporária e imediatamente responde ao Mss_{Mh} com um $\mathtt{Ok}_M$ ou $\mathtt{NOk}_M$, dependendo se a mensagem M pode ou não ser processada localmente pela aplicação executada no *Mh*.

Caso um *Mss* se torne responsável por um novo *Mh*, digamos Mh_{new}, que tenha migrado para sua célula, então o *Mss* envia a mensagem `LocationUpdate` para o *IMS* correspondente a fim de que este retransmita todas as mensagens para as quais Mh_{new} seja destinatário e ainda não as tenha recebido. Quando uma mensagem retransmitida chega no *Mss*, a depender da situação a mensagem pode ser enviada para o *Mh* e o *time-out* T1 prorrogado. Este caso é detalhado na seção 4.6.

Após receber as respostas referentes a todos os $Mhs \in M.Dest$, Mss_{init} verifica se todas as respostas recebidas foram $\mathtt{Ok}_M$ ou se alguma delas foi $\mathtt{NOk}_M$. No primeiro caso, o estado final do multicast é *Confirmado (Commited)* e caso contrário, *Abortado (Aborted)*.

- **FASE II**:

  De forma similar à Fase I, Mss_{init} difunde agora o estado do multicast M para todos os *IMSs* do sistema, os quais retransmitem a mensagem de *status* (*Commited* ou *Aborted*) somente para aqueles *Mss*s de seus respectivos domínios que tenham algum *Mh* local pertencendo a *M.Dest*. Cada *IMS* também armazena o estado final de M em *Outcomes*.

  Uma vez recebida a mensagem de *status*, um *Mss* a retransmite para todo $Mh \in (M.Dest \cap Mss.\mathtt{LocalMhs})$ e espera T2 unidades de tempo pela confirmação do recebimento da mensagem de *status* por parte dos *Mhs*. Seja `C` o conjunto de *Mhs* que responderam. Ocorrido o *time-out* T2, o *Mss* envia uma mensagem `AckSet` com o conjunto `C` de volta a Mss_{init}.

  Quando o *Mh* recebe o estado *Commited*, a mensagem M correspondente (armazenada temporariamente numa área de trabalho) é entregue à aplicação, e caso contrário é descartada.

  Assim como ocorre na Fase I, quando um *Mh* migra para a célula de um *Mss* a mensagem `LocationUpdate` é enviada para o *IMS* correspondente, o qual retransmite para o *Mss* os estados finais de todos os multicasts ainda não finalizados e que têm o *Mh* como destinatário. O *Mss* por sua vez, repassa os estados finais para o *Mh*, independente se o mesmo já os recebeu anteriormente. Neste caso, o *Mss* poderá aguardar por até mais T2 unidades de tempo pela confirmação do recebimento de cada estado pelo *Mh*. Se esta confirmação chegar à tempo, o *Mss* enviará a mensagem `AckSet{Mh}` para Mss_{init} e, caso contrário, a mensagem não será enviada.

- **FASE III**:

  Cada vez que Mss_{init} recebe uma mensagem `AckSet{C}` referente a um multicast *M*, este adiciona os elementos contidos em `C` a uma lista *M.Confirmados*. Assim que *M.Confirmados* equivale a *M.Dest*, então Mss_{init} difunde uma mensagem $\mathtt{Del}_M$ para todos os *IMSs* a fim de que estes removam o estado final de *M* armazenado localmente. Ao receber tal mensagem, cada *IMS* executa a operação solicitada.

A fim de evitar que o iAM^2C sofra do problema de um multicast ficar bloqueado indefinidamente devido à falta de resposta de um $Mh \in M.Dest$, seja na Fase I ou II, define-se um tempo máximo de execução do protocolo. Isto é, depois de esgotado um tempo de espera T3 (onde T3 é bem maior do que max(T1, T2)) o Mss_{init} decide abortar o multicast ou enviar a mensagem $\mathtt{Del}_M$ para cada *IMS*, a fim de completar a Fase I ou a Fase II, respectivamente. Ao mesmo tempo, o Mss_{init} difunde uma mensagem anunciando a exclusão do(s) *Mh*(s) não disponível(eis) no sistema.

Vale observar ainda que os *Mss*s não armazenam as mensagens de multicast *M* ou os estados finais, mas apenas repassam *M* (ou estado final) recebida do *IMS*, e simplesmente indicam no seus respectivos *logs* que a mensagem foi enviada. Sempre que se faz necessária uma retransmissão de uma mensagem para um *Mh*, o *Mss* correspondente tem que solicitar o reenvio de tal mensagem ao seu *IMS*.

4.5 Exemplo de Funcionamento do iAM^2C

Na Figura 4.3 são ilustrados os tipos de mensagens transmitidas pelo protocolo iAM^2C em um cenário composto por dois *Mhs*, três *Msss*, e um *IMS*. Neste caso, estamos considerando um único domínio (do IMS_1) composto pelos três *Mss*s. Inicialmente, tanto Mh_1 como Mh_2 estão localizados na célula de Mss_1. Para simplificar, vamos supor que um dado *Mh_S* já tenha enviado uma requisição de multicast de uma mensagem *M* para seu *Mss* responsável (Mss_{init}), e que os destinatários de *M* sejam apenas Mh_1 e Mh_2.

Ao receber a mensagem, Mss_{init} difunde *M* para todos os *IMSs* do sistema (neste caso apenas um). Quando recebe a mensagem, IMS_1 a armazena em um *buffer* local e verifica, de acordo com o

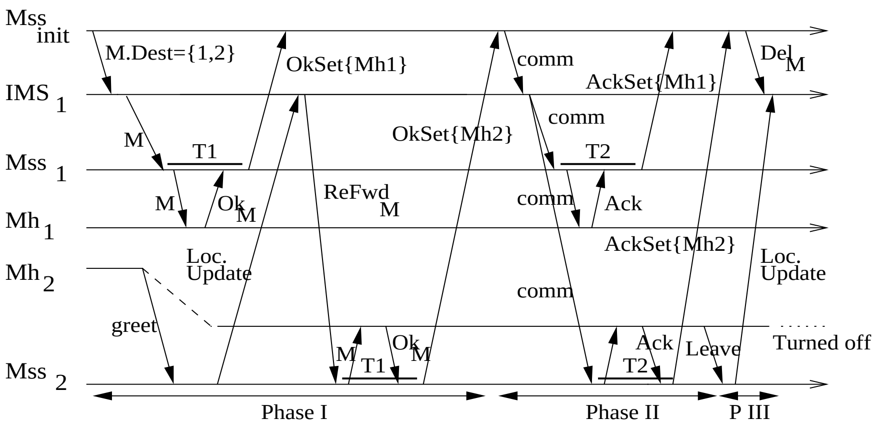

Figura 4.3: Exemplo de Funcionamento do protocolo iAM^2C

`LocMap`, quais são os *Mss*s que possuem algum $Mh \in M.Dest$. Dessa forma, M é enviada somente para Mss_1. Uma vez recebida, Mss_1 envia a mensagem apenas para Mh_1, pois Mh_2 está em processo de migração. Mh_1 responde com um $\texttt{Ok}_M$ para Mss_1. Após decorrido o tempo T1, Mss_1 então envia a mensagem $\texttt{OkSet\{Mh}_1\texttt{\}}$ para Mss_{init} confirmando a aceitação de M por parte de Mh_1 (a mensagem `OkSet` sempre carrega a resposta do Mh, $\texttt{Ok}_M/\texttt{NOk}_M$, que, neste exemplo, consideramos o primeiro caso).

Quando o *hand-off* é completado na célula de Mss_2, este envia a mensagem `LocationUpdate` para IMS_1, informando-o sobre a nova localização de Mh_2 e sobre a última mensagem de Mss_{init} repassada a Mh_2. O *IMS* por sua vez, atualiza o `LocMap` ($\texttt{LocMap(}Mh_2\texttt{)} = Mss_2$) e retransmite M para Mss_2, já que Mh_2 pertence a $M.Dest$ e não recebeu a mensagem. Mss_2 então envia a mensagem para Mh_2, o qual responde com um $\texttt{Ok}_M$ antes de T1 se esgotar. Passadas T1 unidades de tempo, Mss_2 então envia a mensagem $\texttt{OkSet\{Mh}_2\texttt{\}}$ para Mss_{init} informando a aceitação de M pelo Mh_2.

Depois de receber as respostas de todos os $Mhs \in M.Dest$, Mss_{init} verifica que todas são $\texttt{Ok}_M$ e, dessa forma, define o estado final de M como *Confirmado (Commited)*, e envia este para IMS_1 e para o Mh requisitante do multicast. IMS_1 por sua vez envia o estado final para Mss_1 e Mss_2, já que agora ambos os *Mss*s possuem um Mh pertencente ao grupo de destinatários da mensagem original. Mss_1 e Mss_2 enviam o estado final do multicast para Mh_1 e Mh_2, respectivamente, que confirmam o recebimento do mesmo antes de T2 se esgotar. Passado o tempo T2, Mss_1 e Mss_2 enviam as mensagens $\texttt{AckSet\{Mh}_1\texttt{\}}$ e $\texttt{AckSet\{Mh}_2\texttt{\}}$, respectivamente, para Mss_{init}, informando sobre a confirmação do recebimento do estado final de M por parte de Mh_1 e Mh_2.

Mss_{init} então envia a mensagem $\texttt{Del}_M$ para IMS_1 a fim de que este remova a mensagem M

armazenada localmente. Esta operação é realizada em qualquer *IMS* que faça parte do sistema. Finalmente, antes que Mh_2 deixe o sistema, este envia a mensagem `Leave` para Mss_2, que por sua vez envia a mensagem `LocationUpdate` para IMS_1, o qual atualizará a informação de que Mh_2 não se encontra mais no sistema.

De forma similar à seção 3.4, a tabela 4.1 mostra a equivalência das mensagens do protocolo iAM^2C e as mencionadas no exemplo.

No Exemplo	**Na Implementação**
`M.Dest={1,2}`	`FwdMsgFromMssInit`
`M`	`FwdMsgFromIMS` ou `FwdMsgFromMss`
$\texttt{Ok}_M$	`RespFromMh`
`OkSet`	`RespFromMss`
`LocUpdate`	`LocationUpdate`
$\texttt{ReFwd}_M$	`ReFwdMsgFromIMS`
$\texttt{comm}_M$	`StatusFromMssInit` ou `StatusFromIMS` ou `StatusFromMss`
`Ack`	`AckMh`
`AckSet`	`AckMss`
$\texttt{Del}_M$	`Delete`
Requisição Inicial	`MulticastRequested`
Hand-off	`Greet`, `Dereg` e `DeregAck`
Entrada/Saída Sistema	`Join` / `Leave`

Tabela 4.1: Mensagens do protocolo iAM^2C.

4.6 Hand-off no iAM^2C

Esta seção tem o objetivo de apresentar a componente de *hand-off* no protocolo iAM^2C. Todavia, antes de entrarmos nos detalhes relativos ao *hand-off*, é importante apresentar a estrutura de dados *h_RECD* utilizada no iAM^2C, que é responsável pela transferência de informações relevantes sobre um *Mh* entre os *Mss*s envolvidos em uma migração do *Mh*.

4.6.1 ***h_RECD* no *iAM²C***

A estrutura *h_RECD* no *iAM²C* é um pouco diferente da utilizada no *AM²C*. A razão desta diferença é a ausência, no *iAM²C*, da garantia da entrega de mensagens em ordem FIFO e somente uma vez (como no MCAST [1]) para os *Mhs* destinatários. Estes fatos são decorrentes do acréscimo de mais um nível de indireção na entrega de multicasts com a introdução dos *IMSs*. Devido à ausência destas garantias, uma implementação do *h_RECD* como no *AM²C* impossibilitaria a entrega confiável de multicasts aos *Mhs*. Os detalhes de como o acréscimo de mais um nível de indireção no *iAM²C* implica nos fatos mencionados anteriormente, bem como as conseqüências destes, caso fosse utilizada a mesma implementação do *AM²C*, são explicados na seção 6.2.

Assim como no *AM²C*, cada *Mh* do sistema possui um *array h_RECD* associado, e que é mantido pelo *Mss* responsável pelo *Mh*. A diferença entre o *h_RECD* do *AM²C* e do *iAM²C* é que, neste último, cada posição *h_RECD*[i] em vez de conter um número de seqüência, contém uma lista com os números de seqüência de todas as mensagens de multicast iniciados por Mss_i e já recebidas pelo *Mh*.

Seja M uma mensagem de multicast difundida por Mss_i, cujo número de seqüência é j. Quando um *Mss* recebe *M*, a mensagem só será enviada para um *Mh* destinatário se $j \notin h\_RECD[Mss_i]$. Sempre que um *Mh* responde a uma tal mensagem, o número de seqüência desta é então inserido em $h\_RECD[Mss_i]$. Se o *Mh* não é destinatário de uma mensagem, nenhuma atualização é feita em seu *h_RECD*.

Como também acontece no *AM²C*, quando um *Mh* não responde a M dentro do período de T1 unidades de tempo, j é acrescentado em $h\_RECD[Mss_i]$ para indicar que o *Mss* responsável pelo *Mh* já tratou a mensagem (no caso, considerou a resposta do *Mh* como $\mathtt{NOk}_M$). O mesmo ocorre quando o *Mss* da nova célula decide não retransmitir uma mensagem para um *Mh* migratório.

Para evitar que a lista presente em $h\_RECD[Mss_i]$ (para qualquer Mss_i do sistema) cresça indefinidamente com o envio de novos multicasts, faz-se necessária uma "coleta de lixo". Essa coleta é feita sempre que uma mensagem contendo o estado final de um multicast (mensagem de *status*) é transmitida pelo *IMS* aos seus *Mss*s. Assim, além de remover uma mensagem *M* do *buffer* de um *IMS*, elimina-se também o número de seqüência correspondente em $h\_RECD[Mss_i]$, já que o mesmo não será mais usado para verificar se um *Mh* recebeu ou não a mensagem. Como a mensagem de *status* é difundida para qualquer *Mss* que possua um $Mh \in M.Dest$, e como só existe uma única cópia de *h_RECD* por *Mh*, então certamente o número de seqüência de *M* será removido de todos os *h_RECD*s dos *Mhs* destinatários.

4.6.2 O Protocolo de Hand-off

Nesta seção descreveremos o *hand-off* no iAM^2C, ressaltando as principais diferenças com relação ao *hand-off* no AM^2C. Estamos assumindo que um dado *Mh* (Mh_{new}) está em processo de migração da célula de Mss_o para a célula de Mss_n, e que este já enviou a mensagem `Dereg` para Mss_o solicitando o desvinculamento de Mh_{new} da sua antiga célula. A tabela 4.2 mostra a seqüência de eventos mais importantes que ocorrem em Mss_o e Mss_n durante o processo de *hand-off*. As ações tomadas pelos elementos diante da chegada destes eventos são descritas no decorrer desta seção.

Linha	**Seqüência de Eventos**
o_1	`Dereg` $\prec$ `M`
o_2	`M` $\prec$ `Dereg` $\prec$ `M + T1`
o_3	`M` $\prec$ $\mathtt{Ok}_M$ $\prec$ `Dereg`
o_4	`M` $\prec$ $\mathtt{NOk}_M$ ou `M + T1` $\prec$ `Dereg`
n_1	`ReFwd(Pendentes,Finais)` $\prec$ `M`
n_2	`M` $\prec$ `ReFwd(Pendentes,Finais)` $\prec$ `M + T1`
n_3	`M + T1` $\prec$ `ReFwd(Pendentes,Finais)`

Tabela 4.2: Seqüência de eventos mais relevantes ocorridas em Mss_o e Mss_n.

1. O recebimento da mensagem `Dereg` por parte de Mss_o e o consequente envio da mensagem `DeregAck` para Mss_n ocorre em situações idênticas às descritas na seção 3.5. A diferença está no tratamento dado em relação à estrutura *h_RECD* (que é parâmetro da mensagem `DeregAck`), e que apresentamos a seguir. Para isso, seja M uma mensagem de multicast difundida por um dado *Mss* inicializador (Mss_{init}), cujo número de seqüência seja j.

    (a) linha o_1 da tabela: Mss_o envia a mensagem `DeregAck` para Mss_n sem incluir j em *h_RECD[*Mss_{init} *]*;

    (b) linha o_2: se Mss_o já enviou M para Mh_{new}, o qual ainda não respondeu, então Mss_o insere j em *h_RECD[*Mss_{init} *]* e considera a resposta de Mh_{new} como $\mathtt{NOk}_M$. Mss_o envia então a mensagem `DeregAck` para Mss_n;

(c) linha o_3: neste caso, o *h_RECD* incluído na mensagem `DeregAck` contém o número de seqüência j de M, uma vez que tal inclusão é feita no momento da chegada da resposta de Mh_{new};

(d) linha o_4: a mensagem `DeregAck` é enviada com o número de seqüência de M incluído em *h_RECD*, visto que em ambos os casos j é inserido em *h_RECD[*Mss$_{init}$ *]*;

Se Mss_o e Mss_n pertencem a domínios diferentes (i.e., o *IMS* ao qual Mss_o está associado não é o mesmo *IMS* que Mss_n está associado), Mss_o também envia a mensagem `LocationUpdate` para o *IMS* responsável pelo seu domínio, o qual atualizará a informação de que Mh_{new} não mais se encontra em nenhuma célula dos *Mss*s de seu domínio.

2. Ao receber a mensagem `DeregAck` Mss_n oficializa Mh_{new} como parte de seu conjunto `LocalMhs` e armazena o *h_RECD* recebido na mensagem. Mss_n envia então a mensagem `LocationUpdate` para o *IMS* correspondente, incluindo na mensagem o *h_RECD*. Esta mensagem fará o papel de solicitação de atualização de `LocMap` e de reenvio de mensagens pendentes.

3. Na chegada da mensagem `LocationUpdate` o *IMS* atualiza então seu `LocMap`, o qual indicará agora que Mh_{new} encontra-se na célula de Mss_n. Para toda mensagem M armazenada em seu *buffer* e que tenha Mh_{new} como destinatário, o *IMS* verifica se o número de seqüência de M está presente no *h_RECD* transmitido na mensagem recebida. Em caso negativo (ou seja, Mh_{new} não recebeu a mensagem), o *IMS* acrescenta M em uma lista `Pendentes`. Da mesma forma, o *IMS* acrescenta em uma lista `Finais` todos os estados finais dos multicasts que tenham Mh_{new} como destinatário. A mensagem `ReFwdMsgFromIMS(Pendentes,Finais)` é então enviada para Mss_n.

4. Quando Mss_n recebe `ReFwdMsgFromIMS(Pendentes,Finais)` do *IMS*, para cada mensagem $M \in$ `Pendentes`, três situações podem ocorrer. Seja M uma mensagem de multicast difundida por Mss_{init} e com o número de seqüência $= j$. Note-se que M pode chegar em Mss_n no intervalo de tempo entre o envio do `LocationUpdate` e a chegada de `ReFwdMsgFromIMS(Pendentes,Finais)` (por exemplo, pode existir algum outro *Mh* local e pertencente a *M.Dest*). Os possíveis casos e correspondentes ações são:

   (a) A chegada de `ReFwdMsgFromIMS` ocorre antes da chegada de M (linha n_1 da tabela): Mss_n envia M para Mh_{new} e inicia o *timer* com *time-out* T1, aguardando por uma resposta de Mh_{new};

(b) `ReFwdMsgFromIMS` chega em Mss_n após a chegada de M, mas antes da ocorrência do seu *time-out* (linha n_2): neste caso, duas ações podem ser tomadas, de acordo com a política de retransmissão de mensagens adotada. Estas políticas são baseadas no *log* do *Mss*, e podem ser:

P1) *Envio condicional*: M só é enviada caso ainda não tenha sido transmitida para nenhum outro $Mh \in$ `LocalMhs` $\cap$ $M.Dest$ (o que é verificado pelo *Mss* através da consulta a seu *log*). Na situação (b), assume-se que a resposta do novo *Mh* é $\texttt{NOk}_M$ e o Mss_n acrescenta j em $h\_RECD[\text{Mss}_{init}]$;

P2) *Envio incondicional*: a mensagem é enviada para o novo *Mh*, e o tempo de espera é estendido por mais T1 unidades de tempo.

(c) A mensagem `ReFwdMsgFromIMS` chega em Mss_n após o *time-out* de M (linha n_3): como de certa forma M já foi descartada por Mss_n, este responde a Mss_{init} considerando a resposta de Mh_{new} como $\texttt{NOk}_M$. O número de seqüência de M é incluído em $h\_RECD[\text{Mss}_{init}]$;

Em relação aos estados finais presentes na lista `Finais`, estes são retransmitidos para Mh_{new}, independente da política adotada.

A política incondicional de envio de mensagens (considerando um *Mh* na situação (b)) tem a vantagem de ser menos pessimista, isto é, o *Mss* dá uma nova chance ao *Mh* de receber e responder um multicast antes de considerar sua resposta como `NOk`. Todavia, como nestes casos o *time-out* é prorrogado, esta política possui a desvantagem de ser mais demorada do que a política condicional.

A Figura 4.4 mostra um exemplo no qual uma mensagem M chega em Mss_o no momento em que Mh_{new} está em processo de migração. Considere que $Mh_{new} \in M.Dest$ e que ainda não tenha recebido a mensagem. Neste caso, como o *IMS* associado a Mss_o (*IMS2*) não é o mesmo *IMS* associado a Mss_n (*IMS1*), depois de enviar a mensagem `DeregAck` para Mss_n, Mss_o envia a mensagem `LocationUpdate` para *IMS2*.

Depois que `DeregAck` chega em Mss_n, este acrescenta Mh_{new} em `LocalMhs` e também envia a mensagem `LocationUpdate` para *IMS1*, informando-o que Mh_{new} agora se encontra em sua célula. Através de *h_RECD*, *IMS1* percebe que Mh_{new} ainda não recebeu M e a retransmite para Mss_n. Adotando a política de entrega condicional discutida anteriormente, Mss_n envia M para o novo *Mh*, o qual responde antes de T1 se esgotar. Mss_n então envia a mensagem $\texttt{OkSet\{}Mh_{new}\texttt{\}}$ para

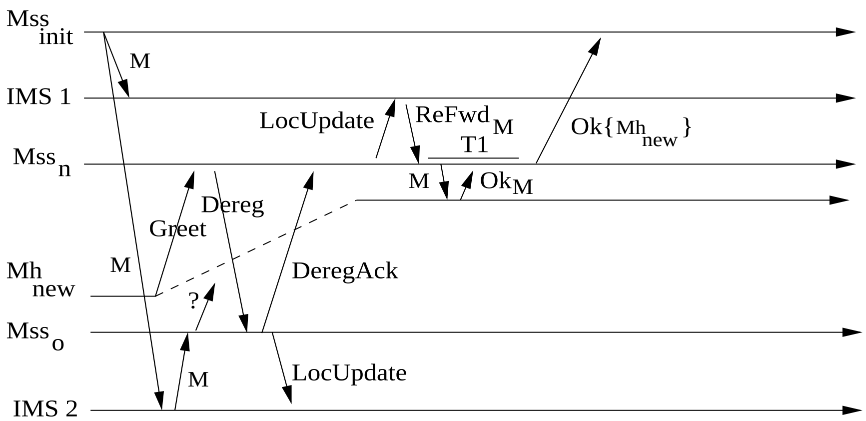

Figura 4.4: Protocolo de Hand-off no iAM^2C

Mss_{init} informando-o sobre a aceitação de M por parte de Mh_{new}. Caso fosse adotada a política de entrega incondicional, mesmo que Mss_n já tivesse enviado M a outro Mh destinatário local, e `ReFwd`$_M$ tivesse chegado antes do *time-out* ser processado, o *Mss* também retransmitiria M para Mh_{new}.

4.7 Discussão

Nesta seção discutiremos alguns pontos importantes do iAM^2C, comparando alguns deles com o AM^2C.

4.7.1 Consequência do Acréscimo de Mais um Nível de Indireção

Um ponto importante a ser discutido é a conseqüência do acréscimo de mais um nível de indireção no gerenciamento de localização do iAM^2C. A introdução dos *IMSs* como elemento intermediário entre os *Mss*s (*Mss* $\leftrightarrow$ *IMS* $\leftrightarrow$ *Mss*) eventualmente pode causar uma inconsistência da informação sobre a localização de um *Mh* nestes elementos. Isto é decorrente dos elementos envolvidos em um *hand-off* no iAM^2C (Mss_n, Mss_o e *IMS*) serem informados da nova localização do *Mh* em momentos diferentes, que acontece, respectivamente, na chegada da mensagem `Greet`, `Dereg` e `LocationUpdate`. Como estes eventos não ocorrem simultaneamente, a informação sobre a localização de um *Mh* fica inconsistente durante um certo período de tempo.

Assim, consideremos um *Mh* (Mh_{new}) que acabou de migrar da célula de Mss_o para a célula de Mss_n. Caso a mensagem `LocationUpdate` chegue no *IMS* após a chegada de uma mensagem de multicast M que tenha Mh_{new} como destinatário, então o *IMS* enviará M para Mss_o, pois o `LocMap` do *IMS* ainda não foi atualizado. Notemos que o envio de M, neste caso, é desnecessário, uma vez que Mss_o nada pode fazer em relação à transmissão da mensagem para Mh_{new} (pois este não mais pertence a `LocalMhs`). Dessa forma, aumenta-se o *overhead* do protocolo.

Todavia, como assumimos que a comunicação na rede fixa é confiável, em algum momento a inconsistência descrita acima deixará de existir. Ainda, o envio da mensagem `LocationUpdate` do Mss_n (*Mss* da nova célula) para seu correspondente *IMS*, ocorre imediatamente após o registro de Mh_{new} em sua célula. Dessa forma, espera-se que o *delay* para a atualização do mapeamento no *IMS* seja pequeno, evitando que mensagens sejam transmitidas desnecessariamente na rede com fio. Vale observar que mesmo nestes casos, não há nenhuma implicação em termos de corretude do protocolo, visto que o Mss_o simplesmente descartará a mensagem. Ainda, o envio de M para Mss_o só constituiria um *overhead* caso Mss_o só tivesse Mh_{new} como elemento destinatário da mensagem. Em todos os outros casos, a chegada de M seria processada normalmente por Mss_o, e não representaria nenhum custo adicional para o protocolo.

4.7.2 Papel Desempenhado Pelos *Mss*s e *IMSs*

Alguns aspectos também merecem atenção quando comparamos o comportamento do iAM^2C com o do AM^2C. Neste último, os *Mss*s tratam tanto do *hand-off* (transferência do *h_RECD*) como do armazenamento das mensagens e estados finais de multicasts. Já no iAM^2C estas funções são bem divididas entre os *Mss*s e os *IMSs*, respectivamente. Esta divisão torna o iAM^2C melhor estruturado, mas tem como conseqüência um maior número de mensagens na rede fixa para a transmissão de um multicast/estado final a um *Mh*, quando ocorre uma migração.

Apesar de implementações diferentes, a estrutura *h_RECD* tem a mesma finalidade nos dois protocolos: determinar quais os multicasts que já foram recebidos por um *Mh* e permitir decidir quais multicasts devem ser retransmitidos. No AM^2C esta decisão é tomada pelos *Mss*s, enquanto que no iAM^2C isto é feito pelos *IMSs*. Por este motivo, após completado um *hand-off* o *h_RECD* também precisa ser passado para o *IMS* correspondente (através da mensagem `LocationUpdate`).

4.7.3 Semântica de Resposta de um *Mss*

Outro ponto é a semântica da resposta de um *Mss* (para o Mss_{init} correspondente) ao término do tempo T1 (Fase I). No AM^2C o `Ok/NOk` sintetiza um conjunto de respostas (dos *Mhs* locais e destinatários de uma mensagem). Já no iAM^2C, o `OkSet` indica cada resposta de um *Mh* explicitamente.

4.7.4 Estimativa do Número de Mensagens Transmitidas

Nesta subseção apresentamos equações que estimam o número de mensagens transmitidas tanto na rede fixa como na rede sem fio, para ambos os protocolos. Em seguida é feita uma avaliação dos resultados obtidos. Antes de iniciarmos, sejam as seguintes considerações:

1. As equações referem-se às mensagens transmitidas em decorrência de uma única requisição de multicast;

2. Consideremos um sistema composto por M *Mhs* destinatários. Suponhamos também uma rede com I *IMSs* e N *Mss*s, dos quais apenas D destes possuem algum *Mh* local $\in M$;

3. Sejam *mig_1* e *mig_2* variáveis que indicam o número de migrações e/ou reconexões dos *Mhs* nas fases I e II, respectivamente, dos protocolos;

4. Estamos desconsiderando a influência de outros parâmetros (como o valor dos *time-outs*) nas análises seguintes;

5. Os parênteses na primeira linha de cada equação indicam as mensagens transmitidas pelos protocolos nas fases I e II, respectivamente.

Mensagens no AM^2C

Mensagens na rede fixa (Wd):

$$\begin{aligned} Wd &= (2N + 2mig\_1) + (2N + 2mig\_2 + mig\_2) + N \\ Wd &= 5N + 2mig\_1 + 3mig\_2 \end{aligned} \tag{4.1}$$

Mensagens na rede sem fio (Wl):

$$\begin{aligned} Wl &= (1 + 2M) + (2M + 2mig\_2) \\ Wl &= 4M + 2mig\_2 + 1 \\ Wl &= 2(2M + mig\_2) + 1 \end{aligned} \tag{4.2}$$

Em se tratando das mensagens na rede com fio, na Fase I do AM^2C temos a difusão do multicast para todos os *Mss*s do sistema, as respostas dos mesmos enviadas para o *Mss* inicializador do multicast, e as mensagens `Dereg` e `DeregAck` geradas durante os *hand-offs*. Este mesmo raciocínio pode ser aplicado à Fase II (a partir da difusão do estado final do multicast), levando-se em consideração também o número de mensagens `AckMss` extras[2]. Na Fase III temos a difusão da mensagem `Delete` para todos os *Msss*. Desta forma, a estimativa do número de mensagens trocadas na rede fixa no AM^2C é dada através da equação 4.1.

Com relação às mensagens enviadas pela rede sem fio, na Fase I do AM^2C, além da requisição do multicast, temos o envio e resposta da mensagem para os M destinatários. Notemos que nesta fase, graças à estrutura *h_RECD* (vide seção 3.5), uma mensagem de multicast é enviada uma única vez para um *Mh* destinatário. Na Fase II, além do envio e confirmação do estado final do multicast para os M destinatários, temos que levar em conta as retransmissões (e confirmações) do estado final sempre que há uma migração. Assim, na equação 4.2 temos o número estimado de mensagens transmitidas no enlace sem fio do protocolo AM^2C.

Mensagens no iAM^2C

Mensagens na rede fixa (Wd):

$$\begin{aligned} Wd &= (I + 2D + 2mig\_1 + 2mig\_1 + mig\_1) + \\ &(I + 2D + 2mig\_2 + 2mig\_2 + mig\_2 + mig\_2) + I \\ Wd &= 3I + 4D + 5mig\_1 + 6mig\_2 \end{aligned} \tag{4.3}$$

Mensagens na rede sem fio (Wl):

$$\begin{aligned} Wl &= (1 + 2M) + (2M + 2mig\_2) \\ Wl &= 4M + 2mig\_2 + 1 \\ Wl &= 2(2M + mig\_2) + 1 \end{aligned} \tag{4.4}$$

Considerando as mensagens enviadas na rede fixa durante a Fase I do protocolo iAM^2C, temos, nesta ordem: difusão do multicast para os *I IMSs*; envio (e resposta) da mensagem para os *D Msss* que possuem algum *Mh* local e destinatário do multicast; mensagens `Dereg` e `DeregAck` relacionadas aos *hand-offs*; atualizações de mapeamento com as migrações (notemos que no pior caso as migrações são sempre inter-domínio, o que implica no envio da mensagem `LocationUpdate` para os *Msss* da antiga e nova células); e retransmissões de mensagens pendentes por parte dos *IMSs*. Na Fase II a análise é semelhante, com a diferença de que temos que considerar as mensagens `AckMss` extras.

[2]Mensagem enviada quando uma resposta de um *Mh* é processada após a ocorrência do *time-out* T2 - vide seção 3.3.

Na Fase III temos a difusão da mensagem `Delete` para os *IMSs*. Desta maneira, a equação 4.3 apresenta uma estimativa do número de mensagens trocadas na rede fixa no iAM^2C.

Em se tratando das mensagens transmitidas no enlace sem fio, vale a mesma análise realizada para o protocolo AM^2C, resultando na equação 4.4. É importante observar que estamos considerando, como pior caso, a adoção da política incondicional de retransmissão de mensagens.

Comparação das Equações

Considerando as equações que estimam o número de mensagens trocadas na rede fixa nos protocolos AM^2C e iAM^2C (equação 4.1 e 4.3, respectivamente), aparentemente o segundo protocolo possui um custo superior ao do primeiro. Todavia, podemos assumir que I é desprezível, se comparado aos valores de N e D (por suposição, o número de *IMSs* é bem menor do que o número de *Mss*s). Além disto, considerando no pior caso que $D = N$, a estimativa do número de mensagens transmitidas na rede fixa do iAM^2C pode ser dada por:

$$Wd(iAM^2C) = 4N + 5mig\_1 + 6mig\_2 \tag{4.5}$$

Assim, resolvendo a inequação $Wd(iAM^2C) < Wd(AM^2C)$, chegamos ao resultado que o uso do protocolo iAM^2C compensa se $N > 3(mig\_1 + mig\_2)$. Isto mostra que o iAM^2C é mais adequado do que o AM^2C em redes com um número grande de *Mss*s, além de indicar que o protocolo é mais sensível à taxa de migração dos *Mhs*. A seção 5.5 mostra resultados de simulações que comprovam esta conclusão.

Em termos das mensagens transmitidas na rede sem fio, notemos que as equações 4.2 (AM^2C) e 4.4 (iAM^2C) são idênticas. Isto mostra que os dois protocolos utilizam estratégias similares para a entrega de mensagens aos seus destinatários. O que vai diferenciar é a escolha da política de transmissão de mensagens adotada no iAM^2C.

Capítulo 5

Implementação e Simulação

Este capítulo descreve a implementação e simulação dos protocolos AM^2C e iAM^2C. O objetivo da implementação foi testar, validar e comparar os dois protocolos, verificando as vantagens e desvantagens de cada um. Para isso, prototipamos e implementamos os protocolos no simulador *MobiCS* [8, 7, 5, 6], como veremos adiante. Para cada um dos protocolos efetuamos testes determinísticos e estocásticos, de maneira a verificar o comportamento dos protocolos em situações específicas e em cenários aleatórios. As estruturas de dados e os algoritmos[1] utilizados na implementação dos protocolos AM^2C e iAM^2C podem ser encontrados, respectivamente, nos apêndices C e D.

5.1 Simulador Mobi*CS*

Ambientes de Computação Móvel se caracterizam principalmente pela topologia dinâmica da rede e a influência da mobilidade sobre o custo implícito da localização e o grau de disponibilidade dos dispositivos móveis. Por este motivo, o uso de simuladores torna-se essencial para a avaliação de protocolos distribuídos para computação móvel, tais como o AM^2C e o iAM^2C.

O Mobi*CS* [8, 7, 5, 6] é um ambiente para prototipagem e simulação de protocolos distribuídos para computação móvel, que permite o uso de abstrações de programação de alto nível e provê total transparência de simulação para o programador. Mobi*CS* é capaz de emular um ambiente de computação móvel e funcionar em dois modos distintos de simulação: o modo *determinístico*, no qual o usuário do simulador pode criar e executar um cenário bem específico de teste; e o modo *estocástico*, no qual os elementos de rede possuem padrões de comportamento probabilísticos definidos pelo usuário. Tais modos de simulação serão discutidos em maiores detalhes adiante.

[1] Os algoritmos também foram desenvolvidos durante o trabalho.

O Mobi*CS* foi projetado para simular um ambiente de rede móvel estruturada, tal como descrito na seção 1. O simulador foi projetado para permitir a implementação e simulação de protocolos que realizam algoritmos distribuídos para ambientes de computação móvel. Além do AM^2C [13] e o iAM^2C [10], o *MobiCS* também foi usado para implementar protocolos como o MCAST [1], o RDP [12], o *Mobile IP* [28] e o *Mobile IP* com Otimizações de Rotas [9]. Estes protocolos levam em consideração a mobilidade e a localização dos dispositivos móveis, tratando explicitamente as desconexões e a topologia dinâmica intrínseca à computação móvel.

5.1.1 Simulação Determinística

No modo de simulação determinístico, o Mobi*CS* executa um *script* de simulação definido pelo usuário que expressa um cenário específico do ambiente computacional móvel a ser simulado. O *script* é composto de eventos como, por exemplo, a movimentação dos elementos móveis, o instante de recebimento de mensagens e eventos locais em cada elemento. Apenas o comportamento do protocolo sendo testado não pode ser descrito em um *script*, pois este é definido por sua implementação. No modo determinístico não existe a noção de tempo. Em vez disso, a simulação ocorre em rodadas definidas no *script* de simulação. Eventos simulados em uma mesma rodada são tratados como se fossem eventos concorrentes, ou seja, a ordem de execução é não-determinística.

Neste modo de simulação pode-se validar um protocolo através de *scripts* que retratam situações bem específicas e críticas para as quais o protocolo deve ser testado. Desta maneira, o protocolo pode ser depurado com relação às situações definidas nos *scripts*, a fim de se ter certeza de que o comportamento do protocolo está pelo menos parcialmente correto. Apesar disto não garantir a corretude do protocolo em todas as possíveis situações, este modo de simulação é uma ferramenta útil para o desenvolvimento de protocolos complexos. Testes bem sucedidos neste modo são um bom indicador de que o protocolo está funcionando adequadamente.

5.1.2 Simulação Estocástica

No modo de simulação estocástica, usualmente implementado pela maioria dos simuladores de rede, o *MobiCS* considera o comportamento dos elementos de rede como sendo aleatório e baseado em padrões probabilísticos definidos na modelagem da simulação. Neste modo, o simulador realiza testes exaustivos do protocolo com o objetivo de avaliar o desempenho dos algoritmos implementados e a complexidade das mensagens. O modelo de execução no modo estocástico é orientado a eventos, como na simulação determinística, e utiliza o conceito de tempo simulado para associar um *timestamp* aos eventos.

A primeira etapa da simulação estocástica é a modelagem do padrão de comportamento de cada elemento a ser simulado, tal como os computadores móveis e as conexões sem fio. Isto é, define-se a geração de eventos periódicos como eventos de migração, eventos de desconexão/conexão, de requisição (do serviço implementado pelo protocolo) e o atraso nas conexões de rede. O Mobi*CS* possibilita a especificação de vários modelos de simulação. Um modelo de simulação tipicamente possui parâmetros como a taxa de mobilidade dos *Mhs*, a probabilidade de desconexão/conexão destes, a probabilidade de atração de cada célula durante um *hand-off*, o atraso da comunicação sem fio em relação à comunicação na rede fixa, entre outros. Para cada um dos elementos de rede simulados definem-se probabilidades associadas a cada parâmetro do modelo de simulação (por exemplo, a probabilidade de migração de um *Mh*). A maioria dos parâmetros que definem tal modelo são baseados em uma noção de uma unidade de tempo simulada (UT). Ou seja, para cada novo valor de UT o modelo determina, de acordo com uma probabilidade indicada pelo usuário, quais eventos deverão ser gerados e processados pelo simulador.

Durante a simulação, a camada de simulação de cada elemento processa eventos gerados por outros elementos de rede ou pela própria camada (eventos de tempo) e ordenados pelo seu *timestamp*. Os eventos descrevem mensagens enviadas entre elementos de rede e o comportamento dinâmico dos elementos, em termos de envio de requisições, movimentações e início e fim de períodos de atividade de *Mhs*.

O simulador Mobi*CS* serviu como principal ferramenta para a prototipação, validação e avaliação de desempenho dos protocolos de multicast atômico AM^2C e iAM^2C para diferentes ambientes de computação móvel.

5.2 Modelo de Composição

Na programação de protocolos utilizando o *MobiCS*, um protocolo é composto de micro-protocolos que implementam uma funcionalidade parcial bem definida do protocolo. Cada micro-protocolo é composto de um estado e de um conjunto de métodos que tratam cada evento que pode ser recebido por ele. Esses eventos são definidos em interfaces Java (vide seção 5.3) para cada um dos elementos de rede a serem simulados (tais como *Mss*s, *Mhs* e *IMSs*). Deste modo, cada micro-protocolo implementa uma ou mais interfaces.

Seguindo este padrão, as Figuras 5.1 e 5.2 apresentam, respectivamente, um esquema da composição dos protocolos AM^2C e iAM^2C. O micro-protocolo *wired* é responsável pelo tratamento dado pelos elementos de rede na chegada de uma mensagem pela rede fixa. Dessa forma, o micro-protocolo *wired* contém interfaces para os elementos de rede que são capazes de receber uma mensagem enviada pela rede com fio (obviamente não há uma interface para os *Mhs*). De forma similar, as mensagens transmitidas pelo meio sem fio são tratadas pelo micro-protocolo *wireless*. Já o micro-protocolo de *hand-off* se encarrega de manipular (por meio dos métodos que fazem parte das interfaces) as mensagens trocadas durante a migração de um *Mh*.

Os diagramas com as principais classes (e seus respectivos métodos) que compõem os protocolos AM^2C e iAM^2C podem ser encontrados, respectivamente, nos apêndices A e B.

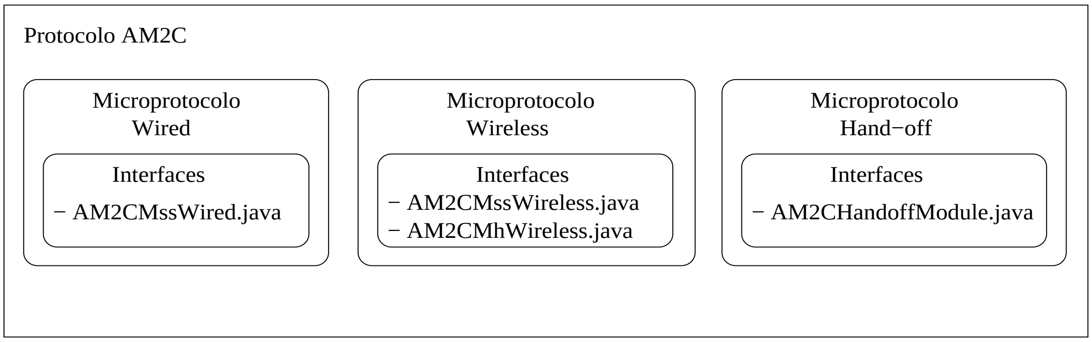

Figura 5.1: Composição do protocolo AM^2C.

5.3 Principais Interfaces

Nesta seção apresentaremos as principais interfaces de cada um dos elementos de rede dos protocolos AM^2C e iAM^2C. Essas interfaces abrangem a comunicação com e sem fio entre os elementos simulados, além do módulo de *hand-off*. É importante observar que o Mobi*CS* estabelece como convenção que a sintaxe para cada método tratado de um evento `Evento` seja denominado `whenEvento`.

5.3.1 Interfaces no AM^2C

AM2CMssWired.java (micro-protocolo *wired*)

Define a interface com fio do *Mss* no protocolo AM^2C. Contém as mensagens tratadas por um *Mss* e provenientes de um outro *Mss*.

- *void whenFwdMsgFromMssInit(Message m)* : recebimento de uma mensagem de multicast difundida pelo *Mss* inicializador do multicast (Mss_{init}). Faz parte da primeira fase do protocolo.

- *void whenRespFromMss(Message m)* : recebimento, por parte do Mss_{init}, de uma resposta de um *Mss* referente a uma mensagem de multicast. Ocorre durante a primeira fase do protocolo.

- *void whenStatusFromMssInit(Message m)* : recebimento do estado final de um multicast difundido pelo Mss_{init}. Faz parte da segunda fase do protocolo.

- *void whenAckMss(Message m)* : confirmação recebida pelo Mss_{init} sobre o recebimento do estado final de um multicast por parte dos *Mhs* locais. Ocorre durante a segunda fase do protocolo.

- *void whenDelete(Message m)* : recebimento da solicitação de remoção do estado final de um multicast armazenado localmente. Faz parte da terceira fase do protocolo.

AM2CMssWireless.java (micro-protocolo *wireless*)

Define a interface sem fio do *Mss* no AM^2C. Contém as mensagens tratadas por um *Mss* e provenientes de um *Mh* local e os eventos de *time-out* gerados.

- *void whenMulticastRequested(Message m)* : requisição de multicast de uma mensagem feito por um *Mh* para o seu *Mss* (que, para efeito do protocolo, será considerado o Mss_{init}).

- *void whenRespFromMh(Message m)* : resposta de um *Mh* sobre a aceitação ou não de uma mensagem. Ocorre durante a primeira fase do protocolo.

- *void whenAckMh(Message m)* : Confirmação de um *Mh* sobre o recebimento do estado final de um multicast. Ocorre durante a segunda fase do protocolo.

- *void whenJoin(Message m)* : pedido de entrada no sistema por um *Mh*.

- *void whenLeave(Message m)* : pedido de saída do sistema por um *Mh*.

- *void whenWaitingForReply(Timer t)* : ocorrência do tempo máximo esperado (*time-out*) por uma resposta de um *Mh*, referente ao envio de uma mensagem durante a primeira fase do protocolo.

- *void whenWaitingForAck(Timer t)* : ocorrência do tempo máximo esperado (*time-out*) por uma confirmação de um *Mh* sobre o recebimento do estado final de um multicast. Ocorre na segunda fase do protocolo.

AM2CMhWireless.java (micro-protocolo *wireless*)

Define a interface sem fio de um *Mh* no protocolo AM^2C. Contém as mensagens que são tratadas por um *Mh* e provenientes do *Mss* responsável pelo mesmo.

- *void whenFwdMsgFromMssN(Message m)* : recebimento de uma mensagem multicast na primeira fase do protocolo.
- *void whenStatusFromMssN(Message m)* : recebimento do estado final de um multicast durante a segunda fase do protocolo.

AM2CHandoffModule.java (micro-protocolo *hand-off*)

Define a interface do módulo de *hand-off* no AM^2C. Estas mensagens podem ocorrer em qualquer uma das fases do protocolo.

- *whenGreet(Message m)* : mensagem recebida pelo *Mss* quando um *Mh* entra na célula do *Mss* ou volta de um período de inatividade.
- *void whenDereg(Message m)* : recebimento, por parte do *Mss* da célula anterior de um *Mh* migratório, da solicitação de desvinculamento do *Mh*, enviada pelo *Mss* da nova célula.
- *void whenDeregAck(Message m)* : recebimento, por parte do *Mss* da célula para a qual o *Mh* migrou, da confirmação de desvinculamento do *Mh* de sua antiga célula.

5.3.2 Interfaces no iAM^2C

Nesta subseção, estaremos apresentando as interfaces do protocolo iAM^2C. É importante dizer que algumas delas são idênticas às interfaces usadas pelo AM^2C, e serão indicadas com um (*). Nestes casos, se o leitor desejar obter alguma explicação sobre a interface, basta consultar a mesma subseção para o protocolo AM^2C. A idéia principal neste momento é apresentar as principais diferenças em termos de interfaces. Claramente, o tratamento dado por um elemento de rede para uma mesma interface pode (e geralmente é) ser diferente nos dois protocolos (vide apêndices C e D).

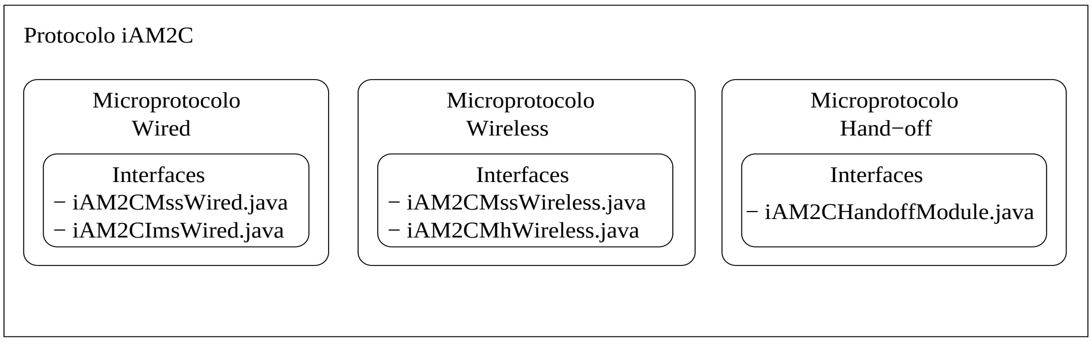

Figura 5.2: Composição do protocolo iAM^2C.

iAM2CMssWired.java (micro-protocolo *wired*)

Define a interface com fio de um *Mss*. Contém as mensagens recebidas por um *Mss* e enviadas por outro *Mss* ou por um *IMS*.

- *void whenFwdMsgFromIms(Message m)* : recebimento de uma mensagem de multicast enviada pelo *IMS* (relativo ao domínio do *Mss*) durante a primeira fase do protocolo.
- *void whenRespFromMss(Message m)* : recebimento, por parte do Mss_{init}, de uma mensagem de um *Mss* com respostas de *Mhs* locais ao *Mss* referente a uma determinada mensagem de multicast. Ocorre durante a primeira fase do protocolo.
- *void whenStatusFromIms(Message m)* : recebimento do estado final de um multicast enviado pelo *IMS* durante a segunda fase do iAM^2C.
- *void whenAckMss(Message m)*(*)
- *void whenReFwdMsgFromIms(Message m)* : recebimento de uma retransmissão de mensagens (em um único pacote) enviada pelo *IMS*.

iAM2CImsWired.java (micro-protocolo *wired*)

Define a interface (com fio) de um *IMS*, isto é, o tratamento de todas as mensagens que são enviadas por um *Mss*. Apesar de ocorrerem em um *IMS*, o que justifica o texto explicativo, as três primeiras mensagens fazem parte da interface com fio de um *Mss* no protocolo AM^2C.

- *void whenFwdMsgFromMssInit(Message m)* : recebimento de uma mensagem de multicast difundida pelo Mss_{init}. Faz parte da primeira fase do protocolo.

- *void whenStatusFromMssInit(Message m)* : recebimento do estado final de um multicast difundido pelo Mss_{init} (segunda fase do protocolo).
- *void whenDelete(Message m)* : recebimento da solicitação de remoção do estado final de um multicast armazenado localmente. Faz parte da terceira fase do protocolo.
- *void whenLocationUpdate(Message m)* : solicitação de atualização de mapeamento (e reenvio de mensagens pendentes), enviada de um *Mss* do domínio do *IMS*.

iAM2CMssWireless.java (micro-protocolo *wireless*)

Interface sem fio do *Mss* no protocolo iAM^2C. Contém as mensagens recebidas dos *Mhs* locais e os eventos de *time-out* gerados.

- *void whenMulticastRequested(Message m)*(*)
- *void whenRespFromMh(Message m)*(*)
- *void whenAckMh(Message m)*(*)
- *void whenJoin(Message m)*(*)
- *void whenLeave(Message m)*(*)
- *void whenWaitingForReply(Timer t)*(*)
- *void whenWaitingForAck(Timer t)*(*)
- *void whenTimeIsOver(Timer t)* : ocorrência do tempo máximo esperado (*time-out*) pelo *Mss* inicializador de um multicast por respostas de todos os *Mhs* destinatários de uma mensagem de multicast (como descrito na seção 4.4).

iAM2CMhWireless.java (micro-protocolo *wireless*)

Define a interface (sem fio) dos *Mhs* no iAM^2C. Corresponde às mensagens enviadas pelo *Mss* responsável pelo *Mh* e tratadas pelo mesmo. Esta interface é a mesma utilizada no protocolo AM^2C.

- *void whenFwdMsgFromMss(Message m)*(*)
- *void whenStatusFromMss(Message m)*(*)

iAM2CHandoffModule.java (micro-protocolo *hand-off*)

Interface do módulo de *hand-off* do protocolo iAM^2C. Também é a mesma utilizada no protocolo AM^2C.

- *whenGreet(Message m)*(*)
- *void whenDereg(Message m)*(*)
- *void whenDeregAck(Message m)*(*)

5.4 Testes

Aqui encontram-se os principais testes realizados com os protocolos através das simulações determinísticas e estocásticas.

5.4.1 Testes Determinísticos

Como mencionado na seção 5.1, os testes determinísticos são importantes no processo de testes e depuração de um protocolo. Nesta etapa focamos em identificar um conjunto de situações críticas e específicas para cada um dos protocolos. Em seguida, definimos e programamos os *scripts* de simulação correspondentes, a fim de testar o comportamento tanto do AM^2C como do iAM^2C diante destas situações.

As figuras 5.3 e 5.4 exemplificam dois cenários determinísticos simulados nos protocolos AM^2C e iAM^2C, respectivamente. Para efeito de simplificação da explanação a seguir, deve-se considerar os dois cenários como sendo somente uma parte de uma execução completa dos protocolos. Isto é, deve-se assumir que já foi feita uma solicitação de multicast por parte de algum *Mh* do sistema, e que os protocolos continuarão sendo executados até que a terceira fase dos mesmos seja completada. Além disto, consideramos um único destinatário para o multicast.

No cenário da figura 5.3 está sendo testada a propriedade do protocolo AM^2C de abortar um multicast (através do *Mss* da antiga célula, Mss_1) quando Mh_1 migra sem responder à mensagem enviada (caso o_2 da tabela 3.2).

A seguir apresentamos parte do *script* utilizado para simular o cenário da figura 5.3 em *MobiCS*. Estamos assumindo que o Mh_1 encontra-se inicialmente na célula de Mss_1, e que o *Mh* faz parte do grupo de destinatários do multicast.

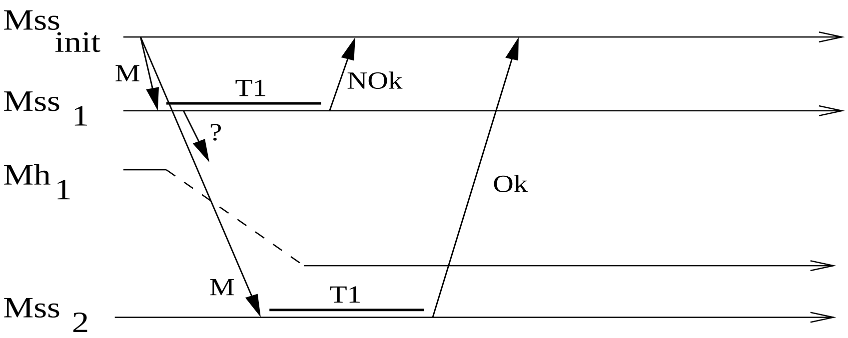

Figura 5.3: Exemplo de cenário determinístico no protocolo AM^2C.

```
1. acceptMessageOn (Mss1, false);
2. next();
3. Mh2.receive (new Start_AM2C( (Group) dest, 100));
4. next();
5. Mh1.moveTo (cell2);
6. next();
7. acceptMessageOn (Mss1, true);
8. next();
```

No *script* acima, inicialmente é desligado o processamento de mensagens no Mss_1 (linha 1), a fim de que o multicast (proveniente da requisição feita por Mh_2 ao Mss_{init} correspondente, linha 3) não seja processado imediatamente pelo *Mss*. Com o comando `moveTo` na linha 5, o protocolo de *hand-off* é iniciado, ou seja, o *Mss* da célula 2 recebe o `Greet` de Mh_1 e envia a mensagem `Dereg` para Mss_1. No momento em que o processamento de mensagens é reativado em Mss_1 (linha 7), a mensagem de multicast é processada, e então Mss_1 envia a mensagem para Mh_1. Logo em seguida Mss_1 processa a mensagem `Dereg`. Como neste momento o *Mss* não recebeu nenhuma resposta do Mh_1, então Mss_1 envia o $\texttt{NOk}_M$ para Mss_{init}. Vale observar que os comandos `next()` são necessários para estabelecer pontos de sincronização no *script*, determinando o início de um novo passo de simulação, que só é iniciado quando todos os elementos terminarem a execução das ações do passo anterior.

O segundo cenário (figura 5.4) tem por objetivo testar o envio do estado final do multicast no protocolo iAM^2C antes e depois da migração do *Mh*. Neste cenário pode-se verificar o envio inicial, por parte do *IMS*, da mensagem de *status* somente para Mss_1 (assumindo-se que Mss_2 não possui nenhum *Mh* local destinatário da mensagem) e a retransmissão do estado final pelo *IMS* para Mss_2

(após o envio da mensagem `LocationUpdate`).

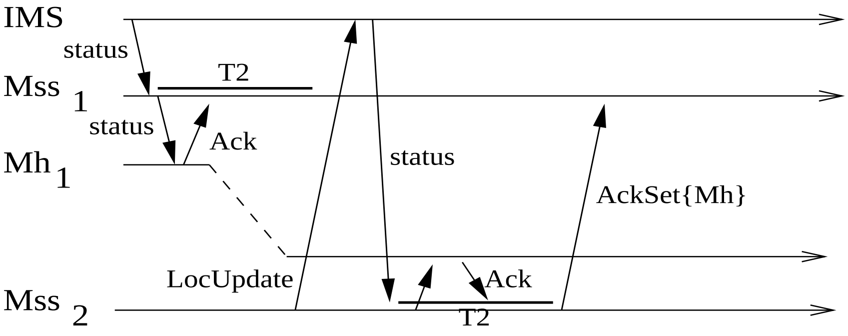

Figura 5.4: Exemplo de cenário determinístico no protocolo *iAM*^{2}*C*.

O cenário da figura 5.4 pode ser simulado através do trecho de *script* apresentado abaixo. Estamos considerando que `mss1` e `mss2` fazem parte do domínio de `ims`, e que `mh1` encontra-se inicialmente na célula de `mss1`. Embora o foco do cenário esteja na segunda fase do protocolo, abordaremos no *script* partes da primeira fase, de forma a ajudar no entendimento da fase seguinte.

A fim de que cada *time-out* possa ser processado através de um comando explícito, inicialmente o processamento de *time-outs* é desativado no Mss_1 (linha 1). Em seguida, uma nova requisição de multicast é feita pelo Mh_2 ao *Mss* correspondente (linha 3), dando início à primeira fase do protocolo. Na linha 5, é invocado explicitamente o processamento do *time-out* T1, que indica a conclusão da primeira fase e início da segunda fase do protocolo. Como o processamento do *time-out* da segunda fase está desligado no Mss_1, o protocolo é executado até a chegada do `Ack` (referente ao recebimento do estado final do multicast por parte de Mh_1) em Mss_1. Na linha 7, o comando `moveTo` desencadeia uma série de eventos definidos pelo protocolo, como o *hand-off* entre Mss_1 e Mss_2 e o conseqüente envio, por parte deste último, da mensagem `LocationUpdate` para *IMS*. Ainda no mesmo passo de simulação, o *IMS* retransmite o estado final do multicast para Mss_2, que por sua vez o repassa para Mh_1. No passo seguinte, o processamento do *time-out* T2 é ativado (linha 9) e o protocolo termina a sua execução.

```
1.  acceptTimeoutOn (Mss1, false);
2.  next();
3.  Mh2.receive (new Start_iAM2C( (Group) dest, 100));
4.  next();
5.  acceptTimeout (Mss1, WaitingForReply.class);
6.  next();
```

```
Mh1.moveTo (cell2);
next();
acceptTimeout (Mss1, WaitingForAck.class);;
next();
```

Para verificar se o protocolo executou um *script* “corretamente”, o programador deve analisar o comportamento esperado pelo protocolo antes de sua execução. Em seguida, deve-se comparar os resultados obtidos como *log* da execução do *script* com os resultados esperados.

A execução dos testes determinísticos no processo de simulação, permitiu não só avaliar o comportamento dos protocolos AM^2C e iAM^2C em diversas situações, mas também foi importante para detectar erros de lógica e uso inadequado de estruturas de dados. A experiência obtida com estes testes será abordada em mais detalhes no capítulo 6.

Passados os testes determinísticos, cresceu a confiança de que os protocolos possuíam as propriedades básicas necessárias para que funcionassem adequadamente nos cenários aleatórios da simulação estocástica.

5.4.2 Testes Estocásticos

A seguir serão apresentados os parâmetros de simulação utilizados nas simulações estocásticas do AM^2C e do iAM^2C, bem como as probabilidades associadas a cada parâmetro neste modelo. Como o objetivo principal foi a comparação do desempenho dos dois protocolos em diferentes cenários, a maioria destes parâmetros foram os mesmos para ambos os protocolos. Aqueles que são específicos de um protocolo serão explicitamente identificados.

Configuração e Características da Rede

- **Número de células (N_C):** na maior parte das simulações usamos uma configuração com 4 ou 8 células. De modo a testar os protocolos em situações mais extremas, realizamos algumas simulações para $N_C = \{10, 20, 30, 40, 50\}$ e $N_C = \{10, 15, 20, 25, 30\}$. A topologia utilizada foi de interconexão total. Para cada célula há um *Mss* responsável pela mesma;

- **Número de *IMSs*:** utilizamos uma configuração com dois *IMSs*, de forma que o domínio de cada *IMS* fosse aproximadamente metade do conjunto total de *Mss*s no sistema. Este parâmetro é exclusivo das simulações do iAM^2C;

- **Número de *Mhs*:** mantivemos sempre o mesmo número de *Mhs* ($N_{Mhs} = 15$) para permitir a comparação dos dois protocolos;

- **Posicionamento inicial dos *Mhs*:** inicialmente todos os *Mhs* foram "posicionados" nas duas primeiras células. O propósito disto foi verificar se de fato o iAM^2C apresenta um desempenho melhor quando há uma distribuição não-uniforme (concentração) de *Mhs* em algumas células.

- **Atraso associado a cada transmissão no meio sem fio:** considerando o atraso na rede com fio de 1 unidade de tempo simulada (1 *UTs*), usamos um atraso 50x maior no meio sem fio. Desta maneira, esta relação equivaleria a uma diferença, por exemplo, entre uma rede com uma taxa de transmissão em torno de 10 Mbps no meio com fio e de 200 kbps no meio sem fio.

Mobilidade

- **Probabilidade de migração** (P_{mig}): todos os *Mhs* tinham a mesma probabilidade de migração, ou seja, periodicamente determinava-se se cada *Mh* iria migrar ou não segundo esta probabilidade. Os experimentos foram realizados para P_{mig}= 0.01 (quase sem migração), 0.2, 0.4, e 0.6 (alta taxa de migração);

- **Fator de Atração das células:** optou-se pela adoção de atrações iguais para todas as células. Ou seja, cada vez que ocorria uma migração, todas as células tinham a mesma probabilidade de serem a célula-destino da migração. O objetivo não foi modelar uma rede específica, como por exemplo, uma cidade onde as células tipicamente têm diferentes fatores de atração (geralmente dependentes da hora do dia), mas sim testar o desempenho dos protocolos para uma mobilidade aleatória.

Disponibilidade e Requisição

- **Probabilidade de desconexão/reconexão:** todos os *Mhs* tinham a mesma probabilidade $P_{desc} = 0.01$ de ficarem desconectados da rede fixa e a probabilidade $P_{recon} = 0.3$ de se (re)conectarem à rede;

- **Probabilidade de requisição de um multicast:** cada *Mh* tinha a probabilidade $P_{req} = 0.1$ de requisitar um multicast para seu *Mss* responsável. Esta probabilidade foi escolhida de modo a gerar razões chamada-mobilidade (*Call-to-Mobility Ratio*) suficientemente distintas, em combinação com os diferentes valores escolhidos para P_{mig};

- **Destinatários dos multicasts:** todos os multicasts tinham todos os *Mhs* como destinatários (pior caso), de forma que possam influenciar os protocolos (tempo de execução, número de mensagens transmitidas, entre outros).

5.4.3 Dados Medidos

Nesta seção descrevemos os dados que foram medidos nas simulações estocásticas do AM^2C e do iAM^2C. Para cada combinação distinta de parâmetros de simulação foram realizadas dez rodadas de testes, e as médias dos valores medidos foram então calculadas. Em cada execução foram solicitadas vinte requisições de multicast pelos *Mhs* do sistema.

- **Porcentagem de multicasts abortados por requisição:** esse dado permite verificarmos o quão "pessimista" é cada um dos protocolos. É importante ressaltar que o valor de T1 na primeira fase dos protocolos tem influência direta sobre as chances de um multicast ser abortado, pois quanto menor for T1, maior a probabilidade de que alguns dos *Mhs* não respondam a uma mensagem a tempo, causando assim o cancelamento (*abort*) do multicast. Por este motivo, as simulações foram realizadas para valores de T1/T2 = {125, 175} *UTs*.

  Os valores de T1/T2 foram selecionados de forma a serem somente um pouco maiores do que duas vezes o atraso na transmissão pelo meio sem fio (50 *UTs*), que corresponderia ao tempo mínimo necessário para o envio de um multicast e da resposta pelo *Mh* através da interface *wireless*. A escolha destes valores para T1/T2 foi devido à constatação de que valores muito menores ou maiores do que os escolhidos eram visivelmente menos sensíveis à probabilidade de migração dos *Mhs*, isto é, abortavam ou confirmavam a maioria dos multicasts.

- **Número de mensagens *wireless* por *Mh* em uma requisição de multicast:** este dado permite compararmos a carga gerada pelo AM^2C e pelo iAM^2C na rede sem fio para diferentes taxas de migração.

- **Sobrecarga na rede fixa:** Estamos considerando como sobrecarga na rede com fio algumas das mensagens que são trocadas entre *hosts* fixos quando ocorre uma migração de um *Mh*. No AM^2C, a sobrecarga são todos os *acknowledgments* enviados de um *Mss* para o

Mss_{init} em decorrência do *time-out* T2 da segunda fase do protocolo[2] (`Acks` extras). Já no iAM^2C, além destas mensagens, consideramos também as atualizações de mapeamento (mensagens `LocationUpdate`) e os reenvios de mensagens pendentes por parte dos *IMSs* (mensagens `ReFwdMsgFromIMS`). A medição deste dado permite, além de constatar que o iAM^2C possui uma sobrecarga maior que a do AM^2C, verificar a influência da sobrecarga no número total de mensagens trocadas na rede fixa pelos dois protocolos.

- **Número de mensagens na rede fixa por requisição de multicast:** um dos problemas do protocolo AM^2C é a quantidade de mensagens difundidas na rede fixa e sua aparente ausência de escalabilidade com relação ao número de *Mss*s no sistema. O iAM^2C por outro lado, apesar de introduzir os *IMSs* na tentativa de ser mais escalável com relação ao número de *Mss*s, possui o *overhead* causado pelas mensagens do tipo `LocationUpdate` e `ReFwdMsgFromIMS`. Dessa forma, os dados medidos aqui permitem a comparação do custo total na rede fixa nos dois protocolos, para diferentes taxas de migração dos *Mhs* e diversos números de *Mss*s na rede.

- **Duração média de um multicast:** a duração é definida como o período entre o momento da difusão de uma mensagem de multicast por parte de Mss_{init} para todos os *Mss*s do sistema (ou para os *IMSs*), e o instante da difusão da mensagem `Delete` correspondente, pelo mesmo Mss_{init}. O objetivo desta medição foi comparar o tempo simulado gasto por cada um dos protocolos, considerando diferentes valores de T1/T2 e das taxas de migração dos *Mhs*. Assim, foi possível avaliar, mesmo tendo como base um tempo simulado, qual dos dois protocolos (e para quais parâmetros de simulação) foi mais eficiente.

5.5 Análise dos Resultados

Nesta seção faremos uma análise dos gráficos obtidos com os resultados das simulações, referentes aos dados mencionados na subseção anterior. O objetivo principal é comparar os protocolos AM^2C e iAM^2C, porém mostraremos também resultados relevantes de um protocolo específico. Como já mencionado, os resultados aqui apresentados referem-se aos valores médios obtidos em cada uma das simulações realizadas.

[2]Uma `Ack` enviado por um *Mss* antes de T2 não foi considerado como sobrecarga, visto que faz parte do processamento normal do protocolo (vide seção 4.4).

5.5.1 Custo *Wired* Total

Em primeiro lugar, é importante mencionar que estamos considerando como o custo *wired* total toda e qualquer mensagem trocada entre dois elementos na rede fixa. Ou seja, o custo *wired* total engloba, além das mensagens (da rede fixa) inerentes ao funcionamento de cada protocolo, as mensagens `Dereg` e `DeregAck` transmitidas durante o protocolo de *hand-off*, e as mensagens relativas à sobrecarga gerada na rede com fio em cada um dos protocolos (isto será abordado na subseção 5.5.3).

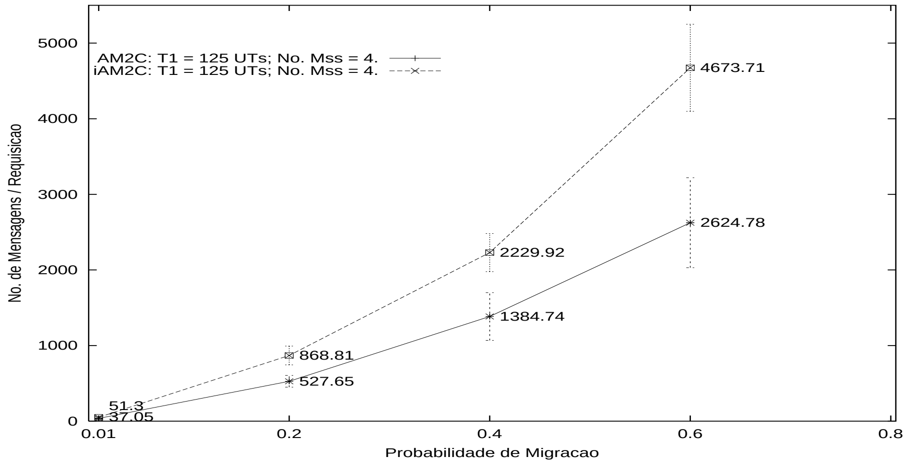

Figura 5.5: Comparação do custo *wired* total.

Na figura 5.5, analisando cada curva individualmente, é clara a influência da probabilidade de migração, P_{mig}, no número de mensagens transmitidas na rede fixa em ambos os protocolos. Isso se deve principalmente ao número de *hand-offs*, que aumenta proporcionalmente à P_{mig} e, no caso do iAM^2C, também ao número de mensagens do tipo `LocationUpdate` (vide subseção 5.5.3) que cresce com o aumento da probabilidade de migração.

Outro ponto importante a ser observado é que o iAM^2C apresenta um desempenho pior do que o AM^2C, onde o número de mensagens por requisição na rede com fio no primeiro protocolo é superior ao segundo em torno de 41 a 98%. Isto mostra que para uma rede com quatro *Mss*s (N_{mss}= 4), considerando-se os demais parâmetros, o acréscimo de mais um nível de indireção no iAM^2C, além do custo das atualizações de mapeamento, representam um aumento no custo *wired* do protocolo maior do que a redução devido ao menor número de *Mss*s envolvidos.

Considerando as duas curvas na figura 5.6 (N_{mss}= 8), fica claro que a introdução dos *IMSs* no iAM^2C (que consequentemente poupa alguns *Mss*s de receberem mensagens desnecessárias) representa um ganho em desempenho deste protocolo em relação ao AM^2C, para este cenário.

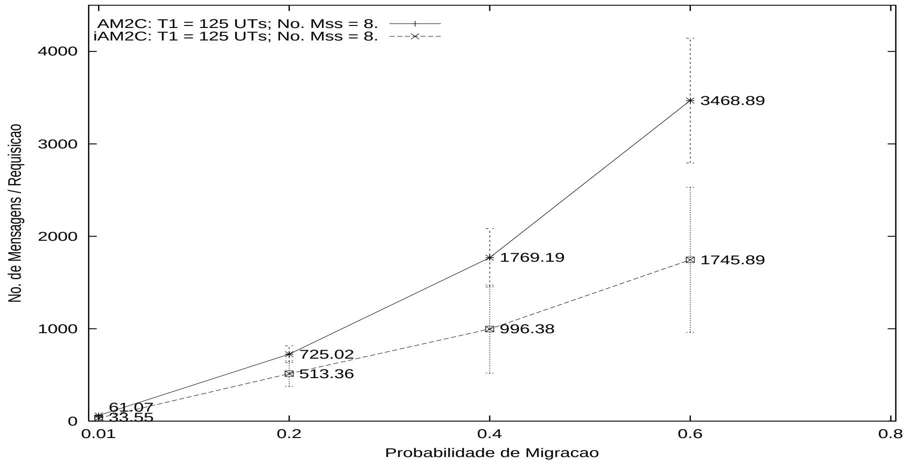

Figura 5.6: Comparação do custo *wired* total.

Um outro aspecto a ser considerado é que, analisando os valores obtidos em cada gráfico para cada um dos protocolos separadamente, e tomando-se um valor específico de P_{mig}, percebemos que enquanto o número de mensagens por requisição de multicast cresce com o aumento de N_{mss} no AM^2C (curvas contínuas), por outro lado diminui no iAM^2C (curvas pontilhadas). Isso já é um indício do problema da falta de escalabilidade do AM^2C. No caso do iAM^2C, fica evidente que o protocolo possui um melhor desempenho quando N_{mss}= 8. Para que possamos analisar melhor a questão da escalabilidade com relação ao número de *Mss*s, a seguir apresentaremos dois gráficos que retratam melhor esta questão nos protocolos.

5.5.2 Influência do Número de *Mss*s no Custo *Wired* Total

Em outra simulação, variamos o número de *Mss*s e mantivemos P_{mig} constante, a fim de avaliar a influência específica do número de *Mss*s no custo *wired* dos protocolos.

Na figura 5.7, percebemos que enquanto a curva sólida (AM^2C) apresenta um custo proporcional ao aumento de N_{mss}, a curva pontilhada (iAM^2C) mostra que o custo *wired* é relativamente constante. Concluímos então que neste caso em que P_{mig} é baixa ($P_{mig} = 0.01$), uma vantagem considerável do iAM^2C deve-se ao fato de boa parte dos *Mss*s não precisarem receber o multicast, pois não têm nenhum *Mh* local pertencente ao grupo de destinatários da mensagem. O AM^2C, por outro lado, não é beneficiado pelo valor de P_{mig}, uma vez que todos os *Mss*s recebem um multicast M independente de possuírem ou não algum $Mh \in M.Dest$.

A figura 5.8 mostra uma análise similar à da figura 5.7, só que para um valor maior de P_{mig}. As curvas confirmam a forte influência do número de *Mss*s no protocolo AM^2C, o que não acontece

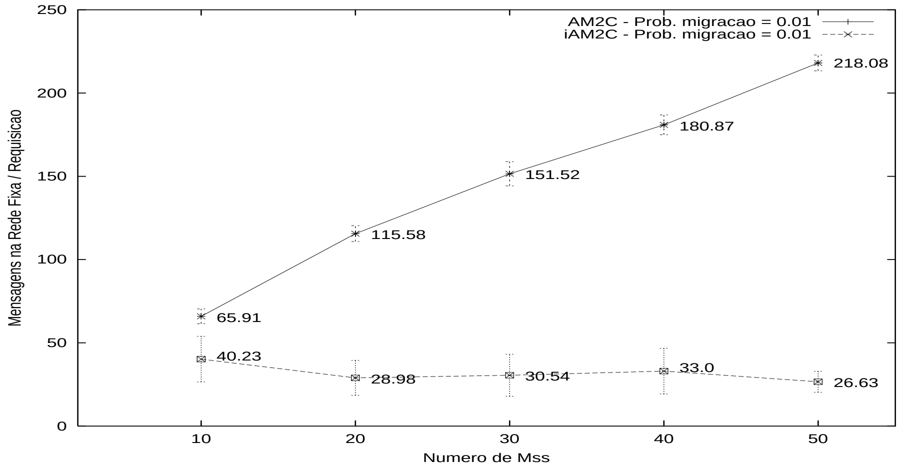

Figura 5.7: Influência do número de *Mss*s no custo *wired*.

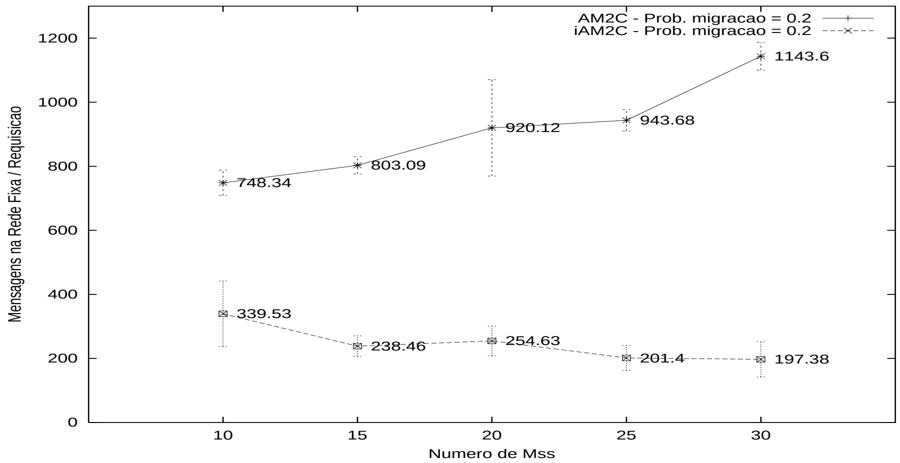

Figura 5.8: Influência do número de *Mss*s no custo *wired*.

com o iAM^2C. Como já era esperado, a maior diferença em relação à figura anterior está no fato dos valores absolutos medidos serem maiores, uma vez que P_{mig} também é maior.

Dessa forma, as figuras 5.7 e 5.8 mostram claramente a escalabilidade do iAM^2C com relação ao número de *Mss*s no sistema. Por outro lado, fica evidente que o custo *wired* do AM^2C é diretamente influenciado pelo valor de N_{mss}.

5.5.3 Distribuição da Sobrecarga no iAM^2C

Como já mencionado na subseção 5.4.3, a sobrecarga na rede fixa do protocolo iAM^2C engloba, além dos `Acks` extras (única sobrecarga do AM^2C), as mensagens de atualização de mapeamento (`LocationUpdate`) e de reenvio de mensagens pendentes (`ReFwdMsgFromIMS`), por parte dos *IMSs*.

Dessa maneira, procuramos analisar nesta simulação como está dividida a sobrecarga do iAM^2C, e qual a influência deste dado sobre o custo *wired* total do protocolo.

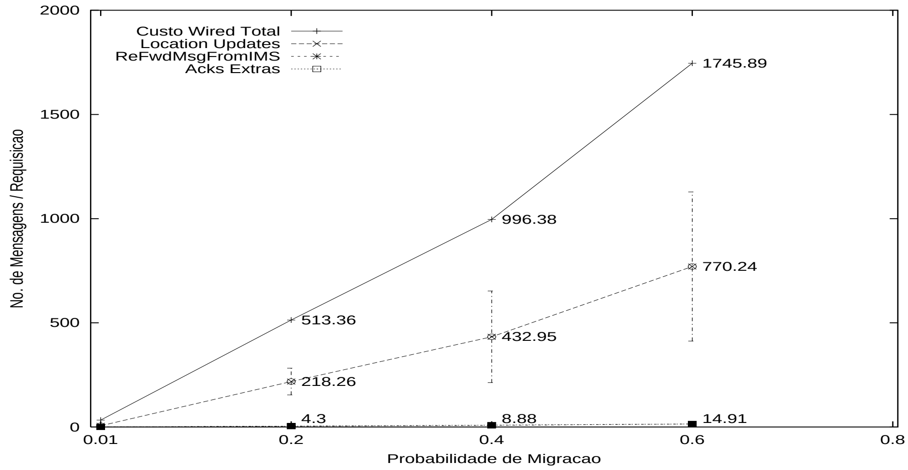

Figura 5.9: Distribuição da sobrecarga no iAM^2C.

Analisando a distribuição da sobrecarga na figura 5.9 percebe-se duas coisas: a primeira é que as grandes responsáveis pela sobrecarga no iAM^2C são as mensagens do tipo `LocationUpdate`, que representam de 88 a 96% da sobrecarga total; a segunda é que no iAM^2C, a sobrecarga (podemos considerar: as mensagens `LocationUpdate`) na rede com fio é responsável por cerca de 43% (exceto para P_{mig}= 0.01, onde este percentual é de 17%) do número total de mensagens enviadas por esse meio de transmissão. É importante mencionar que o iAM^2C otimiza as retransmissões de mensagens pendentes (e estados finais) por parte dos *IMSs*, empacotando todas estas mensagens e fazendo um único envio para o *Mss* solicitante.

A figura 5.9 também mostra a influência da probabilidade de migração dos *Mhs* em cada um dos tipos de mensagens que compõem a sobrecarga na rede fixa. Quanto maior o valor de P_{mig}, nitidamente maior é o número de mensagens `LocationUpdate`, o que implica também no aumento do número de retransmissões dos *IMSs*. Com relação ao número de `Acks` extras, quanto maior for a probabilidade de migrações dos *Mhs*, maiores as chances de que em algumas (ou em várias) dessas migrações o *time-out* T2 já tenha ocorrido no *Mss* responsável pela célula para a qual o *Mh* migrou, fazendo com que uma nova mensagem `AckMss` seja enviada para o *Mss* inicializador do multicast.

Vale ressaltar ainda que no caso do protocolo AM^2C a sobrecarga na rede fixa (que, como dito anteriormente, consiste apenas do número de `Acks` adicionais) é da mesma ordem de grandeza das mensagens `AckMss` extras medidas nas simulações do iAM^2C. Em outras palavras, ao contrário do que ocorre com o iAM^2C, no AM^2C a sobrecarga pouco influencia o custo *wired* total na rede com fio.

5.5.4 Custo *Wireless*

Nos protocolos AM^2C e iAM^2C há um custo fixo, por requisição de multicast, de quatro transmissões na rede sem fio, para cada *Mh* destinatário. Este custo envolve a transmissão, por parte de um *Mss*, de uma mensagem multicast e recebimento da resposta de um *Mh* (Fase I), e o envio do estado final do multicast e recebimento do *acknowledgment* (Fase II). Todavia, em ambos os protocolos os estados finais são retransmitidos quando um *Mh* migra para uma nova célula, além de que no iAM^2C existe a possibilidade de retransmissão *wireless* na Fase I (este caso poderá ser percebido na seção 6.2). Desta forma, estamos interessados em medir o impacto gerado pela probabilidade de migração dos *Mhs* no custo *wireless* total, em cada protocolo.

Notemos que os valores dos *time-outs* T1/T2 exercem influência sobre a quantidade de mensagens transmitidas na rede sem fio. Por exemplo, para um valor de T2 muito baixo, talvez o tempo para um *Mh* confirmar o recebimento do estado final de um multicast seja insuficiente. Como os estados finais são retransmitidos após uma migração, o valor escolhido para T2 causaria um maior número de mensagens *wireless* transmitidas. Entretanto, a influência deste parâmetro não foi percebida nas simulações devido à escolha de valores próximos para os *time-outs*.

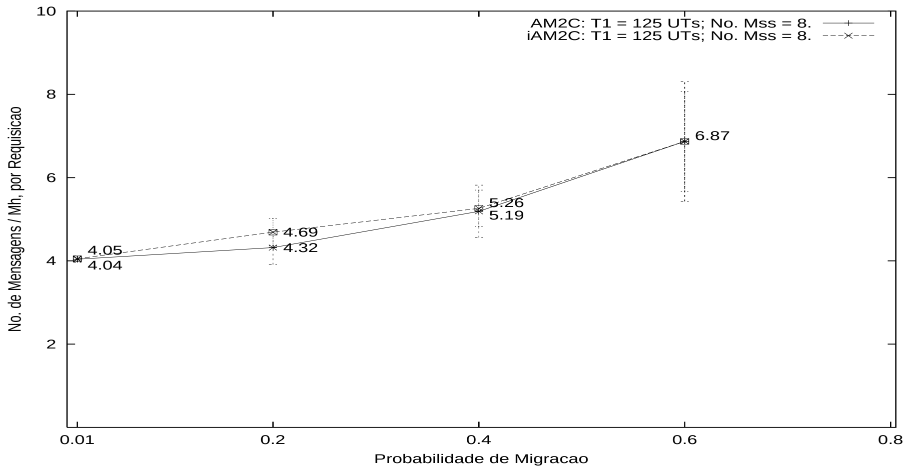

Figura 5.10: Comparação do custo *wireless* nos protocolos.

Analisando as duas curvas, percebemos que tanto o AM^2C como o iAM^2C possuem um custo *wireless* baixo (75% dos valores obtidos apresentam um custo excedente[3], por requisição, não superior a 1.26), e muito próximos para um mesmo valor de P_{mig}. Esta proximidade se deve ao fato

[3]Custo superior às quatro mensagens, por *Mh*, para envio e resposta de um multicast e do seu estado final.

da semelhança comportamental entre os dois protocolos (de uma forma geral), onde a diferença principal está na maneira como as mensagens chegam aos seus destinatários (através da rede fixa).

Deve-se mencionar ainda que adotamos a política incondicional de retransmissão de mensagens, como descrito na seção 4.6. Caso optássemos pela política condicional, muito provavelmente o iAM^2C apresentaria um custo *wireless* ainda menor, visto que esta política causaria menos retransmissões (com a provável conseqüência de uma porcentagem maior de multicasts abortados).

5.5.5 Porcentagem de Multicasts Abortados

Os gráficos a seguir comparam os protocolos AM^2C e iAM^2C com relação à porcentagem média de multicasts abortados. Analisaremos os gráficos de três formas: evidenciando a influência da probabilidade de migração (P_{mig}) dos *Mhs* sobre o dado medido; mostrando a influência do *time-out* T1 (primeira fase de ambos os protocolos); e analisando a porcentagem de multicasts abortados nos dois protocolos em cada gráfico.

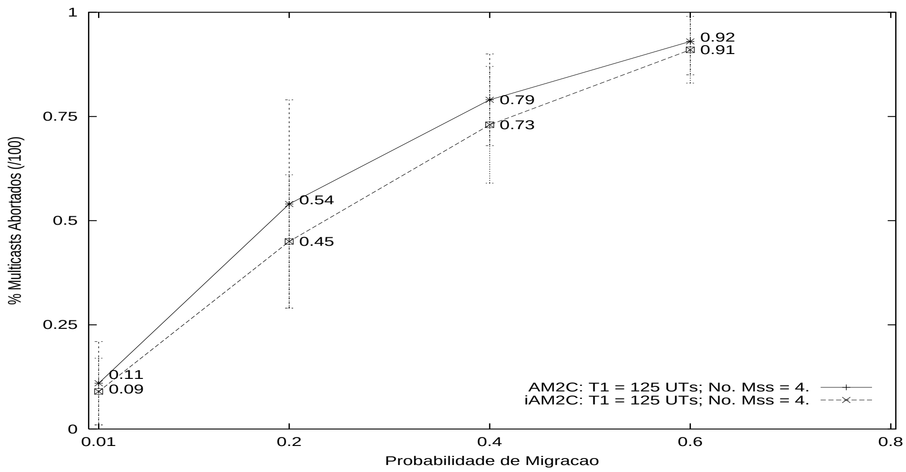

Figura 5.11: Comparação da porcentagem de multicasts abortados.

As figuras 5.11 e 5.12 (para T1 = 125 e 175 UTs, respectivamente) deixam claro a influência de P_{mig} na porcentagem de multicasts abortados (`%Abort`) em ambos os protocolos. Vale ressaltar o aumento brusco de `%Abort` (de até cinco vezes na figura 5.11, e até sete vezes na figura 5.12) quando P_{mig} passa de 0.01 para 0.2. Isso reflete a visão pessimista dos protocolos, onde o *Mss* da antiga célula opta por abortar um multicast diante da incerteza sobre a transmissão do multicast por parte do novo *Mss* para o *Mh* que está migrando (conforme seção 3.5 para o AM^2C e seção 4.6 para o iAM^2C).

Se analisarmos as curvas de cada protocolo separadamente nos dois gráficos, podemos perceber

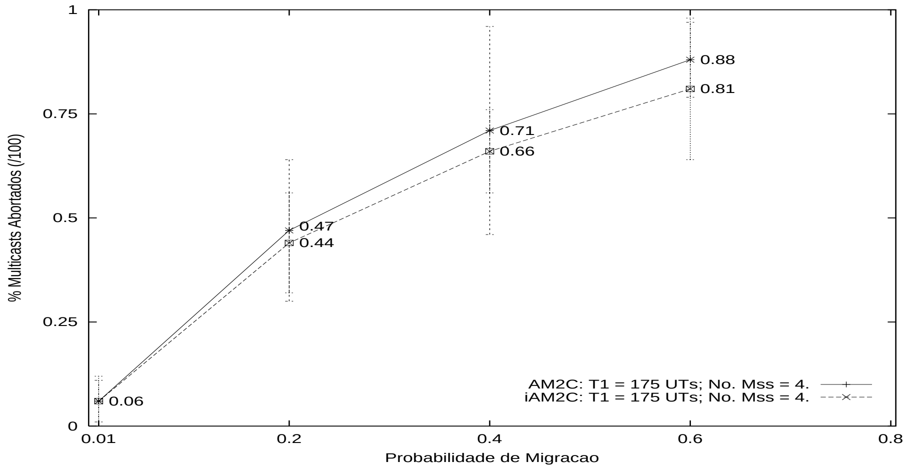

Figura 5.12: Comparação da porcentagem de multicasts abortados.

que a curva do respectivo protocolo na figura 5.11 (onde T1 = 125 UTs) possui valores mais elevados que a mesma curva (e valor de P_{mig}) na figura 5.12 (T1 = 175 UTs). Ou seja, quando o tempo de espera por respostas dos *Mhs* destinatários nos *Mss*s é menor, maior é o valor de `%Abort`. Isto acontece porque quanto menor o *time-out* T1, maiores são as chances de que neste tempo nem todos os *Mhs* locais e destinatários tenham respondido a uma determinada mensagem, fazendo com que o multicast seja abortado.

Comparando o AM^2C e o iAM^2C em ambos os gráficos, podemos observar a proximidade dos valores obtidos para cada valor de P_{mig} (a diferença máxima é de apenas 0.07 ponto percentual), bem como o aumento proporcional de `%Abort` com o incremento da probabilidade de migração dos *Mhs*. A explicação para isso está na similaridade comportamental dos protocolos no que se refere às situações que geram um cancelamento (*abort*) de um multicast, que são: (a) quando nem todos os *Mhs* locais e destinatários de uma mensagem respondem dentro do tempo T1; (b) quando um *Mh* migra sem responder a uma mensagem que lhe foi enviada; e (c), quando um novo *Mh* se registra na célula de um *Mss* sem ter recebido ainda uma determinada mensagem, mas o *time-out* desta já foi processado pelo *Mss*.

Como o envio do `NOk` pelo *Mss* no AM^2C, bem como o envio do `OkSet` (com o `NOk` relativo ao *Mh* correspondente) no iAM^2C, ocorrem nas mesma condições, apesar das curvas do iAM^2C estarem um pouco abaixo das curvas do AM^2C, a diferença entre a média dos valores medidos nos dois protocolos não foi significativa. O fato é que ambos os protocolos implementam uma visão pessimista, que pode acarretar em uma alta porcentagem de multicasts abortados, especialmente em redes onde a taxa de migração dos *Mhs* é alta.

5.5.6 Duração Média de um Multicast

Seguindo a mesma abordagem da subseção anterior, primeiramente analisaremos a influência de P_{mig} na duração média de um multicast nos dois protocolos. Em seguida discutiremos a influência dos *time-outs* (T1/T2) neste dado medido, concluindo com a comparação dos protocolos.

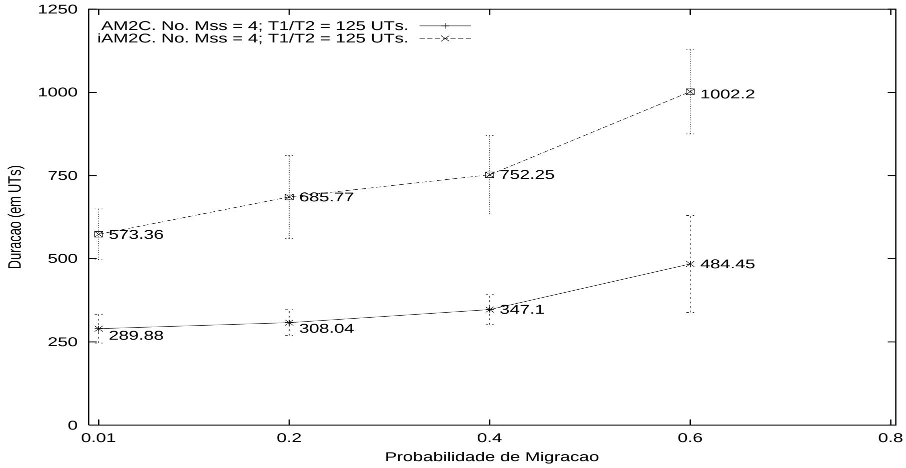

Figura 5.13: Comparação da duração média de um multicast.

De forma geral, nota-se que a duração média de um multicast nas figuras 5.13 e 5.14 (com os *time-outs* T1/T2 = 125 e 175 UTs, respectivamente) aumenta conforme a probabilidade de migração dos *Mhs*. Isso é causado principalmente pela Fase II dos protocolos, onde o estado final de um multicast é sempre retransmitido quando um *Mh* migra, fazendo com que o tempo de espera seja prolongado por mais T2 unidades de tempo. Na Fase I, a prorrogação da espera por respostas dos *Mhs* (um novo período de *time-out* T1) pode acontecer, mas este caso é bem mais raro do que na Fase II, devido à visão pessimista do Mss_o, que quase sempre opta por abortar o multicast.

Como já era esperado, percebemos também que a duração média de um multicast é proporcional ao valor dos *time-outs* (basta comparar as curvas individualmente para cada protocolo nos dois gráficos).

Comparando o AM^2C e o iAM^2C, temos três pontos importantes a serem destacados. O primeiro é o comportamento parecido das duas curvas em ambos os gráficos. Isto nos mostra que a essência dos dois protocolos é a mesma, havendo algum detalhe que torna o tempo de execução médio (em unidades de tempo simuladas) do iAM^2C maior. E este detalhe, que se refere à distância entre as curvas nos gráficos (segundo ponto), está ligado à inclusão dos *IMSs* como elementos intermediários entre os *Mss*s no iAM^2C, acrescentando assim mais um nível de indireção no fluxo das mensagens. Além disto, o iAM^2C gasta um tempo adicional, quando comparado com o AM^2C, com os envios

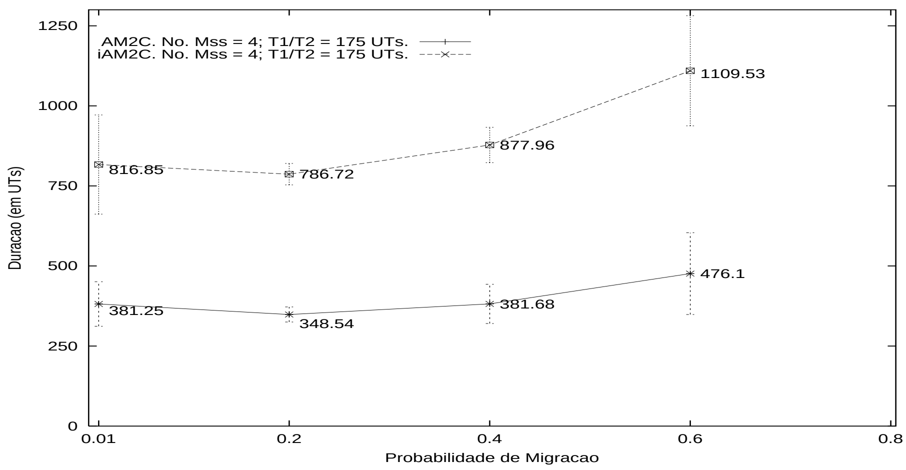

Figura 5.14: Comparação da duração média de um multicast.

das mensagens `LocationUpdate` e `ReFwdMsgFromIMS` que, como vimos anteriormente, representam o principal *overhead* do protocolo.

O terceiro e último ponto está no fato de que, se calcularmos a diferença entre o maior e o menor valor da duração média de um multicast em cada um dos protocolos (considerando os dois gráficos), percebemos que esta diferença é de aproximadamente 187 UTs para o AM^2C e de 536 UTs para o iAM^2C. Assim, concluímos que a variação dos parâmetros de configuração (tais como P_{mig} e T1/T2) influencia bem menos o AM^2C do que o iAM^2C, que nitidamente é mais demorado que o primeiro.

5.5.7 Avaliação Geral

As simulações dos dois protocolos mostraram que o iAM^2C apresenta uma sobrecarga na rede fixa bem maior que a do AM^2C, e que o grande responsável por isto são as mensagens `LocationUpdate`. Todavia, observamos que, exceto para redes onde o número de *Mss*s (N_{mss}) é pequeno, a redução do número de mensagens transmitidas na rede fixa (custo *wired*) com a introdução dos *IMSs* compensa o custo da sobrecarga no iAM^2C. Assim, o desempenho deste protocolo em termos do custo total na rede com fio quando N_{mss} é grande, é consideravelmente melhor do que o desempenho do AM^2C. Além disto, foi constatado que para uma mesma probabilidade de migração dos *Mhs*, o custo *wired* do AM^2C é diretamente proporcional ao número de *Mss*s no sistema. Já no iAM^2C este custo permanece "constante", mesmo quando o número de *Mss*s é grande.

Os resultados dos testes também mostraram que ambos os protocolos apresentam um baixo custo na rede sem fio (custo *wireless*) e uma alta porcentagem de multicasts abortados em redes onde a taxa de migração dos *Mhs* é acentuada. Este último dado confirma a abordagem conservadora dos protocolos para multicasts confiáveis, uma vez que requer a disponibilidade de todos os destinatários de um multicast para que o mesmo seja confirmado (*commited*). No entanto, ao contrário de outros 2PC, o iAM^2C não sofre do problema de bloqueio indefinido, devido a dois fatores:

a) A suposição de que os *hosts* da rede fixa são altamente confiáveis, fazendo com que o estado do 2PC esteja sempre disponível e possa sempre ser retransmitido para os *Mhs* destinatários, independente de sua localização (esta suposição também é valida para o AM^2C);

b) A decisão unilateral de abortar um multicast quando o tempo de validade do mesmo estiver esgotado, e da exclusão forçada dos *Mhs* inacessíveis.

	AM^{2}C	**iAM^{2}C**
`Escalabilidade em Relação a` N_{mss}	não	sim
`Sobrecarga na Rede Fixa`	baixa	alta
`Custo Wired Total`	menor quando N_{mss} é pequeno	menor quando N_{mss} é grande
`Custo na Rede sem Fio`	baixo	baixo
`% Multicast Abortados`	alta para taxas de migração acentuadas	alta para taxas de migração acentuadas
`Tempo de Execução Médio`	baixo	alto
`Passível de Bloqueio Indefinido`	sim	não

Tabela 5.1: Resumo comparativo dos protocolos AM^2C e iAM^2C.

Também pudemos concluir através das simulações que uma vantagem do protocolo AM^2C em relação ao iAM^2C é o tempo de execução médio de um multicast (em unidades de tempo simuladas, UTs). As atualizações de mapeamento e retransmissões de mensagens pendentes, bem como o acréscimo de mais um nível de indireção no iAM^2C aumentam o tempo de execução do protocolo, que em média é aproximadamente 400 UTs a mais que o tempo gasto pelo AM^2C.

A tabela 5.1 apresenta um resumo da comparação entre os protocolos AM^2C e iAM^2C em termos dos dados e parâmetros avaliados nas simulações.

Como podemos observar, nenhum dos protocolos é melhor que o outro em todos os aspectos. O que de fato ocorre é que o AM^2C se enquadra melhor em redes onde o número de *Mss*s é pequeno, visto que sua sobrecarga na rede com fio é baixa e o seu tempo de execução é melhor que o do iAM^2C. Este, por outro lado, é mais adequado para redes com um grande número de *Mss*s, onde a quantidade de mensagens difundidas na rede fixa é bem menor se comparada com o AM^2C. Esta redução é ressaltada especialmente quando a probabilidade de migração dos *Mhs* é baixa, pois, nestes casos, o iAM^2C se beneficia do uso dos *IMSs* como filtros dos *Mss*s que devem receber uma mensagem de multicast.

Em redes onde a taxa de migração dos *Mhs* é alta, o iAM^2C só é adequado quando o número de *Mss*s é significativamente grande. Assim, o custo da sobrecarga gerado pelas atualizações de mapeamento é compensado pela redução das mensagens difundidas na rede fixa.

Capítulo 6

Experiência Adquirida e Dificuldades Encontradas

Neste capítulo primeiramente descreveremos a experiência adquirida na implementação dos protocolos AM^2C e iAM^2C utilizando o simulador *MobiCS* [5]. Em seguida, apresentaremos as principais dificuldades encontradas durante os processos de implementação e validação dos protocolos, que em alguns casos levaram à tomadas de decisões de projeto.

6.1 Implementação Utilizando o *MobiCS*

Depois de implementar e simular os protocolos AM^2C e iAM^2C usando o simulador *MobiCS*, algumas conclusões puderam ser extraídas:

- O *MobiCS* é uma ferramenta simples e de fácil uso, onde a possibilidade de se ter abstrações de alto nível permite que protocolos sejam prototipados rapidamente;
- A simulação determinística é essencial para a detecção de erros. Apesar deste modo de simulação não garantir a corretude do protocolo, a elaboração de scripts de simulação simples mas com um objetivo bem definido, permite que o protocolo seja testado em situações bem específicas e controladas, escolhidas pelo programador. Dessa forma, esta etapa de testes é fundamental para descobrir se há erros no protocolo e para pensar nas mudanças necessárias para que o protocolo execute conforme o esperado. Note que se o projetista/programador do protocolo for capaz de identificar as principais situações "críticas" em que o mesmo precisa ser

testado, existem boas chances de que após a conclusão dos testes determinísticos o protocolo esteja de fato funcionando corretamente;

- Para uma simulação estocástica, basicamente são necessárias três coisas: a configuração da rede, o modelo de simulação dos elementos de rede (neste trabalho, apenas dos *Mhs*) e o protocolo em si. Assim, a depender da complexidade do protocolo e do que se deseja analisar (em termos de desempenho), bastante atenção deve ser dada à definição e implementação destas partes. Erros de lógica ou parâmetros não inicializados de forma correta podem provocar erros na simulação do protocolo, que podem demandar um tempo considerável para detecção do problema;

É importante observar que o alto nível de abstração permitido pelo *MobiCS* faz com que vários aspectos relativos à implementação de um protocolo em um ambiente real sejam desconsiderados, permitindo que o projetista se concentre nos aspectos algorítmicos relevantes de seu protocolo. Devido a esta característica e à possibilidade de submeter o protocolo tanto à simulações controladas (determinísticas), como à simulações estocásticas, o *MobiCS* é uma ferramenta bastante útil e eficaz para a validação e avaliação do desempenho de protocolos para redes móveis.

6.2 Principais Dificuldades

Uma das dificuldades encontradas foi a programação do gerenciamento dos *time-outs*, dado que em *MobiCS* o controle da execução/prorrogação ou não dos mesmos é de total responsabilidade do programador. Além disto, na época do início da programação dos protocolos, o processamento de *time-outs* acabara de ser incorporado à biblioteca do *MobiCS*, de forma que não haviam outros usuários do simulador com um conhecimento sobre o uso deste recurso, e com os quais fosse possível a troca de experiências.

Além do controle da execução dos *time-outs*, durante a etapa de programação dos protocolos percebemos a necessidade de gerenciar a ocorrência ou não de determinadas situações. Por exemplo, ações diferentes deveriam ser tomadas caso um *time-out* T1 (ou T2) já tivesse ocorrido; se a mensagem `Delete` já tivesse ou não sido enviada; caso o estado final de um multicast já tivesse sido definido, entre outras. Dessa maneira, decidimos criar uma estrutura de dados que armazenasse informações para o controle da ocorrência de situações como as exemplificadas anteriormente, considerando cada um dos multicasts em processamento. Denominamos esta estrutura `objMsgManager` (vide apêndices C e D).

Assim, para cada multicast difundido na rede seria criado no *Mss* um `objMsgManager`. Com esta estrutura (que contém apenas atributos *booleanos*), o *Mss* é capaz de gerenciar as ações que devem ser tomadas na chegada de determinadas mensagens do protocolo (por exemplo, na chegada da mensagem `RespFromMh` ou `AckMh`). Ainda, com o `objMsgManager` foi possível determinar se no momento da geração de um evento de *time-out* pelo *MobiCS*, este deveria ou não ser processado ou prorrogado pelo protocolo.

Em termos de simulação, inicialmente encontramos dificuldades em identificar a causa de algumas saídas da simulação estocástica aparentemente não estarem de acordo com o esperado. O código referente ao tratamento dado às mensagens recebidas pelos elementos de rede foi depurado e verificou-se que o mesmo estava coerente com a especificação. Então, percebemos que o problema era decorrente de erros de lógica no modelo de simulação dos *Mhs*, onde várias características do comportamento destes elementos são definidas. Um erro encontrado foi a migração forçada dos *Mhs*, independente da taxa de migração especificada.

Muitas das dificuldades encontradas durante a implementação e simulação do AM^2C não foram encontradas durante estas fases no iAM^2C. Isto aconteceu devido à similaridade dos dois protocolos e à experiência adquirida com a implementação do AM^2C. Todavia, um aspecto da implementação do iAM^2C causou certa dificuldade. Através das simulações determinísticas detectamos que a estrutura *h_RECD*, que até então era idêntica a do AM^2C, não era adequada ao iAM^2C, fazendo com que o protocolo não executasse corretamente certos cenários de testes neste modo de simulação.

A necessidade de uma estrutura de dados diferente para o *h_RECD* foi detectada a partir de dois tipos de situações: **(a)** quando não era possível determinar a ordem de entrega de mensagens provenientes de um mesmo *Mss* inicializador; e **(b)**, quando uma mesma mensagem era entregue mais de uma vez para um mesmo *Mh* destinatário. Para ilustrar a necessidade de uma estrutura *h_RECD* diferente no protocolo iAM^2C, apresentamos a seguir dois exemplos que ilustram as situações mencionadas anteriormente. Para isso, vamos supor que *h_RECD* é um vetor de inteiros, como em AM^2C, que qualquer posição *h_RECD*[i] contém inicialmente o valor "0", e que um *Mh* é destinatário de qualquer mensagem de multicast difundida na rede.

Situação (a): Imaginemos que o *Mh* migre antes que seu Mss_{Mh} tenha recebido uma mensagem de multicast M_1 (cujo número de seqüência seja "1" e que tenha sido difundida por Mss_i). Ao se registrar na nova célula (sob a responsabilidade de Mss_n), a mensagem `LocationUpdate` é enviada para o *IMS* correspondente solicitando também a retransmissão da mensagem M_1. Suponhamos que após o envio de `LocationUpdate` Mss_n receba uma mensagem M_2, proveniente do mesmo

Mss inicializador. Suponhamos que M_2 é enviada logo em seguida para o Mh, o qual responde ao Mss_n. Neste momento, h_RECD[i] passaria a ter o valor "1", correspondente ao incremento após o recebimento da resposta do Mh. Desta forma, quando a mensagem M_1 chegasse em Mss_n (retransmissão do IMS), esta não seria repassada para o Mh, uma vez que $h\_RECD[\mathrm{i}] = 1$, indicando que a mensagem já havia sido recebida.

Situação (b): Imaginemos a mesma situação descrita anteriormente até o envio da mensagem `LocationUpdate` para o IMS correspondente. Suponhamos, no entanto, que neste caso a mesma mensagem M_1 chegue em Mss_n (via difusão) após o envio de `LocationUpdate` (ou seja, existe em Mss_n algum outro $Mh \in M_1.Dest$). Desta maneira, como o Mh que migrou já faz parte do grupo `LocalMhs` e também pertence a $M_1.Dest$, então a mensagem é enviada para o Mh, provocando o incremento de h_RECD[i] no momento da sua resposta. Suponhamos que quando o Mss_n recebe a retransmissão de M_1 do IMS, o *time-out* para M_1 não ocoreu, e que a resposta do Mh ainda não chegou em Mss_n. Assim, a mesma mensagem seria enviada novamente para o Mh. Notemos que, neste caso, duas respostas (do mesmo Mh) seriam processadas por Mss_n, o que implica no incremento duplo de h_RECD[i], que passaria a ter o valor "2". Desta forma, na chegada da próxima mensagem (M_2) difundida por Mss_i, o Mh não receberia tal mensagem, pois também nesta situação seu h_RECD estaria indicando que a mensagem M_2 já teria sido recebida.

Além dos casos apresentados anteriormente, certamente outras situações poderiam ocorrer em que a estrutura h_RECD no protocolo iAM^2C como a do AM^2C não fosse adequada, gerando erros na execução do protocolo. Desta maneira, a adoção de uma estrutura de dados mais apropriada para o iAM^2C (vide seção 4.6.1) tornou-se essencial.

Capítulo 7

Conclusões

Nesta dissertação foram desenvolvidos e comparados dois protocolos de multicast atômico, o AM^2C [13] e o iAM^2C [10]. Estes protocolos são do tipo *Two-Phase Commit* (2PC) [14] com uma etapa adicional de coleta de lixo. O primeiro é uma extensão do protocolo MCAST [1], enquanto que o segundo é uma variante do primeiro que utiliza um gerenciamento hierárquico de localização de unidades móveis (*Mhs*) [25] para difundir mensagens na rede fixa. Os protocolos podem servir de suporte à ações coordenadas entre grupos de dispositivos móveis, que precisem manter uma visão consistente dos dados.

No início deste trabalho o protocolo AM^2C já havia sido especificado e sua corretude comprovada analiticamente. Todavia, percebeu-se a necessidade de avaliar seu desempenho para diferentes taxas de mobilidade e outros parâmetros de configuração de rede. Ao mesmo tempo, chegou-se à conclusão de que o AM^2C sofria do problema de falta de escalabilidade em relação ao número de estações de suporte à mobilidade (*Mss*), o que serviu de motivação para o desenvolvimento de um protocolo similar que fosse escalável.

Diante destes objetivos, este trabalho foi iniciado com a implementação e avaliação do protocolo AM^2C e o projeto do iAM^2C em paralelo, concluindo com a implementação e avaliação deste último e a comparação dos dois protocolos. O AM^2C e o iAM^2C foram avaliados e validados através de simulações, usando o ambiente de prototipação e simulação *MobiCS* [5].

No protocolo AM^2C uma mensagem de multicast é difundida na rede para todos os *Mss*s do sistema, independente se há ou não algum *host* móvel local ao *Mss* que seja destinatário da mensagem. Dessa forma, em redes onde o número de *Mss*s é grande o número de mensagens na rede fixa é expressivo, o que pode comprometer o desempenho do sistema. O AM^2C ainda apresenta um desperdício de memória, visto que todos os *Mss*s armazenam cada mensagem de multicast, em alguns casos desnecessariamente.

Para solucionar a falta de escalabilidade do AM^2C em relação ao número de *Mss*s, o protocolo iAM^2C conta com um tipo especial de servidor denominado *Intermediate Multicast Server*, ou *IMS*, que tem como função armazenar e retransmitir mensagens multicast para as estações de suporte à mobilidade da rede móvel. Neste protocolo uma mensagem só é enviada pelo *IMS* caso o *Mss* possua algum *Mh* destinatário da mensagem. Os *IMSs* são elementos mais simples do que os *Mss*s, pois não tratam dos *hand-off*s e, consequentemente, não precisam interagir entre si, como os *Mss*s. O iAM^2C parte do pressuposto de que o conjunto de *IMSs* é muito menor do que o conjunto dos *Mss*s, tornando o multicast escalável para redes com muitos *Mss*s.

Todavia, a propriedade de atomicidade garantida pelos dois protocolos faz com que tanto o AM^2C como o iAM^2C não sejam escaláveis em relação ao número de *Mhs* no sistema. Em redes com muitos clientes móveis, torna-se "caro" chegar a um consenso, uma vez que são necessárias muitas mensagens na rede fixa para se chegar a uma decisão sobre o processamento conjunto de uma ação.

Também foi visto que no iAM^2C a gerência de localização de unidades móveis (desempenhada pelos *IMSs*) representa um custo adicional para o protocolo. Desta forma, a depeder da taxa de migração dos *Mhs* e do número de *Mss*s, a redução no número de mensagens difundidas na rede fixa não compensa o custo das atualizações de mapeamento.

As simulações mostraram que nenhum dos protocolos é mais eficiente do que o outro em todas as situações, e que os cenários é que determinam qual dos protocolos é mais ou menos adequado. De forma geral, o AM^2C se adequa melhor a redes com poucos *Mss*s, enquanto que o iAM^2C é mais indicado para redes com muitos *Mss*s. Ambos os protocolos possuem um baixo custo na rede sem fio e uma porcentagem relativamente alta de multicasts abortados quando a taxa de migração dos *Mhs* é alta. Durante as simulações não houve a preocupação em determinar uma taxa (probabilidade) realista de migração para os *Mhs*, mas sim apenas comparar o desempenho dos protocolos para situações hipotéticas.

7.1 Trabalhos Futuros

Imaginamos que multicast confiável/atômico para redes sem fio continua sendo uma área para muitas pesquisas, e que *MobiCS* pode ser uma ferramenta valiosa para a prototipação e análise de vários protocolos.

Como trabalho futuro, pensamos em extensões dos protocolos AM^2C e iAM^2C que tornem possível o ajuste dinâmico do tempo máximo esperado por um *Mss* por respostas de seus *Mhs* locais. Desta forma, como o valor dos *time-outs* influencia diretamente no número de multicasts abortados, a depender da carga da rede ou da aplicação, este valor poderia ser mais ou menos estendido.

Outro possível trabalho futuro consiste no desenvolvimento de uma variante do iAM^2C onde os *Mss*s possam manter as mensagens de multicasts em seus *caches* por um período determinado de tempo. Com isso, procuramos diminuir a sobrecarga na rede fixa do protocolo, mantendo sua escalabilidade em relação ao número de *Mss*s no sistema.

Além disto, algumas possíveis direções do trabalho podem ser: adaptação de protocolos para multicast confiável em redes convencionais para redes sem fio, analisar sua complexidade e compará-la com o iAM^2C; definir protocolos para multicast com garantias de entrega menos rigorosas, como por exemplo, entrega para pelo menos k membros do grupo, dentre outras possibilidades; multicasts em redes *ad-hoc*.

Apêndice A

Diagrama de Classes no AM^2C

A seguir apresentamos o diagrama de classes (com os respectivos métodos públicos) do protocolo AM^2C.

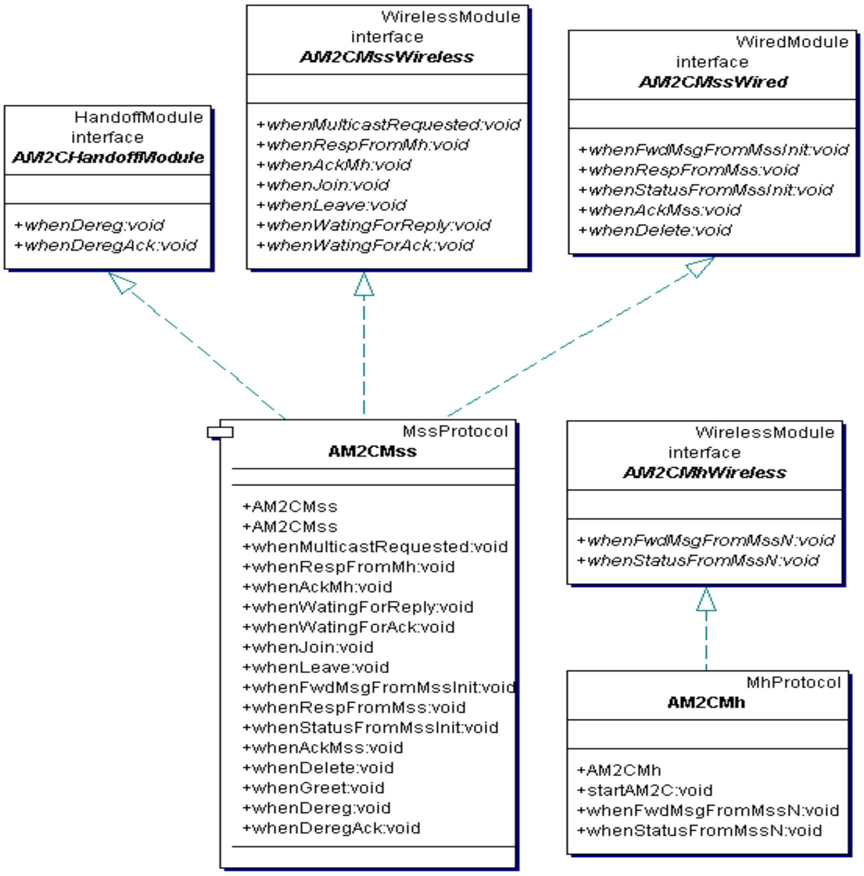

Figura A.1: Diagrama de Classes do AM^2C.

Apêndice B

Diagrama de Classes no iAM^2C

A seguir apresentamos o diagrama de classes (com os respectivos métodos públicos) do protocolo iAM^2C.

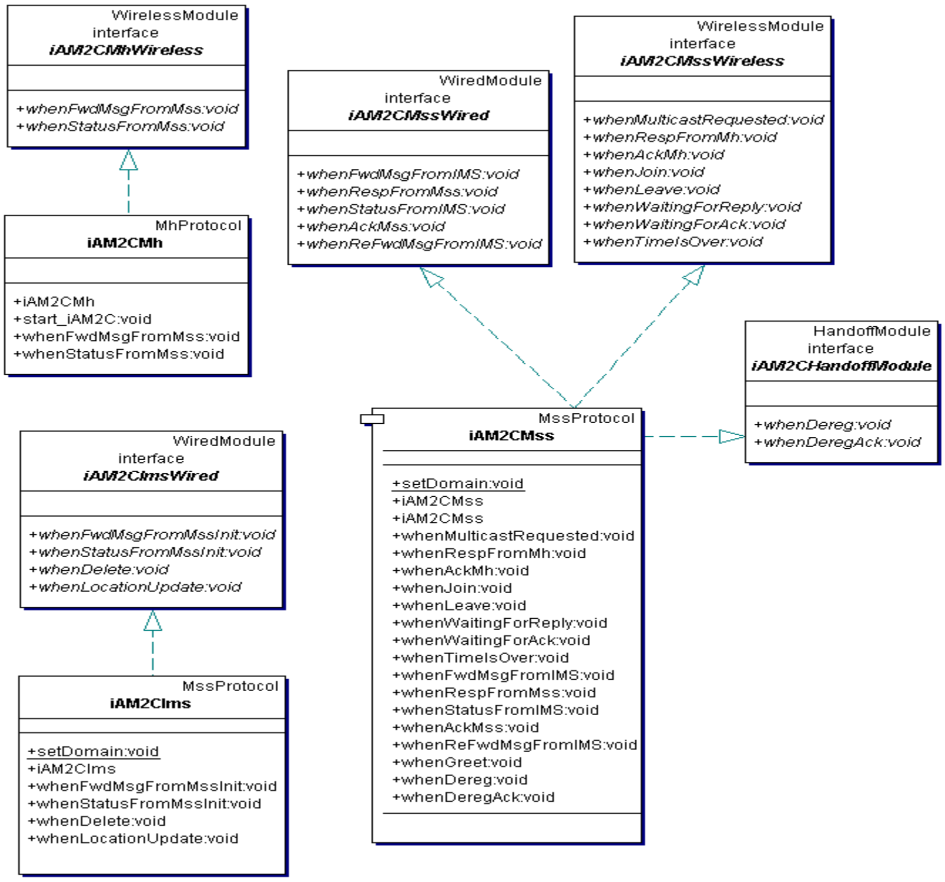

Figura B.1: Diagrama de Classes do iAM^2C.

Apêndice C

Estruturas de Dados e Tratamento de Mensagens no AM^2C

C.1 Introdução

O funcionamento de um protocolo prototipado no simulador *MobiCS* [5, 6] se dá basicamente através de trocas de mensagens. É por meio do tratamento dado a cada uma dessas mensagens que é definido o comportamento do protocolo. Neste documento apresentamos as estruturas de dados que foram utilizadas e os algoritmos desenvolvidos para o funcionamento do protocolo de multicast atômico AM^2C [13].

É importante dizer que estamos considerando dois tipos de elemento de rede, que são os *Mh*s e os *Mss*s. Entretanto, para um melhor entendimento do funcionamento do protocolo, dividimos logicamente os *Mss*s em Mss_{init} e Mss_{part}, de acordo com o *papel* que um mesmo *Mss* assume no sistema. O primeiro refere-se ao *Mss* inicializador de uma mensagem de multicast; o segundo refere-se a um *Mss* participante no processamento de uma mensagem.

Dessa forma, a próxima seção define algumas estruturas de dados utilizadas pelo protocolo. Na seção C.3 encontram-se as definições e o conteúdo de todas as mensagens enviadas durantes as três fases do AM^2C, bem como durante os *hand-off*s. Por fim, a seção C.4 apresenta o tratamento dado (algoritmo) pelo protocolo para cada uma das mensagens descritas na seção C.3.

C.2 Estruturas de Dados

Nesta seção encontram-se as principais estruturas de dados, juntamente com seus respectivos significados, utilizadas pelo protocolo AM^2C. Considere M como uma mensagem em processo de difusão iniciada por Mss_{init} para um conjunto de Mss_{part} (no caso do AM^2C, todos os *Mss*s do sistema). Cada mensagem desse tipo possui um grupo de destinatários que chamaremos de $M.Dest$.

Vale ressaltar ainda que algumas dessas estruturas estão relacionadas, independentemente, a cada mensagem de multicast difundida, ficando isso subtendido. São elas:

C.2.1 Estruturas de Dados Gerais

1. **MsgId**: Uma mensagem é identificada unicamente através de um objeto da classe *MsgId*. Este objeto contém dois atributos: o endereço do *Mss* inicializador do multicast (Mss_{init}) e um número de seqüência gerado pelo mesmo.

2. **ObjReply**: Esta classe implementa um *objeto-resposta* composto pelo endereço do elemento de rede responsável pela resposta e pela resposta em si (`Ok/NOk`).

3. **ObjMsgManager**: Esta classe implementa um objeto da estrutura *MsgManager* (vide seção *Estruturas de Dados em* Mss_{part}), que serve para um *Mss* gerenciar as ações que devem ser tomadas em relação à chegada de uma mensagem. Um *ObjMsgManager* é composto por sete atributos booleanos:
   - *podeProcessar*: indica se um evento de *time-out* na primeira fase do protocolo e associado a uma determinada mensagem M pode ou não ser processado;
   - *tmoutF1Proc*: indica se um evento de *time-out* associado a uma mensagem já foi ou não processado na primeira fase do protocolo;
   - *tmoutF2Proc*: o mesmo, só que para a segunda fase do protocolo;
   - *deleteEnviado*: indica se a mensagem `Delete` referente a um multicast M já foi ou não enviada;
   - *statusDefinido*: indica se o estado final de um multicast já foi ou não definido pelo Mss_{init};
   - *tmoutT1Iniciado*: indica se o *time-out* relacionado a uma mensagem de multicast na primeira fase do protocolo já foi ou não iniciado;
   - *tmoutT2Iniciado*: o mesmo especificado no item anterior, só que vale para a segunda fase do protocolo;

C.2.2 Estruturas de Dados em Mss$_{init}$:

1. **mseq**: variável inteira que armazena o número de seqüência da próxima mensagem de multicast a ser difundida por um dado *Mss* inicializador (Mss_{init}). O *mseq* é um dos atributos que compõem um objeto *MsgId*, como descrito na seção C.2.1.

2. **RepliesFromMss**: tabela *hash* contendo as respostas dos *Mss*s referentes às mensagens de multicast difundidas. Seja *M* uma tal mensagem e *id* sua identificação (um objeto *MsgId*). Cada posição *RepliesFromMss*[*M.id*] retorna um conjunto de respostas (`Ok`/`NOk`) referentes à mensagem *M*.

   Obs.: Utilizaremos no restante deste apêndice o termo *M.id* (ou equivalente) para indexar a posição referente a uma mensagem de multicast *M* em uma estrutura de dados como uma tabela *hash* ou vetor. Todavia, deve-se ter em mente que este índice (tal como outros atributos de uma mensagem) é, na verdade, obtido por um método *get*, tal como *getMsgId()*. De forma similar, deve-se considerar um método *put* para atribuir um valor em uma certa posição de uma estrutura (por exemplo em *RepliesFromMss*[*M.id*] := *valor*). Estas simplificações também serão bastante utilizadas nos algoritmos da seção C.4.

3. **DestM**: tabela *hash* onde cada posição *DestM*[*M.id*] armazena o grupo de destinatários da mensagem *M*.

4. **GConfirmados**: tabela *hash* indexada pelo *id* da mensagem. Cada posição da tabela contém o conjunto de *Mhs* que confirmaram o recebimento do estado final de uma mensagem multicast.

C.2.3 Estruturas de Dados em Mss_{part}:

1. **LocalMhs**: grupo de *Mhs* pertencentes à célula que está sob a responsabilidade de Mss_{part}.

2. **h_RECD**: Esta estrutura (*hash*) é única para cada *Mh* do sistema e está presente no *Mss* responsável pela célula na qual o *Mh* se encontra. Contém n posições, onde n corresponde ao número total de *Mss*s no sistema. Cada posição $i, 0 \leq i < n$, onde $h\_RECD[\text{i}] = j$, corresponde à j-ésima mensagem ($mseq = j$) recebida pelo *Mh* e difundida por Mss_i.

3. **Buffer**: fila que armazena as mensagens de multicast difundidas por um *Mss* inicializador (Mss_{init}). Só é usada durante a primeira fase do protocolo.

4. **Outcomes**: tabela *hash* que armazena o estado final de cada mensagem de multicast difundida até então. É indexado por *M.id*.

5. **RepliesFromMhs**: tabela *hash* de respostas (`Ok`/`NOk`) dos *Mhs* locais à Mss_{part} referente a todas as mensagens já transmitidas pelo *Mss*. Seja *M* uma tal mensagem. Cada posição *RepliesFromMhs*[*M.id*] retorna um vetor que contém um conjunto de objetos do tipo *ObjReply*. Cada objeto desse tipo é composto pelo endereço do *Mh* e por sua resposta à mensagem *M*.

6. **AckMhs**: tabela *hash* usada para armazenar o grupo de *Mhs* locais e destinatários de uma mensagem que confirmaram o recebimento do estado final do multicast. O *M.id* é utilizado como índice.

7. **MsgManager**: tabela *hash* onde cada posição indexada por *M.id* (para uma mensagem de multicast *M*) contém um objeto do tipo *ObjMsgManager*. É por meio desta estrutura que o *Mss* gerencia diversas ações relativas a uma mensagem, tais como o processamento ou não da mensagem, do evento de *time-out* a ela associado, entre outras (como poderá ser observado no tratamento das mensagens na seção C.4).

8. **DestMLocal**: conjunto de *Mhs* locais e destinatários de uma determinada mensagem de multicast.

C.3 Definição das Mensagens

Nesta seção encontram-se as definições e o conteúdo das mensagens segundo o elemento de rede que as processa. Considere *M* como sendo genericamente qualquer uma das mensagens a seguir. *M* contém sempre uma identificação (*M.id*) e um conteúdo que, de forma geral, é alguma informação relevante para o sistema.

É importante observar que quando utilizamos uma variável *M'* (ou *M"*) no lugar da variável *M* para referenciar uma mensagem, deve-se ao fato de que *M* e *M'* (ou *M"*) são sintaticamente diferentes. Todavia, *M'* /*M"* sempre conterá a mesma identificação de *M* (*M'.id* = *M.id*). Ou seja, todas as mensagens referem-se a uma mesma mensagem original, só que contêm atributos diferentes. Esta observação também é valida para a seção C.4.

C.3.1 Mensagens Tratadas por Mss_{init}:

1. **M = MulticastRequested**(*Mh mh, Group dest, int content*):

Mensagem de multicast requisitada por um *Mh*. Sempre enviada para o *Mss* responsável pela célula a qual o *Mh* se encontra no momento do envio da mensagem (Mss_{Mh}). Contém o identificador do *Mh* requisitante, o grupo de destinatários (*M.dest*) e o conteúdo da mensagem.

2. **M = RespFromMss**(*MsgId id, int reply*):

   Resposta proveniente de um *Mss* a respeito de uma determinada mensagem *M'*. Esta resposta refere-se à disponibilidade e aceitação de *M'* por todos os *Mh*s locais ao *Mss* e que pertencem a *M'.Dest*. Contém a identificação da mensagem (onde *M.id = M'.id*) e a resposta, *M.reply* (`Ok/NOk`).

3. **M = AckMss**(*MsgId id, Group C*):

   Mensagem enviada de um *Mss* para o Mss_{init} de uma determinada mensagem *M'*. É composta pela identificação da mensagem (*M.id = M'.id*) e pelo grupo de *Mh*s locais e pertencentes a *M'.Dest* que confirmaram o recebimento do estado final de *M'*.

C.3.2 Mensagens Tratadas pelo Mss_{part}:

1. **M = FwdMsgFromMss**$_{init}$(*MsgId id, Group dest, int content*):

   Recebimento de uma mensagem de multicast *M* inicializada por um Mss_{init}. *M* deve ser retransmitida para todo $Mh \in$ (*Mss.LocalMhs* $\cap$ *M.Dest*). Contém o identificador da mensagem (*M.id*), o grupo de destinatários (*M.Dest*) e o conteúdo (*M.content*).

2. **M = RespFromMh**(*Mh mh, MsgId id, int reply*):

   Resposta de um *Mh* para seu Mss_{Mh} sobre a aceitação ou não de uma mensagem recebida (chamemos de *M'*). Contém a identificação do *Mh*, da mensagem (onde *M.id = M'.id*) e a resposta, *M.reply* (`Ok/NOk`).

3. **M = WaitingForReply**(*Timer T1*):

   Ocorrência do evento de *time-out* no *Mss*. Refere-se ao término do tempo máximo esperado pelo Mss_{part} por respostas de seus *Mhs* locais e destinatários de uma mensagem (associada ao *timer* T1) enviada durante a primeira fase do protocolo.

4. **M = StatusFromMss**$_{init}$(*MsgId id, Group dest, char Status*):

   Recebimento do estado final de um multicast enviado pelo *Mss* inicializador (Mss_{init}) para todos os *Mss*s do sistema. Contém a identificação da mensagem (*M.id*), o grupo de *Mhs* destinatários (*M.dest*) e o estado final ("A" = "*Aborted*"; "C" = "*Commited*").

5. **M = AckMh**(*MsgId id, Mh mh*):

   Recebimento da confirmação de um *Mh* sobre o recebimento do estado final de um multicast. Contém o identificador da mensagem e do *Mh*.

6. **M = WaitingForAck**(*Timer T2*):

   Ocorrência do evento de *time-out* no *Mss*. Refere-se ao término do tempo máximo esperado pelo Mss_{part} por confirmações de seus *Mhs* locais e destinatários de uma mensagem (associada ao *timer* T2) enviada durante a segunda fase do protocolo. Essas confirmações dizem respeito ao recebimento do estado final de um multicast.

7. **M = Delete**(*MsgId id*):

   Recebimento da solicitação do *Mss* inicializador de um multicast *M'* para que todos os *Mss*s removam *M'* de seus respectivos *Outcomes*. Contém apenas a identificação da mensagem (*M'.id*).

8. **M = Greet**(*Mh mh, Mss* Mss_o):

   Mensagem enviada por um *Mh* a um Mss_{part} toda vez que o *Mh* entra na célula que está sob a responsabilidade de Mss_{part}, inicializando assim o protocolo de *hand-off*. Contém a identificação do *Mh* e do *Mss* responsável pela célula da qual o *Mh* migrou (Mss_o).

9. **M = Dereg**(*Mh mh, Mss* Mss_n):

   Solicitação de desvinculação de um *Mh* de sua antiga célula. É enviada do *Mss* responsável pela célula para a qual o *Mh* está migrando, para o *Mss* responsável pela célula da qual o *Mh* migrou. Contém a identificação do *Mh* e do novo *Mss* (Mss_n). Esta mensagem faz parte do protocolo de *hand-off*.

10. **M = DeregAck**(*Mh mh, hashtable* h_RECD):

    Confirmação de desvinculação de um *Mh* de sua antiga célula. Enviada do *Mss* da antiga célula para o *Mss* responsável pela nova célula. Contém a identificação do *Mh* e o *h_RECD* relacionado ao *Mh*. Também faz parte do protocolo de *hand-off*.

11. **M = Join**(*Mh mh*):

    Solicitação de entrada no sistema por parte de um *Mh*. Contém a identificação do *Mh*.

12. **M = Leave** (*Mh mh*):

    Solicitação de saída do sistema de um *Mh* para seu Mss_{Mh}. Contém a identificação do *Mh*.

C.3.3 Mensagens Tratadas por um Mh:

1. **M = FwdMsgFromMss**$_n$(*MsgId id, int content*):

   Recebimento de uma mensagem de multicast enviada pelo *Mss* responsável pelo *Mh* (Mss_{Mh}). Armazena a identificação da mensagem ($M.id$) e o conteúdo.

2. **M = StatusFromMss**$_n$(*MsgId id, char status*):

   Recebimento do estado final de um multicast enviado pelo *Mss* responsável pelo *Mh* (Mss_{Mh}). Contém a identificação da mensagem ($M.id$) o estado final ("C" = "*Commited*"; "A" = "*Aborted*").

C.4 Tratamento das Mensagens

Esta seção apresenta um pseudo-código referente ao tratamento de cada uma das mensagens descritas na seção anterior. Entretanto, antes de iniciarmos faremos algumas considerações:

1. Mais uma vez faremos uma abordagem por elemento de rede envolvido;

2. Faz parte da sintaxe do *MobiCS* [5, 6] o uso da palavra *"When"* antes do nome da mensagem, indicando o tratamento que é dado a uma mensagem quando esta é recebida por um elemento de rede como um *Mss* ou *Mh*.

3. Considere as tabelas C.1 e C.2 que indicam os principais métodos utilizados no tratamento das mensagens.

4. Os eventos de *time-out* `WhenWaitingForReply(`$< timer - event >$`)` e `WhenWaitingForAck(`$< timer - event >$`)` indicam o término do tempo máximo esperado por um *Mss* por uma resposta referente a uma mensagem *M* (associada à $< timer - event >$) de seus respectivos *Mh*s t.q. $Mh \in$ (`LocalMhs` $\cap$ $M.Dest$). No primeiro caso, espera-se uma resposta do tipo `Ok/NOk` e, no segundo, um *acknowledgment* de um *Mh*. As principais ações tomadas quando um evento como estes ocorre estão indicadas na tabela C.3.

Método	**Significado**
New(<MsgType>)	Criação de uma nova mensagem do tipo <MsgType>.
send(elemento, M)	Envio da mensagem M para elemento.
M.bcast	Difusão da mensagem M para todos os *Mss*s.
Object[Mid].put(anObject)	Insere o objeto anObject na posição indexada por Mid do objeto Object.
Object[Mid].add(anObject)	Adiciona o objeto anObject ao objeto Object[Mid].
Object[Mid].remove()	Remove o objeto da posição indexada por Mid.
Object.size()	Retorna o número de posições (tamanho) de Object.
msgNotHandled(Mh, Mid)	Retorna TRUE caso uma mensagem M não tenha sido recebida pelo *Mh* e nenhum *Mss* considerou sua resposta como NOk. Retorna FALSE no caso contrário. Esse teste é feito de acordo com o número de seqüência armazenado em *h_RECD[M.MssInit]* para o *Mh* em questão (o *Mss* inicializador pode ser obtido pelo objeto Mid).
Inc-hRECD(Mh, Mid)	Indica que a mensagem já foi recebida (ou tratada) pelo *Mh*. Isto é feito através do incremento do número de seqüência contido em *h_RECD[M.MssInit]* para o *Mh* em questão.
gethRECD(Mh)	Devolve o *h_RECD* relacionado ao Mh.

Tabela C.1: Principais Métodos Utilizados no Tratamento das Mensagens no AM^2C.

5. Existem alguns eventos tais como: *connect, disconnect, move, leave* e *join* que podem ocorrer em um *Mh*. Os dois últimos são ações voluntárias do usuário que estaria desligando/ligando seu dispositivo móvel. A ocorrência destes eventos é determinada por probabilidades ou por comandos explícitos do usuário.

6. Para uma melhor compreensão dos algoritmos, as variáveis serão expressas da seguinte forma: parâmetros, estruturas de dados ou mensagens serão expressos da mesma forma como descritos nas seções anteriores, porém em *itálico* (ex. "*Par1*","*LocalMhs*"); variáveis temporárias serão escritas em minúsculo (ex. "aux").

7. Considere *TOTAL-MSSs* como uma variável que armazena o número total de *Mss*s no sistema.

8. O termo *owner* corresponde a uma referência para o *Mss/Mh* que está processando a mensa-

Método	**Significado**
timerEvent.getMsg()	Retorna a mensagem associada à timerEvent.
MsgManager[Mid].setInfoAboutDeleteSent(TRUE/FALSE)	Indica se a mensagem Delete referente a uma mensagem cujo identificador é Mid já foi ou não enviada. Este método é utilizado pela estrutura *MsgManager*.
setTimer(<timer-event>, <repeat-order>, <time-to-elapsed>)	Inicia um temporizador para uma mensagem associada à <timer-event>. <repeat-order> (*"ONCE" ou "REPEATING"'*) indica se o evento será gerado apenas uma vez ou repetidamente. <time-to-elapsed> refere-se ao tempo (em UTs) no qual o evento deverá ser gerado ou o período de repetição.
MsgManager[Mid].getPermissionToProcess()	Retorna TRUE caso o evento de *time-out* associado à mensagem M pode ser processado. Retorna FALSE no caso contrário.
MsgManager[Mid].setPermissionToProcess(TRUE/FALSE)	Indica se o evento de *time-out* associado a uma mensagem cujo identificador é Mid pode ou não ser processado. Este método é utilizado pela estrutura *MsgManager*.
MsgManager[Mid].getInfoAboutEventProcessedInF1/F2()	Devolve TRUE se o evento de *time-out* da Fase I/II referente à mensagem indexada por Mid já foi processado. Retorna FALSE no caso contrário.
MsgManager[Mid].setInfoAboutEventProcessedInF1/F2 (TRUE/FALSE)	Indica que o evento de *time-out* associado à mensagem M na Fase I/II do protocolo já foi ou não processado.

Tabela C.2: Principais Métodos Utilizados no Tratamento das Mensagens no AM^2C.

gem.

9. Nas próximas subseções, considere M como sendo cada uma das mensagens tratadas;

C.4.1 Mensagens Tratadas por Mss_{init}:

1. **WhenMulticastRequested**(*Mh, Dest, Content*):

   (a) *mseq* := *mseq* + 1;

   (b) mid := New *MsgId*(*owner*, *mseq*); */* Identificador da mensagem. */*

   (c) *M'* := New *FwdMsgFromMss*$_{init}$(mid, *Dest*, *Content*);

   (d) *DestM*[mid] := Dest;

Evento	**Ações** (Seja M uma mensagem associada a $< timer-event >$)
WhenWatingForReply($< timer - event >$)	**Se** LocalMhs $\cap$ $M.Dest = \emptyset$ **ou** $\forall$ $Mh \in$ (LocalMhs $\cap$ $M.Dest$), a resposta referente à M é Ok, **então** envie um $\texttt{Ok}_M$ para Mss_{init}. **Senão** envie um $\texttt{NOk}_M$.
WhenWatingForAck($< timer - event >$)	Seja C o grupo de *Mhs* locais que confirmaram o recebimento de *M*. Envie um *acknowledgment* (Ack_M) com o grupo C para o Mss_{init} relativo à *M*.

Tabela C.3: Eventos ocorridos em um *Mss* no AM^2C.

(e) *GConfirmados*[mid] := {};

(f) *M'*.bcast; */* Difusão para todos os Msss. */*

2. **WhenRespFromMss**(*Mid, Reply*):
Seja *resp* uma resposta do tipo "OK"/"NOK".

   (a) *RepliesFromMss*[*Mid*].add(*reply*);

   (b) **Se** *RepliesFromMss*[*Mid*].size() = *TOTAL-MSSs* **então**
   / Todos os Mss ja responderam. */*
   status := "C"; */* Commited */*
   $\forall$ *resp* $\in$ *RepliesFromMss[Mid]* **faça:**
   Se *resp* = "NOK"**então**
   status := "A"; */* Aborted */*
   dest := (Group) *DestM*[*Mid*]; **/ destinatarios de M */*
   M' := New $StatusFromMss_{init}$(*Mid*, dest, status);
   M'.bcast;

3. **WhenAckMss**(*Mid, C*):

   (a) $\forall$ *Mh* $\in$ *C* **faça:**
   GConfirmados[*Mid*].add(*Mh*);

   (b) **Se** *GConfirmados*[*Mid*].size() = *DestM*[*Mid*].size() **então**
   / Todos os Mhs $\in$ M.Dest ja confirmaram o recebimento do status de M */*
   MsgManager[Mid].setInfoAboutDeleteSent(TRUE);
   M' := New *Delete*(*Mid*);
   M'.bcast;

C.4.2 Mensagens Tratadas por Mss_{part}:

1. **WhenFwdMsgFromMss**$_{Init}$(*Mid, Dest, Content*):
Seja *M* a *i*-ésima mensagem difundida por uma dado Mss_{init}.

   (a) *Buffer*.add(*M*);

(b) Seja N = {*Mh* t.q. *Mh* $\in$ (*LocalMhs* $\cap$ *M.Dest*)}

```
Se N = ∅ então
  M' := New RespFromMss(Mid,"Ok");
  send(M.Mss_init, M');
Senão
  M' := New FwdMsgFromMss_n(Mid, Content);
  ∀ Mh ∈ N faça:
    Se msgNotHandled(Mh, Mid) então
      send(Mh, M'); /* Mh ainda não recebeu a mensagem. */
  /* Inicia um temporizador para um evento de timeout associado a M'. */
  setTimer(New WaitingForReply(M'), ONCE, T1);
  /* Indica que o evento de timeout associado a M' pode ser processado. */
  MsgManager[Mid].setPermissionToProcess(TRUE);
```

2. **WhenRespFromMh**(*Mh, Mid, Reply*):
   / Verifica se o evento de time-out na 1a.fase para a mensagem ainda nao foi processado. */*

   (a) **Se** *MsgManager*[Mid].getInfoAboutEventProcessedInF1() = FALSE **então**

   ```
     /* Guarda a resposta do Mh. */
     RepliesFromMhs[Mid].add(Reply);
   ```

   (b) *Inc-hRECD(Mh, Mid)*;

3. **WhenWaitingForReply**(*TimerEvent*):
   Seja *resp* uma resposta de um *Mh* do tipo "Ok"/"NOk".

   (a) *M'* := *TimerEvent*.getMsg(); */* Mensagem associada ao time-out. */*

   (b) **Se** *MsgManager*[M'.id].getPermissionToProcess() **então**

   ```
     Seja N = {Mh t.q. Mh ∈ (LocalMhs ∩ M.Dest)}
     Se N = ∅ então
       reply := "OK";
     senão
       /* Testa se TODOS os Mhs ja responderam. */
       Se RepliesFromMhs[M'.id].size() = N.size() então
         reply := "Ok";
         ∀ resp ∈ RepliesFromMhs[M'.id] faça:
           Se resp = "NOk"então
             reply := "NOK";
       Senão
         reply := "NOk";
       M" := New RespFromMss(M'.id, reply);
       send(M'.Mss_init, M");
       /* Indica que o time-out ja foi processado na fase I. */
       MsgManager.setInfoAboutEventProcessedInF1(TRUE);
   Senão
     /* Proximo evento de time-out deve ser processado. */
     MsgManager[M'.id].setPermissionToProcess(TRUE);
   ```

4. **WhenStatusFromMss$_{init}$**(*Mid, Dest, Status*):
   Seja *M* a *i*-ésima mensagem de status difundida por um *Mss$_{init}$*.

(a) *Outcomes*[*Mid*] := *Status*;

(b) *Buffer*.remove(Mid);

(c) *M'* := New $StatusFromMss_n$(*Mid, Status*);
∀ *Mh* ∈ (*LocalMhs* ∩ *M.Dest*) **faça:**
send(*Mh, M'*);

(d) /* *Inicia um temporizador para um evento de timeout associado a M'.* */
setTimer(New *WaitingForAck*(*M'*), ONCE, T2);

5. **WhenAckMh**(*Mid, Mh*):
/* *Testa se o evento de time-out ja foi processado. Neste caso, a mensagem AckMss{Mh} (Ack extra) deve ser enviada para o Mss inicializador do multicast.* */

(a) **Se** *MsgManager*[Mid].getInfoAboutEventProcessedInF2() = TRUE **então**
M' := New *AckMss(Mid, Mh)*;
send(*M.*Mss_{init}, M');
senão
/* *Adiciona o Mh no grupo de Mhs que ja enviaram um acknowledjment para M.* */
AckMhs[*Mid*].add(*Mh*);

6. **WhenWaitingForAck**(*TimerEvent*):

(a) *M'* := TimerEvent.getMsg();

(b) *M"* := New *AckMss*(*M'.id*,*AckMhs*[*M'.id*]);

(c) send(*M'.*Mss_{init}, *M"*);

(d) /* *Remocao do Mhs que ja responderam a M'.* */
∀ *Mh* ∈ *AckMhs*[*M'.id*] **faça**
AckMhs[*M'.id*].remove(*Mh*);

(e) *MsgManager*[*M'.id*].setInfoAboutEventProcessedInF2(TRUE);

7. **WhenDelete**(*Mid*):

(a) *Outcomes*[*Mid*].remove(); /* *Remocao de M de Outcomes.* */

8. **WhenGreet**(Mh_{new}, Mss_o):

(a) /* *Testa se o Mh esta entrando agora no sistema.* */
Se $Mss_o \neq$ null **então**
/* *Testa se o Mh nao voltou de um periodo de inativacao.* */
Se $Mss_o \neq$ *owner* **então**
/* *Envio da mensagem Dereg do Mss responsavel pela celula atual para* Mss_o *solicitando o desligamento de* Mh_{new} *de sua antiga celula.* */
M' := New *Dereg*(Mh_{new}, *owner*);
send(Mss_o, *M'*);
Senão
/* *Varredura do Buffer.* */
∀ *M'* ∈ *Buffer* **faça**:
Se *MsgManager*[*M'.id*].getInfoAboutEventProcessedInF1() = FALSE **então**
Se $Mh_{new} \notin$ *M'*.Dest e msgNotHandled(Mh_{new}, *M'.id*) **então**

```
            Inc-hRECD(Mh_new, M'.Mss_init);
          Senão
            Se msgNotHandled(Mh_new, M'.id) então
              send(Mh_new, M');
      /* Varredura de Outcomes. */
      ∀ M' ∈ Outcomes faça:
        Se Mh_new ∈ M'.Dest então
          /* Retransmissao do status. */
          send(Mh_new, M');
  Senão
    /* Solicitacao de registro no sistema. */
    M' := New Join(Mh_new);
    send(owner, M');
```

9. **WhenDereg**(Mh_{new}, Mss_n):
Esta mensagem é processada em Mss_o.

(a)
```
/* Remocao de Mh_new do grupo LocalMhs. */
LocalMhs.remove(Mh_new);
```

(b)
```
/* Aborta (indiretamente) todas as mensagen que o Mh e destinatario e ainda
nao respondeu. */
∀ M' ∈ Buffer t.q. (Mh_new ∈ M'.Dest) e (Mh_new ∉ RepliesFromMhs[M'.id]) faça:
  M" := New RespFromMss(M'.id, "NOk");
  send(M'.Mss_init, M"); /* Mss inicializador de M'. */
  Inc-hRECD(Mh_new, M'.Mss_init); /* Indicara que M' ja foi tratada. */
  /* Indica que o proximo evento de time-out nao deve ser processado. */
  MsgManager[M'.id].setPermissionToProcess(FALSE);
```

(c)
```
/* Envio da confirmacao de desvinculacao do Mh para Mss_n. */
M' := New DeregAck(Mh_new, gethRECD(Mh_new);
send(Mss_n, M');
```

10. **When DeregAck**(Mh_{new}, $hRECD$):

(a)
```
LocalMhs.add(Mh_new);
```

(b)
```
∀ M' ∈ Buffer faça:
  Se Mh_new ∈ M'.Dest então
    /* Testa se o evento de time-out associado a M' ja foi processado. */
    Se MsgManager[M'.id].getInfoAboutEventProcessedInF1() = FALSE) então
      Se msgNotHandled(Mh_new, M'.id) então
         /* Retransmissao de M' para o Mh. */
        send(Mh_new, M'.id);
         /* Prorrogacao do time-out. */
        setTimer(New WaitingForReply(M'), ONCE, T1);
         /* Indica que o evento de time-out anterior nao deve ser processado.*/
        MsgManager.setPermissionToProcess(FALSE);
    Senão
      Se msgNotHandled(Mh_new, M'.id) então
        /* Envio do "NOk"por parte do Mss. */
```

```
            reply := "NOk";
            M":= New RespFromMss(M'.id, reply);
            send(M'.Mss_init, M");
            Inc-hRECD(Mh_new, M'.Mss_init);
      Senão
        Se msgNotHandled(Mh_new, M'.id) então
          Inc-hRECD(Mh_new, M'.Mss_init);
```

(c) /* *Varredura de Outcomes.* */
$\forall$ *M'* $\in$ Outcomes faça:
  Se $Mh_{new} \in$ *M'.Dest* **então**
    /* *Retransmissao do estado final.* */
    send(Mh_{new}, *M'*);
    setTimer(New *WaitingForAck*(*M'*), ONCE, T2);

11. **WhenJoin**(Mh_{new}):

    (a) *LocalMhs*.add(Mh_{new}); /* *Adiciona* Mh_{new} *no grupo de Mhs locais.* */

12. **WhenLeave**(*Mh*):

    (a) *LocalMhs*.remove(*Mh*); /* *Remove Mh do grupo de Mhs locais.* */

C.4.3 Mensagens Tratadas no Mh:

1. **WhenFwdMsgFromMss**$_n$(*Mid, Content*):

    (a) **Se** a aplicacao e capaz de processar a mensagem **então**
        reply := "Ok";
      Senão reply := "NOk";

    (b) *M'* := New *RespFromMh*(*owner*, *Mid*, reply);

    (c) /* *Envia resposta para o* Mss *resposavel pelo Mh.*/*
      send(Mss_{owner}, *M'*);

2. **WhenStatusFromMss**$_n$(*Mid, Status*):
   /* *Acknowledgment enviado para* Mss *responsável pelo Mh confirmando o recebimento do estado final de uma mensagem.* */

    (a) *M'* := New *AckMh*(*owner, Mid*);

    (b) send(Mss_{owner}, *M'*);

Apêndice D

Estruturas de Dados e Tratamento de Mensagens no iAM^2C

D.1 Introdução

Este apêndice tem como objetivo apresentar as estruturas de dados, mensagens e o tratamento dado a estas pelos elementos de rede envolvidos durante o funcionamento do protocolo iAM^2C[10], através do uso do simulador *MobiCS* [5, 6].

Estamos considerando três tipos de elementos de rede, que são os *Mhs*, os *Mss*s e os *IMSs*. Assim como fizemos no apêndice C, dividimos logicamente os *Mss*s em Mss_{init} e Mss_{part}, de acordo com o *papel* que um mesmo *Mss* assume no sistema.

A próxima seção define as principais estruturas de dados utilizadas pelo protocolo. A seção D.3 apresenta e define o conteúdo de todas as mensagens enviadas durantes as três fases do iAM^2C, bem como durante os *hand-off*s. Finalmente, na seção D.4 encontra-se o tratamento dado pelo protocolo para cada uma das mensagens descritas na seção D.3.

D.2 Estruturas de Dados

Nesta seção encontram-se as principais estruturas de dados, juntamente com suas respectivas utilidades, usadas pelo protocolo iAM^2C. Seja M uma mensagem de multicast difundida por um *Mss* inicializador (Mss_{init}) para um conjunto de Mss_{part}. Cada mensagem desse tipo possui um grupo de destinatários que chamaremos de $M.Dest$.

Muitas das estruturas de dados que serão apresentadas são as mesmas utilizadas pelo protocolo AM^2C (e que já foram vistas). Nestes casos, indicaremos com um (*) tais estruturas, apresentando apenas um resumo das mesmas.

D.2.1 Estruturas de Dados Gerais

1. **MsgId (*)**: identificador de uma mensagem composto pelo endereço do *Mss* inicializador do multicast e de um número de seqüência gerado pelo mesmo.

2. **ObjReply (*)**: Resposta de um elemento de rede referente a uma mensagem de multicast.

3. **ObjMsgManager (*)**: Classe que implementa um objeto da estrutura *MsgManager* (vide seção D.2.3) e que contém sete atributos *booleanos*. Estes atributos, que estão relacionados a uma mensagem de multicast, permitem que o *Mss* gerencie as ações que devem ser tomadas na chegada de uma mensagem.

4. **IMSAddress**: vetor com os endereços de todos os *IMSs* do sistema.

D.2.2 Estruturas de Dados em Mss$_{init}$:

1. **mseq (*)**: número de seqüência da próxima mensagem de multicast a ser difundida pelo *Mss* inicializador (Mss_{init}).

2. **RepliesMhSet**: tabela *hash* indexada pelo identificador de uma mensagem M (*M.id* - objeto *MsgId*). Cada posição *RepliesMhSet*[*M.id*] retorna um vetor de objetos do tipo *ObjReply* com o conjunto de *Mhs* (e suas correspondentes respostas) que responderam à *M*.

3. **DestM (*)**: tabela *hash* que armazena, para cada posição indexada por *M.id*, o grupo de destinatários de um multicast *M*.

4. **GConfirmados (*)**: tabela *hash* que contém o conjunto de *Mhs* que confirmaram o recebimento do estado final de um multicast. Também é indexado pelo *id* da mensagem.

D.2.3 Estruturas de Dados em Mss_{part}:

1. **RespIms**: endereço do *IMS* ao qual o *Mss* está associado.

2. **LocalMhs (*)**: grupo de *Mhs* pertencentes à célula que está sob a responsabilidade de Mss_{part}.

3. **h_RECD (*)**: estrutura (tabela *hash*) semelhante à descrita na subseção C.2.3 no sentido de que está relacionada a cada *Mh* do sistema e é mantida no *Mss* responsável pelo *Mh* no momento. Contém n posições, onde n corresponde ao número total de *Mss*s no sistema. A principal diferença está no fato de que cada posição *h_RECD*[i] ($i, 0 \leq i < n$) contém uma

lista com os números de seqüência de todas as mensagens já recebidas pelo *Mh* e difundidas por Mss_i.

4. **MsgLog**: vetor que guarda em cada posição um objeto *MsgId* referente a uma mensagem multicast enviada para um dos elementos do grupo *LocalMhs* e que é destinatário da mensagem.

5. **RepliesFromMhs (*)**: tabela *hash* semelhante à *RepliesMhSet* descrita na subseção anterior. Cada posição *RepliesFromMhs*[*M.id*] retorna um vetor que com as respostas dos *Mhs* locais e destinatários da mensagem *M*.

6. **AckMhs (*)**: tabela *hash* indexada pelo identificador da mensagem (*M.id*), onde cada posição *AckMhs*[*M.id*] retorna o grupo de *Mhs* que confirmaram o recebimento do estado final de *M*.

7. **MsgManager (*)**: *hash* de objetos do tipo *ObjMsgManager* que auxilia, para cada mensagem *M*, o *Mss* no gerenciamento das ações que devem ser tomadas em relação à mensagem *M*.

8. **DestMLocal**: tabela *hash* onde cada posição indexada por *M.id* devolve o conjunto de *Mhs* locais e destinatários de uma determinada mensagem *M*.

D.2.4 Estruturas de Dados no *IMS*:

1. **Buffer (*)**: fila que armazena as mensagens de multicast difundidas por um *Mss* inicializador. É usado durante a primeira fase do protocolo.

2. **Outcomes (*)**: tabela *hash* que armazena o estado final das mensagens de multicast difundidas até o momento.

3. **Domain**: grupo de *Mss*s que fazem parte do domínio do *IMS* [1].

4. **MappingTable**: tabela *hash* indexada pelo endereço de um *Mss* (seja Mss_n) que pertença a *Domain*. Cada posição *MappingTable*[Mss_n] devolve o grupo *LocalMhs* de Mss_n. Desta forma, esta estrutura é responsável pelo mapeamento *Mh* $\leftrightarrow$ *Mss* no *IMS*.

D.3 Definição das Mensagens

Esta seção contém as definições e o conteúdo das mensagens segundo o elemento de rede que as processa. Mais uma vez, indicaremos com um (*) as mensagens que são comuns ao AM^2C e ao

[1] Esta informação é replicada nos *Mss*s, ou seja, dado que um *Mss* pertence a um domínio A, o *Mss* sabe quais os demais *Mss*s pertencentes a A.

iAM^2C e que não sofreram nenhuma modificação em sua estrutura[2]. Considere M como qualquer uma das mensagens a seguir, a qual possui uma identificação ($M.id$).

D.3.1 Mensagens Tratadas por Mss_{init}:

1. **M = MulticastRequested**(*Mh mh, Group dest, int content*)* :

   Mensagem de multicast requisitada por um *Mh*. Sempre enviada para o *Mss* responsável pela célula na qual o *Mh* se encontra no momento do envio da mensagem (Mss_{Mh}). Contém o identificador do *Mh* requisitante, o grupo de destinatários (*M.Dest*) e o conteúdo.

2. **M = RespFromMss**(*MsgId id, Vector replySet*):

   Recebimento da resposta de um *Mss* a respeito de uma determinada mensagem *M'*. *replySet* contém um conjunto de objetos do tipo *ObjReply* com as respostas de todos os *Mhs* pertencentes a *DestMLocal* referentes à mensagem *M'*.

3. **M = AckMss**(*MsgId id, Group C*)*:

   Mensagem enviada de um *Mss* para o Mss_{init} de uma determinada mensagem *M'*. É composta pela identificação da mensagem (onde *M.id = M'.id*) e pelo grupo de *Mhs* locais e pertencentes a *M'.Dest* que confirmaram o recebimento do estado final de *M'*.

4. **M = TimeIsOver**(*Timer T3*):

   Tempo máximo esperado pelo *Mss* inicializador de um multicast por respostas de todos os *Mhs* destinatários da mensagem. O *timer* T3 está associado uma mensagem do tipo `FwdMsgFromMssInit` difundida por Mss_{init} para todos os *IMSs* do sistema.

D.3.2 Mensagens Tratadas pelo Mss_{part}:

1. **M = FwdMsgFromIMS**(*MsgId id, Group dest, int content*):

   Mensagem enviada do *IMS* ao qual Mss_{part} está associado. Contém a identificação da mensagem, o grupo de destinatários e o conteúdo.

2. **M = RespFromMh**(*Mh mh, MsgId id, int reply*)*:

   Resposta de um *Mh* para seu Mss_{Mh} sobre a aceitação ou não de uma mensagem recebida (chamemos de *M'*). Contém a identificação do *Mh*, da mensagem (*M.id = M'.id*) e a resposta (`Ok`/`NOk`).

[2]O que pode mudar nestes casos é o fluxo dessas mensagens no protocolo.

3. **M = WaitingForReply**(*Timer T1*)*:

   Evento de *time-out* ocorrido em um *Mss*. Neste caso, o *time-out* corresponde ao término do tempo máximo esperado por um Mss_{part} por respostas de seus *Mhs* locais e destinatários da mensagem (Fase I). Contém um *timer* ao qual está associada uma mensagem *M'* do tipo `FwdMsgFromMss`, que é enviada do *Mss* para todo *Mh* ∈ (*LocalMhs* ∩ *M'.Dest*).

4. **M = StatusFromIMS**(*MsgId id, Group dest, char status*):

   Estado final de um multicast recebido do *IMS* ao qual o *Mss* está associado. Contém a identificação da mensagem, o grupo de destinatários e o *status* ("C" = *Commited*; "A" = *Aborted*).

5. **M = AckMh**(*MsgId id, Mh mh*)*:

   Aviso de um *Mh* a seu Mss_{Mh} sobre o recebimento de uma mensagem de *status* (chamemos de *M'*). Contém o identificador da mensagem (onde *M.id* = *M'.id*) e do *Mh*.

6. **M = WaitingForAck**(*Timer T2*)*:

   Evento de *time-out* ocorrido em um *Mss*. Neste caso, o *time-out* corresponde ao término do tempo máximo esperado por um Mss_{part} por confirmações de seus *Mhs* locais e destinatários de uma mensagem sobre o recebimento do estado final do multicast (Fase II). Contém um *timer* ao qual está associada uma mensagem *M'* do tipo `StatusFromMss`, que é enviada do *Mss* para todo *Mh* ∈ (*LocalMhs* ∩ *M'.Dest*).

7. **M = Greet**(*Mh mh, Mss* Mss_o)*:

   Mensagem recebida por Mss_{part} quando um *Mh* está entrando na sua célula (inicializando assim o protocolo de *hand-off*) ou volta de um período de inatividade. Contém a identificação do *Mh* e do *Mss* da antiga célula (Mss_o).

8. **M = Dereg**(*Mh mh, Mss* Mss_n)*:

   Solicitação de desvinculamento de um *Mh* de sua antiga célula enviada do *Mss* responsável pela nova célula (Mss_n). Contém a identificação do *Mh* e do novo *Mss*.

9. **M = DeregAck**(*Mh mh, hashtable* h_RECD*, Mss* Mss_o)*:

   Recebimento da confirmação de desvinculamento de um *Mh* de sua antiga célula (sob a responsabilidade de Mss_o). Contém a identificação do *Mh*, o *h_RECD* relacionado ao *Mh* e o endereço do Mss_o.

10. **M = ReFwdMsgFromIMS**(*Vector Pendentes, Vector Finais, Mh mh, Mss* Mss_o):

    Recebimento de uma retransmissão de mensagens pendentes (que é uma resposta ao envio de uma mensagem do tipo `LocationUpdate`) do *IMS* ao qual o *Mss* está associado. Contém um vetor (*Pendentes*) com todas as mensagens ainda não recebidas por *mh* (desde que seja destinatário) e um vetor (*Finais*) com os estados finais de todos os multicasts ainda armazenados no *IMS* (onde o *Mh* é destinatário). O parâmetro Mss_o indica o endereço do *Mss* responsável pela célula da qual o *Mh* migrou (e que gerou o envio da mensagem `LocationUpdate` - vide seção 4.4).

11. **M = Join**(*Mh mh*)*:

    Solicitação de entrada no sistema por parte de um *Mh*. Contém a identificação do *Mh*.

12. **M = Leave** (*Mh mh*)*:

    Solicitação de saída do sistema de um *Mh* para seu Mss_{Mh}. Contém a identificação do *Mh*.

D.3.3 Mensagens Tratadas por um IMS:

1. **M = FwdMsgFromMssInit**(*MsgId id, Group Dest, int content*)* :

   Mensagem recebida do *Mss* inicializador do multicast (Mss_{init}) e que deve ser enviada para todo $Mss \in Domain$ e que tenha algum $Mh \in (M.Dest \cap Mss.LocalMhs)$. Armazena o identificador da mensagem ($M.id$), o grupo de destinatários ($M.Dest$) e o conteúdo.

2. **M = StatusFromMssInit**(*MsgId id, Group Dest, char Status*)* :

   Recebimento do *status* de uma mensagem enviado pelo respectivo Mss_{init} para todos os *IMSs* do sistema. Contém a identificação da mensagem ($M.id$), o grupo de *Mhs* destinatários ($M.dest$) e o *status* ("A" = "*Aborted*"; "C" = "*Commited*").

3. **M = Delete**(*MsgId id*)*:

   Mensagem que foi enviada pelo *Mss* inicializador do multicast (Mss_{init}) para todos os *IMSs* a fim de que estes removam o estado final de *M* de seus respectivos *Outcomes*. Contém apenas a identificação da mensagem ($M.id$).

4. **M = LocationUpdate**(*Mh mh, Vector h_RECD, Mss* Mss_n*, Mss* Mss_o*, int opcao*):

   Mensagem que foi enviada de um Mss_{part} (e que faz parte do domínio do *IMS*) responsável pela célula-destino de um *Mh* em processo de *hand-off* ou que está voltando de um período de inatividade. É composta pelos endereços do *Mh*, do *Mss* da nova célula (Mss_n) e do *Mss* da antiga célula (Mss_o), respectivamente. Ainda, as ações são tomadas de acordo com o parâmetro *opcao*, o qual pode assumir os seguintes valores:

0: Solicitação de Atualização de Mapeamento (registro do *Mh* no sistema);

1: Solicitação de Reenvio de Mensagens Pendentes (baseado nos valores contidos em h_RECD);

2: Ambos os itens anteriores;

3: Solicitação de Remoção de Mapeamento (caso de migração inter-domínio);

D.3.4 Mensagens Tratadas por um Mh:

1. **M = FwdMsgFromMss**(*MsgId id, int content*)*:

   Recebimento de uma mensagem de multicast enviada pelo *Mss* responsável pelo *Mh* (Mss_{Mh}). Armazena a identificação da mensagem e o conteúdo.

2. **M = StatusFromMss**(*MsgId id, char status*)*:

   Recebimento do estado final de um multicast enviado pelo *Mss* responsável pelo *Mh* (Mss_{Mh}). Contém a identificação da mensagem e o estado final.

D.4 Tratamento das Mensagens

Esta seção apresenta um pseudo-código para o tratamento dado pelos elementos de rede na chegada das mensagens descritas na seção anterior. Sejam as seguintes considerações:

1. A abordagem será por elemento de rede envolvido;

2. As tabelas D.1 e D.2 apresentam os principais métodos utilizados no tratamento das mensagens no iAM^2C.

3. Sendo T1 e T3, respectivamente, os tempos (em UTs) dos eventos de *time-out* `WaitingForReply` e `TimeIsOver`, como T3 >> T1, se o estado final do multicast ainda não foi definido ao término de T1, então existe algum *Mh* que, por algum motivo, não permanece em uma célula durante um tempo suficiente para que lhe seja enviada a mensagem (por exemplo, um *Mh* numa região de transição constante). Caracterizamos isso como um caso "patológico" que dificilmente ocorre, mas que o protocolo deve levar em consideração.

Método	**Significado**
New(<MsgType>)	Criação de uma nova mensagem do tipo <MsgType>.
setTimer(<timer-event>, <repeat-order>, <time-to-elapsed>)	Inicia um temporizador para uma mensagem associada à <timer-event>. <repeat-order> (*"ONCE" ou "REPEATING"*) indica se o evento será gerado apenas uma vez ou repetidamente. <time-to-elapsed> refere-se ao tempo (em UTs) no qual o evento deverá ser gerado ou o período de repetição.
MsgManager[Mid].getInfoAboutEventProcessedInF1/F2()	Devolve TRUE se o evento de *time-out* da Fase I/II referente à mensagem indexada por Mid já foi processado.
MsgManager[Mid].setInfoAboutEventProcessedInF1/F2 (TRUE/FALSE)	Indica que o evento de *time-out* associado à mensagem M na Fase I/II do protocolo já foi ou não processado.
MsgManager[Mid].getInfoAboutDeleteSent()	Retorna TRUE caso a mensagem Delete referente à mensagem indexada por Mid já foi enviada para os *IMSs*. Retorna FALSE no caso contrário.
MsgManager[Mid].setInfoAboutDeleteSent(TRUE/FALSE)	Indica se a mensagem Delete referente a uma mensagem cujo identificador é Mid já foi ou não enviada.
MsgManager[Mid].getPermissionToProcess()	Retorna TRUE caso o evento de *time-out* associado à mensagem M (cujo identificador é Mid) pode ser processado. Retorna FALSE no caso contrário.
MsgManager[Mid].setPermissionToProcess(TRUE/FALSE)	Indica se o evento de *time-out* associado a uma mensagem cujo identificador é Mid pode ou não ser processado.
MsgManager[Mid].getInfoAboutStatusDefined()	Retorna TRUE se o estado final de uma mensagem M já foi definido. Retorna FALSE no caso contrário.
MsgManager[Mid].setInfoAboutStatusDefined(TRUE/FALSE)	Indica se o estado final de uma mensagem indexada por Mid já foi ou não definido.
gethRECD(Mh)	Devolve o *h_RECD* relacionado ao Mh.

Tabela D.1: Principais Métodos Utilizados no Tratamento das Mensagens no iAM^2C.

Dessa forma, considere a tabela D.3 que indica as principais ações tomadas quando um evento de *time-out* ocorre no protocolo iAM^2C.

4. Nos algoritmos, os parâmetros, estruturas de dados ou mensagens serão expressos utilizando-se a mesma notação do AM^2C.

5. O termo *owner* corresponde a uma referência para o *Ims/Mss/Mh* que está processando a mensagem.

Método	Significado
`allMhsHaveReplied (RepliesMhSet[Mid], DestM[Mid])`	Testa se todos os *Mhs* destinatários de uma mensagem *M* (*DestM[Mid]*) já responderam. *RepliesMhSet[Mid]* é um subconjunto de *DestM[Mid]* correspondente ao conjunto (*LocalMhs* ∩ *M.Dest*) que responderam à mensagem *M*.
`allMhsHaveConfirmed (AckMhs[Mid], DestM[Mid])`	Testa se todos os *Mhs* destinatários de uma mensagem *M*(*DestM[Mid]*) já confirmaram o recebimento do estado final da mensagem. *AckMhs[Mid]* é um subconjunto de *DestM[Mid]* correspondente ao conjunto (*LocalMhs* ∩ *M.Dest*) que confirmaram o recebimento do estado final de *M*.
`msgNotHandled (Mh, Mid)`	Retorna TRUE caso a mensagem `M` (cujo identificador é `Mid`) não tenha sido recebida pelo *Mh* e nenhum *Mss* considerou sua resposta como "NOk". Retorna FALSE no caso contrário. Esse teste é feito de acordo com os números de seqüência armazenados em *h_RECD[M.MssInit]* para o *Mh* em questão (o *Mss* inicializador pode ser obtido pelo objeto `Mid`).
`msgNotReceived (Mh, Mid, hRECD)`	Este método tem a mesma finalidade do método `MsgNotHandled(Mh, M.id)`, onde a única diferença está no parâmetro adicional `hRECD`. É por meio deste parâmetro, que é passado pelo *Mss* no envio da mensagem `LocationUpdate`, que o *IMS* correspondente é capaz de saber quais mensagens já foram tratadas/recebidas pelo *Mh* em questão.
`atualizaHRECD(Mh, Mid)`	Acrescenta o número de seqüência de *M* na estrutura *h_RECD[M.MssInit]* (onde o *Mss* inicializador pode ser obtido através do identificador da mensagem) para o *Mh* em questão.
`removeFromHRECD(Mh, Mid)`	Remove o número de seqüência da mensagem *M* presente na estrutura *hRECD[M.MssInit]* para o *Mh* em questão.
`Object[Mid].put(anObject)`	Insere o objeto `anObject` na posição indexada por `Mid` do objeto `Object`.
`Object[Mid].add(anObject)`	Adiciona o objeto `anObject` em `Object[Mid]`.
`Object[Mid].remove()`	Remove o objeto da posição indexada por `Mid`.
`send(elemento, M)`	Envio da mensagem `M` para `elemento`.

Tabela D.2: Principais Métodos Utilizados no Tratamento das Mensagens no iAM^2C.

6. Seja M uma das mensagens descritas na seção D.3. Assim como ocorreu com o AM^2C, quando utilizamos uma variável *M'* (ou *M"*) no lugar da variável M para referenciar uma mensagem, temos que tais mensagens possuem a mesma identificação (*M'.id* = *M.id*). Ou seja, *M' /M"* refere-se a mesma mensagem original (M) só que contém atributos diferentes.

D.4.1 Mensagens Tratadas por Mss_{init}:

1. **WhenMulticastRequested**(*Mh, Dest, Content*):

   (a) *mseq* := *mseq* + 1;

   (b) mid := New *MsgId*(*owner*, *mseq*); /* *Identificador da mensagem.* */

Evento	**Ações** (Seja M uma mensagem associada a $< timer-event >$)
`WhenWaitingForReply(`$< timer - event >$`)`	Seja A o conjunto de *Mhs* formado por (*LocalMhs* $\cap$ *M.Dest*). Seja B um subconjunto de A composto por todos aqueles *Mhs* que responderam à mensagem M, e o conjunto $C = A$ - B. $\forall$ *Mh* $\in C$ considere a resposta como sendo "NOk_M". Envie tais respostas, juntamente com as respostas dos *Mhs* pertencentes a A, para Mss_{init}.
`WhenWaitingForAck(`$< timer - event >$`)`	Seja C o grupo de *Mhs* locais que confirmaram o recebimento de M. Envie um *acknowledgment* com o grupo C para o Mss_{init}.
`WhenTimeIsOver(`$< timer - event >$`)`	**Se** o estado final de M ainda não foi definido **então** defina tal estado como *"Aborted"*. Difunda o estado final para todos os *IMSs* do sistema.

Tabela D.3: Eventos ocorridos em um *Mss*.

```
   (c) M' := New FwdMsgFromMssInit(mid, Dest, Content);
   (d) DestM[mid] := Dest;
   (e) /* Difusao para todos os IMSs do sistema. */
       ∀ ims ∈ IMSAddress faça
         send(ims, M');
   (f) /* Inicia um temporizador para um evento de timeout associado a M'. */
       setTimer(New TimeIsOver(M'), ONCE, T3);
```

2. **WhenRespFromMss**(*Mid, ReplySet*):
Considere que *ReplySet* contém um conjunto de objetos do tipo *ObjReply* com as respostas dos *Mhs* $\in$ *LocalDestM*[*Mid*].

```
   (a) ∀ objReply ∈ ReplySet faça
         RepliesMhSet[Mid].add(objReply);
   (b) Se allMhsHaveReplied(RepliesMhSet[Mid], DestM[Mid]) então
         */ Definição do estado final do multicast. */
         status := "C"; /* Commited */
         ∀ objReply ∈ RepliesMhSet[Mid] faça:
           reply := objReply.getReply();
           Se reply = "NOk"então
             status := "A"; /* Aborted */
             break;
         M' := New StatusFromMssInit(Mid, DestM[Mid], status);
         MsgManager[Mid].setInfoAboutStatusDefined(TRUE);
         */ Difusao para todos os IMSs. */
         ∀ ims ∈ IMSAddress faça
           send(ims, M');
```

3. **WhenAckMss**(*Mid, C*):

(a) ∀ *Mh* ∈ *C* **faça:**
GConfirmados [*Mid*] .add(*Mh*);

(b) **Se** allMhsHaveConfirmed(*GConfirmados[Mid], DestM[Mid]*) **então**
MsgManager [*Mid*] .setInfoAboutDeleteSent(TRUE);
M' := New *Delete* (*Mid*);
**/ Difusão para todos os IMSs. */*
∀ ims ∈ *IMSAddress* **faça**
send(ims, *M'*);

4. **WhenTimerIsOver**(*TimerEvent*):

(a) *M'* := *TimerEvent*.getMsg(); */* Mensagem associada ao time-out. */*

(b) **Se** *MsgManager*[*M'.id*].getInfoAboutStatusDefined() = FALSE **então**
/ Se o status ainda nao foi definido entao significa que nem todos os Mhs responderam.*/*
status := "A"; */* Aborted */*
M" := New *StatusFromMssInit* (*M'.id*, *DestM*[*M'.id*], status);
MsgManager[*M'.id*].setInfoAboutStatusDefined(TRUE);
**/ Difusão para todos os IMSs. */*
∀ ims ∈ *IMSAddress* **faça**
send(ims, *M"*);

D.4.2 Mensagens Tratadas por Mss_{part}:

1. **WhenFwdMsgFromIMS**(*Mid, Dest, Content*):
Seja *M* a *i*-ésima mensagem difundida por um dado Mss_{init}.

(a) */* Filtro dos Mhs locais e que pertencem a M.Dest. */*
∀ *Mh* ∈ *LocalMhs* **faça:**
Se *Mh* ∈ *Dest* **e** msgNotHandled(*Mh*, *Mid*) **então**
DestMLocal[*Mid*].add(*Mh*);

(b) M' := New *FwdMsgFromMss(Mid, Content)*;

(c) *Envio da mensagem para os Mhs locais e destinatarios. */*
∀ *Mh* ∈ *DestMLocal* [*Mid*] **faça:**
send(*Mh*, M');

(d) */* Inicia um temporizador para um evento de timeout associado a M'. */*
setTimer(New *WaitingForReply* (*M'*), ONCE, T1);

(e) */* Indica que o evento de timeout associado a M' pode ser processado. */*
MsgManager [*Mid*] .setPermissionToProcess(TRUE);

(f) */* Indica no Log que a mensagem ja foi transmitida. */*
MsgLog [*Mid*] .add(*Mid*);

2. **WhenRespFromMh**(*Mh, Mid, Reply*):
/ Verifica se o evento de time-out na 1a.fase para a mensagem ainda nao foi processado. */*

(a) **Se** *MsgManager* [*Mid*] .getInfoAboutEventProcessedInF1() = FALSE **então**
objAux := New *ObjReply*(*Mh, Reply*);

RepliesFromMhs [*Mid*].add(objAux);
/ Apenas aqui ha a atualizacao de h_RECD, pois se o time-out ja foi processado entao h_RECD ja foi atualizado. */*
atualizaHRECD(*Mh, Mid*);

3. **WhenWaitingForReply**(*TimerEvent*):

   (a) *M'* := *TimerEvent*.getMsg(); */* Mensagem associada ao time-out. */*

   (b) **Se** *MsgManager* [*M'.id*].getPermissionToProcess() **então**
   / Verifica quais Mhs nao responderam. /
   $\forall$ *Mh* $\in$ *DestMLocal*[*M'.id*] **faça**
   Se *Mh* $\notin$ *RepliesFromMh*[*M'.id*] **então**
   / Considera a resposta do Mh como "Nok". /
   objAux := New *ObjReply*(*Mh* , "Nok");
   RepliesFromMh[*M'.id*].add(objAux);
   / Indica que a mensagem ja foi tratada. /
   atualizaHRECD (*Mh* , *M'.id*);
   M":= New *RespFromMss* (*M'.id*, *RepliesFromMh*[*M'.id*]);
   / Envio da resposta para Mss_{init}. */*
   send(M'.MssInit(), M");
   MsgManager.setInfoAboutEventProcessedInF1(TRUE);
   Senão
   / Proximo evento de time-out deve ser processado. */*
   MsgManager [*M'.id*].setPermissionToProcess(TRUE);

4. **WhenStatusFromIMS**(*Mid, Dest, Status*):

   (a) M' := New *StatusFromMss* (*Mid, Status*);

   (b) $\forall$ *Mh* $\in$ (*LocalMhs* $\cap$ *M.Dest*) **faça:**
   send(*Mh* , M');
   / Remove o numero sequencia da mensagem de h_RECD[M.MssInit] para o Mh em questao, uma vez que esta ja esta obsoleta. */*
   removeFromHRECD(*Mh, Mid*);

   (c) */* Inicia um temporizador para um evento de time-out associado a M'. */*
   setTimer(New *WaitingForAck* (*M'*), ONCE, T2);

5. **WhenAckMh**(*Mid, Mh*):
   / Testa se o evento de time-out ja foi processado. Neste caso, a mensagem AckMss(Mh) deve ser enviada para o Mss inicializador do multicast. */*

   (a) **Se** *MsgManager* [*Mid*].getInfoAboutEventProcessedInF2() = TRUE **então**
   M' := New *AckMss*(*Mid*, New Group(*Mh*));
   send(M.MssInit, *M'*); */* Ack extra enviado. */*
   Senão
   / Adiciona o Mh no grupo de Mhs que ja enviaram um acknowledjment para M. */*
   AckMhs [*Mid*].add(*Mh*);

6. **WhenWaitingForAck**(*TimerEvent*):

(a) *M'* := *TimerEvent*.getMsg();

(b) *M"* := New *AckMss*(*M'.id*,*AckMhs*[*M'.id*]);

(c) send(M'.MssInit, M");

(d) $\forall$ *Mh* $\in$ *AckMhs*[*M'.id*] **faça**
 / Remocao do Mhs que ja responderam a M'. */*
 AckMhs[*M'.id*].remove(*Mh*);

(e) *MsgManager*.setInfoAboutEventProcessedInF2(TRUE);

7. **WhenGreet**(Mh_{new}, Mss_o):

(a) */* Testa se o Mh esta entrando agora no sistema. */*
 Se $Mss_o \neq$ null **então**
 / Testa se o Mh nao voltou de um periodo de inativacao. */*
 Se $Mss_o \neq$ owner **então**
 / Envio da mensagem Dereg do Mss responsavel pela celula atual para* Mss_o *solicitando o desligamento de* Mh_{new} *de sua antiga celula. */*
 M' := New *Dereg*(Mh_{new}, *owner*);
 send(Mss_o , M');
 Senão
 / O Mh esta voltando de um periodo de inativacao. Assim, deve-se enviar a mensagem LocationUpdate para o IMS, a qual tera o papel apenas de solicitante de reenvio de mensagens pendentes. */*
 M' := New *LocationUpdate*(Mh_{new}, gethRECD(Mh_{new}), owner, null, 1);
 send(owner.getIMS(), M');
 Senão
 / Solicitação de registro do Mh no sistema. */*
 M' := New *Join*(Mh_{new});
 send(owner, M');

8. **WhenDereg**(Mh_{new}, Mss_n):
Esta mensagem é processada em Mss_o (*Mss* da antiga celula).

(a) */* Remocao de* Mh_{new} *do grupo LocalMhs. */*
 LocalMhs.remove(Mh_{new});

(b) */* Aborta todas as mensagens enviadas e que o Mh ainda nao respondeu. */*

 $\forall$ mid $\in$ *MsgLog* **faça**
 Se *MsgManager*[mid].getInfoAboutEventProcessedInF1() = FALSE **então**
 Se msgNotHandled(Mh_{new}, mid) **então**
 **/ Considera a resposta do Mh como "Nok". */*
 objAux := New *ObjReply*(Mh_{new}, "Nok");
 RepliesFromMh[mid].add(objAux);
 **/ Indica que a mensagem ja foi tratada. */*
 atualizaHRECD(Mh_{new}, mid);
 **/ Remove o Mh de DestMLocal. */*
 DestMLocal[mid].remove(
 mhnew);

(c) */ *Verifica a necessidade do envio da mensagem LocationUpdate no caso de migracao inter-dominio.* */

```
Se Mss_n ∉ Domain então
  M' := New LocationUpdate(Mh_new, gethRECD(Mh_new), null, owner, 3);
  send(owner.getIMS(), M');
```

(d) /* *Envio da mensagem DeregAck para* Mss_n *informando o desvinculamento do Mh de sua antiga celula.* */

```
M' := New DeregAck(Mh_new, gethRECD(Mh_new), owner);
send(Mss_n, M');
```

9. **WhenDeregAck**(*Mh*$_{new}$, *hRECD*, *Mss*$_o$):

   (a) /* *Registro do novo Mh.* */

   ```
   LocalMhs.add(Mh_new);
   ```

   (b) /* *Envio da mensagem LocationUpdate para o IMS correspondente solicitando a atualizacao do mapeamento e envio de mensagens pendentes.* */

   ```
   M' := New LocationUpdate(Mh, hRECD, owner, Mss_o, 2);
   ```

   (c) send(owner.getIMS(), M');

10. **WhenReFwdMsgFromIMS**(*Pendentes, Finais, Mh,* Mss_o):
    Pendentes contém todas as mensagens que *Mh* ainda não recebeu e que é destinatário.
    Finais contém os estados finais de todos os multicasts ainda não finalizados, independente de *Mh* ter ou não recebido tal estado final.

    (a) */ *Testa se o Mh nao migrou novamente.* */

    ```
    Se Mh ∈ LocalMhs então
      ∀ M ∈ Pendentes faça
        Se msgNotHandled(Mh, M.id) então
          Se MsgManager[M.id].getInfoAboutEventProcessedInF1() = FALSE então
            */ Retransmissao e prorrogacao do time-out. */
            M' := New FwdMsgFromMss(M.id, M.Content);
            send(Mh, M');
            setTimer(New WaitingForReply(M'), ONCE, T1);
            /* Indica que o evento de time-out anterior nao deve ser processado. */
            MsgManager[M.id].setPermitionToProcess(FALSE);
            */ Adiciona o Mh no grupo de destinatarios local. */
            DestMLocal[M'.id].add(Mh);
          Senão /* Time-out ja processado. */
            Se Mss_o ≠ owner então
              /* Multicast deve ser abortado. */
              objAux := New ObjReply(Mh, "Nok");
              M' := New RespFromMss(M.id, New Vector(objAux));
              /* Envio da resposta para o Mss inicializador. */
              send(M.MssInit, M');
              */ Indica que a mensagem ja foi tratada. */
              atualizaHRECD(Mh, M.id);
    ```

*/ *Retransmissao dos estados finais.* */
∀ M ∈ *Finais* **faça**
  M' := New *StatusFromMss*(M.id, M.Status);
  send(*Mh* , M');
  /* *Remocao da mensagem M (ja obsoleta) e h_RECD[M.MssInit].* */
  *removeFromHRECD(*Mh, *M.id)*;

11. **WhenJoin**(Mh_{new}):

    (a) *LocalMhs*.add(Mh_{new}); /* *Adiciona* Mh_{new} *no grupo de Mhs locais.* */

    (b) /* *O IMS deve acrescentar o Mh no seu mapeamento.* */
    M' := New *LocationUpdate*(Mh_{new}, gethRECD(Mh_{new}), owner, null, 0);

    (c) send(owner.getIMS(), M');

12. **WhenLeave**(*Mh*):

    (a) *LocalMhs*.remove(*Mh*); /* *Remove Mh do grupo de Mhs locais.* */

D.4.3 Mensagens Tratadas no IMS:

1. **WhenFwdMsgFromMssInit**(*Mid, Dest, Content*):

    (a) /* *Armazenamento da mensagem.* */
    Buffer[*Mid*].add(M);

    (b) mssDest := New Group();

    (c) /* *Filtro dos* Msss ∈ *Domain e que possuem algum Mh* ∈ *M.Dest.* */

    ∀ mss ∈ *Domain* **faça**
      /* *Grupo de Mhs locais ao Mss pertencente ao dominio do IMS.* */
      gMhs := *Domain*[mss];
      ∀ mh ∈ *gMhs* **faça**
        Se mh ∈ *Dest* **então**
          /* *Adiciona o Mss no grupo de destinatarios.* */
          mssDest.add(mss);
          break;

    (d) M' := New *FwdMsgFromIMS(Mid, Dest, Content)*;

    (e) /* *Envio da mensagem somente para aqueles* Msss *que possuem algum Mh* ∈ *M.Dest.* */

    ∀ mss ∈ mssDest **faça**
      send(mss, M');

2. **WhenStatusFromMssInit**(*Mid, Status, Dest*):

    (a) /* *Armazenamento do estado final em Outcomes.* */
    Outcomes[*Mid*].put(M);

    (b) mssDest := New Group();

    (c) /* *Filtro dos* Msss ∈ *Domain e que possuem algum Mh* ∈ *M.Dest.* */

```
∀ mss ∈ Domain faça
  /* Grupo de Mhs locais ao Mss pertencente ao dominio do IMS. */
  gMhs := Domain[Mss];
  ∀ mh ∈ gMhs faça
    Se mh ∈ Dest então
      /* Adiciona o Mss no grupo de destinatarios.  */
      mssDest.add(mss);
      break;
```

(d) M' := New *StatusFromIMS(Mid, Dest, Status)*;

(e) */* Envio da mensagem somente para aqueles* Msss *que possuem algum Mh* $\in$ *M.Dest. */*

```
∀ mss ∈ mssDest faça
  send(mss, M');
```

(f) */* Remocao da mensagem-conteudo do Buffer. */*
Buffer[*Mid*].remove();

3. **WhenDelete**(*Mid*):

   (a) */* Remocao do estado final de Outcomes. */*
   Outcomes [*Mid*].remove();

4. **WhenLocationUpdate**(*Mh, hRECD,* $\mathtt{Mss}_n$, $\mathtt{Mss}_o$, *Opcao*):
   Sejam Mss_n e Mss_o os *Mss*s da nova e antiga celula, respectivamente (lembremos que uma migracao gera uma mensagem *LocationUpdate* por parte de Mss_n.

   (a) *Pendentes* := New *Vector*();

   (b) *Finais* := New *Vector*();

   (c) **Se** *Opcao* = 1 **ou** *Opcao* = 2 **então**

```
/* Armazena em Pendentes toda mensagem presente em Buffer
que o Mh ainda nao a recebeu e que e destinatario.  */
∀ M ∈ Buffer faça
  Se msgNotReceived(Mh, M.id, hRECD) então
    Pendentes.add(M);
Armazena em Finais todos os estados finais presentes em Outcomes
e que o Mh e destinatario.  */
∀ M ∈ Outcomes faça
  Finais.add(M);
```

   (d) Caso *Opcao* Seja:

```
0:  /* Solicitacao de Atualizacao de Mapeamento.  1o.  registro no sistema.*/
  /* Adicao do Mh no grupo de Mssn.  */
  Domain[Mssn ].add(Mh);
1:  /* Solicitacao de Envio de Mensagens Pendentes.  */
  M' := New ReFwdFromIMS(Mh, Pendentes, Finais);
  send(Mssn, M');
2:  /* Casos 0 e 1.  */
  Domain[Msso ].remove(Mh);
```

```
    Domain[Mss_n ].add(Mh);
    M' := New ReFwdFromIMS(Mh , Pendentes, Finais);
    send(Mss_n , M');
3:  /* Solicitacao de Remocao do Mh do Mapeamento:  migracao inter-dominio.  */
    Domain[Mss_o ].remove(Mh);
```

D.4.4 Mensagens Tratadas no Mh:

1. **WhenFwdMsgFromMss**(*Mid, Content*):

   (a) **Se** a aplicacao e capaz de processar a mensagem **então**
   reply := "Ok";
   Senão reply := "NOk";

   (b) *M'* := New *RespFromMh*(*owner*, *Mid*, reply);

   (c) */* Envia resposta para o* Mss *resposavel pelo Mh.*/*
   send(Mss_{owner}, *M'*);

2. **WhenStatusFromMss**$_n$(*Mid, Status*):
   / Acknowledgment enviado para* Mss *responsável pelo Mh confirmando o recebimento do estado final de uma mensagem. */*

   (a) *M'* := New *AckMh*(*owner, Mid*);

   (b) send(Mss_{owner}, *M'*);

Referências Bibliográficas

[1] A. Acharya and B.R. Badrinath. Delivering Multicast Messages in Networks with Mobile Hosts. In *Proc. of 13th Intl. Conference on Distributed Computing Systems, Pittsburgh*. IEEE Computer Society, May 1993.

[2] S. Alagar and S. Venkatesan. Causal ordering in distributed mobile systems. *IEEE Transactions on Computers*, 46(3):353-361, 1997.

[3] Kenneth P. Birman. Building secure and reliable network applications. In *WWCA*, pages 15–28, 1997.

[4] Uyless Black. *Mobile and Wireless Networks*. Prentice Hall, 1996.

[5] R.C.A. da Rocha and M. Endler. 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*, pages 44–51, August 2001.

[6] Ricardo C.A. da Rocha. *MobiCS: Um Simulador de Protocolos para Computaçâo Móvel*. IME/USP, 2000. www.ime.usp.br/~rcarocha/mobics/.

[7] Ricardo C.A. da Rocha and Markus Endler. Flexible Simulation of Distributed Protocols for Mobile Computing. In *3rd. Workshop on Modeling Analyses and Simulation of Wireless and Mobile Systems (MSWIM), Boston*, pages 123–126, 2000.

[8] Ricardo C.A. da Rocha and Markus Endler. MobiCS: An Environment for Prototyping and Simulating Distributed Protocols for Mobile Networks. In *Proc. 3rd IEEE Intern. Conference on Mobile and Wireless Communications Networks (MWCN2001), Recife - Brazil*, pages 44–51, August 2001.

[9] A. Myles David B. Johnson and Charles Perkins. A mobile host protocol supporting route optimization and authentication. *IEEE Journal on Selected Areas in Communications*, 13(5):839-849, June 1995.

[10] Mateus de Freitas Ribeiro and M. Endler. Um Protocolo Indireto para Multicast Atômico em Computação Móvel. In *III Workshop de Comunicação sem Fio e Computação Móvel (WCSF´2001), Recife, Brazil*, pages 84–91, August 2001.

[11] Carlos de M. Cordeiro, Djamel H. Sadok, and Judith Kelner. Um Mecanismo de Retransmissão Eficiente para Multicast. In *XVIII Simpósio Brasileiro de Redes de Computadores (SBRC), Belo Horizonte, MG*, May 2000.

[12] M. Endler, Dilma M. da Silva, and Kunio Okuda. RDP: A Result Delivery Protocol for Mobile Computing. In *Proc. of the Int. Workshop on Wireless Networks and Mobile Computing (WNMC) with 20th ICDCS, Taiwan, R.O.C.* IEEE, April 2000.

[13] Markus Endler. A protocol for atomic multicast among mobile hosts. In *Dial M Workshop/Mobicom'99, Seatle (USA)*, pages 56–63. ACM, August 1999.

[14] João E. Ferreira and Marcelo Finger. *Controle de concorrência e distribuição de dados: a teoria clássica, suas limitações e extensões modernas.* X Escola de Computação, São Paulo, 2000.

[15] Georges Forman and John Zahorjan. The challenges of mobile computing. *IEEE Computer*, pages 39–47, April 1994. ftp://ftp.cs.washington.edu/tr/1993/11/UW-CSE-93-11-03.PS.Z.

[16] Jim Geier. *Wireless LANs. Implementing High Performance IEEE 802.11 Networks.* SAMS Publishing, 2001.

[17] Alberto Bartoli Giuseppe Anastasi and Francesco Spadoni. A reliable multicast protocol for distributed mobile systems: Design and Evaluation. In *IEEE Transactions on Computers*, October 2001.

[18] T.G. Harrison, C.L. Williamson, W.L Mackrell, and R.B. Bunt. Mobile Multicast (MoM) Protocol: Multicast Support for Mobile Hosts. In *Proc. 3rd Inter. Conference on Mobile Computing and Networking (Mobicom 97), Budapest, Hungary*, pages 151–160, September 1997.

[19] N. Kanodia, T.-C. Lin, Zamora V., and Schlosser S. Adaptive Routing for Mobile Multicast Users. WWW, February 1998. www.andrew.cmu.edu/schlos/mwn/new_proposal.

[20] E. Schooler M. Handy, H. Schulzrinne and J. Rosemberg. SIP: Session initiation protocol. *Rfc. 2543, Columbia University*, March 1999. www.cs.columbia.edu/ hgs/sip/papers.html.

[21] A. Schiper M. Raynal and S. Toueg. The causal ordering abstraction and a simple way to implement it. *Information Processing Letters*, 39(6):343-350, 1991.

[22] Silvano Maffeis, Walter Bischofberger, and Kai-Uwe Maetzel. A generic multicast transport service to support disconnected operation. In USENIX Association, editor, *Proceedings of the second USENIX Symposium on Mobile and Location-Independent Computing: April 10–11, 1995, Ann Arbor, Michigan, USA*, pages 79–90, Berkeley, CA, USA, April 1995. USENIX.

[23] Geraldo Robson Mateus and Antonio A. Ferreira Loureiro. *Introdução à Computação Móvel*. II Escola de Computação Móvel, Rio, 98.

[24] Charles Perkins. IP mobility support. Request for Comments 2002, October 1996. IETF Mobile IP Group.

[25] Evaggelia Pitoura and George Samaras. *Data Management for Mobile Computing*. KAP, 1998.

[26] M. Raynal R. Prakash and M. Singhal. An Adaptive Causal Ordering Algorithm Suited to Mobile Computing Environments. *Journal of Parallel and Distributed Computing*, pages 190–204, March 1997.

[27] Djamel H. Sadok, Carlos de M. Cordeiro, and Judith Kelner. A reliable subcasting protocol for wireless environments. In C. Guy Omidyar, editor, *Mobile and Wireless Communication Networks*, number 1818 in Lecture Notes in Computer Science, pages 174 – 185. Springer-verlag, 2000.

[28] James D. Solomon. *Mobile IP: The Internet Unplugged*. Prentice Hall, 1998.

[29] J.Z. Wang. A fully distributed location registration strategy for universal personal communication systems. *IEEE Journal on Selected Areas in Communications*, 11(6):850-860, August 1993.

[30] C.L. Williamson, T.G. Harrison, W.L Mackrell, and R.B. Bunt. Performance evaluation of the mom mobile multicast protocol. *ACM Baltzer Journal on Mobile Networks and Applications*, 3(2):189–201, August 1998.

[31] George Xylomenos and George C. Polyzos. IP multicast for mobile hosts. *IEEE Communications Magazine*, 35(1):54-58, 1997.