Protocolos para comunicação entre agentes móveis

Rachel de Paula Cavalcanti

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: **Ciência da Computação**
Orientador: **Profª Dr. Kunio Okuda**

São Paulo – 2005

Protocolos para comunicação entre agentes móveis

Este exemplar corresponde à redação final da dissertação devidamente corrigida e defendida por Rachel de Paula Cavalcanti e aprovada pela Comissão Julgadora.

São Paulo, 4 de novembro de 2005.

Banca Examinadora :

- Profº Dr. Kunio Okuda (Orientador) - IME-USP
- Profº Dr. Fábio Kon - IME-USP
- Profº Dr. Edmundo R. M. Madeira - IC-UNICAMP

A mamãe, vó Lourdes,

Mi e ao meu amor Davi

Agradecimentos

Ao professor Kunio Okuda, pelo ótimo trabalho de orientação, por sua dedicação, paciência, compreensão e bom humor. Sua conduta foi fundamental para a realização deste trabalho.

A minha mãe, Ilda, pela educação e genética, pelo exemplo de força, coragem, dedicação e amor ao próximo que sempre tento seguir. Pessoa em quem sempre pude contar e confiar, cujas lembranças e ensinamentos vão estar comigo para sempre.

Ao amor da minha vida, meu maior tesouro, Davi, pelo apoio emocional e financeiro, pelo companheirismo e imenso amor de todos estes anos.

A minha irmã, Mitsue, por toda sua dedicação, admiração, amor e fidelidade. Do melhor que mamãe me deixou, certamente, você foi o melhor presente.

A minha vó Lourdes, pela comida bem feita, pelos afagos, pelas roupas cheirosas, por todo o conforto e amor que me foi dado. A minha tia Marcia por ser tão carinhosa e alegre, tornando a minha vida muito feliz durante este trabalho.

A toda minha família, tios, primos, sogros e cunhados, que sempre me incentivaram e demonstraram sua confiança em mim. Em particular, aos meus primos Walter e Andréa Ragnev, por terem me ajudado financeira e pscicológicamente no momento em que mais precisei, sem essa ajuda não conseguiria nem mesmo ter iniciado a graduação.

A todos os meus amigos, pelo incentivo, preocupação, conselhos e por partilhar momentos maravilhosos de diversão, imprescindíveis para o bom andamento deste trabalho. A todos os membros da Ala Oeste, amigos com quem pude contar durante toda a graduação e pós-graduação.

Em especial, agradeço a Deus, por atender aos meus pedidos e por me dar saúde, inteligência, oportunidades e condições para que eu pudesse atingir meus objetivos.

Resumo

O número de computadores móveis ligados à rede é cada vez maior, desse modo, faz-se necessário prover suporte eficiente à mobilidade, que garantirá, assim, um bom desempenho. Nesse sentido, grande parte dessa responsabilidade se deve aos protocolos de comunicação. Existem vários grupos de trabalho discutindo, projetando e fazendo propostas de protocolos de comunicação para sistemas de computação móvel. O que faz desta área muito ativa. Contudo, em vários pontos ainda não existe um consenso sobre a melhor abordagem a ser escolhida.

Esta dissertação apresenta, compara e analisa alguns protocolos que provêem comunicação entre agentes móveis. Inicialmente serão apresentados três protocolos: o protocolo RDP (*Result Delivery Protocol*), o protocolo RCP (*A Reliable Connectionless Protocol for Mobile Clients*), o protocolo SRDP (*Safe Result Delivery Protocol*) e, por fim, será apresentado o protocolo ARCP (*Ack* RCP), este último, desenvolvido neste trabalho, possui algumas vantagens em relação aos protocolos anteriores.

Com o objetivo de simular e comparar os protocolos citados acima, foi utilizado o simulador MobiCS (*Mobile Computing Simulator*), por ser um simulador de protocolos flexível e por permitir validar e acompanhar o desempenho de diferentes algoritmos que simulam protocolos para comunicação móvel. Também foram desenvolvidas algumas funcionalidades adicionais no simulador com o propósito de obter, ao final da simulação, um relatório contendo métricas que facilitam a comparação entre os protocolos.

Abstract

The number of mobile computers connected to the net grows every day. An efficient support for mobility shows necessary to allow a good performance in the net. Communication protocols are one of the main drivers in providing good net support. Nowadays there are several workgroups discussing, projecting and proposing new communication protocols for mobile computer systems. This is a very active working area, but still in many scenarios there is no agreement on a "best approach" to use.

This work shows, compares and analyses some protocols to provide communication between mobile agents. Initially three protocols will be presented: RDP (Result Delivery Protocol), RCP (A Reliable Connectionless Protocol for Mobile Clients) and SRDP (Safe Result Delivery Protocol). After those, will be presented the ARCP protocol (Ack RCP), which was developed during this work and has some advantages over the previous protocols.

To simulate and compare the protocols, the simulator MobiCS (Mobile Computing Simulator) was used. MobiCS is a flexible protocol simulator, and allows the validation and performance tracking of different algorithms that simulate mobile communication protocols. Finally, in this work some new functionalities were added to MobiCS to allow reporting. At the end of the simulation a specific report is generated, with some metrics to make the process of comparing the protocols easier.

Sumário

1 Introdução

1.1 Introdução à Computação Móvel

Nas últimas décadas observa-se o forte crescimento que houve nas áreas de serviços via satélite, redes locais sem fio e comunicação celular, permitindo, assim, que informações e recursos sejam acessados e utilizados a qualquer lugar em qualquer momento. O número de pessoas que hoje possuem algum tipo de dispositivo que possa estar ligado à rede sem fio como *laptop*, *palmtop* e celulares é cada vez maior. A maior parte desses dispositivos deve ter a capacidade de se comunicar com a parte fixa da rede e com outros computadores móveis. Nesse ambiente, chamado de computação móvel ou computação nômade, o dispositivo não precisa ter uma posição fixa na rede [ML98].

Lembrando que o número de computadores móveis ligados à rede é cada vez maior, e que para garantir um bom desempenho é necessário prover suporte eficiente à mobilidade; tal desempenho se deve, em grande parte, aos protocolos de comunicação.

Os sistemas para computação móvel devem levar em consideração três aspectos essenciais: uso da rede sem fio, mobilidade das estações e portabilidade dos dispositivos. Tais aspectos acarretam em uma série de problemas a serem observados [FZ94].

Os problemas relacionados com o uso da rede sem fio incluem número frequente de desconexões, baixa largura de banda se comparada a redes com fio, grande variação na qualidade do serviço, rede heterogênea e aumento dos riscos na segurança da comunicação. Em decorrência da mobilidade, o endereço de uma estação móvel muda dinamicamente. Sua localização afeta o envio de respostas das suas requisições. Finalmente, a portabilidade dos dispositivos introduz várias restrições como economia de energia, interfaces reduzidas e limitação na capacidade de armazenamento.

Devido a toda esta complexidade, simuladores têm sido utilizados com o objetivo de facilitar a prototipagem, testes e avaliação dos protocolos.

Existem duas classificações de simuladores: simuladores de redes e simuladores de protocolos. Os simuladores de rede têm como objetivo testar os protocolos em cenários com características específicas dos elementos de rede e topologia. Desse modo, os protocolos podem ser implementados de maneira mais eficiente, pois podem fazer uso de tais informações. Em contrapartida, os simuladores de protocolos visam ser mais flexíveis e genéricos, facilitando a prototipagem dos protocolos. Resumindo, os simuladores de protocolos são utilizados para testar a corretude dos protocolos em um ambiente de rede simulado e genérico, enquanto os simuladores de rede podem ser utilizados para testar os protocolos em ambientes de redes específicos [Roc01].

Neste trabalho, será utilizado um simulador de protocolos. A ênfase da comparação será dada no comportamento e desempenho dos protocolos de acordo com as diferenças em suas respectivas especificações.

Sendo assim, foi desenvolvido um novo protocolo de entrega de mensagens que será analisado, avaliado e comparado com outros protocolos similares. Tais protocolos serão simulados em vários cenários distintos com o objetivo de identificar o melhor protocolo para cada situação. Para a avaliação dos protocolos será utilizado o simulador MobiCS. Isto porque, entre os simuladores analisados, o MobiCS é o único simulador que permite de uma maneira flexível e genérica a análise de desempenho, testes e validação de protocolos em um ambiente único. Também foram realizadas algumas modificações neste simulador para que ele consiga, ao final da simulação, emitir um relatório com métricas que possibilitem a avaliação dos protocolos, tais como: número de mensagens enviadas e retransmitidas separadas por tipo, duração média de uma requisição, quantidade de requisições concluídas e não concluídas, número de mensagens enviadas na rede fixa, número de mensagens enviadas no meio sem fio e número de mensagens enviadas no processo de *hand-off*.

O modelo de sistema utilizado é estruturado em células. Uma célula representa uma área geográfica atendida por um transmissor (estação base ou Mss, de *Mobility Support Sta-*

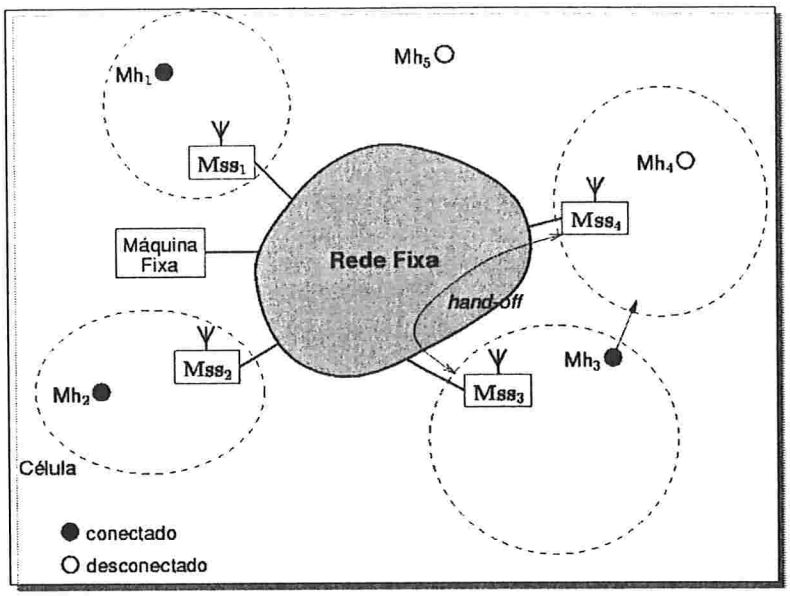

Figura 1: Modelo de Sistema

tion). As estações base têm como objetivo atender as unidades móveis (ou *Mh*, de *Mobile Host*) dentro de sua área de cobertura e fazer a comunicação com a rede fixa. Portanto, a comunicação entre uma unidade móvel e a rede fixa ou outra unidade móvel será sempre intermediada por uma estação base. Este modo de comunicação é chamado de modelo indireto para interações cliente-servidor que foi originalmente proposto por Badrinath [BBIM93] e atualmente é adotado em vários trabalhos. A Figura 1 ilustra o modelo de sistema adotado.

O simulador MobiCS e os protocolos estudados neste trabalho fazem parte do projeto SIDAM (Sistemas de Informação Distribuídos para Agentes Móveis)[ESS+00]. A finalidade do projeto SIDAM é analisar os principais problemas vinculados à implementação de serviços de informação descentralizados para consulta por agentes móveis, assim como, pesquisar as metodologias e tecnologias existentes e, por fim, desenvolver novos algoritmos, teorias e métodos nessa área.

1.2 Descrição do Trabalho

Esta dissertação apresenta, compara e analisa alguns protocolos que provêem comunicação entre agentes móveis. Inicialmente serão apresentados três protocolos. O protocolo RDP (*Result Delivery Protocol*) que provê entrega confiável de mensagens entre servidores e clientes móveis. Neste protocolo um *proxy* é criado para cada requisição feita pela unidade móvel para o servidor. O protocolo RCP (RCP - *A Reliable Connectionless Protocol for Mobile Clients*) é uma versão melhorada do protocolo RDP, neste uma unidade móvel possui apenas um *proxy* a cada instante. O protocolo SRDP (*Safe Result Delivery Protocol*) [Dur01] é uma modificação do protocolo RCP que, por sua vez, busca manter as mesmas garantias de confiabilidade na entrega sem que a confiabilidade dos $Msss$ seja crítica para o funcionamento do protocolo. Por fim, será apresentado o protocolo ARCP (*Ack* RCP), desenvolvido neste trabalho, que assim como o protocolo SRDP garante a entrega confiável de mensagens mesmo que haja falha na comunicação entre os $Msss$, porém este protocolo diminui a quantidade de informações armazenadas no Mh e reduz o número de requisições feitas para o servidor para evitar a sobrecarga do mesmo.

A fim de simular e comparar os protocolos, foi utilizado o simulador MobiCS (*Mobile Computing Simulator*). O MobiCS é um simulador de protocolos flexível que permite validar e acompanhar o desempenho de diferentes algoritmos que simulam protocolos para comunicação móvel [Roc01]. Neste simulador, estão disponíveis dois modos de simulação. No modo estocástico os protocolos são testados através de simulações exaustivas e aleatórias, em contrapartida, no modo determinístico o cenário de simulação é descrito detalhadamente pelo usuário através de um *script*. Somado a isto, foram desenvolvidas algumas funcionalidades adicionais no simulador para que este emita, ao final da simulação, um relatório contendo métricas como: o número de mensagens enviadas e retransmitidas separadas por tipo, duração média de uma requisição, quantidade de requisições concluídas e não concluídas,

número de mensagens enviadas na rede fixa, número de mensagens enviadas no meio sem fio e número de mensagens enviadas no processo de *hand-off*. Todas estas funcionalidades facilitam a comparação entre os protocolos.

Posteriormente, uma vez apresentados os resultados obtidos com a implementação e simulação dos protocolos no MobiCS, é realizada uma análise dos mesmos com o objetivo de identificar as vantagens e desvantagens de cada protocolo em determinados cenários.

1.3 Estrutura do Trabalho

- **Problemas relacionados à computação móvel:** No Capítulo 2 serão apresentados os problemas relacionados à computação móvel que devem ser levados em consideração por todos os protocolos descritos neste trabalho.

- **Protocolos:** No Capítulo 3 são descritos os protocolos RDP (*Result Delivery Protocol*), RCP (*A Reliable Connectionless Protocol for Mobile Clients*), SRDP (*Safe Result Delivery Protocol*) e um protocolo desenvolvido neste trabalho, o protocolo ARCP (*Ack* RCP). Neste capítulo serão detalhados os processo de cada protocolo e exemplificados através da simulação de cenários de execução. Além disso, é feita uma comparação inicial que listará todos os tipos de mensagens enviadas e dados mantidos por cada protocolo.

- **Simuladores de Protocolos:** No Capítulo 4 é apresentada uma descrição geral dos simuladores de protocolos. Em seguida, são listadas as características dos principais simuladores existentes. Por fim, são apresentadas as justificativas da escolha do simulador MobiCS.

- **Visão Geral do MobiCS:** No Capítulo 5 o simulador MobiCS, escolhido para a realização do trabalho, é detalhado. São listadas todas as modificações que tiveram de

ser feitas no MobiCS com o fim de obter métricas para as comparações

- **Implementação do Protocolo ARCP:** O Capítulo 6 contém a descrição do processo de implementação do protocolo ARCP utilizando o simulador MobiCS. Além disso, são apresentadas as situações críticas escolhidas para os testes do protocolo.

- **Comparação dos Protocolos:** No Capítulo 7 é feita a comparação entre os protocolos apresentados. Primeiramente são detalhados os parâmetros utilizados e os dados medidos nas simulações. A partir dos resultados obtidos é feita a análise dos protocolos.

- **Conclusão:** No Capítulo 8 é apresentado o resumo da dissertação. Em seguida são listadas todas as contribuições desta pesquisa. Finalmente, são apresentadas propostas para trabalhos futuros.

2 Problemas relacionados à computação móvel

2.1 Sumário

Neste capítulo são apresentados os problemas relacionados à computação móvel que devem ser levados em consideração por todos os protocolos que serão descritos neste trabalho.

São relevantes nos sistemas para computação móvel três aspectos essenciais: uso da rede sem fio, mobilidade das estações e portabilidade dos dispositivos. Tais aspectos acarretam em uma série de problemas a serem observados[FZ94].

Os problemas relacionados à rede sem fio, mobilidade e portabilidade são apresentados nas Seções 2.2, 2.3 e 2.4 respectivamente.

2.2 Rede sem fio

Os problemas relacionados ao uso da rede sem fio incluem: número frequente de desconexões, baixa largura de banda, grande variação na qualidade do serviço, rede heterogênea e aumento dos riscos na segurança da comunicação.

Para lidar com as desconexões frequentes, os dispositivos móveis podem executar as aplicações localmente o máximo possível, reduzindo, assim, a comunicação com a rede. Além disso, as aplicações podem tornar as desconexões transparentes para o usuário através de comunicação assíncrona, isto é, o cliente pode enviar várias requisições antes de receber alguma resposta.

A largura de banda é dividida entre os usuários de uma célula, desse modo, a mesma depende do tamanho e distribuição da população de usuários. Existem duas possibilidades para melhorar a capacidade da rede: sobreposição de células ou redução do tamanho das mesmas com o objetivo de aumentar o número de células em uma área. Algumas técnicas de programação também podem contribuir, como a compressão dos dados e a redução do número

de pequenas requisições através de requisições maiores capazes de englobar as menores.

Com relação à variação na qualidade de serviço, as aplicações podem operar de três formas distintas. Podem assumir alta largura de banda e somente rodar em tais condições ou assumir baixa largura de banda não aproveitando os momentos de alta largura de banda ou, ainda, identificar a largura de banda e modificar o nível de detalhes e qualidade do serviço proporcionalmente.

A heterogeneidade da rede também torna a comunicação sem fio mais complexa, pois os dispositivos podem ter que mudar de interface dependendo da rede, por exemplo, interfaces com infravermelho não podem ser utilizadas na luz do sol[FZ94]. Além disso, pode vir a ser necessário mudar o protocolo de acesso para diferentes redes.

A segurança na comunicação sem fio é comprometida pela facilidade de se conectar a um canal sem fio. A solução é criptografar os dados. A criptografia pode ser feita via software ou através de hardware especializado.

2.3 Mobilidade

Em decorrência da mobilidade, o endereço de uma estação móvel muda dinamicamente. Sua localização afeta o envio de respostas das suas requisições. Existem quatro formas básicas de se obter o endereço atual de uma estação móvel:

- ***Broadcast:*** uma mensagem é enviada para um grupo de células. Desse modo, somente a estação base que estiver responsável pela estação móvel irá responder. Essa técnica só é utilizada quando se possui informação aproximada da localização, pois enviar mensagens para um grande número de células é muito custoso.

- ***Central Services:*** o endereço atual de todos as estações móveis é armazenado em um banco de dados central. A cada migração a estação móvel envia uma mensagem para a base central atualizando a sua localização. Apesar do banco ser centralizado,

técnicas como distribuição, replicação e *caching* podem ser utilizadas para melhorar o desempenho.

- ***Home Bases:*** somente um servidor possui a localização da estação móvel. Nesta técnica, se o *home base* se desconectar a comunicação com a estação móvel não será mais possível.

- ***Forwarding Pointers:*** a cada migração uma cópia da nova localização é enviada para o endereço antigo da estação. As mensagens enviadas para a estação móvel são reenviadas entre as localizações anteriores até chegar ao endereço atual.

2.4 Portabilidade

A portabilidade dos dispositivos introduz várias restrições como economia de energia, interfaces reduzidas e limitação na capacidade de armazenamento.

Minimizar o consumo de energia aumenta a portabilidade dos dispositivos permitindo que a bateria seja menor e mais leve, além disso, poderá ser recarregada em períodos mais longos. As aplicações podem economizar energia reduzindo o processamento, comunicação, uso de memória e fazendo suas operações periódicas com menos frequência.

A capacidade de armazenamento de um dispositivo é limitada pelo seu tamanho e consumo de energia. As soluções para economia de espaço incluem compressão de dados, acesso a dados através da rede e compartilhamento de bibliotecas de código pelos programas.

3 Protocolos

3.1 Sumário

O protocolo RDP (*Result Delivery Protocol*) [ES00], o protocolo RCP (*A Reliable Connectionless Protocol for Mobile Clients*)[ESO00], o protocolo SRDP (*Safe Result Delivery Protocol*) [Dur01] e o protocolo ARCP (*Ack* RCP) são protocolos que definem regras para a entrega confiável de mensagens de servidores a clientes móveis. Por confiável queremos dizer que, para cada requisição do cliente móvel a algum servidor haverá garantia de resposta, mesmo que não seja instantânea e que ocorra queda do serviço ou migrações.

Todos os protocolos utilizam o conceito de *proxy*. O *proxy* representa o Mh na rede fixa, tem como função receber as mensagens do servidor e reenviá-las para o Mh. Os protocolos também se baseiam no modelo indireto, neste modelo a comunicação entre os Mhs e os servidores é sempre intermediada por estações base. A Seção 3.2 apresenta mais detalhes sobre este assunto.

Em cada seção deste capítulo serão analisados, individualmente, os protocolos RDP (*Result Delivery Protocol*), RCP (*A Reliable Connectionless Protocol for Mobile Clients*), SRDP (*Safe Result Delivery Protocol*) e finalmente ARCP (*Ack* RCP).

Na Seção 3.7 é feita uma comparação inicial entre os protocolos acima citados de acordo com suas especificações, também serão listados todos os tipos de mensagens e dados mantidos por cada protocolo. Finalmente, na Seção 3.8 são apresentadas duas alternativas de tolerância a falha nos Msss.

Alguns exemplos de aplicações práticas que poderiam fazer uso destes protocolos de entrega de mensagens são: requisições de informações de trânsito, clima, cotação de preços e notícas.

3.2 *Proxy* e Modelo Indireto

3.2.1 *Proxy*

O *proxy* representa o Mh na rede fixa, tem como função receber as mensagens do servidor e reenviá-las para o Mh, para isso, faz o recebimento, armazenamento e reenvio de respostas do servidor para o Mh em sua localização atual. O *proxy* possui uma variável contendo o endereço corrente do Mh, esta variável sempre é atualizada a cada migração do Mh.

3.2.2 Modelo Indireto

Em todos protocolos foi utilizado o modelo indireto para interações cliente-servidor. Proposto originalmente por Badrinath [BBIM93] é atualmente adotado em vários trabalhos. Nesse modelo, a mobilidade é feita explicitamente em todas as camadas do protocolo e os $Msss$ itermediam qualquer comunicação entre os Mhs e os servidores. A maior vantagem dessa abordagem consiste na possibilidade de construir protocolos de alto nível que percebem a mobilidade das estações, desse modo é possível utilizar tal informação para se adaptar a fatores como queda do sinal, proximidade do servidor, largura de banda, etc. O modelo também permite que os $Msss$ se comuniquem tanto na rede fixa quanto no meio sem fio, além de possibilitar que eles armazenem informações específicas para cada aplicação relacionadas com os status dos Mhs locais.

3.3 RDP - *Result Delivery Protocol*

Esta seção descreve o protocolo RDP [ES00]. Neste protocolo o cliente móvel instancia *proxies* em várias estações base, cada *proxy* corresponde a uma das requisições pendentes. Assume-se que toda a comunicação na rede fixa e entre os $Msss$ é confiável, isto é, não há perdas e/ou corrupção de mensagens. Uma verificação formal deste protocolo pode ser vista em [Tab02].

3.3.1 Visão Geral

Para cada requisição, o protocolo RDP cria um *proxy* no Mss responsável pelo Mh cliente no momento exato em que a requisição é enviada. O *proxy* age como representante do Mh na rede fixa e faz o recebimento, armazenamento e reenvio de respostas do servidor para o Mh em sua localização atual.

Para cada *proxy*, a variável *currentLoc* guarda uma referência para o Mss responsável pelo Mh depois do último *hand-off*. Para cada Mh, guarda-se uma lista de *proxies* ($PList$), cada entrada é uma referência para um Mss que tenha algum *proxy* criado pelo Mh e que esteja na pendência da resposta do servidor.

No momento em que a resposta de um servidor é recebida pelo *proxy*, o Mss_{prxy} reenvia a resposta através da mensagem $ForwardRes$ para a localização corrente do Mh guardada em *currentLoc*. Depois de recebida a confirmação do Mh (Ack), o Mss_{prxy} providencia para que os registros de pendência dessa resposta sejam apagados através do envio da mensagem $RemPList$. O envio da $RemPList$ a todos os $Msss$ garante que, mesmo que o Mh migre para outra célula, a remoção do *proxy* será concluída em toda a rede móvel.

3.3.2 *Hand-off*

Quando ocorre uma migração do Mh da célula do Mss_o para a célula do Mss_n, o protocolo de *hand-off* no Mss_n envia uma mensagem do tipo $Dereg$ para o Mss_o pedindo a ele a lista de *proxies* do Mh. Ao receber a mensagem $Dereg$, o Mss_o envia ao Mss_n uma mensagem do tipo $PList$ contendo a lista de todos os *proxies* do Mh e, em seguida o Mss_o remove o Mh em questão de sua lista de Mhs locais.

Ao receber a lista de *proxies* do Mh, o Mss_n toma a responsabilidade pelo Mh, guardando-o em sua lista de Mhs locais. Em seguida, o Mss_n atualiza o endereço do Mh nos *proxies* enviando a mensagem $UpdateCurrLoc$ para o Mss_{prxy} de cada *proxy*.

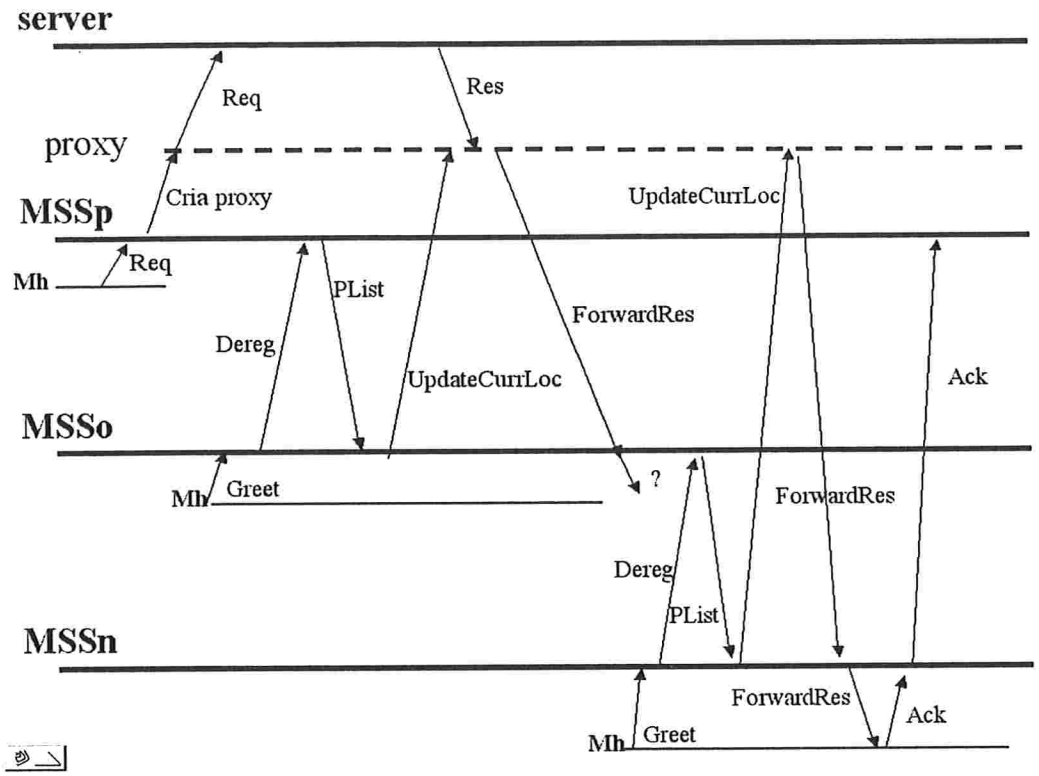

Figura 2: Cenário do Protocolo RDP

Cada Mss_{prxy} corrige a posição do Mh guardando o endereço do Mss_n na variável *currentLoc* do *proxy*. Depois, reenvia o resultado das requisições pendentes para a nova localização do Mh através da mensagem $ForwardRes$.

3.3.3 Exemplo

A Figura 2 representa um exemplo típico de execução do protocolo. Nela, estão indicadas todas as trocas de mensagens entre servidor, $Msss$, Mh e *proxy* (linha tracejada). Podemos, também, ver as mensagens geradas pelo protocolo quando o Mh envia uma requisição.

Neste exemplo, o Mh troca de célula antes de receber a mensagem $ForwardRes$ do Mss_o. Porém, sabemos que a mensagem $ForwardRes$ será reenviada quando Mss_o receber a mensagem $UpdateCurrLoc$, gerada no *hand-off* do Mh, garantindo, assim, que o Mh

receba a resposta do servidor, como já foi dito, mesmo não sendo de forma síncrona e ainda que haja migrações.

3.4 RCP - *A Reliable Connectionless Protocol*

Esta seção descreve o protocolo RCP [ESO00], versão modificada do protocolo RDP, na qual um cliente móvel cria e mantém no máximo um *proxy* a cada instante. Como no protocolo anterior, assume-se que toda a comunicação na rede fixa e entre os $Msss$ é confiável, isto é, não há perdas e/ou corrupção de mensagens.

3.4.1 Visão Geral

Um *proxy* é criado para cada Mh que deseja fazer requisições para o servidor. Os *proxies* são gerados no Mss responsável (Mss_{resp}) pela célula onde o Mh se encontra no momento em que ele iniciou a série de requisições e existirão até que sejam recebidas as confirmações (*Acks*) para todas as requisições feitas pelo Mh. Posteriormente, o mesmo Mh poderá causar a criação de um novo *proxy* no mesmo ou em outro Mss, o que depende da execução ou não de migrações. No entanto, um Mh possui no máximo um *proxy* a cada instante.

O *proxy* possui uma variável *currentLoc* que armazena o endereço do Mss responsável pelo Mh após o último *hand-off*, também possui uma lista (*requestList*) que contém identificadores para todas as requisições pendentes do Mh.

Para ser capaz de atualizar a variável *currentLoc* a cada migração, cada Mh possui uma referência para o *proxy* ($pRef$), o qual armazena o endereço do Mss onde o *proxy* está localizado. Quando um Mh não possui um *proxy* esta variável tem o valor *null*.

A cada momento em que um Mss recebe uma nova requisição de um Mh, ele checa a sua variável $pRef$. Se o valor dela é *null*, então um novo *proxy* é criado localmente para o Mh e a variável $pRef$ é atualizada. Caso contrário, o Mss reenvia a requisição para o *proxy*

armazenado em $pRef$.

Quando o resultado de uma requisição pendente chega no *proxy* (localizado no Mss_{prxy}), o Mss_{prxy} reenvia a mensagem para o Mss indicado na variável $currentLoc$, então a mensagem é entregue para o Mh através do meio sem fio.

Em condições normais, isto é, quando um Mh está ativo e permanece na célula por tempo suficiente, o Mh envia um Ack desta mensagem, que é reenviado do Mss_{resp} para o Mss_{prxy}. Quando o Ack chega ao *proxy*, este marca a mensagem como concluída, o quer dizer que já foi recebida pelo Mh e, em seguida a remove da lista $requestList$.

Se o Mss_{resp} não conseguir localizar o Mh, devido a migrações, falha na comunicação ou queda do serviço, o Mss_{resp} não reenvia a mensagem, o *proxy* irá fazê-lo quando receber a nova localização do Mh.

Enquanto o *proxy* não receber um Ack_R confirmando que o resultado da mensagem R foi recebido pelo Mh, ele irá reenviar o resultado toda vez que a variável $currentLoc$ for atualizada. Isto garante que todo resultado de uma requisição será eventualmente recebido pelo Mh. Porém, este resultado poderá ser recebido pelo Mh mais de uma vez.

A variável $pRef$ também contém uma *flag* chamada "Ready to Kill pRef" ($RKpR$), que indica quando um *proxy* já reenviou o resultado da sua última requisição pendente. Quando um *proxy* envia o resultado de sua última requisição pendente, ele seta a *flag* $RKpR = true$ do resultado e a envia para a localização atual do Mh. Ao receber a mensagem, o Mss_{resp} seta a *flag* $RKpR = true$ da variável $pRef$ e o resultado é reenviado para o Mh, isso se nenhuma outra requisição for iniciada até que o Mh envie o Ack do resultado para o Mss_{resp}, então $pRef$ é setada com $null$ e um Ack com a *flag* $RKpR = true$ é enviado para o Mss_{prxy}, ao receber o Ack o *proxy* é removido.

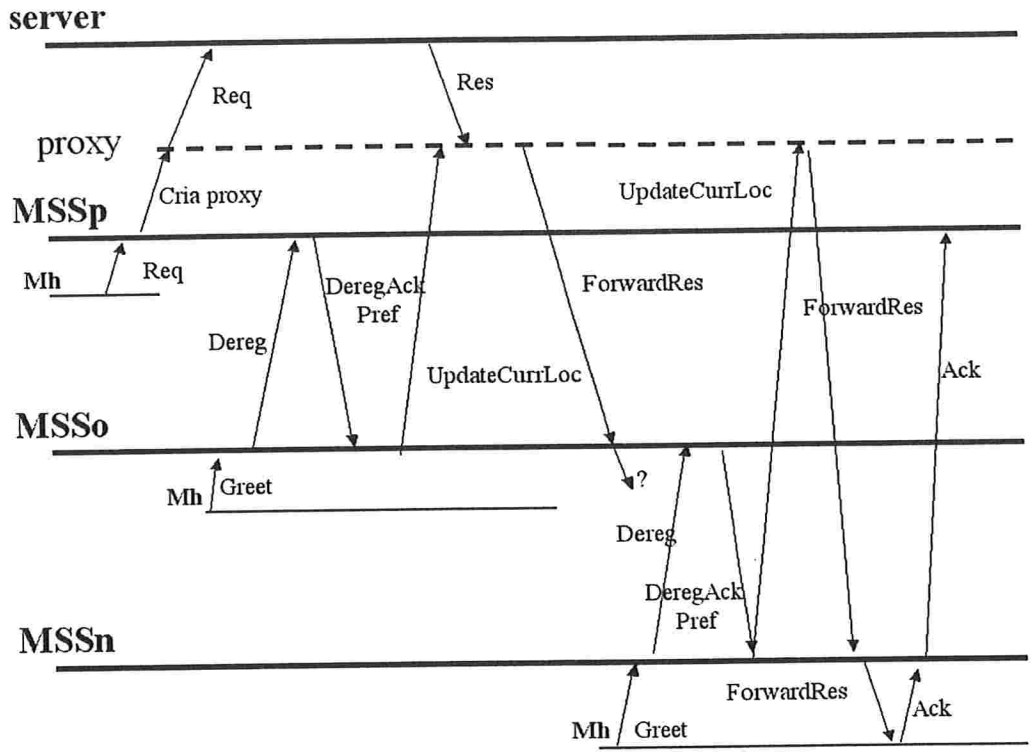

Figura 3: Cenário do Protocolo RCP

3.4.2 *Hand-off*

Quando o Mh migra da célula do Mss_o para a célula do Mss_n, o protocolo de *hand-off* no Mss_n envia uma mensagem $Dereg$ para o Mss_o requisitando a variável $pRef$ daquele Mh. No recebimento de $Dereg$, o Mss_o envia uma mensagem $DeregAck$ para o Mss_n que contém a $pRef$ do Mh, e o remove da sua lista de Mhs locais.

Ao receber a variável $pRef$ do Mh, o Mss_n torna-se responsável pelo Mh colocando-o na sua lista de Mhs locais, e envia seu endereço na mensagem $UpdateCurrLoc$ para o *proxy* identificado em $pRef$. O *proxy* atualiza a localização do Mh armazenando o endereço do Mss_n na variável $currentLoc$. Em seguida, reenvia o resultado de todas as requisições pendentes do Mh para sua nova localização.

3.4.3 Exemplo

A Figura 3 mostra um cenário onde o Mh faz uma única requisição para o Mss_p, então migra para o Mss_o e depois para o Mss_n. Assumindo que esta é a primeira requisição do Mh, um novo *proxy* é criado no Mss_p, sua variável *currentLoc* e a variável $pRef$ do Mh são setadas com Mss_p.

Agora, cada vez que o Mh migra, o novo Mss envia a mensagem $UpdateCurrLoc$ para o *proxy*, para que este atualize sua variável *currentLoc*. Quando o resultado da requisição chega no *proxy*, o Mss_p o reenvia para o Mss indicado em *currentLoc*, mesmo que durante este processo o Mh tenha migrado, como demonstrado na Figura 3. A mensagem contendo o resultado também contém uma *flag* $RKpR$ setada com *true*, isso porque a lista *requestList* do *proxy* tem somente uma entrada. Como o Mh neste espaço de tempo migrou para uma outra célula, o *proxy* não recebe um *Ack* do resultado, então ele irá reenviar a mensagem toda vez que receber a mensagem $UpdateCurrLoc$, até que finalmente receba um *Ack* do Mh.

Quando o Mss_n recebe o resultado com $RKpR = true$, ele seta a flag $RKpR = true$ da variável $pRef$, reenvia o resultado para o Mh e espera até receber um *Ack*. Assim que o *Ack* chega, e como $RKpR = true$, pois nenhuma requisição foi iniciada neste período, o Mss_n seta a referência $pRef$ do Mh para *null* e reenvia o *Ack* para o Mss_p com a *flag* $RKpR = true$. Finalmente, quando o *Ack* chega no Mss_p o *proxy* é removido.

3.5 SRDP - *Safe Result Delivery Protocol*

Esta seção descreve o protocolo SRDP [Dur01]. Sendo este uma modificação do protocolo RDP capaz de garantir a entrega confiável das mensagens de servidores a clientes móveis, mesmo que ocorram falhas na comunicação com os $Msss$. Nele assume-se que a comunicação entre $Msss$ e o servidor é confiável.

3.5.1 Visão Geral

Assim como no RCP, um Mh mantém no máximo um *proxy* a cada instante. Porém, toda vez que o Mh migra de uma célula para outra, um novo *proxy* será criado no Mss atual do Mh. O *proxy* antigo é descartado e, posteriormente, apagado pelo sistema através de técnicas de coleta de lixo [PS95].

Para possibilitar a criação de um novo *proxy* a cada migração, o Mh armazena localmente uma estrutura que implementa o padrão Memento [GHJV95]. Esta estrutura contém todas as informações das requisições pendentes do Mh. O Memento será criado no próprio Mh e enviado para o Mss na mensagem de *Greet*. A mensagem de *Greet* é enviada de uma estação móvel a uma estação base para iniciar o processo de *hand-off* durante a migração de célula ou após retornar de um estado de inatividade.

A cada nova requisição, o *proxy* criado no Mss atual irá redirecionar a mensagem para o servidor. Assim que o resultado da requisição chegar no *proxy*, ele irá redirecionar a mensagem para o Mh. Ao receber a confirmação, o *proxy* atualiza o estado do Memento.

Sempre que o Mh migra, ele envia uma mensagem de *Greet* para o novo Mss contendo o Memento que armazena as informações necessárias para a criação de um novo *proxy*. O Mss atual cria um novo *proxy* e reenvia todas as requisições pendentes para o servidor. Ao receber as requisições, se o servidor já tiver enviado a resposta da requisição, ele a processa novamente. Caso contrário, o servidor apenas atualiza o endereço para onde a resposta será enviada.

3.5.2 *Hand-off*

Nesse protocolo não há troca de mensagens entre os $Msss$ no processo *hand-off*. Na migração, o *proxy* antigo sempre é descartado. Para apagar os *proxies* obsoletos dos $Msss$ pode-se utilizar técnicas de coleta de lixo global [PS95]. Outra alternativa seria configurar

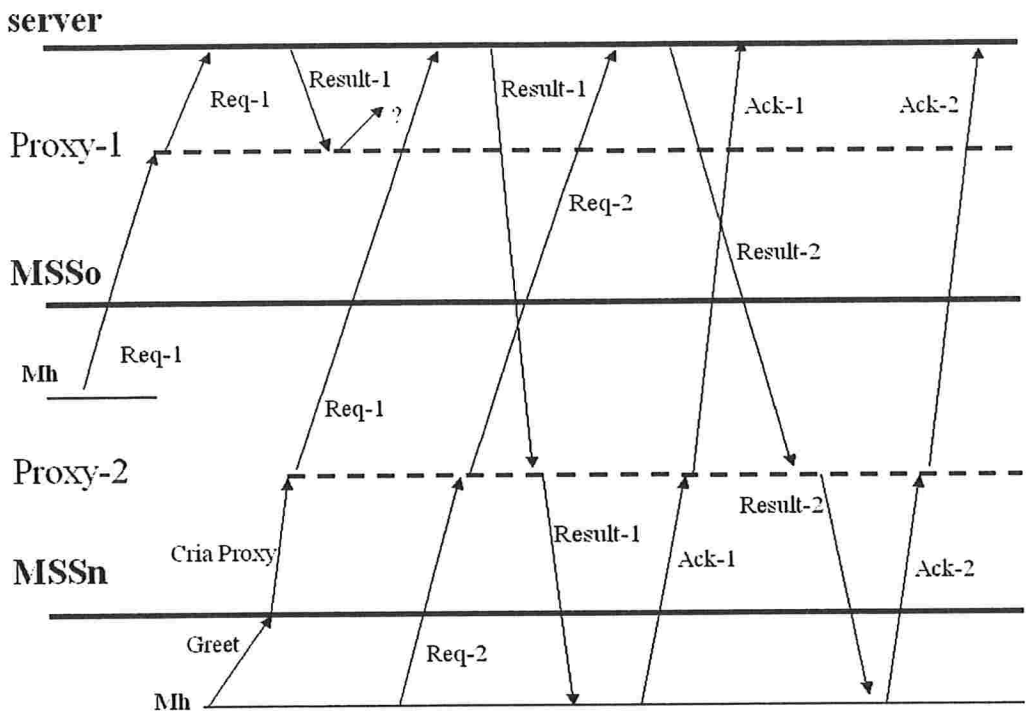

Figura 4: Cenário do Protocolo SRDP

um tempo limite para o recebimento de mensagens pelo *proxy*, no caso de nenhuma mensagem ser recebida neste período, o *proxy* é apagado. Esta técnica não afeta a confiabilidade do protocolo, pois o Mh armazena todas as informações necessárias para a criação de um novo *proxy* quando necessário.

3.5.3 Exemplo

A Figura 4 ilustra um cenário de execução do protocolo SRDP. O Mh faz uma requisição no Mss_o, nesse momento, é criado o $proxy1$ e a requisição é reenviada para o servidor. O servidor termina de processar a requisição e envia o resultado para o $proxy1$. Porém, neste período de tempo, o Mh migra para o Mss_n. Um novo *proxy* ($proxy2$) é criado no Mss_n e a requisição é reenviada para o servidor.

O servidor processa a requisição novamente e reenvia o resultado para o *proxy*2 e, este, por sua vez, reenvia a requisição para o Mh. Por fim, os *Ack*s são reenviados até chegar ao servidor.

3.6 ARCP - *Ack* RCP

Esta seção descreve o protocolo ARCP, um protocolo novo de entrega de mensagens desenvolvido neste trabalho. Este protocolo é similar ao protocolo RCP mas, assim como no protocolo SRDP, suporta falhas na comunicação entre $Msss$. Comparado ao protocolo SRDP, o ARCP reduz a quantidade de informações armazenadas no Mh, reduz o número de requisições feitas ao servidor e garante que o servidor processará as mensagens somente uma vez, evitando, assim, a sobrecarga do mesmo. Neste protocolo supõe-se que a comunicação entre os $Msss$ e o servidor nunca falha. Além disso, as falhas nos $Msss$ devem ser temporárias, pois se um Mss contém um *proxy* de um Mh, este Mh só irá receber o resultado de suas requisições pendentes quando o Mss estiver ativo novamente.

3.6.1 Visão Geral

O endereço do *proxy* é armazenado no Mh ($pRef$). O Mh enviará este endereço para o Mss atual na mensagem de *Greet* toda vez que fizer uma requisição. Desse modo, garantimos que o Mss atual sempre conhecerá o endereço do *proxy* do Mh. No protocolo RCP este endereço era trocado entre os $Msss$ antigo e atual no processo de *hand-off*. Neste caso, isto não será possível, já que há a possibilidade de ocorrer falhas na comunicação entre os $Msss$. Além disso, o Mh também terá que armazenar os identificadores das suas requisições pendentes ($requestIdList$). Ao contrário do protocolo SRDP, que armazena as requisições completas para ser capaz de reenviá-las ao servidor, no protocolo ARCP o Mh armazena somente os identificadores das mensagens pendentes. Assim, quando o Mh recebe um resul-

tado, se ele não tiver mais requisições pendentes, ele seta a variável $pRef$ com *null*. Este protocolo, diferente dos demais, é capaz de identificar o envio redundante de resultados. Ao chegar um resultado, se o identificador não estiver armazenado na lista que contém as requisições pendentes (*requestIdList*), então o protocolo assume que este resultado já foi recebido anteriormente.

Assim como no protocolo RCP, um *proxy* é criado para cada Mh que deseja fazer requisições para o servidor, sendo criado no Mss responsável (Mss_{resp}) pela célula onde o Mh se encontra no momento em que foi iniciada a série de requisições. O *proxy* existirá até que sejam recebidas as confirmações (*Acks*) para todas as requisições feitas pelo Mh. Os Mhs podem possuir no máximo um *proxy* a cada instante.

O *proxy* possui uma variável *currentLoc* que armazena o endereço do Mss responsável pelo Mh após o último *hand-off*. Também possui uma lista (*requestList*) que contém os identificadores para todas as requisições pendentes do Mh. A variável *currentLoc* é atualizada toda vez que o Mh migra ou passa do estado inativo para ativo. Quando o Mss_{resp} recebe a mensagem *Greet* do Mh contendo o endereço do *proxy*, ele envia uma mensagem $UpdateCurrLoc$ para o Mss_{prxy} do Mh.

Antes de um Mh enviar uma requisição para o Mss_{resp}, ele verificará a variável $pRef$. Se o valor dela é *null*, então, o Mh seta a variável com o endereço do Mss_{resp}, indicando que um *proxy* deve ser criado neste Mss. Quando o Mss_{resp} recebe uma requisição e $pRef$ está setada com o seu endereço, ele cria um *proxy* localmente caso não exista um, em seguida, adiciona a requisição na lista de requisições pendentes deste *proxy*. Caso $pRef$ esteja setada com o endereço de outro Mss, será reenviada a requisição para o Mss armazenado em $pRef$ e ficará armazenada no Mss_{resp} e, então, o Mss_{resp} irá reenviar a requisição para o Mss_{prxy} periodicamente, até receber a confirmação de que o Mss_{prxy} recebeu a requisição.

Quando o resultado de uma requisição chega no *proxy*, o Mss_{prxy} reenvia a mensagem para o Mss indicado na variável *currentLoc* (Mss_{resp}). Ao receber a mensagem, o Mss_{resp}

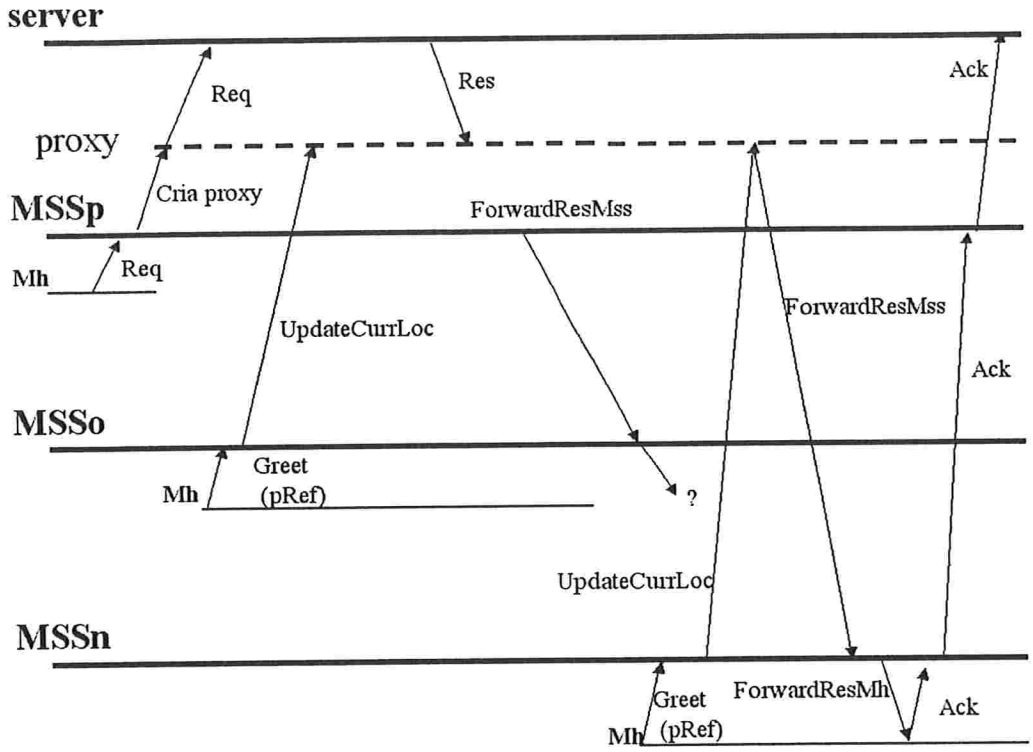

Figura 5: Cenário do ARCP

reenvia o resultado para o Mh. Uma vez recebida a mensagem, o Mh verifica se este é o último resultado pendente. Neste caso, o Mh seta a variável $pRef$ com *null*, logo em seguida, é setada também uma *flag* no *Ack* ($RKpR$) indicando que o *proxy* pode ser removido, essa mensagem é enviada para o Mss_{resp}, que ao receber o *Ack*, reenvia a mensagem para o Mss_{prxy}.

Quando o *Ack* chega no *proxy*, se $RKpR = true$ o *proxy* é removido. Caso o *Ack* não chegue ao *proxy* devido a falhas na comunicação, apesar de ser inutilizado, este *proxy* não será apagado. Os *proxies* inutilizados poderão ser apagados posteriormente com algoritmos de coleta de lixo [PS95].

3.6.2 *Hand-off*

Quando o Mh migra da célula do Mss_o para a célula do Mss_n, o primeiro envia uma mensagem de *Greet* para o Mss_n contendo o valor de $pRef$. Então, o Mss_n envia uma mensagem *UpdateCurrLoc* para o Mss_{prxy} contendo a nova localização do Mh. Ao receber a mensagem *UpdateCurrLoc*, o Mss_{prxy} envia todos os resultados de requisições pendentes para a nova localização do Mh.

3.6.3 Exemplo

A Figura 5 ilustra um cenário de execução do protocolo ARCP. Nela é possível notar que a troca de mensagens é similar a do protocolo RCP. No entanto, uma diferença é que nesse protocolo, o valor de $pRef$ é armazenado no Mh e não no Mss. Ao enviar uma mensagem de *Greet* ou uma requisição, o Mh envia o valor de $pRef$. No protocolo ARCP, não há troca de informações entre o Mss antigo e o atual durante o processo de *hand-off*.

3.7 Comparação Inicial

Nesta seção é feita uma comparação inicial dos protocolos baseada em suas especificações. Na Seção 3.7.1 são listadas todas as mensagens trocadas nos protocolos e é feita uma comparação com relação ao funcionamento dos protocolos. A Seção 3.7.2 contém os dados que são mantidos por cada protocolo.

3.7.1 Funcionamento e Especificação das Mensagens

Nesta seção são listadas todas as mensagens trocadas nos protocolos apresentados neste capítulo, permitindo, assim, uma melhor visualização do funcionamento de cada um. Primeiro, nas tabelas 1, 2, 3 e 4, são listadas todas as mensagens trocadas entre os elementos do sistema nos protocolos RDP, RCP, SRDP e ARCP respectivamente.

Mensagem	Elementos	Definição
Ack	$Mh \rightarrow Mss; Mss \rightarrow Mss$	Confirma o recebimento do resultado
Dereg	$Mss \rightarrow Mss$	Desregistra o Mh no Mss antigo
PList	$Mss \rightarrow Mss$	Envia a lista de *proxies* do Mh
AckProxy	$Mss \rightarrow Mss$	Encaminha o *Ack* para o *Proxy*
ForwardResMss	$Mss \rightarrow Mss$	Encaminha o resultado para o Mss atual do Mh
ForwardResMh	$Mss \rightarrow Mh$	Encaminha o resultado para o Mh
Greet	$Mh \rightarrow Mss$	Mudança para a célula controlada pelo Mss
RemPList	$Mss \rightarrow Mss$	Mensagem enviada para todos os $Msss$ para que removam o *proxy* da sua lista de *proxies*
Request	$Mh \rightarrow Mss; Mss \rightarrow Mss$	Requisição de um Mh
Result	$Server \rightarrow Mss$	Resultado de uma requisição
ServerRequest	$Mss \rightarrow Server$	Envia a requisição para o servidor
UpdateCurrLoc	$Mss \rightarrow Mss$	Atualiza a localização do Mh no *proxy*

Tabela 1: Mensagens trocadas no protocolo RDP

A diferença principal entre os protocolos RDP e RCP é o fato do protocolo RDP manter um *proxy* para cada requisição do Mh, ou seja, um Mh poderá ter vários *proxies*, já que um *proxy* é criado a cada requisição. Em contrapartida, o protocolo RCP mantém somente um *proxy* para cada Mh em cada instante, isto é, um *proxy* para cada série de requisições. No RDP, o *proxy* irá existir até que seja recebido o *Ack* referente à requisição tratada por ele. Por outro lado, no RCP, o *proxy* irá existir até que sejam recebidas as confirmações de todas as requisições tratadas por ele e , ainda, se o Mh iniciar uma nova série de requisições, um novo *proxy* será criado.

No protocolo RDP, após o recebimento do *Ack* pelo *proxy*, é necessário enviar a mensagem $RemPList$ para todos os $Msss$ da rede, garantindo, assim, que a referência para o *proxy*, na lista de *proxies* do Mh, seja apagada. Se a mensagem fosse enviada somente para o Mss atualmente responsável pelo Mh, e no momento do envio da mensagem o Mh migrasse para

Mensagem	Elementos	Definição
Ack	$Mh \rightarrow Mss; Mss \rightarrow Mss$	Confirma o recebimento do resultado
Dereg	$Mss \rightarrow Mss$	Desregistra o Mh no Mss antigo
DeregAck	$Mss \rightarrow Mss$	Envia a referência para o *proxy* do Mh
AckProxy	$Mss \rightarrow Mss$	Encaminha o *Ack* para o *Proxy*
ForwardResMss	$Mss \rightarrow Mss$	Encaminha o resultado para o Mss atual do Mh
ForwardResMh	$Mss \rightarrow Mh$	Encaminha o resultado para o Mh
Greet	$Mh \rightarrow Mss$	Mudança para a célula controlada pelo Mss
Request	$Mh \rightarrow Mss; Mss \rightarrow Mss$	Requisição de um Mh
Result	$Server \rightarrow Mss$	Resultado de uma requisição
ServerRequest	$Mss \rightarrow Server$	Envia a requisição para o servidor
UpdateCurrLoc	$Mss \rightarrow Mss$	Atualiza a localização do Mh no *proxy*

Tabela 2: Mensagens trocadas no protocolo RCP

outra célula, esta informação não seria removida da lista de *proxies*.

No protocolo RCP, quando o Mss responsável pelo Mh recebe o resultado de uma requisição com uma *flag* indicando que esta é a última requisição do Mh, o Mss_{resp} reenvia o resultado para o Mh e, assim que o *Ack* desta requisição é recebido por ele, caso o Mh não haja iniciado nenhuma requisição neste período, a referência para o *proxy* do Mh é setada como *null*. Em seguida, o *Ack* é enviado para o Mss_{prxy} indicando que o *proxy* pode ser removido. Desse modo, não é necessário o envio de uma mensagem para todos os $Msss$ da rede, o que é uma vantagem, visto que, trata-se de um processo muito custoso.

Outra desvantagem na abordagem de vários *proxies*, adotada pelo protocolo RDP, decorre da necessidade de, a cada migração do Mh, enviar uma mensagem para atualizar a sua localização para cada Mss que contém pelo menos um *proxy* do Mh. No protocolo RCP, só é necessário o envio de uma mensagem de atualização a cada migração, pois o Mh terá apenas um *proxy* a cada instante.

Mensagem	Elementos	Definição
Ack	$Mh \rightarrow Mss$; $Mss \rightarrow Mss$	Confirma o recebimento do resultado
AckProxy	$Mss \rightarrow Server$	Encaminha o *Ack* para o Servidor
ForwardResMh	$Mss \rightarrow Mh$	Encaminha o resultado para o Mh
Greet	$Mh \rightarrow Mss$	Mudança para a célula controlada pelo Mss
Request	$Mh \rightarrow Mss$	Requisição de um Mh
Result	$Server \rightarrow Mss$	Resultado de uma requisição
ServerRequest	$Mss \rightarrow Server$	Envia a requisição para o servidor

Tabela 3: Mensagens trocadas no protocolo SRDP

Em ambos protocolos, RDP e RCP, supõe-se que a comunicação na rede fixa é confiável, isto é, não há perdas e/ou corrupção de mensagens. Esta restrição se trata de um requisito de confiabilidade muito forte que impossibilita a utilização dos protocolos em ambientes reais.

Os protocolo SRDP e ARCP, diferentemente dos demais, garantem a entrega confiável das mensagens mesmo que ocorram falhas na comunicação entre os $Msss$. Nesses dois protocolos, assume-se que a comunicação entre $Msss$ e servidores é confiável, porém a comunicação entre os $Msss$ pode falhar. O protocolo ARCP requer que as falhas na comunicação entre os $Msss$ sejam temporárias.

No protocolo SRDP, uma mesma requisição poderá ter que ser reprocessada pelo servidor mais de uma vez. Requisições que demandam muito tempo de processamento podem causar sobrecarga no servidor. Além disso, ao invés de receber o pedido de uma requisição somente uma vez, como ocorre nos protocolos RDP, RCP e ARCP, o servidor, no protocolo SRDP, irá receber os pedidos de requisição toda vez que um Mh com requisições pendentes migrar.

As vantagens do protocolo ARCP em relação ao SRDP são, portanto, redução da quantidade de informações armazenadas pelo Mh, redução do número de requisições feitas para o servidor e garantia de que uma requisição será processada somente uma vez pelo servidor.

Já as desvantagens do protocolo ARCP em relação ao SRDP são duas, em primeiro lugar, o aumento no número de mensagens enviadas na rede fixa; por último, a restrição de

Mensagem	Elementos	Definição
Ack	$Mh \rightarrow Mss$; $Mss \rightarrow Mss$	Confirma o recebimento do resultado
AckProxy	$Mss \rightarrow Mss$	Encaminha o *Ack* para o *Proxy*
ForwardResMss	$Mss \rightarrow Mss$	Encaminha o resultado para o Mss atual do Mh
ForwardResMh	$Mss \rightarrow Mh$	Encaminha o resultado para o Mh
Greet	$Mh \rightarrow Mss$	Mudança para a célula controlada pelo Mss
Request	$Mh \rightarrow Mss$;$Mss \rightarrow Mss$	Requisição de um Mh
RequestAck	$Mss \rightarrow Mss$	Confirma o recebimento da requisição pelo *proxy*
Result	$Server \rightarrow Mss$	Resultado de uma requisição
UpdateCurrLoc	$Mss \rightarrow Mss$	Atualiza a localização do Mh no *proxy*

Tabela 4: Mensagens trocadas no protocolo ARCP

confiabilidade dos $Msss$, uma vez que o protocolo ARCP requer que as falhas de comunicação entre os $Msss$ sejam temporárias.

3.7.2 Dados armazenados

Nesta seção são listados todos os dados armazenados pelos protocolos apresentados neste capítulo, bem como, suas respectivas definições e o local onde são armazenados. Nas tabelas 5, 6, 7 e 8 são listados todos os dados armazenados nos protocolos RDP, RCP, SRDP e ARCP, respectivamente.

Com relação aos dados mantidos, a diferença dos protocolos se deve ao fato do protocolo SRDP armazenar as informações das requisições no Mh, informações necessárias para que um novo *proxy* seja criado a cada migração em sua nova localização. O tamanho de tais informações poderá aumentar indefinidamente, dependendo do número de requisições feitas pelo Mh. Esta característica do protocolo pode ser um problema, pois a memória disponível em um dispositivo móvel pode ser escassa. Os protocolos RDP e RCP não armazenam nenhum tipo de informação no Mh, somente nos $Msss$.

Dado	Local onde é armazenado	Definição
currentLoc	Mss_{prxy}	Armazena endereço do atual Mss_{resp} do Mh. Esta variável é atualizada a cada migração.
pList	Mss_{resp}	Armazena o endereço de todos os $Msss$ que contém pelo menos um *proxy* do Mh.
proxy	$Msss$. Os *proxies* são criados no Mss responsável pelo Mh no momento em que a requisição é enviada.	Representante do Mh na rede fixa.

Tabela 5: Dados mantidos no protocolo RDP

Dado	Local onde é armazenado	Definição
currentLoc	Mss_{prxy}	Armazena endereço do atual Mss_{resp} do Mh. Esta variável é atualizada a cada migração.
requestList	Mss_{prxy}	Lista de requisições pendentes de um Mh.
pRef	Mss_{resp}	Armazena endereço do Mss que contém o *proxy* do Mh
proxy	Mss_{prxy}	Representante do Mh na rede fixa.

Tabela 6: Dados mantidos no protocolo RCP

O protocolo ARCP armazena duas informações no Mh: o endereço do *proxy* e os identificadores das requisições, esta última informação, apesar de ser menor do que a requisição completa, também não tem tamanho constante.

Todos os protocolos armazenam as informações referentes aos *proxies* dos Mhs nos $Msss$. Nos protocolos RDP e RCP, os *proxies* são sempre removidos na execução dos protocolos. Em contrapartida, nos protocolos SRDP e ARCP, é necessário um processo de coleta de lixo para remover os *proxies* inutilizados.

Dado	Local onde é armazenado	Definição
proxy	Mss_{resp}	Representante do Mh na rede fixa.
Memento	Mh	Lista de requisições pendentes de um Mh.

Tabela 7: Dados mantidos no protocolo SRDP

Dado	Local onde é armazenado	Definição
currentLoc	Mss_{prxy}	Armazena endereço do atual Mss_{resp} do Mh. É atualizada a cada migração.
requestList	Mss_{prxy}	Lista de requisições pendentes de um Mh.
proxy	Mss_{prxy}	Representante do Mh na rede fixa.
pRef	Mh	Armazena endereço do Mss que contém o *proxy* do Mh
requestIdList	Mh	Armazena os ids das requisições pendentes

Tabela 8: Dados mantidos no protocolo ARCP

3.8 Tolerância a Falhas nos *Msss*

Os protocolos aqui apresentados armazenam informações cruciais para entrega de mensagens somente nos $Msss$, exceto o protocolo SRDP, no qual todas as informações importantes para entrega de mensagens também são armazenadas no Mh. Neste sentido, nos protocolos RDP e RCP, caso ocorra alguma falha nos $Msss$ as mensagens podem não ser entregues, ou senão, o Mh só voltará a receber as mensagens quando o Mss que contém o seu *proxy* ficar ativo novamente, como ocorre no protocolo ARCP.

Em [ARV93] são apresentadas duas alternativas para tolerância a falhas nos $Msss$. As duas abordagens consistem em replicação de dados para um conjunto de $Msss$ secundários, porém a diferença está em como esta replicação é realizada em cada uma delas. O número de falhas simultâneas toleradas é igual ao tamanho do conjunto de $Msss$ secundários. Sendo assim, quando ocorrer uma falha no Mss primário do Mh, basta que o Mh migre para a célula de um dos $Msss$ secundários. Se esta migração não for possível, uma alternativa seria

que cada célula pudesse ser coberta por mais de um Mss. Desse modo o Mh não precisará esperar que o Mss que contém o seu *proxy* volte a ficar ativo para receber o resultado de suas requisições. Em ambas abordagens, considera-se que existe um mecanismo para que a falha de um Mss possa ser detectada pelos $Msss$ das células vizinhas e pelo Mh.

A primeira alternativa é chamada de "Esquema Pessimista de Replicação". Neste caso, antes de um Mss alterar os dados referentes a um Mh, ele envia uma mensagem para os $Msss$ secundários, para que estes executem a mesma alteração, desse modo os dados ficarão consistentes no Mss primário e em todos os $Msss$ secundários. Após receber as confirmações (*Acks*) de todos os $Msss$ secundários, o Mss primário altera os dados localmente e envia uma mensagem para o Mh. Provocando, assim, um atraso nas mensagens enviadas para o Mh, contudo, é importante frisar que caso o Mss primário falhe, basta o Mh migrar para um dos $Msss$ secundários sem a necessidade de um procedimento de recuperação.

A segunda abordagem é chamada de "Esquema Otimista de Replicação", nela o Mss altera os dados referentes ao Mh localmente e envia as mensagens de atualização para os $Msss$ secundários e para o Mh de forma assíncrona. Dessa forma, não há atraso no envio de mensagens para o Mh. Entretanto, caso ocorra alguma falha no Mss primário, será necessário executar um procedimento de recuperação antes que o Mh migre para um dos $Msss$ secundários, que podem estar desatualizados em relação ao Mss primário. Uma maneira de sincronizar os dados de um dos $Msss$ secundários seria esperar que este Mss receba todas as mensagens de atualização antes de assumir a responsabilidade pelo Mh. Outra alternativa seria fazer o *rollback* no Mh, para que este fique consistente em relação ao seu novo Mss primário.

4 Simuladores de Protocolos

4.1 Sumário

Neste capítulo serão apresentados alguns simuladores de protocolos. A Seção 4.2 apresenta uma visão geral dos simuladores de protocolos. Os simuladores Coyote, GloMoSim e NS serão apresentados nas seções 4.3, 4.4 e 4.5 respectivamente. Na Seção 4.6 será apresentado o simulador MobiCS. Finalmente, na Seção 4.7, são apresentadas as justificativas da escolha do simulador MobiCS para a realização deste trabalho.

4.2 Visão Geral

Existem duas classes de simuladores: simuladores de redes e simuladores de protocolos. A função dos simuladores de rede é testar os protocolos em diferentes aspectos específicos dos elementos de rede e de topologia. Com as informações obtidas nas simulações, os protocolos podem ser implementados de maneira mais eficiente. Já, os simuladores de protocolos visam ser mais flexíveis e genéricos, o que facilita a prototipagem dos protocolos. Em outras palavras, os simuladores de protocolos são utilizados para testar a corretude dos protocolos em um ambiente de rede simulado e genérico, enquanto os simuladores de rede podem ser utilizados para testar os protocolos em ambientes de redes específicos [Roc01].

Para este trabalho, será utilizado um simulador de protocolos. A ênfase da comparação será dada no comportamento e desempenho dos protocolos de acordo com as diferenças em suas respectivas especificações. Isto é, determinar por exemplo a suposta melhoria no desempenho que foi causada pelo fato de se utilizar apenas um *proxy* para cada unidade móvel, principal diferença entre os protocolos RDP e RCP. Tendo em vista que não faz parte do escopo deste trabalho testar os protocolos em diferentes ambientes de rede, os simuladores de rede não serão utilizados nesta pesquisa.

4.3 x-Kernel e Coyote

O x-Kernel [HP91] é um arcabouço que provê uma arquitetura para construção e composição de protocolos de rede. Foi desenvolvido com o intuito de facilitar a implementação de protocolos de comunicação, diante desta perspectiva, nele um protocolo é visto como uma especificação de uma abstração de comunicação na qual os participantes trocam mensagens. Nesse modelo pode-se garantir ou não a entrega de mensagens, a troca de mensagens pode ser síncrona ou assíncrona, um número arbitrário de participantes pode estar envolvido na comunicação e as mensagens podem ter tamanho fixo.

Neste sentido, para suportar esse modelo são definidos três elementos básicos: protocolos, sessões e mensagens. Protocolos representam protocolos convencionais como IP, UDP e TCP, a relação entre os protocolos é definida na configuração do sistema; sessões são instâncias dos protocolos e mensagens representam os dados trocados entre as sessões e os protocolos.

Coyote [BHSC98] é uma extensão do x-kernel e nele os protocolos são desenvolvidos de forma modular e configurável. Aqui é utilizado o conceito de micro protocolos, isto é, módulos que implementam funcionalidades específicas dos protocolos. Portanto, um protocolo consiste em uma composição de micro-protocolos capazes de compartilhar dados entre si. Através dessa abordagem, obtém-se maior configurabilidade, eficiência, reusabilidade e flexibilidade.

Os micro-protocolos se comunicam através de eventos. Os eventos podem ser definidos pelo usuário no corpo dos micro-protocolos implementados. Há também os eventos pré-definidos, geralmente relacionados a mensagens recebidas de outras camadas. Para cada micro-protocolo pode-se definir um conjunto de eventos que serão tratados, condições para o recebimento de eventos e uma sequência de ações associadas ao recebimento de cada evento. Os eventos gerados são captados pelo sistema em tempo de execução e seus correspondentes tratadores são executados.

4.4 GloMoSim

O GloMoSim [ZBG98] é um ambiente escalável de simulação para sistemas de computação móvel. Foi desenvolvido em Parsec [BMT+98], linguagem baseada em C que provê simulação paralela utilizando eventos discretos. Além disso, é baseado em uma arquitetura de camadas que utilizam interfaces padrões para comunicação, ou seja, os modelos de protocolos pertencentes a uma de suas camadas só se comunicam com protocolos da camada acima ou abaixo através destas interfaces.

No Parsec uma entidade modela um processo físico e os eventos são modelados por mensagens trocadas entre as entidades.

O modelo utilizado é o de computação em grade, no qual cada processador é modelado como uma entidade Parsec e representa uma partição dos nós da rede. Assim, em cada entidade podem ser simulados vários nós da rede. Essa abordagem torna o sistema bastante escalável e, devido a isso, é possível aumentar consideravelmente o número de nós na simulação mantendo o número de entidades Parsec. Além disso, é utilizada somente uma entidade Parsec para representar toda a estrutura de camadas, desse modo, o número de camadas do sistema não afeta a quantidade de entidades utilizadas. Outra razão para essa abordagem se deve ao fato das camadas do sistema compartilharem dados, isto não seria possível caso cada camada fosse representada por uma entidade diferente.

A configuração dos modelos da simulação, o mapeamento das entidades nos processadores e o projeto de programas podem ser especificados utilizando-se o PAVE (Parsec Visual Environment), interface gráfica utilizada para desenvolver e configurar modelos de simulação Parsec visualmente.

Em sua dissertação, Rocha [Roc01] faz uma comparação dos simuladores aqui apresentados e constata que o GloMoSim fornece o melhor desempenho. No entanto, as abstrações de programação no GloMoSim são limitadas ao modelo de programação Parsec.

Alguns trabalhos recentes como [JBRAS03, BDM01, GLAM02] são exemplos de uso do GloMoSim. Em [JBRAS03] são acrescentados obstáculos a fim de restringir os movimentos das unidades móveis e a transmissão de dados no meio sem fio com o intuito de tornar os cenários mais realistas. Através de simulações no GloMoSim, foi constatado que tais obstáculos tem grande impacto na avaliação do desempenho dos protocolos.

Em [BDM01] é feita uma comparação entre três protocolos de acesso à midia com o objetivo de verificar quais os impactos causados no desempenho desses protocolos com relação ao tamanho da rede, número de conexões abertas, velocidade das unidades móveis e protocolos das camadas superiores. Por último, em [GLAM02] é feito um estudo da comunicação via broadcast.

O GloMoSim não é capaz de simular um ambiente com máquinas fixas e móveis [Lim03], o que torna sua utilização neste trabalho inviável, pois, os protocolos de entrega de mensagens descritos no capítulo anterior são especificados em ambientes híbridos.

4.5 Ns

Ns [BEF+00] é um simulador de eventos discretos e é utilizado para simulação de uma vasta gama de protocolos de rede. Entre os protocolos suportados pelo Ns, temos o TCP e suas variações e extensões, protocolos de roteamento e protocolos de multicast e, ainda, podem ser simuladas tanto redes fixas quanto redes sem fio.

O modelo de programação do Ns é divido em dois níveis [BEF+00]. Um deles é o núcleo da simulação que é implementado em uma linguagem de programação, o C++, esta fração do sistema será raramente modificada depois de implementada. O segundo nível, onde são especificados o controle, a configuração e a definição da simulação, é descrito em uma linguagem de *script* chamada *Tcl*. Dessa forma, no NS, as alterações na configuração da simulação e a construção de novas configurações podem ser implementadas com mais rapidez.

Uma das maiores vantagens desse simulador é a capacidade de emulação, que permite a troca de dados entre o simulador e uma rede real.

Para visualização, o Ns dispõe de uma ferramenta de animação chamada **nam** [Est00], que provê uma representação dinâmica da simulação.

Outra funcionalidade do simulador é a possibilidade de validação dos protocolos. A validação é baseada em simulações exaustivas, porém não é genérica o suficiente para ser aplicada a qualquer protocolo [Roc01].

O Ns fornece, através de uma abordagem orientada a objetos, maiores possibilidades de abstrações do que o GloMoSim.

Contudo, por não ser escalável, simular redes com mais de 100 nós torna-se uma tarefa muito difícil. Em [GN03] são descritas várias melhorias no simulador com o objetivo de torná-lo escalável. As alterações levam em consideração o fato de que as interferências causadas pela comunicação sem fio são limitadas. O simulador modificado é capaz de realizar simulações com mais de 3000 nós e tornou-se 30 vezes mais rápido do que a versão original.

O Ns é um dos simuladores mais utilizados e conhecidos, [AJK04, CM02, FML03] são exemplos atuais do uso do Ns. Nesses trabalhos, o protocolo TCP e algumas variações são analisados e simulados em situações diversas.

Além do fato de que a validação no Ns não é genérica o suficiente para ser aplicada a todos os tipos de protocolo, outro fator impediu a escolha desse simulador, assim como no GloMoSim, o módulo responsável pela mobilidade tem como objetivo simular os detalhes de mais baixo nível na comunicação entre as unidades móveis e as estações base, tais como controle de acesso ao meio, camada de enlace e recursos de rádio. Portanto, a simulação de protocolos de alto nível requer um grau de detalhamento muito maior do que o necessário. Considerando tudo isto, pode ser mais prático desenvolver um novo simulador para reproduzir e avaliar o comportamento de protocolos de alto nível do que utilizar simuladores como o Ns e o GloMoSim.

4.6 MobiCS

O MobiCS é um simulador de protocolos flexível que permite validar e acompanhar o desempenho de diferentes algoritmos que simulam protocolos para comunicação móvel [Roc01]. Desenvolvido como trabalho de mestrado de Rocha, aluno do Instituto de Matemática e Estatística da USP, no contexto do projeto SIDAM (Sistemas de Informação Distribuídos para Agentes Móveis)[ESS+00]. São dois os modos de simulação disponíveis neste simulador. No modo estocástico os protocolos são testados através de simulações exaustivas e aleatórias. No modo determinístico, o cenário de simulação é descrito detalhadamente pelo usuário através de um *script*.

O simulador MobiCS incorporou o conceito de micro-protocolos do ambiente Coyote implementando micro-protocolos, eventos e tratadores de eventos de uma forma mais adequada que o próprio Coyote [Roc01]. Isso ocorre porque o Coyote não especifica uma linguagem de programação de protocolos, os tratadores de eventos são funções que devem ser explicitamente registradas para tratar as respectivas mensagens.

Em [Lim03] é apresentada uma extensão do MobiCS com o objetivo de criar um conjunto de classes que melhorariam a usabilidade da biblioteca uma vez que reduz o trabalho do usuário na descrição de um novo protocolo de entrega de mensagens *unicast*. As particularidades dessa classe de protocolos foram analisadas visando determinar funcionalidades comuns.

Dessa forma, o MobiCS é o simulador que contém as características necessárias para a análise dos dados desta pesquisa, devido a fatores que serão apresentados na conclusão deste capítulo. No Capítulo 5 o simulador MobiCS é apresentado mais detalhadamente.

4.7 Conclusão

Entre os simuladores apresentados, o MobiCS apresenta maiores possibilidades de abstração e modularidade. É orientado a objetos como o Ns, porém, adiciona outro nível de abstração no modelo de programação através dos micro-protocolos, cujo conceito é implementado de uma maneira mais adequada que o Coyote que possui limitações.

Somente os simuladores Ns e MobiCS possuem módulos para validação dos protocolos. No entanto, a abordagem apresentada no Ns não é genérica o suficiente para ser utilizada em qualquer protocolo [Roc01]. Em contrapartida, o MobiCS apresenta dois modos de simulação: o determinístico e o estocástico. O modo determinístico permite a avaliação dos protocolos em situações críticas conhecidas. No modo estocástico são executadas simulações exaustivas.

Apesar do MobiCS ser mais lento se comparado aos demais simuladores, não possuir uma ferramenta gráfica para a especificação da simulação e ser incapaz de emulação de rede, este foi o simulador escolhido, pois abordaremos protocolos de alto nível que não estão diretamente associados a protocolos de rede. Nesse caso, deve-se priorizar a facilidade de programação e flexibilidade do simulador. Os simuladores Ns e GloMoSim requerem um grau de detalhamento muito maior do que o necessário, já que foram desenvolvidos visando a simulação de protocolos de camadas inferiores ou a simulação de algoritmos bem específicos. Além disso, o MobiCS é o único simulador que permite, de uma maneira flexível e genérica, a análise de desempenho, testes e validação de protocolos em um único ambiente.

5 Visão Geral do MobiCS

5.1 Sumário

O objetivo deste capítulo é descrever o simulador MobiCS. A Seção 5.2 contém uma visão geral do simulador. As seções 5.3 e 5.4 descrevem os pacotes e a programação de protocolos respectivamente. Em seguida, a Seção 5.5 descreve os tipos de simulação que o MobiCS suporta.

Uma nova biblioteca, que estende as funcionalidades do simulador visando simplificar o seu uso sem reduzir a flexibilidade e extensibilidade [Lim03], foi desenvolvida para o MobiCS. A mesma será descrita na Seção 5.6.

Finalmente, na Seção 5.7 serão descritas as funcionalidades adicionais implementadas no MobiCS neste trabalho, o que permitiu, ao final da simulação, emitir um relatório com algumas métricas que serão utilizadas para avaliar os protocolos, tais como, número de mensagens enviadas e retransmitidas separadas por tipo, duração média de uma requisição, quantidade de requisições concluídas e não concluídas, número de mensagens enviadas na rede fixa, número de mensagens enviadas no meio sem fio e número de mensagens enviadas no processo de *hand-off*.

5.2 Visão Geral

O MobiCS [Roc01, ER00, RE01b] é um simulador de protocolos flexível que permite validar e acompanhar o desempenho de diferentes algoritmos que simulam protocolos para comunicação móvel. Como já foi dito, o MobiCS foi desenvolvido por Rocha [Roc01] para o projeto SIDAM [ESS+00]. Nesse simulador, estão disponíveis dois modos de simulação. O modo estocástico no qual protocolos são testados através de simulações exaustivas e randômicas e o modo determinístico, cujo cenário de simulação é descrito detalhadamente pelo usuário

através de um *script*. Este simulador já foi utilizado para implementar e simular protocolos como o Mobile IP [JM96], MCAST [AB93], RDP [ES00], RCP [ESO00], SRDP [Dur01], AM^2C [End99] e o iAM^2C [RE01a].

5.3 Pacotes do MobiCS

O MobiCS é um arcabouço Java cujas classes definem o comportamento de elementos de um sistema de computação móvel, como estação base, unidade móvel, etc. Além disso, contém outras classes que representam o protocolo que será implementado e as mensagens que serão enviadas ao longo da simulação.

Os principais pacotes do MobiCS são: o mobics.ppi, que contém as classes que representam os protocolos e, dentro do qual, podemos encontrar dois subpacotes (o mobics.ppi.protocol que é o protocolo propriamente dito e o mobics.ppi.message que são as mensagens enviadas); também temos o mobics.simulation no qual se apresentam as classes que representam a simulação, isto é, os cenários que serão executados; depois o mobics.network que contém as classes necessárias para a configuração e criação da simulação e, neste pacote, estão representadas as máquinas fixas, estações móveis, canais de comunicação e estações base, todas devem estender uma classe base (mobics.network.Element) que contém um conjunto de atributos e métodos comuns a todos os tipos de máquinas simuladas.

5.4 Protocolos no MobiCS

Devido à finalidade de reutilização de código, encapsulamento e transparência para o programador, as classes do pacote mobics.ppi provêem a comunicação entre as várias camadas de software do simulador, desse modo, não é necessário que o usuário se preocupe com detalhes, concentrando, assim, seu esforço somente na lógica dos protocolos. Em resumo, o MobiCS permite programação de protocolos em uma linguagem de alto nível. Além disso, é

importante ressaltar a forma como o MobiCS foi construído, isto é, um protocolo é dividido em micro-protocolos, módulos que implementam funcionalidades específicas dos protocolos, sendo um exemplo de micro-protocolo o modo como o protocolo geral se comporta no *hand-off*.

Estes micro-protocolos são declarados como interfaces e definem quais as mensagens devem ser tratadas por ele. Por exemplo, o micro-protocolo de *hand-off* geralmente deve tratar a mensagem *Greet*.

5.4.1 Declaração de Mensagens

As mensagens são classificadas de acordo com a situação em que são enviadas. Os possíveis tipos de mensagens são: HandoffMessage, enviadas no processo de *hand-off*; WiredMessage, enviadas através da rede fixa; WirelessMessage, enviadas através do meio sem fio e AppMessage, mensagens de aplicação. Para definir o tipo de uma mensagem basta implementar a interface correspondente ao seu tipo. Ou seja, existe uma interface para cada tipo de mensagem, com os mesmos nomes citados acima, localizada no pacote mobics.ppi.message. Toda mensagem no MobiCS deve implementar uma dessas interfaces e estender a classe mobics.ppi.Message, classe que determina um conjunto mínimo de informações que uma mensagem deve conter.

O pacote mobics.ppi.message declara, ainda, uma mensagem padrão do MobiCS, a mensagem Greet, enviada de uma estação móvel a uma estação base para iniciar o processo de *hand-off* durante a migração de célula ou após retornar de um estado de inatividade.

5.4.2 Programação de Micro-Protocolos

Para declarar um micro-protocolo é necessário criar uma interface que estende alguma das interfaces WiredModule, WirelessModule e HandoffModule. Tais interfaces estão localizadas no pacote mobics.ppi.protocol. Assim como na declaração de mensagens, essas interfaces

determinam a funcionalidade do micro-protocolo. Nelas devem ser declarados todos os tratadores para as mensagens recebidas pelos micro-protocolos. Os tratadores são implementados como métodos e sua assinatura deve ser *whenMensagem()*, onde Mensagem é o nome da mensagem que deve ser tratada, por exemplo, o micro-protocolo de *hand-off* deve conter o método *whenGreet()*, tratador da mensagem Greet.

5.5 Modos de Simulação

No pacote mobics.simulation estão contidas as classes básicas para criar e iniciar uma simulação. Os dois modos de simulação possíveis são determinístico, representado pela classe DetermSimulation e o modo estocástico, representado pela classe StochSimulation.

Para criar uma simulação, o usuário deve estender uma dessas classes, de acordo com o modo de simulação desejado, e ainda declarar os atributos e métodos necessários para a configuração da simulação.

5.5.1 Modo Determinístico

É objetivo da simulação determinística testar o protocolo em cenários críticos que serão detalhadamente descritos pelo usuário. Assim, é possível analisar o comportamento do protocolo em um sequência de passos específica.

Uma simulação determinística deve estender a classe DetermSimulation, o *script* determinístico que será seguido durante a simulação é implementado no método *script()* e deve ser definido pelo usuário.

Para configurar a simulação são oferecidas várias interfaces ao usuário, com elas, o usuário pode determinar os tipos de mensagens que serão exibidos, criar pontos de sincronização e controlar as máquinas simuladas.

5.5.2 Modo Estocástico

A simulação estocástica consiste em executar o protocolo de maneira exaustiva e aleatória, possibilitando, assim, a avaliação do desempenho do protocolo em um cenário realista e aleatório. É impossível criar todos os possíveis cenários determinísticos, portanto a simulação estocástica complementa a avaliação da corretude do protocolo.

A principal diferença entre o modo de simulação determinístico e estocástico é o modo como o usuário implementa a simulação. Na simulação determinística o usuário deve especificar os elementos de simulação e cada ação que será efetuada durante a execução do protocolo. Enquanto que na simulação estocástica o usuário estende a classe StochSimulation e apenas define os elementos de simulação (máquinas fixas, estações base, estações móveis, etc.) no método *configure()* e o simulador, por sua vez, se encarrega de criar a sequência de passos que será executada.

5.6 Biblioteca que provê usabilidade

Em [Lim03] é descrita uma nova biblioteca desenvolvida para o MobiCS que visa simplificar o seu uso para implementação de protocolos de entrega de mensagens. Essa biblioteca será utilizada na implementação dos protocolos descritos neste trabalho. Além da biblioteca, Lima também implementou algumas funcionalidades novas no simulador.

Uma das funcionalidades implementadas foi proporcionar ao simulador o envio de informações específicas no processo de *hand-off*. A única informação enviada para o Mss na versão inicial do MobiCS era o endereço do antigo Mss através da mensagem $Greet$. Desse modo, é possível definir uma mensagem de migração específica para cada protocolo.

Outra é a possibilidade de limitar a migração dos Mhs somente para células adjacentes a célula atual do Mh. Na versão inicial do simulador um Mh poderia migrar para qualquer célula do sistema, não permitindo, assim, uma simulação realista. No mundo real um Mh

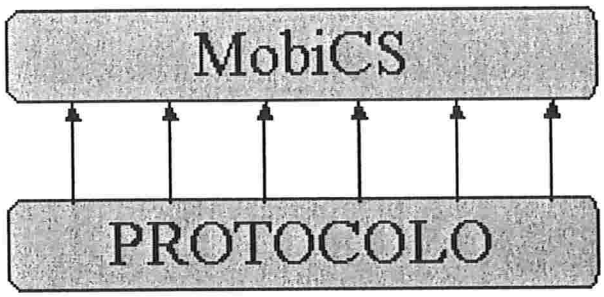

Figura 6: Interação com o MobiCS sem a biblioteca

só pode migrar para células vizinhas da sua célula atual e somente nos casos de inatividade, um Mh poderá migrar para uma célula mais distante.

Também foi adicionado ao simulador a possibilidade de falha nos Mss. Na versão original, supunha-se que não ocorriam falhas na rede fixa, o que tornava a simulação menos realista e muito limitada.

Por fim, introduziu-se o conceito de um servidor mais realista. Na versão inicial, o servidor era considerado um Mss com algumas funções adicionais. Agora, um servidor não possui mais as funcionalidades de um Mss e só é capaz de se comunicar com a rede fixa e não com o meio sem fio.

Então, a biblioteca, apresentada em [Lim03], contém a implementação de diversas funções comuns aos protocolos de entrega de mensagens, simplificando, desse modo, o uso do simulador para essa classe de protocolos. Algumas das vantagens de tal abordagem são redução de linhas de código necessárias para o desenvolvimento de um protocolo e utilização de um código menos passível de erro, uma vez que já foi verificado pelo desenvolvedor da biblioteca.

A implementação de um protocolo no MobiCS sem o uso da biblioteca é ilustrada na Figura 6. Para construir um protocolo, o desenvolvedor instancia ou estende as classes

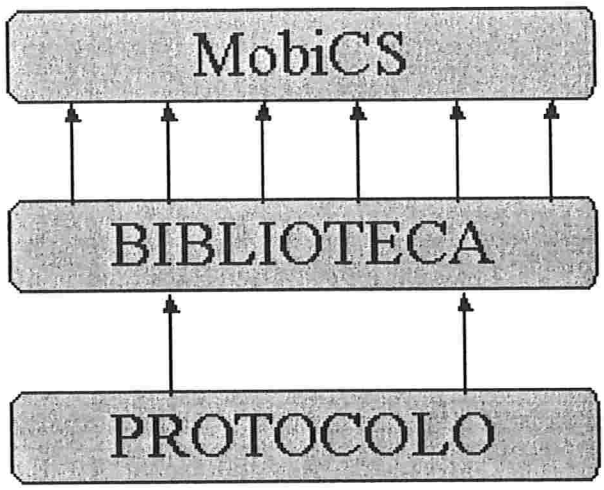

Figura 7: Interação com o MobiCS utilizando a biblioteca

do simulador. Lima inseriu uma nova camada na estrutura e, sendo assim, a biblioteca instancia ou estende as classes do simulador e o desenvolvedor, por sua vez, utiliza as classes da biblioteca. A Figura 7 ilustra a nova estrutura.

5.7 Modificações realizadas no MobiCS para este trabalho

Com o intuito de facilitar a comparação dos protocolos apresentados no Capítulo 3, o simulador MobiCS precisou passar por algumas alterações. Devido a tais modificações, o simulador agora é capaz de emitir um relatório, ao final da execução, contendo alguns dados sobre as mensagens que foram trocadas durante a simulação. Na versão original do MobiCS, apenas era impresso o roteiro detalhado da simulação, isto é, todas as ações que foram executadas na simulação, não havia um relatório resumido com informações gerais das mensagens trocadas, o que dificultava muito a comparação de protocolos.

A escolha dos dados que deveriam estar presentes neste relatório foi baseada nos dados necessários para a comparação dos protocolos do capítulo 3 e, ainda, em alguns trabalhos,

como [Rib02, Lim03], que utilizaram o simulador MobiCS para comparação de protocolos. Nos trabalhos citados, foram extraídos os dados utilizados pelos usuários para realizar as comparações. Sendo assim, os usuários do MobiCS tiveram que adicionar rotinas no simulador ou nos protocolos para adquirir tais informações. Portanto, as modificações realizadas no MobiCS foram feitas para que usuários futuros não tenham que implementar tais funcionalidades novamente.

No relatório são listados os seguintes dados para o usuário:

- Número total de mensagens enviadas separadas por tipo;
- número total de mensagens retransmitidas;
- custo *wired*, mensagens trocadas na rede fixa;
- custo *wireless*, mensagens trocadas no meio sem fio;
- custo *hand-off*, mensagens trocadas no processo de migração dos Mhs;
- tempo médio simulado levado desde o início de uma requisição até que o destinatário final receba o resultado;
- número total de requisições concluídas e não concluídas;
- lista com os identificadores das requisições não concluídas;
- possibilidade de complementar o relatório com uma classe implementada pelo usuário do simulador.

5.7.1 Número total de mensagens enviadas separadas por tipo

Criamos o pacote mobics.controller.report contendo as classes utilizadas para implementar o relatório descrito neste capítulo. Dentro desse pacote, foi criada uma classe nova,

MessageCounter, que possui um método capaz de contabilizar as mensagens enviadas separadas por tipo. O método *mobics.network.Element.onSend(NetMessage)* é executado toda vez que uma mensagem é enviada na simulação. Por isso, foi colocada uma chamada para *MessageCounter.addMessage(NetMessage)* no corpo desse método. No método *MessageCounter.addMessage(NetMessage)* verifica-se o tipo (classe) da mensagem e, em seguida, um contador para este tipo de mensagem é criado, caso ainda não exista, então, atualiza-se o contador.

5.7.2 Número total de mensagens retransmitidas

Para o simulador não há diferenças entre uma mensagem enviada pela primeira vez ou uma mensagem retransmitida. Portanto, para conseguir contabilizar as mensagens foi necessário adicionar um atributo à classe mobics.ppi.message.Message que indica se a mensagem é uma retransmissão ou não. Tal atributo deve ser setado pelo usuário do simulador na implementação do protocolo antes da mensagem ser retransmitida. Assim, no método *mobics.controller.report.MessageCounter.addMessage(NetMessage)* é feito um teste que verifica se é uma retransmissão, em caso positivo, o contador de retransmissões para aquele determinado tipo de mensagem é criado, caso ainda não exista, então o contador é atualizado.

O código a seguir é a implementação do método *addMessage()* da classe mobics.controller.report.MessageCounter,

```
// Contabiliza as mensagens enviadas e retransmitidas separadas pelo tipo
// de mensagem. Além disto, contabiliza o tempo levado desde de o envio
// de uma requisição até que o destinatario final receba a resposta.
public static void addMessage(NetMessage msg){
    Class msgClass = msg.getType();
    // Se existe um contador para este tipo de mensagem, atualiza o contador.
    if(messageCounter.containsKey(msgClass)){
        int value = ((Integer)messageCounter.get(msgClass)).intValue();
        value++;
        messageCounter.put(msgClass, new Integer(value));
    }else{// Senão cria um novo contador
        messageCounter.put(msgClass, new Integer(1));
    }
    // Se a mensagem é uma retransmissão atualiza o contador
    if(msg.getMessage().getRetrans()){
        if(retransCounter.containsKey(msgClass)){
            int value = ((Integer)retransCounter.get(msgClass)).intValue();
            value++;
            retransCounter.put(msgClass, new Integer(value));
        }else{
            retransCounter.put(msgClass, new Integer(1));
        }
    }
    // Atualiza o contador de mensagens dependendo do micro-protocolo
    if(msg.getMessage() instanceof WirelessMessage)
        totalWirelessMsg++;
    if(msg.getMessage() instanceof WiredMessage)
        totalWiredMsg++;
    if(msg.getMessage() instanceof HandoffMessage)
        totalHandoffMsg++;
}
```

5.7.3 Custo Wired, Wireless e Handoff

No método *mobics.controller.report.MessageCounter.addMessage(NetMessage)*, verifica-se o tipo da mensagem através da interface implementada (mobics.ppi.message.WiredMessage, mobics.ppi.message.WirelessMessage ou mobics.ppi.message.HandoffMessage) e, em seguida, é atualizado o contador respectivo.

5.7.4 Tempo médio simulado levado desde o início de uma requisição até que o destinatário final receba o resultado

Devido à importância de se saber o tempo médio simulado desde o início de uma requisição até que o destinatário final receba o resultado, foi preciso criar uma interface nova mobics.controller.report.IdMessage. As mensagens que indicam o início e o final de uma requisição devem implementar essa interface. As classes que implementam a interface devem implementar o método *getId()*. Como somente o envio de uma mensagem não caracteriza o início ou o final de uma requisição, o usuário do simulador deverá incluir na implementação de seus protocolos chamadas para os métodos *MessageCounter.initialRequest(NetMessage)* e *MessageCounter.finalRequest(NetMessage)*, sempre que uma requisição é iniciada ou finalizada.

No método *MessageCounter.initialRequest(NetMessage)*, armazena-se o *id* e a unidade de tempo simulado em que a requisição foi iniciada. No método *mobics.controller.report.MessageCounter.finalRequest(NetMessage)*, através do método *getId()*, obtém-se a unidade de tempo simulado em que a requisição foi iniciada e calcula-se o tempo simulado total levado desde o início de uma requisição até que o destinatário final receba o resultado. Este tempo é somado a uma variável que armazena a soma dos tempos de todas as requisições, no final da simulação, calculamos a média de tempo das requisições concluídas.

A unidade de tempo simulado é obtida através da classe mobics.controller.stochastic.StochController, a mesma classe controla a execução dos eventos, na ordem de seus respectivos tempos simulados.

5.7.5 Número total de requisições concluídas e não concluídas

No método *MessageCounter.finalRequest(NetMessage)*, após o cálculo do tempo simulado, a requisição é removida da lista de requisições não concluídas e o contador de requisições concluídas é atualizado. Sendo assim, no final da simulação, tem-se o número total de requisições concluídas, assim como os identificadores de todas as requisições que não foram concluídas durante a simulação.

5.7.6 Lista com os identificadores das requisições não concluídas

No final da simulação é impressa a lista com os identificadores das requisições não concluídas

5.7.7 Possibilidade de complementar o relatório com uma classe implementada pelo usuário do simulador

Caso o usuário queira complementar o relatório impresso pelo simulador, ele pode construir uma classe que implemente a interface mobics.controller.report.Report. Esta classe deverá implementar o método *report()*, onde serão impressos os dados do relatório desenvolvido pelo usuário. Na classe que o usuário irá configurar a simulação, deverá ser incluída uma chamada para o método *mobics.controller.report.MessageCounter.setUserReport(Report)* passando o relatório implementado.

Segue abaixo a impressão do relatório:

```
*************** REPORT *************
Message Type: <class rcp.impl.messages.Dereg> occurences: 14385
Message Type: <class frame.messages.ForwardResMss> occurences: 1332
Message Type: <class frame.messages.Ack> occurences: 997
Message Type: <class rcp.impl.messages.Arrive> occurences: 14432
Message Type: <class rcp.impl.messages.UpdateCurrLoc> occurences: 12360
Message Type: <class rcp.impl.messages.DeRegAck> occurences: 14376
Message Type: <class frame.messages.ForwardResMh> occurences: 1524
Message Type: <class frame.messages.ServerRequest> occurences: 997
Message Type: <class frame.messages.Result> occurences: 997
Message Type: <class frame.messages.AckProxy> occurences: 863
Message Type: <class frame.messages.SystemRequest> occurences: 1869

*************** RETRANSMISSIONS *************
Message Type: <class frame.messages.SystemRequest> occurences: 867

Wireless Messages: 18822
Wired Messages: 45310
Handoff Messages: 55553
Total: 64132
Total Requests Concluded: 998
Average Time: 382.8014042126379 UTs
Mensagens não concluídas 1:
Request Id: frame.util.IdRequisition@1292d26 request time 2282280
```

6 Implementação do Protocolo ARCP

6.1 Sumário

Este capítulo descreve o processo de implementação do Protocolo ARCP utilizando o simulador MobiCS. Na Seção 6.2 é apresentada a implementação do protocolo desenvolvido neste trabalho, o ARCP. A Seção 6.3 descreve os testes determinísticos que foram realizados a fim de testar a corretude do protocolo.

6.2 Implementação do Protocolo ARCP

Nesta seção serão apresentados os métodos que tratam as mensagem recebidas pelas principais classes representantes de cada um dos elementos de rede (*Mh*, *Mss*, *Server*) do protocolo ARCP. Tais métodos são capazes de tratar mensagens enviadas na rede fixa, meio sem fio e também no processo de *hand-off*. O MobiCS estabelece como convenção que a sintaxe para cada método usado no tratamento de uma mensagem determinada seja *when-Mensagem*.

ARCPMss.java

Contém métodos para tratar todas as mensagens recebidas por um Mss.

- *void whenGreet(Message)*: trata mensagem do tipo *Greet*, recebida pelo Mss quando um Mh migra ou volta de um período de inatividade. A mensagem irá conter a referência para o *proxy* ($pRef$) do Mh. Se o valor de $pRef$ não é *null*, isto é, já existe um *proxy* para o Mh, é enviada uma mensagem $UpdateCurrLoc$ para o Mss_{proxy} com o objetivo de atualizar a localização do Mh no *proxy*.

- *void whenUpdateCurrLoc(Message)*: trata a mensagem *UpdateCurrentLoc*, enviada toda vez que o Mh migra de uma célula para outra ou, ainda, quando volta de um

período de inatividade. Após receber uma mensagem desse tipo, o Mss_{prxy} envia todos os resultados das requisições pendentes para a nova localização do Mh.

- *void whenRequest(Message)*: trata a mensagem do tipo *Request* enviada pelo Mh. Nela encontra-se a referência para o *proxy* ($pRef$) do Mh. Se o valor de $pRef$ é o endereço do Mss que recebeu a mensagem, cria-se um *proxy* para o Mh localmente, isso se ainda não existir um *proxy*, e a requisição é adicionada a lista de requisições pendentes do *proxy*. No caso de $pRef$ conter o endereço de outro Mss, a requisição é reenviada para o Mss armazenado em $pRef$ e, em função disso, a requisição ficará armazenada nesse Mss e deverá ser enviada, periodicamente, até que o Mss destinatário envie a mensagem *RequestAck*. Sendo assim, quando um Mss recebe a requisição de outro Mss, uma mensagem *RequestAck* deve ser enviada confirmando o recebimento da requisição.

- *void whenRequestAck(Message)*: trata a mensagem do tipo *RequestAck* enviada pelo Mss_{prxy}. Tal mensagem indica que o M_{resp} não precisa mais enviar a requisição ao Mss_{prxy}, pois a mesma já foi recebida.

- *void whenResult(Message)*: trata a mensagem do tipo *Result* enviada pelo servidor. A função dessa mensagem é enviar o resultado de uma requisição feita pelo Mh. Após recebida, o Mss_{prxy} verifica se o Mh é local, em caso afirmativo, envia o resultado para o Mh através da mensagem $ForwardResMh$, caso contrário, reenvia o resultado para a atual localização do Mh através da mensagem $ForwardResMss$.

- *void whenForwardResMss(Message)*: trata a mensagem do tipo $ForwardResMss$, enviada do Mss_{prxy} para o Mss atualmente responsável pelo Mh. O Mss_{resp} reenvia o resultado para o Mh através da mensagem $ForwardResMh$.

- *void whenAck(Message)*: trata a mensagem do tipo *Ack* enviada pelo Mh. Quando o

proxy é local, é verificada a *flag* da mensagem $RKpR = true$ e remove-se o *proxy*, senão apenas é removida a requisição da lista de requisições pendentes. Caso o *proxy* não seja local, a mensagem $AckProxy$ é enviada para o Mss_{prxy} indicado na mensagem.

- *void whenAckProxy(Message)*: trata a mensagem do tipo *Ack* enviada pelo Mss responsável pelo Mh. Primeiro, verifica-se a *flag* da mensagem $RKpR = true$ e remove-se o *proxy*, caso contrário apenas remove-se a requisição da lista de requisições pendentes.

ARCPMh.java

A classe ARCPMh.java contém métodos para tratar todas as mensagens recebidas por um Mh.

- *void whenUserRequest(Address, int)*: mensagem recebida de uma aplicação que indica o início de uma requisição. Verifica se $pRef = null$, caso positivo, seta $pRef$ com o endereço do Mss atualmente responsável pelo Mh, indicando que um *proxy* deve ser criado neste Mss. E ainda envia a mensagem *Request* para o Mss_{resp} contendo o valor de $pRef$

- *void whenForwardResMh(Message)*: trata o recebimento da mensagem $ForwardResMh$, enviada por um Mss, contendo o resultado de uma requisição. É verificado se não existem mais mensagens pendentes, caso haja, seta $pRef = null$. Depois, é enviada a mensagem *Ack* para o Mss_{resp}, se esta for a última requisição pendente, seta a *flag* $RKpR = true$ do *Ack* indicando que o *proxy* deve ser removido.

ARCPServer.java

A classe ARCPServer contém métodos necessários para tratar todas as mensagens recebidas por um servidor.

- *void whenServerRequest(Message)*: mensagem recebida de um Mss que contém a requisição de um Mh. Sua função é processar a requisição e enviar o resultado para o

Mss que enviou a requisição.

6.3 Testes determinísticos

No modo de simulação determinístico, o simulador executa um *script*, definido pelo usuário, que descreve exatamente todos os eventos ocorridos na simulação. Os eventos podem ser envio ou recebimento de mensagens, movimentação dos Mhs ou mudança de estado (ativo, inativo) dos elementos do sistema. No modo estócástico a simulação é executada em rodadas. A cada rodada são executados concorrentemente todos os eventos determinados para a mesma. Os eventos de uma rodada só começam a ser executados depois que todos os eventos da rodada anterior tiverem sido executados.

A simulação determinística é utilizada para testar as situações críticas do protocolo. Desse modo, verifica-se o comportamento do protocolo nos cenários descritos nos *scripts*, averiguando, assim, a corretude do protocolo em uma situação bem definida. Apesar de não garantir a corretude do protocolo em todas as situações possíveis, esse modo de simulação é uma ferramenta muito útil para a avaliação de protocolos complexos, visto que, desse modo é possível verificar o seu comportamento nos cenários mais críticos. A escolha criteriosa dos cenários a serem simulados, em conjunto com testes bem sucedidos, são indícios de que o protocolo está funcionando adequadamente, tais indícios permitem iniciar os testes estocásticos que irão simular o protocolo em situações aleatórias.

O modo determinístico gera um *log* de execução contendo todos os eventos que ocorreram durante a simulação, o usuário pode desabilitar a exibição de tipos de mensagens para que sejam exibidos somente os eventos relevantes para a avaliação do protocolo. Após a simulação, deve-se comparar a sequência descrita no *log* com o comportamento esperado do protocolo verificando, assim, sua corretude.

Nesta seção serão apresentados dois cenários críticos simulados para o protocolo ARCP

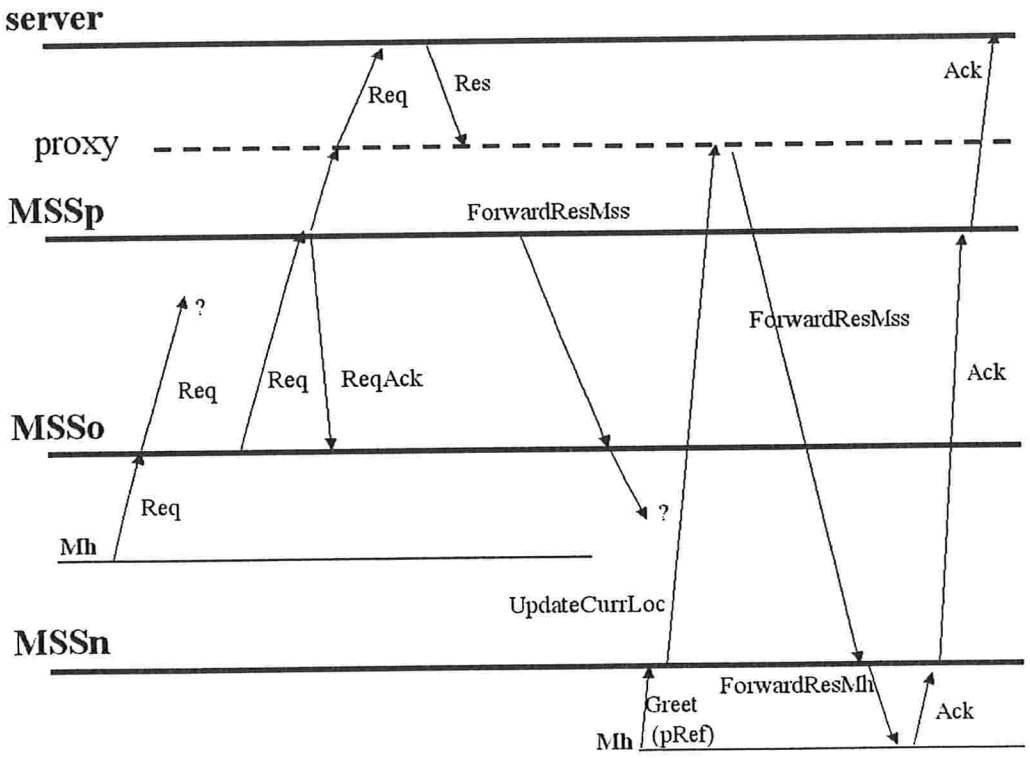

Figura 8: Cenário do ARCP, falha no Mss

através de *scripts* determinísticos. Graças ao modo determinístico, foi possível detectar erros de lógica no desenvolvimento do protocolo e aprimorar o seu funcionamento. Após a execução dos testes determinísticos, o protocolo ARCP pode ser testado nos cenários aleatórios da simulação estocástica.

A Figura 8 ilustra um cenário onde ocorre falha no Mss_{prxy} antes deste receber a requisição repassada pelo Mss_{resp}. Pode-se verificar que, mesmo com esta falha, o Mh irá, infalivelmente, receber o resultado da requisição, pois o Mss_{resp} reenviará a requisição até que receba a mensagem *RequestAck* confirmando o recebimento da requisição pelo Mss_{prxy}. A Figura ilustra apenas uma parte do cenário executado. Sendo assim, assume-se, nessa Figura, que o Mh já havia feito uma requisição anterior para o *proxy* ter sido criado.

```
...
acceptTurnOn(server, false);
...
next();
   mh.moveTo(cell_o);
next();
   mss_p.unavailable();
   mh.receive(new UserRequest(ARCPMh.class,server.getAddress(), 20));
   acceptTimeoutOn(mss_o, false);
next();
   mss_p.available();
next();
   acceptTimeout(mss_o, WaitingForRequestAck.class);
next();
   acceptTurnOn(server, true);
```

```
        mh.moveTo (cell_n);
        changeOrderInQueue(mss_p, UpdateCurrLoc.class);
    next();
```

No *script* acima, inicialmente é desligado o processamento de mensagens no servidor, evitando, assim, que a requisição enviada seja processada antes das migrações do Mh. O Mh migra para a célula do Mss_o, supõe-se que o Mh encontrava-se na célula do Mss_p onde já havia feito uma requisição para que o *proxy* tenha sido criado. No próximo passo, o Mss_p fica inativo, isto é, não está mais habilitado a receber mensagens. Em seguida, o Mh faz uma nova requisição. O Mss_o irá tentar redirecionar a requisição para o Mss_p, mas a mensagem não será recebida, visto que o Mss_p está inativo. O processamento de *timeout* é desligado temporariamente no Mss_o para que, dessa forma, seja processado depois que o Mss_p tornar a ficar ativo, só então o Mss_o processa o timeout e envia novamente a requisição para o Mss_p. No último passo, o servidor processa as requisições e o Mh migra para a célula do Mss_n. Garantimos que o Mss_p irá, assim que receber o resultado do servidor, enviar a mensagem $ForwardResMss$ para o Mss_o, pois esta é a última localização armazenada no *proxy*. Nesse momento, a mensagem referente ao último *hand-off* do Mh será a última mensagem processada pelo Mss_p. Somente após essa atualização o Mss_p enviará a mensagem $ForwardResMss$ para o Mss_n, que encaminhará o resultado para o Mh. Por fim, o Mh envia o *Ack* da requisição que é encaminhado para o Mss_p.

A Figura 9 ilustra um cenário em que o Mh migra antes de receber o resultado de uma requisição. Após receber o resultado de uma requisição, o Mss_p irá armazená-lo até receber um *Ack* do Mh que confirme o recebimento do resultado pelo Mh. Podemos observar na figura que quando o Mh migra, ele envia a mensagem de $Greet$ para o Mss, esta mensagem contém a referência para o *proxy* daquele Mh. Em seguida, o Mss envia a mensagem $UpdateCurrLoc$ para o Mss_p, ao receber a mensagem o *proxy* reenvia o resultado para a

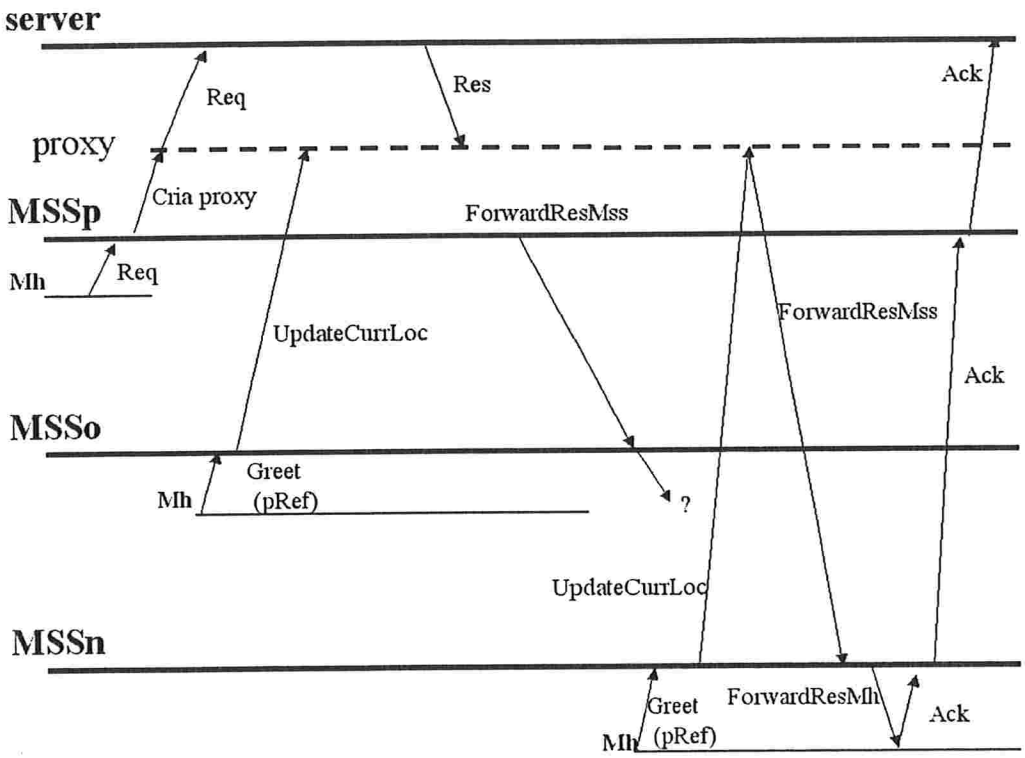

Figura 9: Cenário do ARCP, *Mh* migra antes de receber resultado

atual localização do Mh. Reenviado o resultado ao Mh, este envia, finalmente, um Ack confirmando o recebimento.

Abaixo está listado parte do *script* utilizado para executar o cenário ilustrado na Figura 9.

```
...
acceptTurnOn(server, false);
next();
   mh.moveTo (cell_p);
next();
   mh.receive(new UserRequest(ARCPMh.class,server.getAddress(), 10));
next();
   mh.moveTo(cell_o);
next();
   acceptTurnOn(server, true);
   mh.moveTo (cell_n);
   changeOrderInQueue(mss_p, UpdateCurrLoc.class);
next();
```

Podemos observar que neste *script*, inicialmente é desligado o processamento de mensagens no servidor, isso para que a requisição enviada não seja processada antes das migrações do Mh. Vemos, também, que o Mh a princípio encontra-se na célula do Mss_p, no passo seguinte o Mh inicia uma requisição. Antes da requisição ser processada, pois o servidor não está habilitado a receber mensagens, o Mh migra para a célula do Mss_o. O Mss_p não irá enviar uma mensagem $ForwardResMss$, visto que ainda não recebeu o resultado do servidor. No último passo, o servidor processa a requisição e o Mh migra para a célula do Mss_n. Garantimos que o Mss_p irá, assim que receber o resultado do servidor, enviar

a mensagem $ForwardResMss$ para o Mss_o, última localização armazenada no *proxy*. A mensagem referente ao último *hand-off* do Mh será a última mensagem processada pelo Mss_p. Somente após a atualização o Mss_p enviará a mensagem $ForwardResMss$ para o Mss_n que, por sua vez, encaminhará o resultado para o Mh que, por fim, envia o Ack da requisição que é encaminhada para o Mss_p.

7 Comparação dos Protocolos

7.1 Sumário

Este capítulo descreve, com mais detalhes, a comparação entre os protocolos apresentados anteriormente. Na Seção 7.2, são detalhados os parâmetros utilizados na simulação estocástica dos protocolos. Na Seção 7.3, são listados os dados medidos nas simulações. Na Seção 7.4, são apresentadas as restrições de cada protocolo, algumas delas detectadas através das simulações. A Seção 7.5 apresenta a sobrecarga dos protocolos. Na Seção 7.6, os resultados obtidos são apresentados e com base nesses dados é feita a análise dos protocolos. Por fim, na Seção 7.7 é apresentada a conclusão do capítulo.

É importante informar que através das simulações estocásticas foi possível detectar erros na implementação dos protocolos decorrentes de situações que não haviam sido testadas nas simulações determinísticas.

7.2 Parâmetros da simulação

Nesta seção são apresentados os parâmetros de simulação utilizados nas simulações estocásticas dos protocolos.

Configuração e Características da Rede

- Número de células(Nc): foi utilizada uma configuração com oito células. Para cada célula há um Mss responsável pela mesma.
- Número de Mhs: manteve-se sempre o mesmo número de Mhs ($N_{mhs} = 10$).
- Atraso associado a cada transmissão no meio sem fio: considerando o atraso na rede com fio de 1 unidade de tempo simulada (1 UT), usamos um atraso 100x maior no meio sem fio.

Mobilidade

- Probabilidade de migração(P_{mig}): todos os Mhs têm a mesma probabilidade de migração, ou seja, periodicamente o simulador determina se cada Mh irá migrar ou não segundo essa probabilidade. Os experimentos foram realizados para P_{mig}=0.00 (sem migração),0.2, 0.4, 0.6, 0.8 (alta taxa de migração);

- Fator de atração das células: optamos pela adoção de atrações diferentes para todas as células, ou seja, cada vez que ocorria uma migração, as células tinham diferentes probabilidades de serem a célula destino da migração. Nosso objetivo foi modelar uma cidade onde, tipicamente, as células têm diferentes fatores de atração, o que torna a simulação mais realística. Na simulação foram utilizadas oito células com os seguintes fatores de atração: 1 célula com fator 0.2, 2 células com fator 0.15 e 5 células com fator 0.1.

Disponibilidade e Requisição

- Probabilidade de desconexão/reconexão: todos os Mhs têm a mesma probabilidade $P_{desc} = 0.02$ de serem desconectados da rede fixa e a probabilidade $P_{recon} = 0.4$ de se (re)conectarem a rede;

- Probabilidade de requisição: cada Mh têm a probabilidade $P_{req} = 0.1$ de fazer uma requisição para seu Mss responsável.

Processamento de uma Requisição

- Tempo de Processamento de uma Requisição: o tempo para que uma requisição seja processada foi definido como sendo 99 UTs. Desse modo, o tempo necessário para que uma mensagem contendo o resultado de uma requisição seja enviada é de 100 UTs, pois o envio de uma mensagem na rede fixa tem o custo de 1 UT.

7.3 Dados Medidos

Nesta seção são descritos os dados medidos nas simulações estocásticas dos protocolos RDP, RCP, SRDP e ARCP. Para cada combinação distinta de parâmetros foram feitas dez rodadas de testes, em cada rodada foram realizadas mil requisições. As médias dos valores obtidos e seus respectivos desvios padrão foram, então, calculados.

Segue abaixo a lista dos dados medidos:

- Número de mensagens no meio sem fio: esse dado permite compararmos a carga gerada pelos protocolos na rede sem fio para diferentes taxas de migração.

- Número de mensagens na rede fixa: permite a comparação do custo total na rede fixa em todos os protocolos, para diferentes taxas de migração dos Mhs.

- Sobrecarga na rede fixa: todos os protocolos necessitam das mensagens *Greet*, *Request*, *ServerRequest*, *Result*, *ForwardResMh* e *Ack*. Tais mensagens representam o custo fixo para o processamento de uma requisição. Portanto, as demais mensagens de cada protocolo ou a retransmissão dessas mensagens fixas são o que constituem a sobrecarga dos protocolos. A medição desse dado permite verificar a influência da sobrecarga no número total de mensagens trocadas na rede fixa pelos protocolos.

- Duração média de uma requisição: o período é definido a partir da requisição do Mh até o recebimento do resultado pelo mesmo. O objetivo da medição é comparar o tempo simulado gasto por cada um dos protocolos, considerando diferentes taxas de migração dos Mhs. Assim, é possível avaliar qual dos protocolos (em determinados parâmetros de simulação) é o mais eficiente.

7.4 Restrições dos Protocolos

Citaremos a seguir as restrições de cada um dos protocolos. Sendo algumas descritas nas especificações dos mesmos, outras, detectadas neste trabalho através das simulações dos protocolos. A tabela 9 contém o resumo das restrições citadas.

RDP	RCP	SRDP	ARCP
Comunicação Confiável na Rede Fixa	Comunicação Confiável Rede Fixa	Comunicação Confiável $Mss \rightarrow$ Servidor	Comunicação Confiável $Mss \rightarrow$ Servidor
Handoff Atômico	Handoff Atômico		
Envio da mensagem Greet não pode falhar	Envio da mensagem Greet não pode falhar		
Não pode haver falha no envio de requisições para o Mss responsável	Não pode haver falha no envio de requisições para o Mss responsável		Não pode haver falha no envio de requisições para o Mss responsável
	Requisições só podem ser enviadas depois do handoff terminado		

Tabela 9: Restrições dos Protocolos

RDP

- Toda a comunicação na rede fixa e entre os $Msss$ é confiável, isto é, não há perdas e/ou corrupção de mensagens. $Msss$ e servidores nunca ficam inativos.
- O processo de *hand-off* deve ser terminado antes que outro comece, caso contrário, a lista de requisições passada para o novo Mss poderá estar desatualizada.
- O envio da mensagem *Greet* não pode falhar nunca, uma vez que a mesma inicia o processo de *hand-off*.
- Não pode haver falha no envio das requisições do Mh para o Mss_{resp}. Se as requisições não são recebidas pelo Mss_{resp} (o Mh fica inativo e não consegue enviar a requisição) elas não serão atendidas.

RCP

- Toda a comunicação na rede fixa e entre os $Msss$ é confiável, isto é, não há perdas e/ou corrupção de mensagens. $Msss$ e servidores nunca ficam inativos.
- O processo de *hand-off* deve ser terminado antes que outro comece, caso contrário, a lista de requisições passada para o novo Mss poderá estar desatualizada.
- O envio da mensagem *Greet* não pode falhar nunca, uma vez que a mesma inicia o processo de *hand-off*.
- Não pode haver falha no envio das requisições do Mh para o Mss_{resp}. Se as requisições não são recebidas pelo Mss_{resp} (o Mh fica inativo e não consegue enviar a requisição) elas não serão atendidas.
- Se uma requisição chega antes de todo o processo de *hand-off* ter sido executado ela não é atendida, já que o Mss_{resp} ainda não tem a referência para o *proxy* deste Mh (tal restrição não ocorre no RDP, sempre é criado localmente um novo *proxy* para cada requisição).

SRDP

- A comunicação entre os $Msss$ e servidores é confiável, isto é, não há perdas e/ou corrupção de mensagens. Servidores nunca ficam inativos, porém, os $Msss$ podem ficar inativos.
- O envio da mensagem *Greet* não pode falhar, pois através desta mensagem o *proxy* é enviado do Mh para o Mss_{resp}. No entanto, esta restrição não compromete a confiabilidade do protocolo, somente irá atrasar o recebimento das mensagens pelo Mh. Já que se o envio da mensagem falhar, o Mh irá enviá-la novamente quando efetuar uma migração ou mudar do estado inativo para ativo. Assim, as requisições feitas pelo

Mh nesse espaço de tempo, que não foram recebidas pelo Mss_{resp}, serão enviadas na próxima mensagem de *Greet*. Isso ocorre, pois, as informações referentes às requisições são armazenadas no Mh.

- Se uma requisição chega no Mss_{resp} antes da mensagem de *Greet* ela não é tratada imediatamente, pois o Mss_{resp} ainda não construiu o *proxy* do Mh. Porém, a requisição será reenviada quando o Mh migrar ou passar do estado inativo para ativo.

ARCP

- A comunicação entre os $Msss$ e servidores é confiável, isto é, não há perdas e/ou corrupção de mensagens. Os servidores nunca ficam inativos, porém, os $Msss$ podem ficar inativos. A falha nos $Msss$ deve ser temporária, pois se o Mss_{prxy} de algum Mh falhar, o Mh só irá receber os resultados de suas requisições caso este Mss volte para o estado ativo.

- Não pode haver falha no envio das requisições do Mh para o Mss_{resp}. Se as requisições não são recebidas pelo Mss_{resp} (Mh fica inativo e não consegue enviar a requisição) elas não serão atendidas.

7.5 Sobrecarga gerada pelos Protocolos

Nesta seção, são listadas as mensagens que compõem a sobrecarga de cada um dos protocolos. Todos os protocolos necessitam das mensagens *Greet*, *Request*, *ServerRequest*, *Result*, *ForwardResMh* e *Ack*. Tais mensagens representam o custo fixo para o processamento de uma requisição. Portanto, as demais mensagens de cada protocolo ou a retransmissão dessas mensagens fixas constituem a sobrecarga dos protocolos.

RDP

- *Dereg* enviada no processo de *hand-off* para o Mss antigo;
- *PList* enviada no processo de *hand-off* do Mss antigo para o Mss atual;
- *UpdateCurrLoc* enviada no processo de *hand-off* do Mss atual para o Mss_{prxy}, atualiza a localização do Mh no *proxy*;
- *RemPList* enviada para todos os Mss da rede assim que o *proxy* recebe o *Ack* confirmando o recebimento do resultado pelo Mh;
- $ForwardResMss$ enviada do Mss_{prxy} para a atual localização do Mh;
- $AckProxy$ enviada do Mss_{resp} para o Mss_{prxy};
- reenvio das mensagens $ForwardResMh$ e do *Ack*;
- reenvio da mensagem *Request* do Mss_{resp} para o Mss_{prxy}.

RCP

- *Dereg* enviada no processo de *hand-off* para o Mss antigo;
- *DeregAck* enviada no processo de *hand-off* do Mss antigo para o Mss atual;
- *UpdateCurrLoc* enviada no processo de *hand-off* do Mss atual para o Mss_{prxy}, atualiza a localização do Mh no *proxy*;
- $ForwardResMss$ enviada do Mss_{prxy} para a atual localização do Mh;
- $AckProxy$ enviada do Mss_{resp} para o Mss_{prxy};
- reenvio da mensagem *Request* do Mss_{resp} para o Mss_{prxy};

- reenvio das mensagens $ForwardResMh$ e Ack.

SRDP

- Reenvio da mensagem *Request* para o servidor toda vez que um Mh migra ou se torna ativo;
- reenvio da mensagem *Result*;
- reenvio da mensagem $ForwardResMh$.

ARCP

- $UpdateCurrLoc$ enviada no processo de *hand-off* do Mss atual para o Mss_{prxy}, atualiza a localização do Mh no *proxy*;
- $ForwardResMss$ enviada do Mss_{prxy} para a atual localização do Mh;
- $AckProxy$ enviada do Mss_{resp} para o Mss_{prxy};
- $RequestAck$ enviada do Mss_{prxy} para o Mss_{resp}, confirma o recebimento da requisição pelo *proxy*;
- reenvio da mensagem *Request* do Mss_{resp} para o Mss_{prxy};
- reenvio das mensagens $ForwardResMh$ e Ack.

7.6 Análise dos Resultados obtidos

Nesta seção, são apresentados e analisados os dados obtidos através da simulação estocástica dos protocolos. O dados são apresentados através de gráficos. Como dito na subseção anterior, para cada combinação distinta dos parâmetros, apresentados na Subseção 7.2, foram feitas dez simulações e, em cada simulação, foram realizadas mil requisições. Os

gráficos apresentados nesta seção utilizam os valores médios dos resultados obtidos e seus respectivos desvios padrão, os pontos nos gráficos indicam as médias e as barras verticais indicam os desvios padrão. Na Subseção 7.4.1 é apresentado o custo total na rede fixa de cada protocolo. Na Subseção 7.4.2 é detalhada a sobrecarga na rede fixa em cada um dos protocolos. Na Subseção 7.4.3 é apresentado o custo total no meio sem fio. Na Seção 7.4.4 é feita a comparação da duração média de uma requisição nos protocolos.

7.6.1 Custo Total na Rede Fixa

Considera-se custo total, toda e qualquer mensagem trocada entre os elementos da rede fixa ($Msss$ e Servidores). Portanto, além das mensagens contendo as requisições e seus resultados, também estão inclusas nesse custo as mensagens trocadas na rede fixa no processo de *hand-off*.

O custo mínimo para o processamento de uma requisição em todos os protocolos é constituído por duas mensagens:

- *ServerRequest*, mensagem que contém uma requisição. É enviada de um Mss para o servidor que irá processá-la
- *Result*, mensagem que contém o resultado de uma requisição. É enviada do servidor para um Mss.

Na Figura 10 é possível observar que a probabilidade de migração (P_{mig}) tem grande influência sobre o número de mensagens transmitidas na rede fixa nos protocolos RDP, RCP e ARCP. Isso se deve, principalmente, ao número de *hand-offs* que aumentam proporcionalmente ao valor de P_{mig}. Nesses três protocolos há troca de mensagens entre os $Msss$ durante o processo de *hand-off*. A tabela 10 contém os dados que foram utilizados para a construção do gráfico da Figura 10.

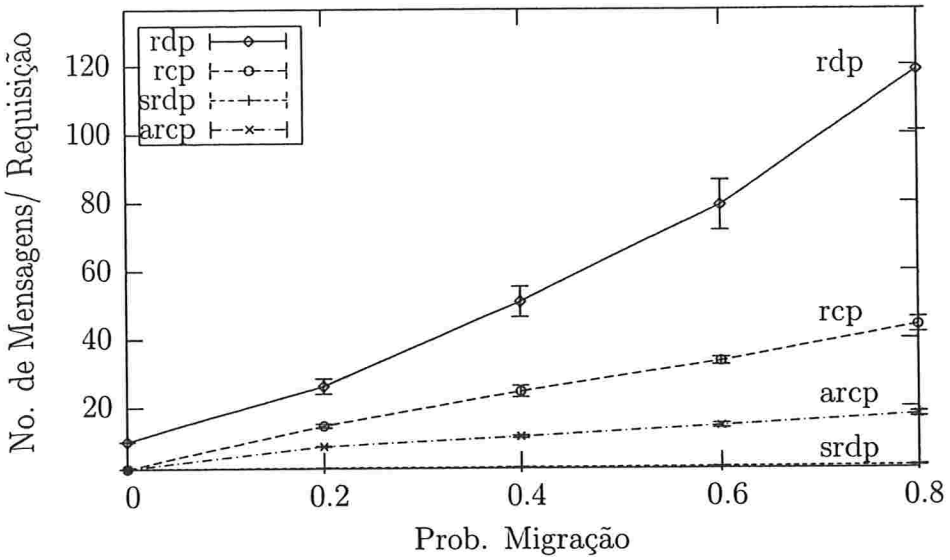

Figura 10: Custo Total na Rede Fixa

No protocolo SRDP a taxa de migração não tem grande influência sobre o número de mensagens na rede fixa, visto que, nesse protocolo, não há troca de mensagens entre os $Msss$ da rede durante o processo de *hand-off*. O número de mensagens também aumenta de acordo com o valor de P_{mig}, ainda que em uma escala bem menor que a dos outros protocolos. Outro aspecto é o aumento no número de mensagens na rede fixa que ocorre devido ao reenvio das requisições para o servidor toda vez que um Mh com mensagens pendentes migra e, também, devido ao reprocessamento e reenvio de resultados pelo servidor.

Comparado aos demais protocolos, o RDP possui o pior desempenho. As simulações comprovam que o fato de possuir apenas um *proxy* para cada Mh, o que ocorre nos protocolos RCP e ARCP, causa uma queda significativa no número de mensagens transmitidas na rede fixa. Com um agravante, no protocolo RDP o número de mensagens é maior, pois, é necessário enviar uma mensagem de atualização da localização do Mh para todos os $Msss$ que possuem pelo menos um *proxy* do Mh. Diferentemente, nos protocolos RCP e ARCP é necessário enviar somente uma mensagem de atualização a cada migração, uma vez que o Mh terá apenas um *proxy* a cada instante. O número de mensagens por requisição na rede

Protocolo	Prob. de Migração	Média	Desvio Padrão
RDP	0.0	9.987690736	0.020391915
	0.2	26.13453495	2.226879642
	0.4	50.87176928	4.452352272
	0.6	79.15084525	7.324353258
	0.8	118.5671895	17.75897837
RCP	0.0	2	0.00752028
	0.2	14.50949399	0.587570885
	0.4	24.54552724	1.685662826
	0.6	33.4018009	1.08939725
	0.8	43.95061728	2.216702386
SRDP	0.0	2.000198236	0.013314462
	0.2	2.183715177	0.01617641
	0.4	2.320628058	0.038540181
	0.6	2.470304783	0.049036798
	0.8	2.723236936	0.133219374
ARCP	0.0	2	0.012029052
	0.2	8.474061535	0.110361111
	0.4	11.28228138	0.342614012
	0.6	14.49119295	0.692690812
	0.8	17.68509615	0.752645189

Tabela 10: Custo Total na Rede Fixa

fixa do protocolo RDP é superior ao protocolo RCP em torno de 80% a 170%. Em relação ao ARCP, o número de mensagens no RDP é maior em torno de 209% a 570%. Isso se desconsiderarmos quando $P_{mig} = 0.0$, pois, nesse caso, a média de mensagens por requisição do protocolos RCP e ARCP é dois, já no protocolo RDP a média é 10, devido as mensagens *RemPList* que devem ser enviadas para todos os $Msss$ da rede, nesse caso $N_{mss} = 8$.

Como já foi dito, o protocolo ARCP possui um desempenho melhor do que o RCP. Primeiro, porque a cada migração somente uma mensagem é enviada na rede fixa, a mensagem de atualização de localização para o *proxy*. Isso é devido ao fato de que no protocolo ARCP a referência para o *proxy* do Mh é armazenada no próprio Mh, desse modo, não são necessárias as mensagens *Dereg* e *DeregAck*, enviadas no protocolo RCP para que o Mss atual receba a referência para o *proxy* do Mh armazenada no Mss que era responsável por ele antes do

hand-off. Sendo assim, o número de mensagens enviadas no protocolo RCP é superior ao ARCP em torno de 71% a 148%. Contudo, em relação ao protocolo SRDP, o número de mensagens enviadas no protocolo ARCP é superior em torno de 289% a 550%.

7.6.2 Distribuição da Sobrecarga na Rede Fixa

O que constitui a sobrecarga na rede fixa é o número de mensagens enviadas entre os elementos da rede fixa ($Msss$ e Servidores). Não contribuem para tal sobrecarga, no entanto, o envio das mensagens *ServerRequest* e *Result*. Já o reenvio das mesmas é considerado parte da sobrecarga.

7.6.3 Sobrecarga na rede fixa do protocolo RDP

Como visto na Seção 7.5, a sobrecarga do RDP na rede fixa é constituída pelas mensagens *Dereg*, *PList*, *UpdateCurrLoc*, *RemPList*, *ForwardResMss* e *AckProxy*.

Na Figura 11 fica evidente que a mensagem *UpdateCurrLoc* é a maior responsável pela sobrecarga no protocolo RDP. Mensagens desse tipo representam em torno de 31% a 66% do número total de mensagens enviadas na rede fixa pelo protocolo, exceto quando $P_{mig} = 0$, cuja porcentagem é de 0%. Isso devido ao fato descrito na subseção anterior, ou seja, no protocolo RDP é necessário, a cada migração, o envio de uma mensagem de atualização para cada Mss que possui pelo menos um *proxy* do Mh. A tabela 11 contém os dados que foram utilizados para a construção do gráfico da Figura 11.

As mensagens *Dereg* e *PList*, enviadas no processo de *hand-off* para que o Mss atual receba a lista de *proxies* do Mh do Mss anterior, representam, somadas, em torno de 27% a 23% do número total de mensagens enviadas na rede fixa, mais uma vez, exceto quando $P_{mig} = 0$, cuja porcentagem é de 0%. Nota-se que a porcentagem de mensagens destes tipos diminui conforme a probabilidade de migração dos Mhs aumenta, uma vez que constituem a segunda maior sobrecarga na rede fixa do protocolo RDP.

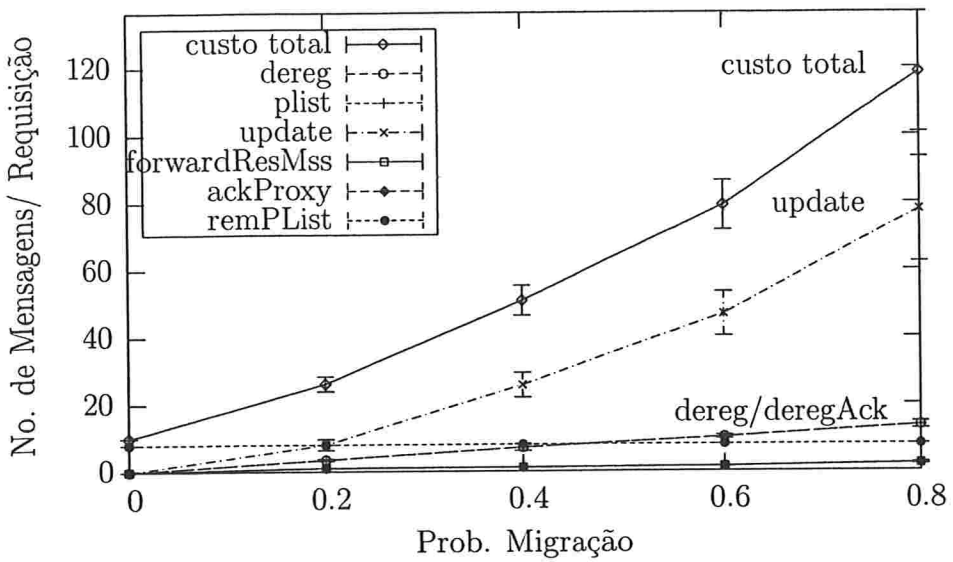

Figura 11: Sobrecarga na rede fixa do protocolo RDP

Na Figura 11 também é possível notar que a probabilidade de migração do Mh não tem influência sobre as mensagens do tipo *RemPList*. O número de mensagens deste tipo enviadas a cada requisição é constante e igual ao número de $Msss$ da rede, neste caso $N_{Mss} = 8$. O que faz do protocolo RDP não escalável em relação ao número de $Msss$.

As mensagens do tipo $ForwardResMss$ e $AckProxy$ representam somadas, respectivamente, em torno de 8% a 4% do número total de mensagens enviadas na rede fixa, exceto quando $P_{mig} = 0$, pois, a porcentagem é de 0%. A incidência de mensagens desses tipos é menor, pois elas são enviadas somente quando o *proxy* do Mh, no momento do envio do resultado ou do Ack, não está localizado no Mss responsável pelo Mh naquele momento.

Mensagem	Prob. de Migração	Média	Desvio Padrão
UpdateCurrLoc	0.0	0.0	0.0
	0.2	7.996215893	1.659830228
	0.4	25.7046203	3.688334979
	0.6	46.8193458	6.575762927
	0.8	77.82315916	15.53406029
Dereg / PList	0.0	0.0	0.0
	0.2	3.475204143	0.301090912
	0.4	7.101686458	1.018703968
	0.6	10.10453136	0.514183059
	0.8	13.52702568	1.075702951
RemPList	0.0, 0.2, 0.4, 0.6 e 0.8	8.0	0.0
ForwardResMss	0.0	0.0	0.0
	0.2	1.057558255	0.008305116
	0.4	1.163955693	0.043551538
	0.6	1.379713914	0.064560569
	0.8	2.142172045	0.262577281
AckProxy	0.0	0.0	0.0
	0.2	1.057558255	0.008305116
	0.4	1.163955693	0.043551538
	0.6	1.379713914	0.064560569
	0.8	2.142172045	0.262577281

Tabela 11: Sobrecarga na rede fixa do protocolo RDP

7.6.4 Sobrecarga na rede fixa do protocolo RCP

Como visto na Seção 7.5, a sobrecarga do RCP na rede fixa é constituída pelas mensagens *Dereg*, *DeregAck*, *UpdateCurrLoc*, *ForwardResMss*, *AckProxy*, além do reenvio da mensagem *Request* para a localização do *proxy*.

Na Figura 12 podemos observar que as mensagens *Dereg* e *DeregAck*, enviadas no processo de *hand-off* para que o Mss atual receba a referência para o *proxy* do Mh do Mss anterior, correspondem a maior parte da sobrecarga deste protocolo. Elas representam somadas em torno de 48% a 62% do número total de mensagens enviadas na rede fixa (exceto quando $P_{mig} = 0$ cuja porcentagem é de 0%). A tabela 12 contém os dados que foram utilizados para a construção do gráfico da Figura 12.

Mensagem	Prob. de Migração	Média	Desvio Padrão
Dereg / DeregAck	0.0	0.0	0.0
	0.2	3.456407198	0.220299371
	0.4	6.983108446	0.573565734
	0.6	10.02871436	0.369313677
	0.8	13.63545117	0.761681112
UpdateCurrLoc	0.0	0.0	0.0
	0.2	2.979520827	0.178640484
	0.4	5.935132434	0.524883771
	0.6	8.582191096	0.345933144
	0.8	11.6539195	0.706733243
ForwardResMss	0.0	0.0	0.0
	0.2	0.904364251	0.026774251
	0.4	0.949925037	0.026356838
	0.6	1.060930465	0.031755093
	0.8	1.314664258	0.037709608
AckProxy	0.0	0.0	0.0
	0.2	0.855254001	0.022288972
	0.4	0.850174913	0.018413002
	0.6	0.854127064	0.011800909
	0.8	0.859981933	0.010931831
Request	0.0	0.0	0.0
	0.2	0.857540511	0.021861922
	0.4	0.846376812	0.019114598
	0.6	0.851425713	0.010609287
	0.8	0.856368564	0.014589762

Tabela 12: Sobrecarga na rede fixa do protocolo RCP

As mensagens do tipo *UpdateCurrLoc* correspondem em torno de 21% a 27% do número total de mensagens enviadas na rede fixa (exceto quando $P_{mig} = 0$ cuja porcentagem é de 0%). Esta mensagem é enviada no processo de *hand-off* e atualiza a localização do Mh no *proxy*. A incidência desse tipo de mensagem é um pouco menor do que as mensagens do tipo *Dereg* e *DeregAck*, pois, aquela primeira, não é enviada em todos os processos de *hand-off*, mas somente naqueles em que o Mh possui um *proxy*, isto é, $pRef$ não é *null*.

As mensagens do tipo $ForwardResMss$, $AckProxy$ e o reenvio de *Request* representam, respectivamente, em torno de 6% a 3%, 6% a 2% e 6% a 2% do número total de mensagens

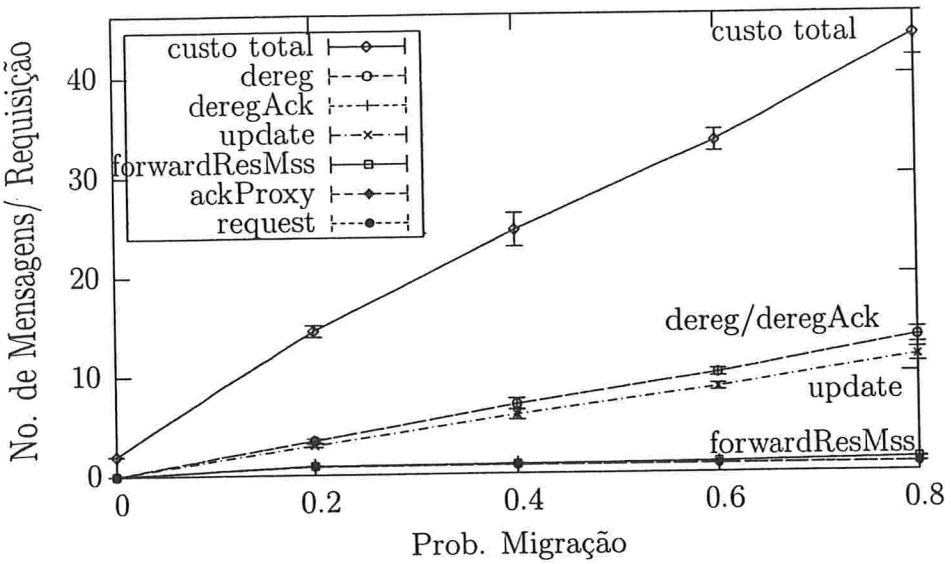

Figura 12: Sobrecarga na rede fixa do protocolo RCP

enviadas na rede fixa (exceto quando $P_{mig} = 0$ cuja porcentagem é de 0%). A incidência desses tipos de mensagens é menor, pois, são enviadas somente quando o *proxy* do Mh, no momento do envio do resultado ou do *Ack*, não está localizado no Mss atualmente responsável pelo Mh.

Mensagem	Prob. de Migração	Média	Desvio Padrão
ServerRequest	0.0	0.00010	0.006657231
	0.2	0.091857588	0.008088205
	0.4	0.160314029	0.019270091
	0.6	0.235152391	0.024518399
	0.8	0.361618468	0.066609687
Result	0.0	0.00010	0.006657231
	0.2	0.09186	0.008088205
	0.4	0.16031	0.019270091
	0.6	0.23515	0.024518399
	0.8	0.36162	0.066609687

Tabela 13: Sobrecarga na rede fixa do protocolo SRDP

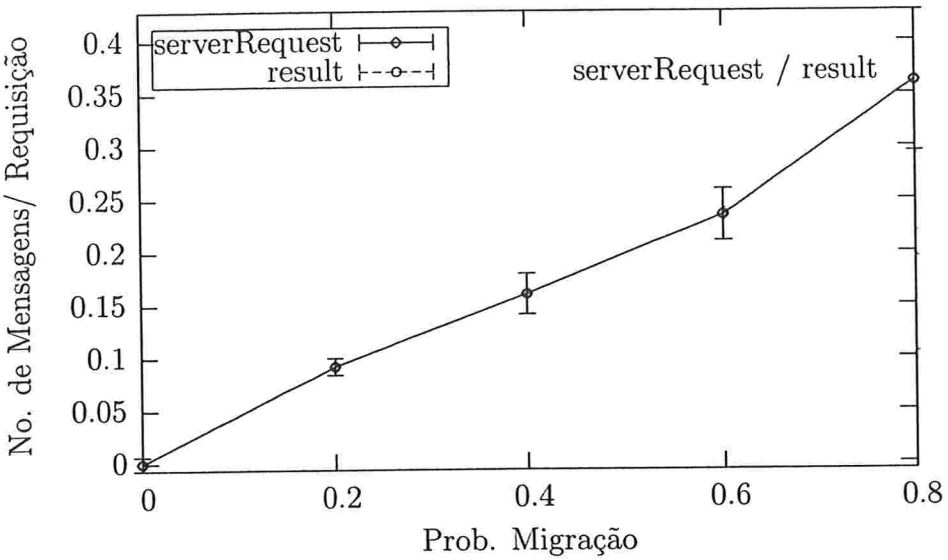

Figura 13: Sobrecarga na rede fixa do protocolo SRDP

7.6.5 Sobrecarga na rede fixa do protocolo SRDP

Como visto na Seção 7.5, a sobrecarga do SRDP na rede fixa é constituída pelo reenvio das mensagens *Request* e *Result*.

A mensagem *Request* é reenviada ao servidor toda vez que um Mh com requisições pendentes migra ou passa do estado inativo para ativo. A mensagem *Result* é reenviada quando o resultado de uma requisição não foi recebido pelo Mh, por exemplo, o Mh migrou ou ficou inativo quando o Mss repassou o resultado. Nesse caso, a requisição será reenviada para o servidor que terá que reprocessá-la e reenviá-la novamente.

A Figura 13 mostra que a probabilidade de migração do Mh influencia o número de reenvios realizados. Por exemplo, quando $P_{mig} = 0.2$ o número de requisições reenviadas e resultados reprocessados é em torno de 9%, para $P_{mig} = 0.4$ este número é 16%, para $P_{mig} = 0.6$ é 24% e, finalmente, para $P_{mig} = 0.8$ este número é 36%. A tabela 13 contém os dados que foram utilizados para a construção do gráfico da Figura 13.

Ao contrário dos outros protocolos, toda a sobrecarga na rede fixa do protocolo SRDP

envolve servidores, nos outros três protocolos a sobrecarga envolve apenas $Msss$.

7.6.6 Sobrecarga na rede fixa do protocolo ARCP

Como visto na Seção 7.5, a sobrecarga do ARCP na rede fixa é constituída pelas mensagens $UpdateCurrLoc$, $ForwardResMss$, $AckProxy$ e, ainda, o reenvio da mensagem *Request* para a localização do *proxy* e o envio da mensagem *RequestAck* que confirma o recebimento da mensagem *Request* pelo Mss_{prxy}.

Na Figura 14, nota-se que as mensagens $UpdateCurrLoc$ são as maiores responsáveis pela sobrecarga gerada pelo protocolo. Tais mensagens correspondem em torno de 35% a 67% do número total de mensagens enviadas na rede fixa (exceto quando $P_{mig} = 0$ cuja porcentagem é de 0%). A mensagem $UpdateCurrLoc$ é enviada a cada migração para atualizar a localização do Mh no *proxy*. A tabela 14 contém os dados que foram utilizados para a construção do gráfico da Figura 14.

Mensagem	Prob. de Migração	Média	Desvio Padrão
UpdateCurrLoc	0.0	0.0	0.0
	0.2	2.983869362	0.117465751
	0.4	5.757004686	0.33683806
	0.6	8.862790232	0.669499151
	0.8	11.82381811	0.775333041
ForwardResMss	0.0	0.0	0.0
	0.2	0.910086628	0.022569617
	0.4	0.964403231	0.019304916
	0.6	1.054643715	0.028089059
	0.8	1.297175481	0.054502173
AckProxy	0.0	0.0	0.0
	0.2	0.860101563	0.02242664
	0.4	0.853624489	0.012061394
	0.6	0.85918735	0.012520012
	0.8	0.857371795	0.013708911
Request	0.0	0.0	0.0
	0.2	0.860001991	0.020928781
	0.4	0.853624489	0.0136024
	0.6	0.857285829	0.01220448
	0.8	0.853365385	0.010536681
RequestAck	0.0	0.0	0.0
	0.2	0.860001991	0.020928781
	0.4	0.853624489	0.0136024
	0.6	0.857285829	0.01220448
	0.8	0.853365385	0.010536681

Tabela 14: Sobrecarga na rede fixa do protocolo ARCP

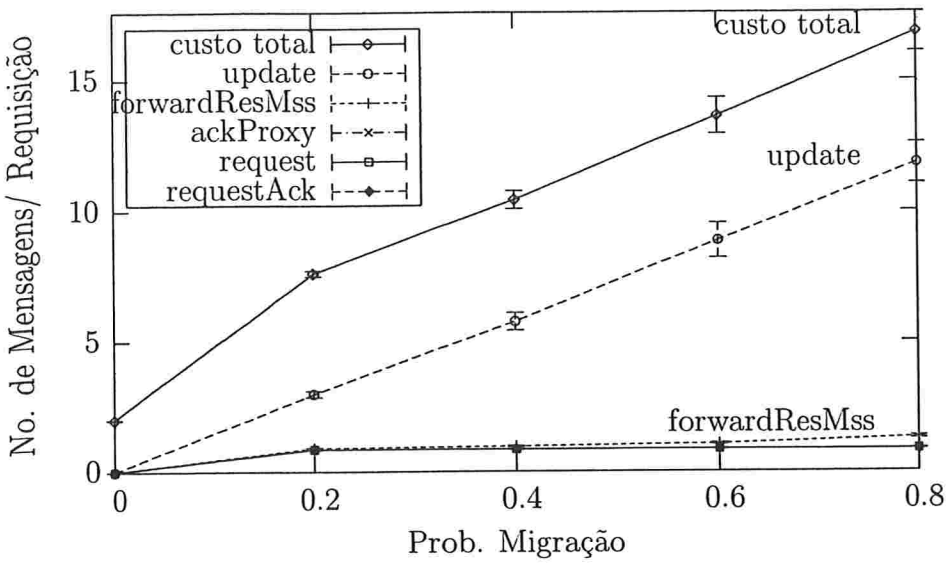

Figura 14: Sobrecarga na rede fixa do protocolo ARCP

As mensagens *ForwardResMss*, *AckProxy*, reenvio de *Request* e o envio da mensagem *RequestAck* representam, respectivamente, em torno de 11% a 7%, 10% a 5%, 10% a 5% e 10% a 5% (exceto quando $P_{mig} = 0$ cuja porcentagem é de 0%). A porcentagem desses tipos de mensagem diminui a medida que o valor de P_{mig} aumenta.

7.6.7 Custo Total no ambiente sem fio

No ambiente sem fio, o custo total é toda e qualquer mensagem trocada entre os elementos do sistema através do meio sem fio. Desse modo, além das mensagens contendo as requisições e seus resultados, também estão inclusos no custo, os *Acks* enviados pelo Mh para confirmar o recebimento de um resultado.

O custo mínimo no ambiente sem fio para o processamento de uma requisição em todos os protocolos constitui-se de três mensagens:

- *Request*, mensagem que contém uma requisição. É enviada de um Mh para o Mss atualmente responsável por ele.

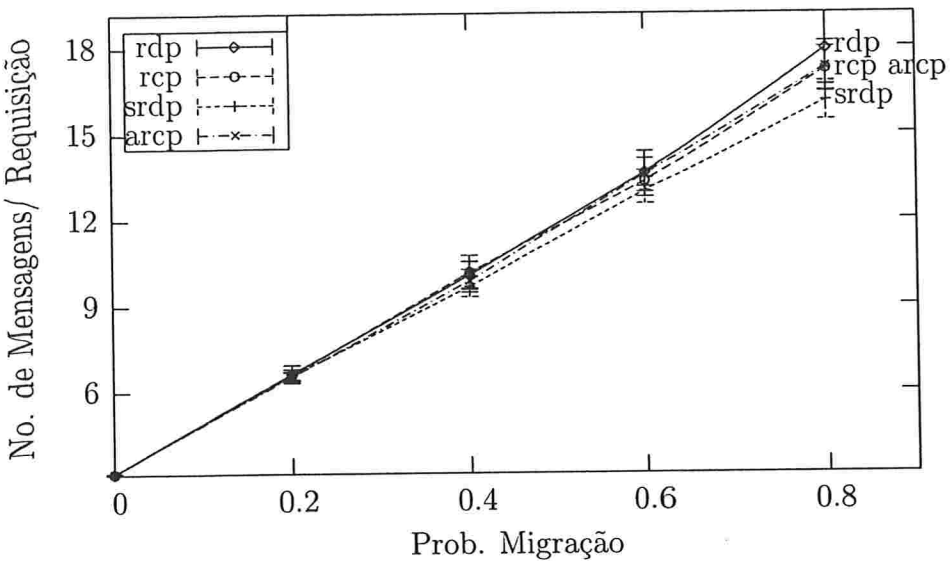

Figura 15: Custo Total no meio sem fio

- *ForwardResMh*, mensagem que contém o resultado de uma requisição. É enviada de um *Mss* para um *Mh*.

- *Ack*, confirma o recebimento da mensagem *Result* pelo *Mh*.

Na Figura 15, pode-se observar que a probabilidade de migração (P_{mig}) tem grande influência sobre o número de mensagens transmitidas na rede fixa em todos os protocolos. Isso ocorre, pois, o número de mensagens *Greet* enviadas aumenta proporcionalmente ao valor de P_{mig}. O aumento de P_{mig} também aumenta a probabilidade de um *Mh* migrar assim que o *Mss* recebe o resultado de uma requisição e tenta fazer o reenvio para o *Mh*. Nesse caso, a mensagem *ForwardResMh* terá que ser enviada novamente. A tabela 15 contém os dados que foram utilizados para a construção do gráfico da Figura 15.

Na figura fica evidente que o custo no meio sem fio de todos os protocolos é equivalente.

Protocolo	Prob. de Migração	Média	Desvio Padrão
RDP	0.0	3.153144217	0.017804552
	0.2	6.627165903	0.308922519
	0.4	10.06226923	0.467842114
	0.6	13.61808543	0.487567147
	0.8	17.91557598	1.258851782
RCP	0.0	3.146848989	0.013680738
	0.2	6.589422408	0.200780436
	0.4	10.15722139	0.588506496
	0.6	13.32046023	0.373571842
	0.8	17.21208471	0.762797819
SRDP	0.0	3.148676777	0.025399668
	0.2	6.569538561	0.14411383
	0.4	9.660939811	0.355051585
	0.6	12.99096494	0.449975776
	0.8	16.11085743	0.666678111
ARCP	0.0	3.159486618	0.014250574
	0.2	6.529722195	0.146703451
	0.4	9.860803669	0.397079906
	0.6	13.56755404	0.793969917
	0.8	17.29497196	0.880378717

Tabela 15: Custo Total no meio sem fio

7.6.8 Duração média de uma requisição

A duração de uma requisição é definida como o período entre o momento em que o Mh faz uma requisição e o instante em que o Mh recebe o resultado. O objetivo dessa medição é comparar o tempo simulado gasto por cada um dos protocolos, considerando diferentes taxas de migração dos Mhs.

A duração mínima de uma requisição é de 301 UTs (Unidade de Tempo), pois as seguintes mensagens com seus respectivos tempos devem ser enviadas:

- *Request*, mensagem enviada no meio sem fio de um Mh para o Mss responsável por este. Toda mensagem no meio sem fio demora 100 UTs para chegar ao seu destino. (O atraso no meio sem fio foi configurado como sendo 100x maior do que na rede fixa).

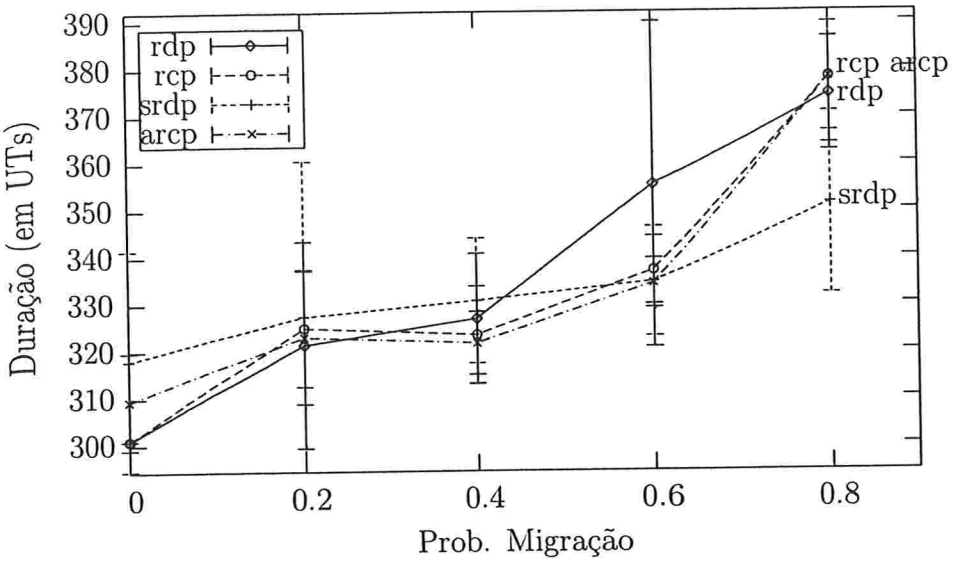

Figura 16: Duração média de uma requisição

- *ServerRequest*, mensagem enviada na rede fixa de um Mss para o servidor, toda mensagem na rede fixa demora 1 UT para chegar ao seu destino.

- *Result*, mensagem enviada na rede fixa do servidor para um Mss. Esta mensagem demora 100 UTs para chegar ao seu destino, este atraso foi configurado como o tempo necessário para processar uma requisição mais o tempo de envio na rede fixa.

- $ForwardResMh$, mensagem enviada no meio sem fio de um Mss para um Mh. Toda mensagem no meio sem fio demora 100 UTs para chegar ao seu destino.

Na Figura 16, percebe-se que as médias obtidas com as simulações não são muito significativas devido aos desvios padrão muito altos. Porém, através desse gráfico nota-se que o tempo simulado médio de uma requisição é similar em todos os protocolos. A tabela 16 contém os dados que foram utilizados para a construção do gráfico da Figura 16.

O desvio padrão é alto para todos os protocolos. Na média, uma requisição demora entre 301 UTs e 378 UTs, porém, existem casos em que um resultado deve ser reenviado para o Mh muitas vezes, pois, com o aumento de P_{mig} também aumenta a probabilidade de um Mh

migrar no momento em que o resultado está sendo enviado a ele. No protocolo SRDP, caso o Mh não receba o resultado, a requisição terá que ser reprocessada, aumentando, desse modo, o tempo para que a requisição seja recebida. Nos demais protocolos, o Mss, que recebeu a mensagem de *Greet* do Mh, irá enviar uma mensagem de $UpdateCurrLoc$ para o Mss que contém o *proxy* do Mh, então, o Mss_{prxy} irá reenviar o resultado para a atual localização do Mh; este processo pode ser repetido várias vezes dependendo das migrações do Mh.

Protocolo	Prob. de Migração	Média	Desvio Padrão
RDP	0.0	301	0
	0.2	321.367	22.05847668
	0.4	326.8007	13.900976
	0.6	355.2384	34.58832939
	0.8	374.4017	12.01534364
RCP	0.0	301	0
	0.2	324.9438	12.50854556
	0.4	323.357	10.36980827
	0.6	337.0124	7.204619
	0.8	377.9953	11.53081353
SRDP	0.0	318.0511	23.52037649
	0.2	327.3766	33.13467101
	0.4	330.6297	13.32702064
	0.6	334.6062	11.64011273
	0.8	351.2349	19.35213314
ARCP	0.0	309.443	10.3810897
	0.2	322.9741	14.2092099
	0.4	321.5809	6.716364153
	0.6	334.2602	5.243329345
	0.8	377.9385	14.06462428

Tabela 16: Duração média de uma requisição

7.7 Conclusão

Através dos dados obtidos com as simulações, observa-se que o protocolo SRDP apresenta a menor sobrecarga na rede fixa. Porém, ao contrário dos outros protocolos toda a sua sobrecarga envolve servidores, o que é uma desvantagem, uma vez que, geralmente, o número

de servidores em uma rede é muito menor do que o número de $Msss$. Nos outros três protocolos a sobrecarga envolve apenas $Msss$.

Nos protocolos RCP e ARCP, apesar de a cada momento somente um Mss ser responsável em retransmitir o resultado de uma requisição para um dado Mh, o protocolo provê balanceamento de carga global dinâmico entre o conjunto de $Msss$ devido às migrações do Mh.

As simulações comprovam que o fato de se possuir apenas um *proxy* para cada Mh, como nos protocolos RCP e ARCP, causa uma queda significativa no número de mensagens transmitidas na rede fixa se comparado ao protocolo RDP, em que um *proxy* é criado para cada requisição. Outra melhoria observada no protocolo ARCP, deve-se a redução do número de mensagens na rede fixa devido ao fato de a referência para o *proxy* ser armazenada no próprio Mh e não no Mss responsável por ele, como é feito no protocolo RCP. Assim, as mensagens *Dereg* e *DeregAck* podem ser descartadas.

Com relação ao custo no meio sem fio e a duração média de uma requisição, todos os protocolos são equivalentes. Sendo a duração de uma requisição definida como o período entre o momento em que o Mh faz uma requisição e o instante em que o Mh recebe o resultado.

8 Conclusão

8.1 Sumário

Finalmente, neste capítulo, é apresentada a conclusão desta dissertação, bem como as contribuições da mesma. Por fim, serão apresentados possíveis trabalhos futuros.

A Seção 8.2 apresenta a conclusão e contribuições deste trabalho. Na Seção 8.3, são listados os possíveis trabalhos futuros.

8.2 Conclusão e Contribuições

Esta dissertação apresentou alguns protocolos que provêem comunicação entre agentes móveis. Como o protocolo RDP (*Result Delivery Protocol*) que provê entrega confiável de mensagens entre servidores e clientes móveis, neste protocolo um *proxy* é criado para cada requisição feita pela unidade móvel. O protocolo RCP (RCP - *A Reliable Connectionless Protocol for Mobile Clients*), versão melhorada do protocolo RDP, e no qual uma unidade móvel possui apenas um *proxy* a cada instante. O protocolo SRDP (*Safe Result Delivery Protocol*) é uma modificação do protocolo RCP que busca manter as mesmas garantias na entrega de mensagens sem que a confiabilidade dos $Msss$ seja crítica para o funcionamento do protocolo. E, finalmente, o protocolo ARCP (*Ack* RCP), desenvolvido neste trabalho, e que, assim como o protocolo SRDP garante a entrega confiável de mensagens mesmo que haja falha na comunicação entre os $Msss$. No entanto, é importante observar que no ARCP a quantidade de informações armazenadas na unidade móvel é diminuída, também é reduzido o número de requisições feitas para o servidor com intuito de evitar a sobrecarga do mesmo.

Inicialmente todos os processos de cada protocolo foram detalhados e comparados tendo como base as especificações de cada um. Também foram listados todos os tipos de mensagens enviadas e dados mantidos por cada protocolo.

Nesse sentido, para realizar a comparação dos protocolos foi utilizado o simulador MobiCS, que precisou de melhorias a fim de se obter métricas para as comparações. O MobiCS original não emitia nenhum relatório com dados desse tipo, sendo assim, a saída do simulador se resumia a um *log* de execução dos eventos que foram executados durante a simulação, por isso, a necessidade de alterar o simulador. Tais alterações foram devidamente listadas.

Esta dissertação também descreve a implementação do protocolo desenvolvido neste trabalho, o ARCP. Além do processo de implementação, são apresentadas as situações críticas escolhidas para os testes do protocolo.

Foram realizadas simulações estocásticas para todos os protocolos com o intuito de verificar quais as vantagens e desvantagens de cada protocolo em determinados cenários de execução. Através da análise e simulação dos protocolos percebe-se que o protocolo SRDP é o que menos apresenta restrições que interfiram na sua confiabilidade, sendo a única deste tipo, a de que a comunicação entre $Msss$ e servidores deve ser confiável.

O protocolo SRDP é o protocolo que armazena a maior quantidade de informações no Mh. Neste protocolo é necessário armazenar os dados das informações pendentes para que elas possam ser reconstruídas posteriormente. Em contrapartida, no protocolo ARCP são armazenados somente os identificadores das requisições pendentes e a referência para o *proxy* do Mh. Exceto a referência para o *proxy*, os demais dados armazenados por esses protocolos não têm tamanho constante, eles podem crescer indefinidamente dependendo da quantidade de requisições feitas pelo Mh e do recebimento dos resultados. Os protocolos RDP e RCP não armazenam nenhum tipo de informação no Mh.

Outra desvantagem dos protocolos SRDP e ARCP é a necessidade de um algoritmo de coleta de lixo para eliminar os *proxies* sem uso.

Portanto, através dos resultados obtidos nas simulações nota-se que - exceto o protocolo RDP no qual não foi encontrada nenhuma vantagem em relação ao protocolo RCP - os outros três protocolos possuem tanto vantagens como desvantagens. Sendo assim, a escolha

do protocolo será determinada de acordo com o cenário de execução.

O protocolo RCP possui o pior desempenho no que diz respeito ao número de mensagens enviadas na rede fixa, porém neste protocolo não é necessário um processo de coleta de lixo para remover os *proxies* sem uso, assim como não é armazenada nenhuma informação no Mh. No protocolo SRDP, apesar de ter o melhor desempenho com relação ao número de mensagens enviadas na rede fixa, há sobrecarga no servidor, além disso, entre os três protocolos este é o que armazena a maior quantidade de informações no Mh. No protocolo ARCP a sobrecarga na rede fixa é maior do que a do SRDP, porém, menor do que a do RCP. O ARCP armazena mais informações no Mh do que o protocolo RCP, mas se comparado ao SRDP, a quantidade de informação armazenada no Mh é menor.

Portanto o melhor cenário para a execução do protocolo ARCP, desenvolvido neste trabalho, é aquele em que haja a possibilidade de ocorrerem falhas temporárias na rede fixa, contexto que descarta o uso dos protocolos RDP e RCP. Além disso, neste cenário as requisições devem ocupar um espaço considerável na memória. Neste caso o ARCP é mais vantajoso do que o SRDP, pois armazena menos informações relativas às requisições no Mh. Por fim, o processamento de uma requisição pode ser uma operação custosa ou pode ser grande o número de requisições feitas pelos Mhs, neste contexto o ARCP é melhor do que o SRDP pois ele garante que cada requisição será processada e enviada somente uma vez para o servidor.

Dessa forma, as principais contribuições deste trabalho foram:

- A criação e implementação de um novo protocolo de entrega de mensagens para agentes móveis.
- A simulação e comparação de quatro protocolos de entrega de mensagens para agentes móveis, incluindo o protocolo ARCP desenvolvido neste trabalho. Através dos resultados obtidos, foi possível fazer uma análise detalhada dos protocolos, verificando assim

as vantagens e desvantagens de cada um. Além disso, as melhorias feitas nos protocolos puderam ser comprovadas.

- O desenvolvimento de novas funcionalidades no simulador MobiCS, fazendo com que este emita, ao final da simulação, um relatório contendo métricas necessárias para comparação entre protocolos.

8.3 Trabalhos Futuros

Os protocolos de comunicação têm grande responsabilidade em prover suporte eficiente à mobilidade em ambientes de computação móvel. Sendo assim, um possível trabalho futuro seria o desenvolvimento de novos protocolos para comunicação entre agentes móveis. Através da análise apresentada nesta dissertação, tais protocolos podem ser aprimorados.

O simulador MobiCS não foi construído com o intuito de ser uma ferramenta completa de simulação, podendo ser visto, assim, como um núcleo de simulação onde novas funcionalidades e características podem ser adicionadas. Neste trabalho, como já dissemos, foram adicionadas algumas funcionalidades ao simulador para que ele emita ao final da simulação um relatório contendo métricas necessárias para comparação entre protocolos. Dessa forma, o desenvolvimento de novas funcionalidades para o simulador seria uma possibilidade de trabalhos futuros.

Segue abaixo a lista de alguns possíveis trabalhos futuros que podem dar sequência ao estudo realizado nesta dissertação:

- A criação de novos protocolos para comunicação entre agentes móveis. O desenvolvimento de tais protocolos pode levar em consideração a análise feita neste trabalho que mostra as vantagens e desvantagens de alguns protocolos.

- O simulador MobiCS não apresenta um bom desempenho se comparado aos outros simuladores citados neste trabalho. Portanto, um possível trabalho futuro seria melhorar

o desempenho deste simulador. Essa melhoria poderia ser feita através de algoritmos paralelos ou através da distribuição do simulador em várias máquinas.

- Com relação à usabilidade do simulador MobiCS, poderia ser implementada uma interface gráfica que permita a visualização dinâmica dos elementos do sistema na simulação.

- Implementar emulação de rede no MobiCS, permitindo a troca de dados entre o simulador e uma rede real.

9 Bibliografia

Referências

[AB93] A. Acharya and B. R. Badrinath, *Delivering multicast messages in networks with mobile hosts*, In 13th International Conference on Distributed Computing Systems (Pittsburgh, US), Maio 1993, pp. 292–300.

[AJK04] Eitan Altman, Tania Jiménez, and Daniel Kofman, *Dps queues with stationary ergodic service times and the performance of tcp in overload*, IEEE Infocom 2004, Março 2004.

[ARV93] Sridhar Alagar, R. Rajagopalan, and S. Venkatesan, *Tolerating mobile support station failures*, Tech. report, University of Texas at Dallas, Novembro 1993.

[BBIM93] B. R. Badrinath, A. Bakre, Tomasz Imielinski, and R. Marantz, *Handling mobile clients: A case for indirect intercation*, In 4th Workshop on Workstation Operating System (Napa, US), Outubro 1993, pp. 91–97.

[BDM01] Christopher L. Barrett, Martin Drozda, and Madhav V. Marathe, *A comparative experimental study of media access protocols for wireless radio networks*, Tech. Report LA-UR-01-6217, Los Alamos National Laboratory, 2001.

[BEF+00] Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, John Heidemann, Ahmed Helmy, Polly Huang, Steven McCanne, Kannan Varadhan, Ya Xu, and Haobo Yu, *Advances in network simulation*, IEE Computer **33** (2000), no. 5, 59–67.

[BHSC98] Nina T. Bhatti, Matti A. Hiltunen, Richard D. Schlichting, and Wanda Chiu, *Coyote: A system for constructing fine-grain configurable COmmunication services*, Transaction on Computer Systems **16** (1998), no. 4, 321–366.

[BMT+98] Rajive Bagrodia, Richard Meyer, Mineo Takai, Yu an Chen, Xiang Zeng, Jay Martin, and Ha Yoon Song News, *Parsec: A parallel simulation environment for complex systems*, IEE Computer **31** (1998), no. 10, 77–85.

[CM02] Antonio Capone and Fabio Martignon, *TCP with bandwidth estimation over wireless networks*, Vehicular Technology Conference **3** (2002), 1422–1426.

[Dur01] Alan Mitchell Durham, *A connectionless protocol for mobile agents*, Relatório técnico, Instituto de Matemática e Estatística, Universidade de São Paulo, Janeiro 2001.

[End99] Markus Endler, *A protocol for atomic multicast among mobile hosts*, In Dial M Workshop / Mobicom´99 (Seatle, US), Agosto 1999, pp. 56–63.

[ER00] Markus Endler and Ricardo C. A. Da Rocha, *Flexible simulation of distributed protocols for mobile computing*, In Proceedings of 3rd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems - MSWiM´2000 (Boston, US), Maio 2000, pp. 33–48.

[ES00] Markus Endler and Dilma M. Silva, *RDP: A result delivery protocol for mobile computing*, Relatório técnico, Instituto de Matemática e Estatística, Universidade de São Paulo, Janeiro 2000.

[ESO00] Markus Endler, Dilma M. Silva, and Kunio Okuda, *A reliable connectionless protocol for mobile clients*, In Int. Workshop on Wireless Networks and Mobile Computing (WNMC) with 20th ICDCS (Taiwan), Abril 2000.

[ESS+00] Markus Endler, Dilma M. Da Silva, Francisco J. Silva E Silva, Marcos A. De Moura, and Ricardo C. A. Da Rocha, *Project SIDAM: Overview and preliminary*

results, In Proceedings of the 2nd Brazilian Wireless Communication Workshop (Belo Horizonte, Brazil), Maio 2000, pp. 48–64.

[Est00] D. Estrin, *Network visualization with nam, the VINT network animator*, IEEE Computer (2000), 63–68.

[FML03] Cheng P. Fu, Senior Member, and Soung C. Liew, *A remedy for performance degradation of tcp vegas in asymmetric networks*, IEEE Communications Letter, Janeiro 2003.

[FZ94] G. H. Forman and J. Zahorjan, *The challenges of mobile computing*, IEEE Computer **27** (1994), no. 4, 38–47.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, *Design patterns — elements of reusable object-oriented software*, Addison-Wesley, 1995.

[GLAM02] J. J. Garcia-Luna-Aceves and Marc Mosko, *Performance of group communication over ad-hoc networks*, Proc. IEEE Int. Symp. Computers and Communications ISCC (Taormina, Italy), Julho 2002, pp. 545–552.

[GN03] Thomas Gross and Valeri Naoumov, *Simulation of large ad hoc networks*, Proceedings of the 6th ACM international workshop on Modeling analysis and simulation of wireless and mobile systems (San Diego, US), Setembro 2003.

[HP91] Norman C. Hutchinson and Larry L. Peterson, *The x-Kernel: An Architecture for Implementing Network Protocols*, IEEE Transactions on Software Engineering **17** (1991), no. 1, 64–76.

[JBRAS03] Amit Jardosh, Elizabeth M. Belding-Royer, Kevin C. Almeroth, and Subhash Suri, *Towards realistic mobility models*, ACM MobiCom 03, 2003, pp. 217–229.

[JM96] David B. Johnson and David A. Maltz, *Protocols for adaptative wireless and mobile networking*, In IEEE Personal Communications, Fevereiro 1996.

[Lim03] Weslley Emmanuel Martins Lima, *Uma biblioteca para a simulação de protocolos de entrega de mensagens em redes móveis*, Master's thesis, Instituto de Matemática e Estatística, Universidade de São Paulo, Junho 2003.

[ML98] Geraldo Robson Mateus and Antonio A. Ferreira Loureiro, *Introdução à computação móvel*, 11 Escola de Computação Móvel Rio, 1998.

[PS95] D. Plainfosse and M. Shapiro, *A survey of distributed garbage collection techniques*, In Proceedings of the International Workshop on memory Management (Kinross, Scotland), Setembro 1995.

[RE01a] Mateus De Freitas Ribeiro and M. Endler, *Um protocolo indireto para multicast atômico em computação móvel*, Anais do 3o. Workshop de Comunicação sem Fio (WCSF), Recife, Agosto 2001, pp. 84–91.

[RE01b] R.C.A Da Rocha and M. Endler, *MobiCS: An environment for prototyping and simulating distributed protocols for mobile networks*, Proc. of MWCN´2001, Agosto 2001, pp. 44–51.

[Rib02] Mateus De Freitas Ribeiro, *Desenvolvimento e comparação de dois protocolos para multicast atômico em computação móvel*, Master's thesis, Instituto de Matemática e Estatística, Universidade de São Paulo, Setembro 2002.

[Roc01] Ricardo Couto Antunes Da Rocha, *Uma arquitetura para simulação flexível de protocolos para computação móvel*, Master's thesis, Instituto de Matemática e Estatística, Universidade de São Paulo, Maio 2001.

[Tab02] Lucy Mari Tabuti, *Um estudo de um protocolo de comunicação para dispositivos móveis usando distributed join-calculus*, Master's thesis, Instituto de Matemática e Estatística, Universidade de São Paulo, Setembro 2002.

[ZBG98] X. Zeng, R. Bagrodia, and M. Gerla, *GloMoSim: A library for parallel simulation of large-scale wireless networks*, Proceedings of the 12th Workshop on Parallel and Distributed Simulation (PADS-98) (Los Alamitos), IEEE Computer Society, Maio 1998, pp. 154–161.