Departamento de Engenharia Electrotécnica

Rede P2P sobre WLANs em modo ad hoc

Por:
David Manuel Avelar Paulino

Dissertação apresentada na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa para a obtenção do grau de Mestre em Engenharia Electrotécnica e de Computadores

Orientador: Doutor Luís Filipe Lourenço Bernardo

Monte de Caparica
2008

Resumo

Na dissertação é realizado um estudo à partilha de recursos em redes sem fios muito dinâmicas, com o objectivo de identificar alguns dos seus problemas e apresentar soluções para estes.

É complicado manter a conectividade entre equipamentos presentes numa rede sem fios não. A mobilidade e instabilidade introduz dificuldades no de informação entre os diferentes nós presentes na rede.

Foi idealizado e implementado um algoritmo de localização assíncrona de recursos para operar em redes muito dinâmicas, que permite uma melhor gestão da difusão de pesquisas.

Para demonstrar a eficácia do algoritmo implementado, foi desenvolvida uma aplicação de pesquisa e partilha de ficheiros.

A tecnologia JXTA foi estendida com a implementação do algoritmo de gestão de pesquisas. Durante a implementação deste algoritmo foi criado um novo módulo, com objectivo ser uma base para todas as implementações de suporte a redes dinâmicas sem fios.

Para melhorar o débito na comunicação foi implementado um módulo de transporte independente da plataforma.

Palavras-Chave:

JXTA, Manet, *ad-hoc*, *peer-to-peer*, partilha de ficheiros

Abstract

This thesis studies file sharing in very dynamic ad hoc wireless networks. The main objective is to identify problems and propose solutions for them.

Keeping all nodes communicating in an ad hoc network is hard. Routing between nodes is more complex because of mobility and instability.

A new asynchronous resource location algorithm was designed and implemented to operate in highly dynamic networks. It allows a better search diffusion management.

One file sharing application was developed to prove the algorithm effectiveness.

The JXTA technology was extended with the implementation of the Search Management algorithm. During this work a new module was created to support the implementation of applications for very high dynamic wireless networks.

A platform independent transport module was implemented to improve the communication throughput.

Keywords:

JXTA, Manet, ad-hoc, peer-to-peer, file sharing

Agradecimentos

Gostaria agradecer a todos aqueles que me deram apoio durante o decorrer deste projecto. Também agradeço o acompanhamento, especialmente na fase de implementação, do Prof. Dr. Luís Bernardo.

Quero ainda agradecer o bom ambiente e pelo espírito de entreajuda proporcionado pelos meus colegas que se encontravam a terminar o curso na área de Telecomunicações: Diogo Rodrigues, João Martins, Sérgio Gaspar e Sérgio Vieira.

Por fim agradeço à minha família que sempre me apoiou durante o decorrer desta dissertação.

Faculdade de Ciências e Tecnologia, Março de 2008

David Manuel Avelar Paulino

Índice

Índice de Figuras

Índice de Tabelas

Lista de Siglas

API	Application Programming Interface
BMP	Windows bitmap
BSS	Basic Service Set
CSMA/CA	Carrier Sense Multiple Access / Collision Avoidance
CTS	Clear To Send
DCF	Distributed Coordination Function
ERP	Endpoint Routing Protocol
HTTP	HyperText Transport Protocol
IBBS	Independent Basic Service Set
IEEE	Institute of Electrical and Electronics Engineering
iMAQ	The Integrated Mobile Ad-hoc QoS Framework
JINI	tecnologia de base para arquitecturas de descoberta de serviços
JXTA	diminutivo da palavra inglesa juxtapose
LLC	Logical Link Control
MAC	Media Access Control
MANET	Mobile Ad-Hoc Network
MD5	Message Digest 5
MP3	MPEG Audio Layer-3
NAT	Network Address Translation
NAV	Network Allocation Vector
P2P	Peer to peer – comunicações nó a nó
PBP	Pipe Binding Protocol
PDA	Personal Digital Assistant
PDP	Peer Discovery Protocol
PIP	Peer Information Protocol
PRP	Peer Resolver Protocol
QoS	Quality of Service
RTS	Request To Send
RVP	RendezVous Protocol
SLP	Service Location Protocol
SyncML	Synchronization Markup Language

TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VCS	Virtual Carrier Sense
XML	eXtensible Markup Language

Capítulo 1. Introdução

1.1 Introdução

Até ao desenvolvimento do standard 802.11 por parte do IEEE (*Institute of Electrical and Electronic Engeneering*), as redes locais sem fios (WLANS) não tiveram muita aceitação a nível comercial.

Em 1999 foram lançadas duas normas: 802.11a [80211a] e a 802.11b [80211b]. A norma 802.11a teve dificuldades a ser implementada devido à gama de frequência utilizada (5.2GHz) estar licenciada na Europa, o que limita o uso desta norma ao continente Americano e Asiático. Para além desta limitação, o custo da interface de rádio é mais dispendioso que a concorrente 802.11b. Assim, a 802.11b ganhou esta batalha.

A norma 802.11b usa uma banda não licenciada na região dos 2.4GHz e tem um débito de transmissão de dados máximo de 11Mbps. Actualmente, as placas vendidas no mercado suportam as normas 802.11b/g e por vezes 802.11n draft. As normas 802.11g [80211g] e 802.11n [80211n] permitem débitos mais elevados (até 54Mbps e 300Mbps respectivamente) e funcionam na mesma gama de frequências que a norma 802.11b.

As redes locais sem fios possuem dois modos de funcionamento: um modo estruturado e outro modo não estruturado.

No modo estruturado, o acesso à rede é realizado de forma centralizada. Existe um dispositivo central (o ponto de acesso) que permite o funcionamento da rede. Nesta forma, são permitidas algumas funcionalidades, tais como, fazer ponte entre uma rede com fios e a outra rede sem fios. O outro modo de funcionamento é o modo não estruturado. Neste modo não existe um dispositivo central mas sim vários dispositivos interligados que fazem o encaminhamento de mensagens entre eles. Assim, o modo não estruturado é denominado de modo *ad hoc*.

As interfaces para redes sem fios são utilizadas normalmente em dispositivos móveis. Assim sendo, é questionável o facto de como poderá ser possível manter uma rede estruturada onde não existe uma estrutura? Para responder a esta pergunta, surge um novo conceito denominado MANET (*Mobile Ad-Hoc Network*). Uma rede MANET é uma rede *ad-hoc* onde existe mobilidade por parte dos utilizadores.

Um dos maiores problemas numa MANET é lidar com a instabilidade, que pode ser muito elevada devido à elevada mobilidade dos dispositivos dentro da rede. Numa MANET os algoritmos de encaminhamento tradicionais não são eficazes porque a validade das rotas pode ser muito curta (ver Anexo A).

Neste trabalho, foi realizada uma adaptação da plataforma JXTA [JXTAspec] para MANETS. Inicialmente a plataforma JXTA foi desenvolvida para suportar sistemas *peer-to-peer* em redes estruturadas. Mais tarde, teve uma contribuição importante para o suporte desta plataforma em redes MANET por Nelson Ruivo [Nelson05]. A localização de recursos numa rede MANET contínua foi facilitada. Contudo, só a localização síncrona de utilizadores não é suficiente. Por isso, surgiu este projecto onde um novo módulo foi implementado. Este módulo permite fazer a pesquisa diferida numa rede MANET, guardando as pesquisas e fazendo a sua propagação assincronamente quando a vizinhança dos nós muda. Para testar este módulo, foi realizada uma aplicação que serve de exemplo para futuras implementações de aplicações. O módulo de localização de recursos foi apresentado num artigo científico [Bernardo06], publicado na conferência WINSYS (*International Conference on Wireless Information Networks and Systems*), que decorreu de 7 a 10 de Agosto de 2006 em Setúbal. Este artigo foi posteriormente publicado num livro [Bernardo08] que reuniu os melhores artigos da conferência.

1.2 Objectivos

Diversas abordagens foram concebidas para possibilitar a descoberta de recursos em redes P2P (*peer to peer*). No caso das redes estruturadas, a abordagem mais frequente é a existência de um nó centralizado ou de vários nós concentradores, onde é guardado o registo das localizações dos diversos recursos existentes na rede (e.g. [Chord]). Apesar desta abordagem funcionar bem para as redes estruturadas o mesmo não acontece quando o cenário em questão é uma rede MANET, visto nestas redes os nós serem móveis.

Com a mobilidade dos nós a validade dos registos diminui. Surge então a necessidade de actualizar regularmente os registos. Para manter os registos actualizados é necessário aumentar o volume de troca de dados entre os nós, consequentemente ocupando mais a largura de banda disponível na rede.

Outro ponto importante é definir quem deve armazenar os registos. Ter um nó central ou vários nós centralizadores não é viável numa rede MANET, já que os nós que armazenam esta informação podem ficar fora de alcance, introduzindo custos significativos de descoberta de rota para esses nós.

Para além da descoberta fica pendente a comunicação entre os nós. Numa rede estruturada a largura de banda existente é normalmente superior e mais estável do que a existente numa rede MANET. Com estas limitações deve existir uma preocupação de como os módulos comunicam, ou seja, existência de mecanismos eficazes de comunicação entre os nós.

Como ponto de partida foi escolhida a plataforma JXTA [JXTAspec]. Esta foi desenvolvida para facilitar a implementação de aplicações para redes P2P. Na primeira fase, foi realizado um estudo para determinar o modo de funcionamento desta plataforma, e que problemas surgiam quando esta fosse utilizada no cenário pretendido. Após identificar os problemas foram idealizadas, implementadas e testadas soluções com vista a melhorar a plataforma. Com estas implementações o suporte desta plataforma às redes MANET será melhorado.

Após a conclusão da primeira fase, foi feita uma análise do que se poderia fazer para alargar ainda mais o suporte. Surgiram novas ideias para melhorar a aplicação, com naturalidade surgiu uma segunda fase. Nesta fase foi redesenhada a interface da aplicação com vista a melhorar a interacção da aplicação com a plataforma. Aproveitando a experiência da primeira fase foi desenvolvido um novo módulo de transporte que poderia vir a substituir o da plataforma. Para esta nova fase foi colocado um novo objectivo: alterar o método de transporte da aplicação de modo a permitir comunicações *multi-hop* entre dois nós.

Um problema da pesquisa de recursos JXTA nas redes MANET é quando os nós que estão ao alcance não possuem o recurso desejado. Se nenhum nó ao alcance possuir o recurso, a pesquisa não obtém resultados e obriga o nó a realizar novamente a pesquisa. Esta solução não é a mais eficiente porque o nó nunca sabe quando deve repetir novamente a pesquisa. A solução ideal para resolver este problema é utilizar um mecanismo inteligente de controlo das pesquisas realizadas.

Outro problema prende-se com o acesso a um recurso após a sua descoberta. Para solucionar esta questão é necessário existir um método que seja responsável por

identificar as rotas de modo a ser possível a comunicação entre dois nós independentemente do número de nós que existam entre eles.

Com as rotas definidas é necessário manter e optimizar a comunicação entre os nós. Numa rede *ad hoc* a largura de banda disponível é limitada e depende da ligação estabelecida via rádio entre os nós. Se entre dois nós com ligação directa entre si a largura é limitada, com vários *hops* essa limitação é maior. É necessário manter a troca de informação apenas ao essencial.

1.3 Arquitectura Geral

Os mecanismos existentes para fazer procuras baseiam-se na inundação da rede com pesquisas dos vários utilizadores [Gnutella04]. Numa rede MANET esta situação é problemática. Se todos os nós retransmitirem mensagens existe um aumento na transmissão de dados e, com isto, a probabilidade de existirem colisões aumenta [Oliveira05b]. Outra situação derivada da mobilidade dos nós é o facto de a um dado instante os nós com determinados recursos estarem fora do alcance de rádio. Portanto, não recebem a pesquisa e consequentemente não partilham os seus recursos.

Na plataforma JXTA o mecanismo implementado mais adaptado para realizar pesquisas é baseado em inundação síncrona de pesquisa com limite de retransmissões. Uma pesquisa é efectuada no grupo a que o nó pertence, e destina-se a todos os nós pertencentes a esse grupo.

Existem nós que possuem características especiais, denominados por nós *Rendezvous*. Estes, além de processarem as pesquisas, são responsáveis pela propagação destas pela rede JXTA. Esta propagação é feita com base em pacotes multicast e num mecanismo que retransmite as pesquisas pelos diversos nós Rendezvous conhecidos. No caso das redes MANET, repropagar as pesquisas através do multicast é benéfico já que os nós ao alcance de rádio as recebem. Contudo, falta um mecanismo que propague as pesquisas para além do alcance de rádio do nó em questão. Os outros mecanismos que os nós Rendezvous [JXTAspec] possuem são bastante lentos e não são adequados para as redes MANET.

As mensagens de pesquisa propagadas na rede JXTA contêm um campo onde é registada uma lista que contém a rota por onde esta pesquisa passou. Sempre que um nó propaga a mensagem adiciona-se à lista. Desta forma, quando é construída a

resposta à pesquisa, é aproveitado este campo como rota para a resposta. Além disso, todos os nós que propagaram a mensagem aproveitam para actualizar as suas tabelas de encaminhamento.

Para forçar a propagação da pesquisa para além da fronteira rádio do nó que a realizou é possível utilizar inundação. Ou seja, todos os nós assim que recebessem essa pesquisa repropagavam-na imediatamente. Ter um mecanismo desde género que apenas inunde a rede não é o mais indicado para uma rede MANET.

Do ponto de vista da comunicação *multi hop* é necessário manter as trocas de informação entre os nós no mínimo, ou seja, reduzir a carga dos protocolos utilizados de modo a obter uma ligação mais eficiente.

Este trabalho surge para resolver os problemas acima descritos e implementar na plataforma JXTA um mecanismo tolerante a atraso (*delay tolerant*) que seja mais adequado para redes MANET esparsas. Inicialmente o método de inundação periódico foi desligado. Ao desligar este método de propagação surge um novo problema que é como propagar as pesquisas dum modo eficiente e que não seja tão exigente a nível de largura de banda como o anterior. A solução foi aproveitar parte das ideias do mecanismo de inundação, ou seja, uma inundação controlada. Neste novo mecanismo as pesquisas são propagadas apenas quando um novo vizinho entra ao alcance. Ao adicionar um tempo de vida às pesquisas é permitida uma gestão mais elaborada. Assim, as pesquisas com tempo de vida expirado são descartadas poupando recursos da rede. Como a plataforma não possuía este tempo de vida para as pesquisas, foi necessário modificar as mensagens XML definidas no JXTA [JXTAspec] e acrescentá-lo, estudando ao pormenor o protocolo e procedendo às modificações. Durante as modificações foi mantida a estrutura existente, permitindo que aplicações que não tenham sido desenhadas para tirar proveito destas funcionalidades possam usufruir, dum modo transparente (é adicionado um valor de tempo de vida por defeito), das novas alterações introduzidas.

Outra funcionalidade necessária é que todos os nós que possuam esta plataforma JXTA modificada propaguem as pesquisas, mesmo que estes não pertençam ao mesmo grupo. Nas alterações efectuadas foi introduzido um mecanismo que garante que as pesquisas são sempre propagadas no grupo onde foram inicialmente submetidas.

No modo normal as pesquisas são propagadas com um intervalo de tempo fixo. Contudo, quando um novo vizinho é detectado existem duas abordagens: Na primeira o nó detecta e repropaga imediatamente as pesquisas que possui armazenadas; a outra é o nó aguardar um tempo aleatório e monitorizar o meio, para verificar se recebe alguma das pesquisas que estejam à espera de ser propagadas. Neste caso elimina-as da lista de espera.

Após a localização dos recursos é necessário que exista um meio de comunicação que permita aceder a estes. Quando falamos de redes MANET a comunicação entre os nós deve ser mantida no mínimo possível.

A plataforma JXTA oferece já alguns meios para os nós comunicarem entre si. Contudo estes meios foram idealizados tendo em conta redes estruturadas, o que implica que não foram tidas em conta as limitações das redes MANET. Ou seja, baseia-se em estruturas de redes fixas onde normalmente a largura de banda disponível entre nós é superior e mais estável. Cada mensagem que o JXTA envia possui uma carga elevada em relação aos dados que transmite. Para reduzir essa carga foi desenhado e implementado um mecanismo de transporte alternativo, mais eficiente numa rede MANET.

1.4 Estrutura da Dissertação

O próximo capítulo descreve trabalho relacionado com o tema desta dissertação.

Os conteúdos teóricos são abordados no terceiro capítulo exemplificando os problemas associados à tecnologia utilizada no projecto.

No capítulo quatro estão descritos os pormenores de implementação, bem como as funcionalidades utilizadas e implementadas na aplicação desenvolvida.

A apresentação dos resultados é feita no quinto capítulo. E finalmente, no último capítulo são expostas as conclusões alcançadas.

Capítulo 2. Plataforma JXTA

Quando se deseja implementar uma aplicação para funcionar num ambiente P2P (*peer to peer*) é necessário ter em conta diversas situações que estão associadas a este tipo de implementações. Como nem sempre é viável começar uma aplicação de "raiz" e pensar em todas as situações, os programadores acabam por desistir de realizar a aplicação. Outra solução é adaptar alguma aplicação existente sob uma licença *"open source"*. Esta solução, por sua vez, não se enquadra nas especificações desejadas.

Com estes obstáculos em mente, começou-se a desenvolver plataformas que implementassem uma rede P2P mas apenas com alguns serviços básicos, como pesquisa e transporte. Assim, uma aplicação utilizaria a plataforma sem se preocupar com as situações associadas a uma rede P2P e focar o desenvolvimento na aplicação desejada.

Com a evolução tecnológica surgiram as redes sem fios. Com este novo tipo de rede, novas situações foram adicionadas a uma rede P2P. Numa rede sem fios é muito difícil garantir estabilidade. Os nós podem entrar e sair de alcance a qualquer momento, o que torna ainda mais difícil desenvolver aplicações P2P.

Tendo em conta o carácter descentralizado das redes P2P, as soluções que implementam estas redes têm um interesse especial quando aplicadas a uma rede *ad hoc*.

2.1 A Plataforma JXTA

A plataforma JXTA [JXTAspec] é uma das soluções que surgiram para facilitar a implementação de redes P2P. Esta plataforma tem um certo peso na comunidade em relação às suas rivais devido a ter sido pioneira e de proporcionar uma interface simples de desenvolvimento. A SUN Microsystems, detentora da sua licença, teve desde o início o objectivo de deixar a plataforma *open source* a fim de explorar comercialmente potenciais ideias que fossem introduzidas.

Ao desenvolver esta plataforma foram tidas em conta três premissas:

- Interoperabilidade – entre vários sistemas P2P e comunidades;
- Independência da Plataforma – diversas linguagens, sistemas e redes;
- Omnipresença – em todos os dispositivos digitais.

O objectivo destas premissas é a independência da plataforma a nível do sistema operativo onde esta é utilizada, assim como sua a linguagem de programação. Com a plataforma, é acrescentada uma camada entre o software e o hardware denominada por *"middleware"*. Esta camada permite aos programadores de aplicações abstraírem-se dos problemas associados às implementações de redes P2P.

Com a introdução desta camada, passa a existir a possibilidade de o meio de transporte ser qualquer um. De momento, apenas existe suporte para TCP e UDP, contudo, é possível adicionar mais protocolos de transporte sem mudar a implementação das aplicações existentes.

Para permitir que as mensagens atravessem *firewalls* ou NAT (*network address translation routers*), existe cooperação entre os diversos nós. No caso das mensagens "atravessarem" *firewalls,* existe o recurso ao protocolo HTTP.

A segurança não é obrigatória. Cada nó deve escolher caso a caso o protocolo que achar necessário tanto para o caso de autenticação de utilizadores como de encriptação das mensagens.

A plataforma JXTA contém um serviço de encaminhamento que permite o funcionamento em redes *ad hoc*. Assim, cada nó faz a retransmissão das mensagens. Por haver associação de um nó a um identificador único que não depende de endereço de rede, é possível haver mobilidade numa rede *ad hoc*. Esta mobilidade é permitida à custa do serviço de descobertas da plataforma JXTA.

2.1.1 Arquitectura JXTA

O JXTA é composto por conjunto de camadas. Cada camada está bem definida e com objectivos concretos.

A camada "JXTA Core" contém as ferramentas essenciais para o funcionamento duma rede P2P, criação de grupos e de nós, descoberta de nós, transporte de

mensagem onde vários protocolos podem ser suportados. Até à data deste projecto existem 3 tipos suportados (TCP, UDP e HTTP o último lida com o transporte para nós por detrás de *"firewalls"*).

A camada seguinte é a de serviços, composta pelos serviços que são essenciais para uma rede P2P. Esta camada pode ser expandida pelos programadores de aplicações, havendo no entanto distinção nos serviços *standard* do JXTA que são inicializados sempre que é inicializada a plataforma e os serviços considerados extra. Todos os serviços extra devem ser inicializados apenas no grupo ao qual se destina a aplicação. É possível que um serviço que seja útil passe de extra para um serviço bem conhecido caso este seja bem aceite pela comunidade JXTA, nomeadamente por quem está a desenvolver a plataforma.

Finalmente vem a camada das aplicações, onde estão as aplicações implementadas: desde aplicações de conversação até partilha de ficheiros.

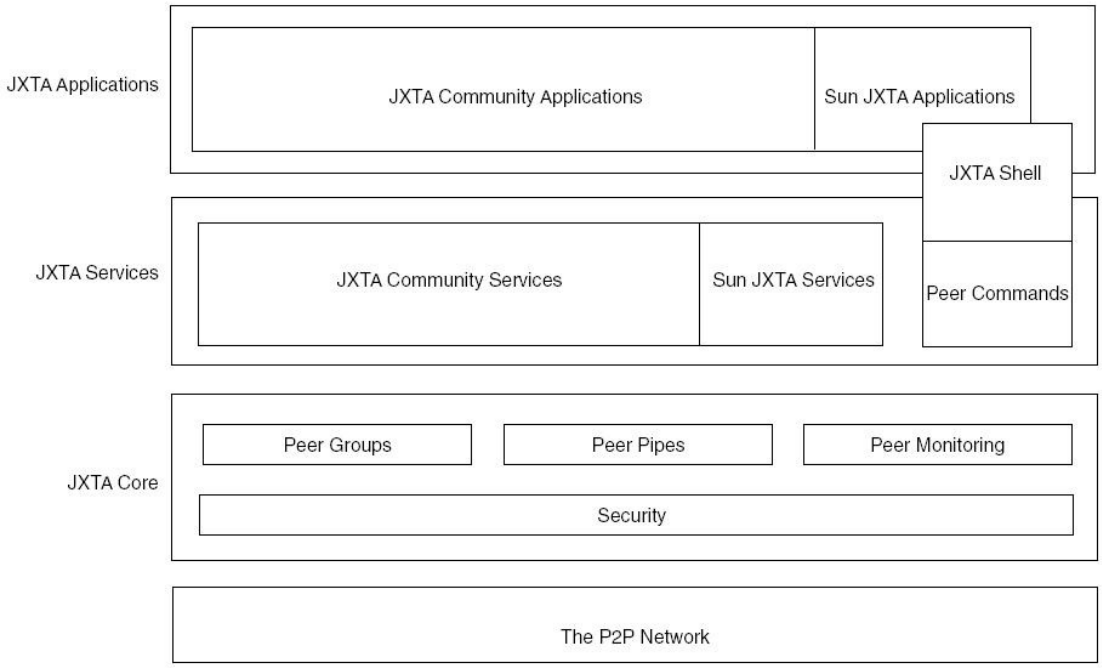

Figura 2.1 – Modelo da Plataforma JXTA

2.1.2 Conceitos JXTA

Existem alguns conceitos introduzidos na rede JXTA:

- **Nós** – Um nó é qualquer dispositivo que implemente um ou mais protocolos de JXTA. Cada nó é independente dos restantes e opera num modo assíncrono.

- **Grupo de Nós** – É um conjunto de nós que concordam a implementar os mesmos serviços. Quando é inicializado o JXTA o nó entra no grupo por omissão (*"NetPeerGroup"*), este grupo implementa os protocolos básicos. Quando é criado outro grupo é utilizada a especificação deste como base. O JXTA já possui mecanismos para criar, publicar grupos, assim como, juntar e abandonar grupos. As pesquisas são propagadas no grupo onde são feitas.

- **Pipes** – Representam um canal de comunicação entre nós, que são uma abstracção das ligações de rede. Os canais de comunicação são assíncronos e unidireccionais. Mesmo que exista a possibilidade de comunicação bidireccional no nível físico são criados dois canais. Suportam o transporte de qualquer objecto entre nós.

- **Módulos** – São uma abstracção de como um programa implementa uma rotina em JXTA. Para além dos módulos existentes no núcleo é possível adicionar novos módulos. A implementação destes módulos extra fica a cargo das aplicações.

- **Anúncios** – Os recursos do JXTA são representados por anúncios em XML. Quando os nós procuram os recursos a resposta é um anúncio. Num anúncio está contida toda a informação relativa ao recurso que este representa. Por exemplo, num anúncio de um canal

(*PipeAdvertisement*) está contida a informação do ID do canal, o nome, tipo, e a sua descrição.

2.1.3 Protocolos JXTA

A rede JXTA tem seis protocolos especificados:

- *Peer Resolver Protocol* (PRP) – Define o formato e transmissão das pesquisas realizadas pelos nós com destino a outro nó ou a um grupo de nós, e respostas a essas pesquisas na rede JXTA. A cada pesquisa tem associado o identificador único do nó que o fez. Este identificador é incluído na resposta quando esta é gerada. Assim, o nó sabe a que pesquisa pertence a resposta que recebeu. A unicidade do identificador tem de ser garantida pela aplicação ou serviço que realiza a pesquisa.

- *Peer Discovery Protocol* (PDP) – Contém as especificações de como os recursos disponíveis são anunciados na rede JXTA. Cada recurso tem um anúncio. Estes anúncios são formatados em XML (*eXtensible Markup Language*) para permitir uma independência da linguagem de programação.

- *Peer Information Protocol* (PIP) – Após a descoberta do nó é importante obter mais informações sobre o mesmo para poder tomar decisões. Com este protocolo é possível monitorizar outros nós. Tempo *on-line*, largura de banda disponível são exemplos de informação possível de obter.

- *Pipe Binding Protocol* (PBP) – Quando um canal é criado este não pode ser utilizado até estar associado a um endereço do nó. A gestão destas operações fica a cargo deste protocolo.

- *Endpoint Routing Protocol* (ERP) – Este protocolo permite gerir as rotas entre dois nós quando não é possível existir conectividade entre ambos.

A rota é definida recorrendo a um ou mais nós intermédios. Para este protocolo funcionar necessita de nós que aceitem ser nós *rendezvouz*.

- *Rendezvous Protocol* (RVP) – Este protocolo define como são propagadas as mensagens pela rede JXTA. Contém o Rendezvouz Walk que faz a propagação das mensagens pelos vários nós Rendezvouz existentes na rede. A propagação das mensagens é essencial para aumentar o alcance das pesquisas na rede JXTA.

Estes protocolos estão agrupados em três grupos, representados na figura 3.2. Em cada grupo os protocolos são independentes entre si, entre os grupos pode haver dependência. Por exemplo, o *Peer Discovery Protocol* depende do *Peer Resolver Protocol*.

Para uma aplicação ser considerada um nó na rede JXTA tem de implementar pelo menos o *Peer Endpoint Protocol*, os restantes são considerados opcionais e têm como função melhorar a interoperabilidade entre os diversos nós.

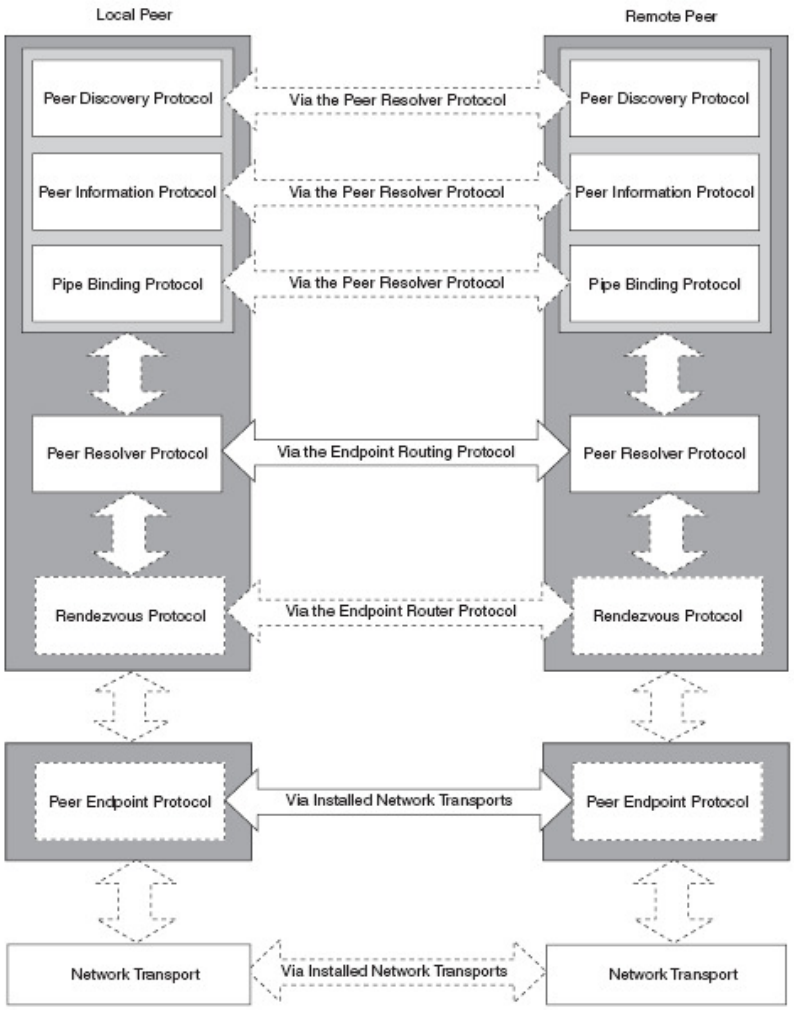

Figura 2.2 – Pilha de protocolos JXTA

2.1.4 Serviços JXTA

O ponto anterior explicitava quais os protocolos existentes na rede JXTA, que protocolos estão implementados na forma de serviços. Estes serviços constituem um nível da plataforma JXTA. Existem já por defeito alguns serviços básicos que são inicializados no arranque da plataforma, quando é feita a entrada no grupo pré definido, o *NetPeerGroup*. Este grupo tem como objectivo ser um ponto de partida para os restantes. Cada grupo tem um identificador único (*GroupPeerID*). Assim, o grupo pré definido terá o mesmo identificador em qualquer aplicação que inicialize a plataforma JXTA.

Apesar de não ser possível implementar uma aplicação P2P recorrendo apenas ao grupo *NetPeerGroup*, este é muito importante porque permite o envio de anúncios

relativos aos serviços básicos da plataforma JXTA. Assim, por exemplo, um nó pode fazer uma pesquisa por grupos existentes.

Além destes serviços incluídos na plataforma JXTA é dada a possibilidade de estender estes. Deste modo é possível implementar um serviço que esteja mais optimizado para a aplicação que se pretende implementar. O novo serviço terá que depender de pelo menos um serviço existente para funcionar. Para tal são disponibilizadas funcionalidades através de uma API.

Quando é criado um grupo, os serviços por defeito são inicializados mas também podem ser especificados quais os serviços adicionais para o novo grupo de nós. Se um nó quiser juntar a um grupo terá de inicializar todos os serviços especificados para esse grupo. Deste modo, todos os nós nesse grupo possuem os mesmos serviços. Esta funcionalidade permite que uma aplicação que utilize a plataforma JXTA pode adicionar serviços sem alterar o núcleo da plataforma.

Os serviços disponíveis na plataforma são os seguintes:

- Serviço de Descoberta (*Discovery Service*) – É responsável por pedir anúncios remotos, verificar os anúncios locais, publicar novos anúncios tanto localmente como remotamente. Também é responsável pela gestão dos anúncios, apagando-os caso o seu tempo de validade tenha expirado.

- Serviços de canais (Pipe Service) – Tem como objectivo o estabelecimento de canais entre nós (Pipes). Estes canais podem ser unidireccionais, bidireccionais ou multi direccionais (multicast).

- Endpoint Service – É responsável por associar os vários endereços virtuais a um endereço físico. Este serviço possui diversos mecanismos de transporte de mensagens.

- Rendezvouz Service – Este serviço é responsável pela propagação das mensagens pelos vários nós. Este serviço apenas é inicializado caso seja especificado que o nó deseja fornecer as funcionalidades aos restantes nós do seu grupo. Este serviço possibilita que exista um nó que faça a ligação entre outros dois que não conseguem estabelecer ligação directamente um com o

outro. Existe ainda o mecanismo denominado Rendezvouz Walk que permite que as pesquisas circulem entre os vários nós Rendezvouz aumentando assim o alcance destas.

- Relay Service – É semelhante ao serviço de rendezvous, mas um pouco mais "leve" pois apenas serve para interligar dois nós. Não contém nenhum mecanismo de propagação de pesquisas, como acontece no Rendezvous Service.

- Peer Info Service – Serviço que serve para a troca de informações entre os nós referentes a outros nós. Por exemplo, um nó armazena informação sobre os nós conhecidos e pode enviar esta informação para um novo nó que surja na rede.

- Serviço de Pesquisas (*Resolver Service*) – É o mecanismo mais simples de pesquisas. Estas ficam associadas ao grupo que são efectuadas, e as respostas às pesquisas são associadas ao mesmo grupo.

Não está definido o método de invocação dos serviços, nem o formato que a especificação do serviço deve ter. Esta liberdade é para permitir que os programadores possam criar serviços com invocações e formatos à sua medida. Na rede JXTA apenas se especifica como devem ser pesquisados os serviços e quais os métodos que são publicados.

2.1.5 Anúncios JXTA

Todos os recursos existentes na rede JXTA estão representados por anúncios. Para a plataforma JXTA foi definido pelos programadores que estes anúncios teriam o formato XML (eXtensible Markup Language).

- *Peer Advertisement*
- *PeerGroup Advertisement*
- *ModuleClass Advertisement*
- *ModuleSpec Advertisement*
- *ModuleImpl Advertisement*
- *Pipe Advertisement*
- *Rendezvous Advertisement*

Os anúncios possuem um tempo de validade, após o qual terão que ser revalidados. A gestão destes anúncios assim como a sua publicação está a cargo do serviço de descobertas.

A existência dum mecanismo deste tipo possibilita a criação de anúncios personalizados, ou seja, é uma mais valia da plataforma JXTA que facilita a implementação duma aplicação de partilha de recursos.

Representar os recursos fornecidos com anúncios, permite a uma aplicação aceder às funcionalidades do serviço de descobertas. Este serviço lida de forma transparente os diversos tipos de anúncios de forma transparente.

2.1.6 Propagação de pesquisas na rede JXTA

Para distribuir as pesquisas pela rede JXTA, foi concebido um método na plataforma que propaga estas pelos nós dentro da rede. Para que exista um melhor controlo sobre a largura de banda ocupada, os nós não são obrigados a propagar as pesquisas que recebem. O serviço responsável pela propagação é o serviço de *Rendezvous*. Sendo este opcional, somente os nós que possuem este serviço activo é que poderão propagar pesquisas utilizando os dois mecanismos existentes neste

serviço. Outra característica do serviço de *Rendezvous*, é que pode estar activo apenas para um determinado grupo. Ou seja, um nó pode estar em vários grupos mas apenas repropaga as mensagens de alguns deles.

O primeiro mecanismo existente é básico e trata-se de repropagar as mensagens recorrendo ao multicast. Este mecanismo pode ser muito útil numa rede com fios, ao contrário da rede sem fios, em que basta estar um nó temporariamente fora de alcance para não receber esta informação.

O outro mecanismo é um algoritmo mais elaborado, e muito mais lento, que obriga a que os nós *rendezvous* possuam uma lista de nós *rendezvous* da rede. Este é designado por *Rendezvous Walk*.

Assim que uma pesquisa é processada pelo nó, é feita uma verificação se possui o serviço de *rendezvous* activo para o grupo a que pertence esta pesquisa. Após cumprir este teste, são inicializados os dois mecanismos. Para o primeiro caso, é feito o envio da pesquisa via multicast. O *Rendezvous Walk* é mais complicado: O primeiro passo é colocar a mensagem numa lista de espera. Só então, é percorrida uma lista de nós *rendezvous* e enviados as pesquisas. Este processo é lento porque não são repropagadas todas as pesquisas imediatamente para cada nó pertencente a lista. O atraso que existe é para evitar uma sobrecarga no meio de comunicação. Sem este atraso, a probabilidade de o meio ficar congestionado será elevada. Para uma rede MANET este processo não tem muita utilidade já que os nós *rendezvous* conhecidos podem estar fora de alcance ou desligados quando chegar a vez de receber as pesquisas.

Capítulo 3. Trabalho Relacionado

Este capítulo destina-se a referenciar as diversas abordagens existentes. São abordadas algumas implementações que foram consideradas relevantes.

Algumas plataformas tiveram inicialmente um grande entusiasmo, mas por fim acabaram por desenquadrar-se com o objectivo inicial, e com isto, o seu desenvolvimento estagnou.

As redes MANET são um dos temas frequentemente abordados em diversos artigos científicos. Diversas soluções têm sido apresentadas nos últimos anos, contudo, existe um factor comum entre elas na tentativa de resolução de dois problemas: o reencaminhamento das mensagens e a pesquisa de recursos dentro das redes MANET.

Destas duas premissas a mais abordada sem dúvida é a maneira como deve ser feito o reencaminhamento de mensagens.

3.1 The Triana Project

Triana [Taylor03] é uma aplicação que foi desenvolvida na Universidade de Cardiff. Trata-se duma aplicação contendo um conjunto de ferramentas desenhadas para processamento distribuído. A aplicação em si possui uma interface gráfica onde é possível utilizar realizar diversas operações de análise de dados. Estas operações são normalmente dispendiosas. Assim, o uso de processamento distribuído é uma mais valia.

Para estabelecer uma rede onde os utilizadores disponibilizem os seus recursos no auxílio da análise de dados foi implementada uma interface *Grid Application Prototype* (GAP). Inicialmente a GAP foi desenvolvida tendo como base a plataforma JXTA. Devido aos diversos obstáculos técnicos e legais que surgiram foi repensada uma nova plataforma para substituir a interface JXTA.

Um requisito era que os diversos nós pudessem comunicar de forma eficiente. Com o decorrer dos testes foi identificado que a plataforma JXTA por vezes falhava no estabelecimento de ligação entre os nós.

Para a aplicação Triana não era necessária a complexidade que plataforma JXTA possui, foi então idealizada e implementada a P2PS, esta está descrita em pormenor mais à frente.

Para garantir a retrocompatibilidade foi mantida a interface com JXTA e adicionada a interface para P2PS. Deste modo era possível para as versões anteriores continuarem a comunicar recorrendo à plataforma JXTA.

O facto de existir a GAP não permite a aplicação Triana comunicar directamente com a P2PS. Sendo assim, no futuro existe a possibilidade de substituir completamente a GAP. Assim a aplicação teria acesso a P2PS de forma a poder utilizar todas a potencialidades desta.

3.2 Peer to Peer Simple (P2PS)

A plataforma *Peer to Peer Simple* (P2PS) [Wang05] foi desenvolvida na universidade de Cardiff. Foi idealizada tendo em conta a substituição da plataforma JXTA na aplicação Triana (anteriormente mencionada).

A equipa de desenvolvimento aproveitou a experiência que possuía ao trabalhar com a plataforma JXTA. Para o funcionamento da aplicação Triana não eram utilizadas todas as funcionalidades existentes na plataforma JXTA.

Um dos argumentos mais relevantes referido em [Wang05] é a instabilidade dos canais de comunicação da plataforma JXTA e a sua limitação ao protocolo TCP/IP. Nesse artigo é referido que os canais por vezes não são criados o que leva a realizar um novo pedido para a criação dos mesmos.

A plataforma P2PS assemelha-se a plataforma JXTA. Uma diferença notória é o facto que na plataforma P2PS o mecanismo de descobertas é diferente, ou seja, todas as pesquisas são anúncios, enquanto que na plataforma JXTA existe uma separação entre as pesquisas e anúncios. Esta diferença permite que a plataforma P2PS possua o mesmo mecanismo para efectuar uma pesquisa e anunciar recursos.

Em relação às semelhanças a plataforma P2PS possui três serviços que existem na plataforma JXTA, o *Pipe Service*, o *Discovery Service* e o *Rendezvous Service*. Apesar dos nomes serem semelhantes a sua funcionalidade é diferente. Por exemplo no *Discovery Service* da P2PS foram englobados o serviço de pesquisas (*Resolver Service*) e o serviço de anúncios (*Discovery Service*) da plataforma JXTA.

As premissas existentes na plataforma JXTA são passadas para a plataforma P2PS, ou seja, manter a independência dos sistemas utilizados e linguagens de programação. As mensagens são em XML tal com na plataforma JXTA. A diferença está na constituição destas. Na plataforma JXTA adiciona um "envelope" à mensagem por cada nível que esta passa. Ao fazer isto o tamanho da mensagem aumenta. Ao contrário, na plataforma P2PS o excesso de envelopes foi removido.

A plataforma P2PS possui de raiz transporte UDP integrado, o que não acontece com a plataforma JXTA.

3.3 Agile Computing Middleware

A descoberta de recursos e serviços em redes ad hoc utilizando *Agile Computing* está descrita em [Suri03].

O mecanismo de gestão de grupos foi pensado de forma a minimizar a utilização dos recursos dos nós (CPU, memória, armazenamento e rede). A gestão de grupos foi idealizada para as redes MANET. Um nó que esteja associado a um grupo apenas consegue ver os nós que pertencem a esse grupo que estejam a seu alcance. Este facto não impede que um nó não possa pesquisar recursos que possam existir em nós fora do alcance.

A gestão de grupos suporta dois tipos de propagação de informação, uma proactiva e outra reactiva. Existe controlo na propagação de mensagens de modo a impedir inundações na rede.

A versão actual da gestão dos grupos utiliza apenas o protocolo UDP tanto para mensagens *multicast* como *unicast*. Apenas seis tipos de mensagens são utilizados para gerir os recursos entre os nós: três para gestão de informação (PING, INFO e GROUP_DATA) e três focados nas pesquisas (PEER_SEARCH, PEER_SEARCH_RPG e PEER_SEARCH_REPLY).

Futuramente é mencionado que devem ser integradas as mensagens de PING e INFO na parte de encaminhamento de mensagens para reduzir ainda mais a utilização de largura de banda por parte da gestão de grupos.

3.4 Mockets Middleware

A plataforma Mockets [Tortonesi06] foi desenhada com o intuito de apresentar um substituto do protocolo TCP nas comunicações em redes sem fios. A sua proposta passa por utilização de *mobile sockets* (*Mockets*) em vez de TCP *sockets*.

Esta plataforma tem como objectivos o suporte à mobilidade nas comunicações, e facilitar a implementação de aplicações. A plataforma oferece meios de controlo e monitorização da rede às aplicações que a utilizem. A monitorização é fornecida para que certas decisões fiquem a cargo da aplicação em vez de ser a plataforma a decidir o que fazer. Outra funcionalidade é a diferenciação do tipo de mensagens que são enviadas.

A arquitectura da plataforma é composta por dois níveis. A primeira camada é composta por dois módulos: um emissor e outro receptor. Esta camada utiliza o protocolo UDP para comunicar. A segunda camada é composta pelos módulos principais, *Message Management*, *Connection Status Monitor*, *Session Management* e *Traffic Differentiation*.

Para facilitar a mudança das implementações na plataforma ainda é possível utilizar o protocolo TCP.

A comparação entre a utilização de *sockets* TCP e *Mockets* é feita em [Tortonesi06]. Pelos resultados apresentados, os *Mockets* poderão ser uma alternativa para as comunicações nas redes sem fios. O artigo refere que um dos problemas com a utilização de *sockets* TCP é o facto de a perda de um pacote ser interpretada como um problema de congestionamento na rede, o que leva a uma redução no débito dos dados enviados por parte do emissor.

3.5 P2P Mobile Sensors Networks

Uma das utilizações possíveis para as redes sem fios é a implementação de redes de sensores onde estes comunicam através de rádio. Nestes cenários a rede sem fios é composta por vários sensores sem fios. Para ler a informação dos sensores existe um nó da rede que funciona como Gateway.

Em [Srdjan05] é especificada uma possível *Gateway* entre a rede de sensores e os utilizadores que pretendam aceder a informação disponibilizada pelos sensores.

Segundo esta abordagem a *Gateway* é o nó central do ponto de vista da rede de sensores, sendo responsável por fazer a gestão dos sensores e centralizando a informação.

Nesta implementação foi utilizado o JXTA como meio de comunicação entre a *Gateway* e os nós que acedem à informação dos sensores.

3.6 Service Location Protocol (SLP)

O protocolo *Service Location Protocol* (SLP) [SLP99] foi proposto pela IETF (*Internet Engineering Task Force*) para suportar a pesquisa de recursos na Internet. Este protocolo não foi idealizado tendo em conta as redes *ad hoc*. O protocolo especifica que os nós realizam uma pesquisa aos serviços através de um *"User Agent"* e que os serviços encontram-se associados a um *"Service Agent"* O SLP prevê a existência de *"Directory Agents"*, que são responsáveis por armazenar informações sobre os *"Service Agent"* que se registem. Quando um *"User Agent"* pesquisa um determinado serviço o *"Directory Agent"* deverá responder indicando como aceder ao serviço pretendido.

Em 2005 foi proposta uma extensão ao SLP [Penz05] onde o SLP era aplicado à gestão de serviços em redes *ad hoc*. Esta implementação baseia-se essencialmente na especificação de uma arquitectura de descoberta de serviços. Deixa de lado os problemas de encaminhamento de mensagens, e assume que os nós estão todos ao alcance rádio. As pesquisas nesta extensão baseada em SLP são feitas por multicast, ou seja, o *"User Agent"* envia uma pesquisa e os *"Service Agent"* que possuírem o serviço pretendido respondem.

3.7 Delay tolerant networks

Uma proposta para o problemas das redes descontínuas é apresentada em [Tariq06]. Esta consiste em ter um mecanismo de transporte de mensagens para

permitir a comunicação numa rede onde os diversos nós se encontram afastados entre si.

A proposta baseia-se num "*Message Ferry*", responsável por distribuir as mensagens pelos vários nós.

O princípio do "*Message Ferry*" é que em dada altura os nós que se movimentam numa determinada área passam por uma sequência de pontos específicos a cada altura. Sendo assim o "*ferry*" possui uma rota com pontos de paragem que são calculados de modo a minimizar a duração da rota.

É referido que poderão existir casos onde um ponto de paragem seja coincidente para dois nós móveis, contudo é referido que neste tipo de redes estas situações são raras.

Em cada ponto de paragem o "*Message Ferry*" espera um determinado tempo antes de prosseguir a sua rota, ou após terminar a troca de mensagens com o nó associado a esse ponto de paragem. O tempo de espera permite que o ferry não fique parado à espera do nó, já que a proposta assume que os nós possuem rotas livres e que nem sempre passam pelo ponto de paragem do *ferry*.

O *ferry* limita o número de mensagens máximo que armazena por nó, de forma a optimizar os recursos do *ferry*. A proposta especifica que caso seja atingido o número máximo de mensagens as novas serão descartadas até que este seja despejado no próximo encontro com o nó a que estas se destinam.

Para optimizar o percurso do *message ferry* duas situações são tidas em conta: se o ponto de paragem de um nó deve ser mais distante e com menos tempo de paragem ou mais perto e o tempo de espera ser superior.

Nesta especificação os pontos de paragem ficam associados a um determinado nó, ou seja, não é flexível à introdução de novos pontos de paragem, nem a rotas não programadas (em resposta a um evento não previsto).

3.8 Resumo

A tabela 3.1 apresenta uma relação entre as vantagens e desvantagens das abordagens anteriormente mencionadas.

A solução abordada nesta dissertação é a JXTA + Transporte. Esta solução híbrida pretende optimizar a plataforma JXTA de forma a melhorar o seu desempenho nas redes MANET.

	Vantagens	**Desvantagens**
JXTA	• Facilidade de implementação de aplicações para partilha • Interoperabilidade	○ Elevado "overhead" introduzido ○ Limitado ao transporte TCP ○ Arquitectura Complexa ○ Canais de comunicação pouco fiáveis ○ Não optimizado a redes MANET
JXTA + Transporte	• Novo transporte reduz a carga do protocolo • Optimizado para redes MANET • Transporte TCP e UCP	○ Continua a depender das pesquisas JXTA ○ Transporte não possui QoS
Agile	• Optimizado para redes MANET • Propagação de informação pró activa e reactiva	○ Limitado ao transporte UDP ○ Excesso de mensagens "PING"
iMAQ	• Optimizado para redes MANET • Possui QoS	
Mockets	• Optimizado para redes MANET • Diferenciação de serviços • Alternativa ao transporte TCP	
P2PS	• Menos "overhead" que a plataforma JXTA • Arquitectura mais simples que a plataforma JXTA • Transporte TCP e UCP	○ Utilização de XML nas mensagens ○ Não optimizado a redes MANET
Message Ferry	• Uma boa solução para redes onde não é possível estabelecer rotas entre todos os nós da rede • Poupança de recursos	○ Poderá levar a perda de mensagens ○ Demora na entrega de mensagens

Tabela **Erro! Estilo não definido.**.1 – Comparativo de plataformas.

Capítulo 4. Arquitectura Desenvolvida

4.1 Melhoramentos à plataforma JXTA

Para que uma aplicação seja considerada um nó da rede JXTA terá que implementar pelo menos os protocolos essenciais para assegurar o funcionamento deste na rede. Uma das mais-valias da rede JXTA é a possibilidade de efectuar troca de mensagens entre aplicações implementadas com linguagens de programação diferentes. Isto é possível devido ao formato das mensagens ser XML (*eXtensible Markup Language*).

Para simplificar a utilização da rede JXTA foram desenvolvidas duas plataformas, uma implementada em JAVA e a outra em C. Estas plataformas implementam todos os protocolos definidos na especificação da rede JXTA. Ambas foram inicialmente desenvolvidas por um grupo de profissionais da SUN Microsystems, mas estas plataformas encontram-se sob uma licença de código aberto, permitindo que vários programadores contribuam para o seu desenvolvimento.

Neste trabalho foi utilizada a versão 2.3.5 da plataforma em JAVA, que era a versão corrente à data de arranque. Foram adicionadas as alterações que tinham sido desenvolvidas num trabalho anterior [Oliveira05a].

Quando uma aplicação é desenhada para funcionar sobre uma das plataformas JXTA alguns requisitos são necessários. O mais importante é inicializar a plataforma através da aplicação. Ao realizar esta operação o nó entra no grupo predefinido (*netPeerGroup*). Embora seja possível usar este grupo é aconselhável criar um novo ou juntar a um existente.

Com o arranque da plataforma os serviços por esta disponibilizados são inicializados, contudo apenas alguns entram imediatamente em funcionamento. Para além destes, outros poderão ser utilizados. Cabe aos programadores das aplicações definirem quais são os serviços adicionais que devem ser postos em funcionamento. Alguns serviços estão associados a um grupo, sendo possível ter grupos com serviços diferentes activos.

Na plataforma JXTA é disponibilizada uma interface (API – *Application Programming Interface*) que permite às aplicações acederem a classes que implementam os protocolos da rede JXTA. Existe uma clara divisão entre a interface

para as aplicações e a interface do núcleo da plataforma. Esta divisão permite evoluir os diversos módulos que fazem parte do núcleo sem afectar significativamente as aplicações que tenham sido desenvolvidas para versões anteriores da plataforma.

Existe um documento [JXTAb] que descreve como deve ser implementada uma aplicação para utilizar a plataforma JXTA. Neste documento é referido que apenas deve ser utilizada a interface API da plataforma. No entanto é possível utilizar a interface do núcleo por parte das aplicações. A utilização da interface do núcleo permite adicionar funcionalidades à aplicação, porque permite aceder directamente a serviços que não estão disponibilizados na interface API. Contudo, pode levar a problemas quando é feita uma actualização da plataforma. Na evolução da plataforma o essencial é manter a interface para as aplicações inalterada enquanto a interface do núcleo pode sofrer alterações.

Na realização deste trabalho foram introduzidas algumas alterações tanto a nível da API da plataforma JXTA como a introdução de um novo serviço. Com a definição do serviço existe a possibilidade deste ser activado apenas pelos nós que o pretendam fazer. Foi alterado o módulo de agrupamento de nós realizado em [Nelson05] de modo a existir uma melhor integração com o novo serviço implementado.

4.1.1 Protocolo de gestão de pesquisas

Este protocolo foi desenvolvido de modo a permitir aos nós da rede JXTA a possibilidade de armazenar as pesquisas (*"querys"*) e no momento oportuno propagá-las.

O protocolo possui três blocos. O primeiro trata de processar e armazenar as pesquisas recebidas. Existe a possibilidade de configurar o tipo de pesquisas que são armazenadas. As pesquisas possuem tempo de vida. Se o tempo de vida for igual a zero segundos significa que a pesquisa não deve ser armazenada por nenhum nó que a receba.

O segundo bloco é responsável por validar as pesquisas recebidas e indicar quais estão nas condições de serem propagadas. Esta validação é feita através do tempo de vida de cada uma. Se já tiver esgotado o tempo de vida a pesquisa é eliminada. Caso seja válido a pesquisa é colocada numa lista de espera para repropagação. Este bloco pode é activado caso seja detectado um novo vizinho.

E finalmente o último bloco que é responsável pela repropagação das pesquisas. A função deste é percorrer a lista de espera e propagar as pesquisas uma a uma. Este bloco apenas é activado se o bloco anterior encontrar pesquisas que sejam válidas para serem propagadas. Para evitar colisões este bloco tem um tempo de espera aleatório para iniciar a propagação.

O funcionamento deste protocolo baseia-se no facto que dentro do seu alcance de rádio (figura 4.1) os nós recebem sempre as pesquisas independentemente de o nó possuir o serviço, ou não, que é destinado esse pesquisa. Na figura 4.1 é possível visualizar o nó número 3 que está fora do alcance de rádio dos restantes nós. Neste caso, este não recebe a pesquisa do nó número 1. Considerando que o nó número 3 é móvel este pode aproximar-se do nó número 2.

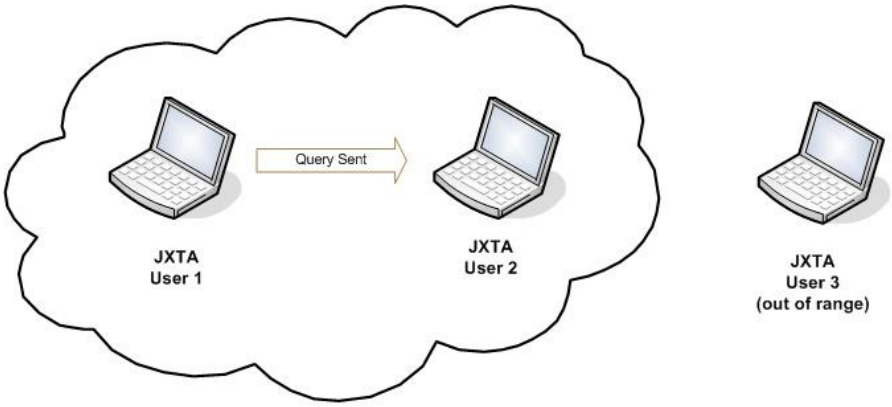

Figura 4.1 – Recepção de uma pesquisa (*"query"*)

Na figura 4.2 é possível visualizar a sequência de acontecimentos quando o nó número 3 entra no alcance rádio do nó número 2.

O primeiro passo é o nó número 2 detectar que um novo vizinho entrou no alcance de rádio, através da recepção de um *beacon*. Como nas redes MANET existe instabilidade, um vizinho pode entrar e sair do alcance várias vezes. Logo não deve ser sempre encarado como um novo vizinho. Se sempre que um nó fosse detectado ocorresse uma propagação das pesquisas, isto poderia originar uma congestão no meio de transmissão. Desta forma, quando se detecta um vizinho é feita uma verificação para ver ele não existe na lista. Para evitar a situação dum vizinho instável (entra e sai de alcance constantemente), é associado ao identificador deste um tempo de vida. Se

um vizinho ficou para além do tempo de vida associado fora do alcance de rádio, quando surgir novamente passa a ser considerado novo vizinho.

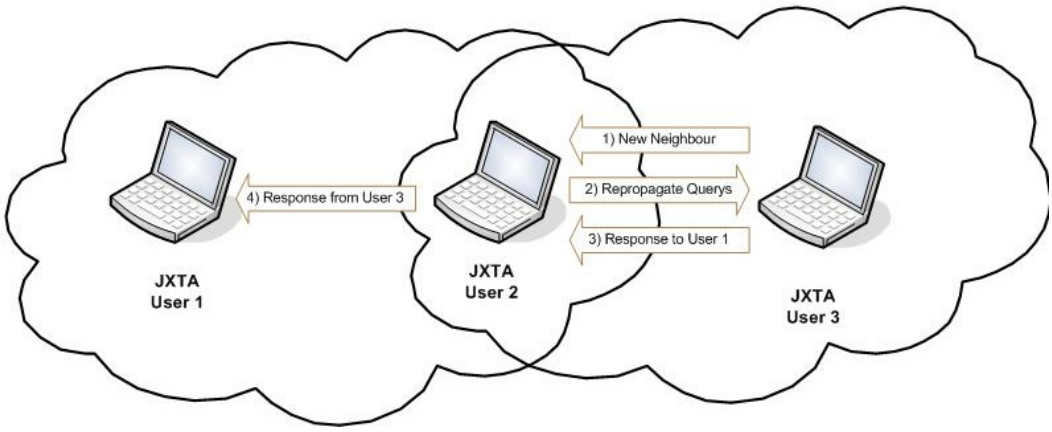

Figura 4.2 – Repropagação das pesquisas.

No caso de estar na presença de um novo vizinho, são validadas e propagadas as pesquisas presentes na *"cache"* do nó número 2 para o nó número 3. A verificação das pesquisas antes de as propagar é muito importante porque existe a probabilidade de existirem pesquisas que já não estejam válidas. Repropagar apenas as pesquisas válidas é bastante útil para diminuir os recursos consumidos.

Ao receber as pesquisas o nó poderá responder caso possua uma resposta válida para fornecer. Mesmo que este nó possua a resposta à pesquisa, a pesquisa é armazenada e poderá ser propagada futuramente por este nó para um novo vizinho, por exemplo, se um quarto nó entrar no alcance rádio.

Existe a possibilidade de o nó número 2 fazer a ligação entre ambos os nós números 1 e 3. Este mecanismo de encaminhamento faz parte da plataforma JXTA. Este protocolo apenas é responsável por armazenar, validar e propagar as pesquisas que recebe. Contudo ao fazer a propagação das pesquisas este protocolo permite a construção de uma rota válida entre o nó número 1 e 3, isto porque quando uma pesquisa é propagada, esta contém todos os nós por onde passou. Quando for enviada a resposta à pesquisa, esta rota é utilizada no percurso de retorno.

4.1.2 O serviço "ManetService"

Como já foi referido a plataforma JXTA possui vários serviços, contudo estes serviços foram sempre desenvolvidos para redes com fios. No decorrer do desenvolvimento deste projecto surgiu a ideia de criar um serviço dentro da plataforma JXTA que fosse dedicado exclusivamente às redes MANET.

Este novo serviço tem como objectivo agrupar todas as alterações feitas à plataforma dedicadas a redes MANET. O serviço possui uma interface para as aplicações (API) comuns. Além desta interface cada módulo deste serviço tem a sua interface própria.

Os serviços na plataforma JXTA estão implementados através de módulos. Estes módulos são inicializados pela plataforma no arranque. Depois cada módulo tem de ser posto em funcionamento. Esta tarefa é feita pela plataforma automaticamente, no arranque, ou quando o módulo faz parte do anúncio dum grupo.

Para ser possível adicionar um novo módulo à plataforma JXTA é necessário analisar o funcionamento dos módulos na plataforma assim como a sua implementação.

4.1.2.1 Módulos na plataforma JXTA

Ao iniciar a plataforma JXTA é criado o grupo *PeerGroup*. Neste grupo são inicializados todos os módulos que estejam descritos na configuração da plataforma, armazenada no ficheiro *PlatformConfig* e na classe interface da API *PeerGroup*.

Cada módulo possui uma função de *init()* onde é feita a sua inicialização, ou seja, é nesta função que estão os procedimentos de preparação do módulo. Outras funções existentes são a *startApp()* e *stopApp()*. A primeira contém os procedimentos iniciais para colocar o módulo em funcionamento, e antes de ser chamada é sempre necessário fazer a inicialização. A função *stopApp()* serve para retirar o módulo de funcionamento garantindo que os recursos que este utiliza são libertados. Os módulos podem ser postos em funcionamento e retirados diversas vezes, mas só podem ser inicializados uma única vez.

Durante o arranque da plataforma é criado o primeiro grupo, o *"WorldPeerGroup"*. Este grupo não é um grupo da rede, mas apenas serve como modelo para permitir à plataforma proceder às configurações necessárias para o nó se ligar à rede JXTA. Um dos passos do arranque é garantir que todos os módulos bem conhecidos da plataforma são inicializados.

Assim que o nó entra na rede JXTA é feita a associação ao grupo por defeito, o *"netPeerGroup"*. Este grupo é o ponto de partida para todos os restantes grupos que possam existir. Este grupo contém os módulos bem conhecidos da plataforma. Todos os grupos que derivem deste possuem pelo menos estes módulos, mas outros podem ser adicionados ao novo grupo.

Com a criação do *"netPeerGroup"* são postos em funcionamento os módulos pré-definidos. Quando é feita a associação a um novo grupo alguns módulos têm de ser postos em funcionamento para esse grupo como é o caso do serviço de pesquisas (*"ResolverService"*).

4.1.2.2 Módulo "ManetService"

Este módulo foi implementado com o objectivo de agrupar todos os módulos de suporte às redes MANET. Como anteriormente foi referido, na plataforma JXTA os serviços são implementados por módulos. No caso deste novo módulo trata-se de um conjunto de serviços.

Para melhorar a integração na plataforma JXTA foram implementadas duas interfaces: uma no núcleo da plataforma e outra destinada às aplicações. A interface das aplicações tem como objectivo permitir a estas aceder a funcionalidades fornecidas pelos serviços existentes neste módulo.

O primeiro serviço a ser integrado neste módulo foi o serviço de *beacon*. Este serviço fazia parte da plataforma JXTA anteriormente alterada [Nelson05].

Este serviço tem como objectivo diferenciar os nós estáveis dos nós instáveis e seleccionar um subconjunto onde existe um nó líder. O mecanismo base deste serviço é o envio periódico de mensagens de *beacon* (farol). Com as informações contidas nas mensagens de *beacon* os nós agrupam-se em grupos e seleccionam um líder para esse grupo.

Para minimizar a carga deste serviço as mensagens contêm poucos dados. Existe uma diferenciação no tratamento dos nós estáveis para o tratamento dos nós instáveis.

Cabe a cada nó eleger um líder de grupo, para esta escolha o nó armazena numa tabela a estabilidade dos seus nós vizinhos. Na mensagem de *beacon* é incluído um campo com o identificador do nó escolhido como líder.

Neste serviço foram feitas algumas alterações para permitir a integração do novo serviço de gestão de pesquisas. A primeira alteração foi a introdução da interface do novo serviço de gestão de pesquisas. Através desta interface o serviço de *beacon* assim que detecta um novo vizinho envia os dados referentes a este para o serviço de pesquisas.

No serviço de *beacon* inicial estava incluído um mecanismo de repropagação não selectiva de mensagens, ou seja, todas as mensagens eram repropagadas. Com a introdução do novo serviço de gestão de pesquisas esta repropagação não é necessária. O outro serviço incluído no módulo *ManetService* é o responsável pela gestão de pesquisas. Nas secções seguintes é explicada a estrutura deste serviço e o seu modo de funcionamento.

4.1.3 Classe ResolverCache

A implementação do protocolo de gestão de pesquisas é feita pela classe ResolverCache, representada na figura 4.3.

Antes de começar a armazenar as pesquisas (*"querys"*) é preciso receber um aviso de que um novo nó entrou no alcance. Para conseguir isto modificou-se a plataforma anterior [Nelson05] de modo a incluir um apontador para a interface deste novo serviço. Assim, quando um novo nó é detectado pelo serviço os seus dados são passados para a interface da classe ResolverCache. Com estes dados recebidos, a função *processNewNeighbour* cria um novo item da tabela *"QueryTable"*, sendo utilizado o identificador do nó (*"PeerID"*) como apontador para garantir a unicidade nesta tabela. Cada item é constituído por três campos: o primeiro é o nome do nó que foi detectado, o seguinte é o campo que indica a última vez que foi recebida uma pesquisa do nó e finalmente o último campo que é uma tabela de todas as pesquisas recebidas efectuadas pelo nó.

Uma vez identificado o novo nó, é possível começar a armazenar as pesquisas efectuadas por este. Caso surja uma pesquisa dum nó desconhecido, primeiro é adicionado como novo nó e em seguida adicionada a pesquisa à sua tabela. Os elementos da tabela de pesquisas (*"QueryTable"*) que existem para cada nó são indexados pelo identificador da pesquisa que tem de ser único. Dentro da plataforma os vários serviços que utilizam o mecanismo de pesquisas garantem a unicidade deste identificador. As aplicações que necessitem do serviço de gestão de pesquisas terão de garantir a unicidade, caso contrário é armazenada sempre a pesquisa mais recente descartando a anterior. Cada elemento é constituído por seis campos:

- Identificador do grupo (QIGroup) – Este campo serve para registar qual é o grupo onde foi efectuada a pesquisa. Sem este, não era possível propagar correctamente as pesquisas armazenadas.

- Identificador de serviço (QIHandlerName) – Para permitir um maior controlo sobre o que pesquisas que são armazenadas é utilizado este campo. Na interface gráfica deste serviço é exibido este identificador, e além disso é possível definir que as pesquisas do serviço associado a este identificador não sejam armazenadas.

- Tempo de vida (QIttl) – Especifica o tempo de vida da pesquisa. Este campo é sempre verificado antes que seja propagada a pesquisa. Caso o seu tempo tenha expirado esta é descartada. O valor deste campo é um inteiro que representa o número de segundos.

- Registo temporal (QItimeStamp) – Para determinar se a pesquisa ainda é valida é necessário saber o tempo que esta foi armazenada na tabela. É essa informação que é registada neste campo.

- Mensagem (QImsg) – Este campo é guardado apenas para permitir a propagação. O serviço Resolver processa e propaga as mensagens. Na mensagem estão contidos todos os campos da mensagem original incluindo os campos associados ao serviço de *"endpoint"*.

- Pesquisa (QIquery) – Este campo contém a pesquisa originalmente recebida. É armazenado de forma a permitir a repropagação.

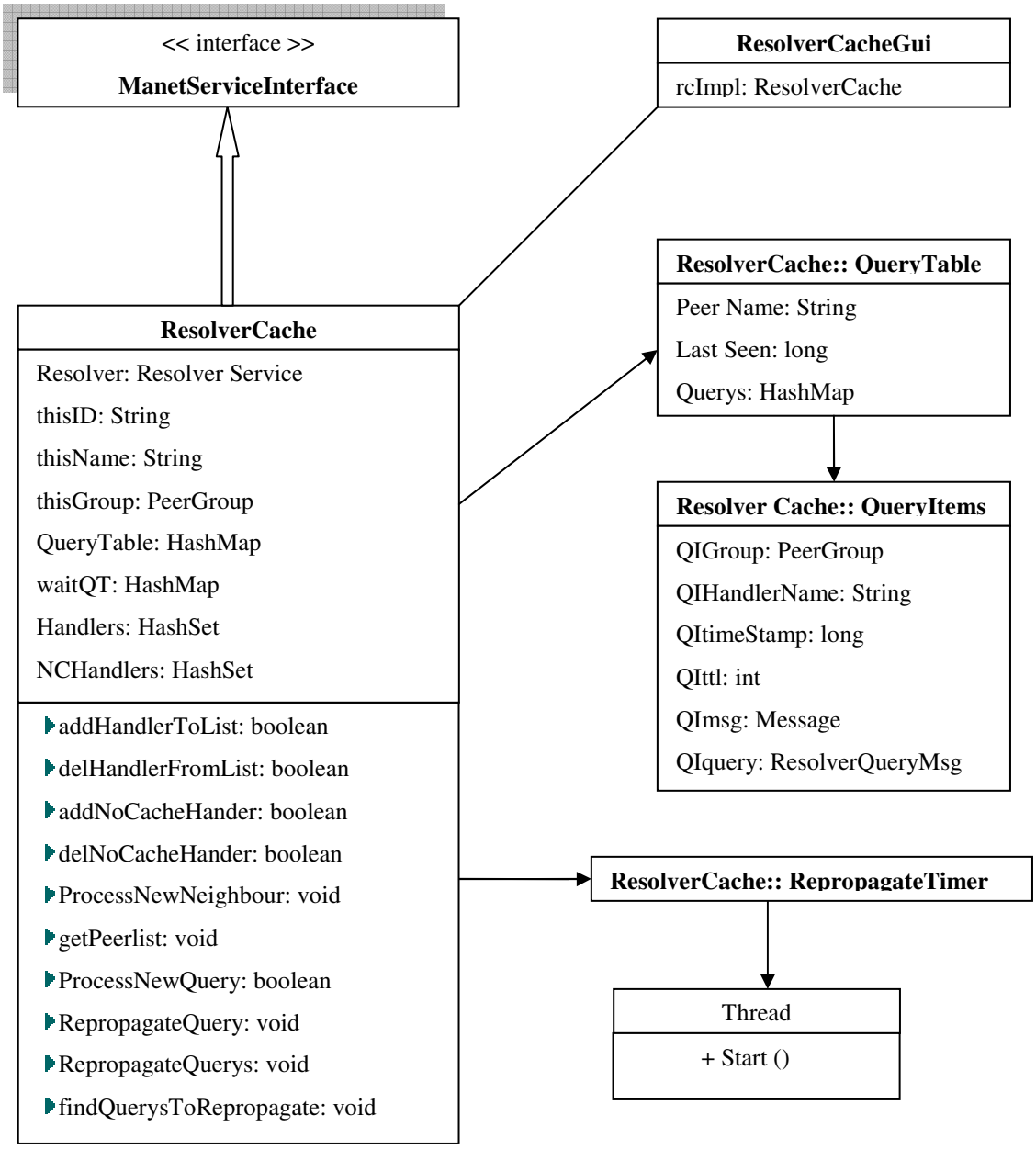

Figura 4.3 – Diagrama de classes do serviço de cache de pesquisas.

Para permitir observar e configurar o comportamento desta classe foi implementada uma interface gráfica *ResolverCacheGui*, composta por dois painéis.

O primeiro painel está representado na figura 4.4. Este painel é composto por duas tabelas, onde a primeira indica os nós conhecidos. Cada elemento desta tabela é

composto por três campos: o primeiro é o identificador do nó (*"PeerID"*), o nome do nó caso seja conhecido. Esta informação é fornecida pelo serviço de beacons. Caso o nó esteja a mais de um salto de distância não é possível obter o nome. O último campo indica o número de pesquisas recebidas do nó em questão.

A segunda tabela contém informação específica de cada nó. Ao seleccionar um elemento na primeira tabela é alterado o conteúdo da segunda. Cada elemento desta tabela é composto por quatro campos: o primeiro contém o identificador da pesquisa (*"QueryID"*), no campo seguinte é exibido o nome do serviço à qual esta pesquisa se destina. Caso o serviço não esteja associado a nenhum nome é mostrado o identificador do serviço. Ainda existe um campo de texto onde é exibido o registo dos acontecimentos como, por exemplo, se um nó foi detectado ou as pesquisas foram propagados. Finalmente três botões: um para forçar a propagação das pesquisas, outro para exibir no campo de texto a pesquisa seleccionada e o último limpa o campo de texto.

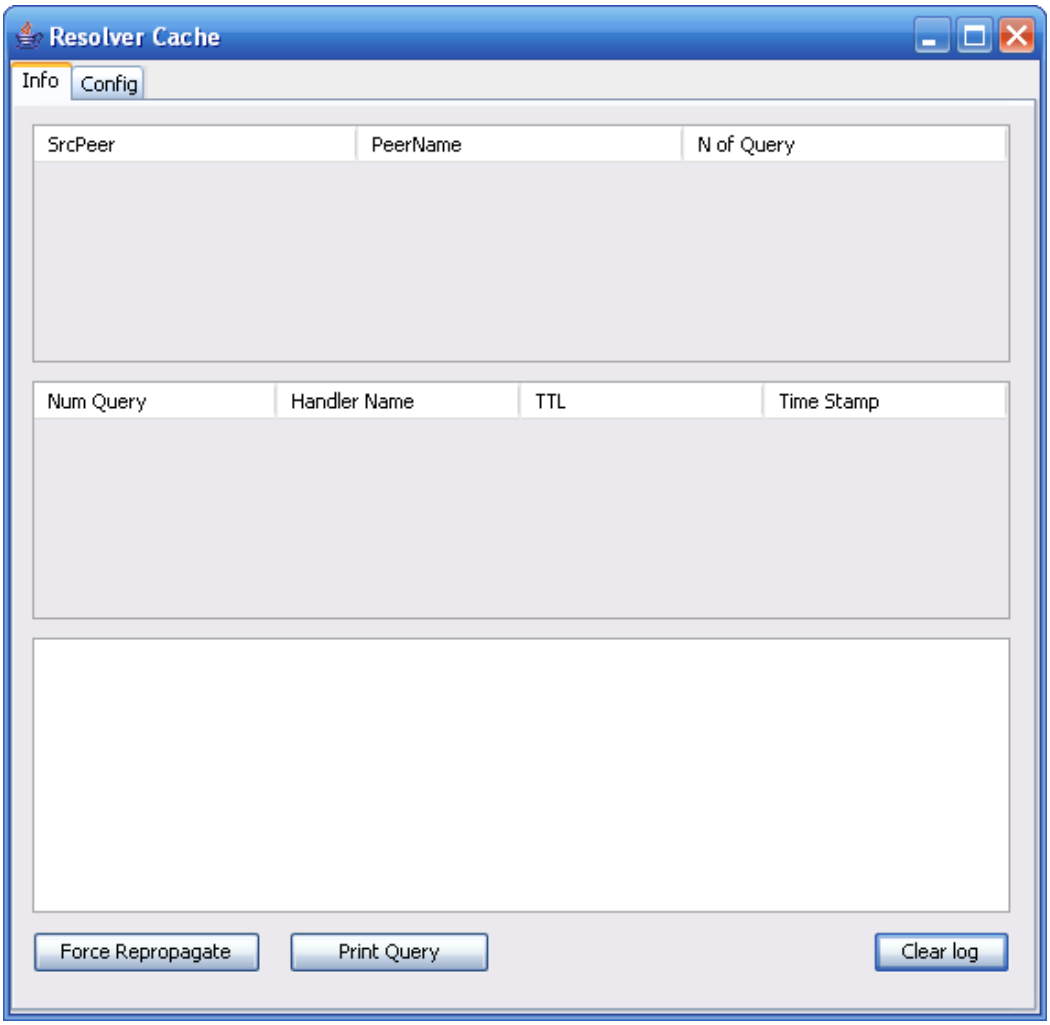

Figura 4.4 – Painel informativo do Resolver Cache.

O segundo painel disponível nesta interface gráfica é o que permite configurar os parâmetros deste serviço. No campo *"Handlers"* é possível visualizar duas listas: a lista da esquerda contém os *"Handlers"* que são armazenados; enquanto a lista da direita contém os que são ignorados pelo serviço de gestão de pesquisas.

Existe ainda a hipótese de definir o tempo de espera mínimo para a repropagação, assim como o número máximo de pesquisas que são armazenadas por nó.

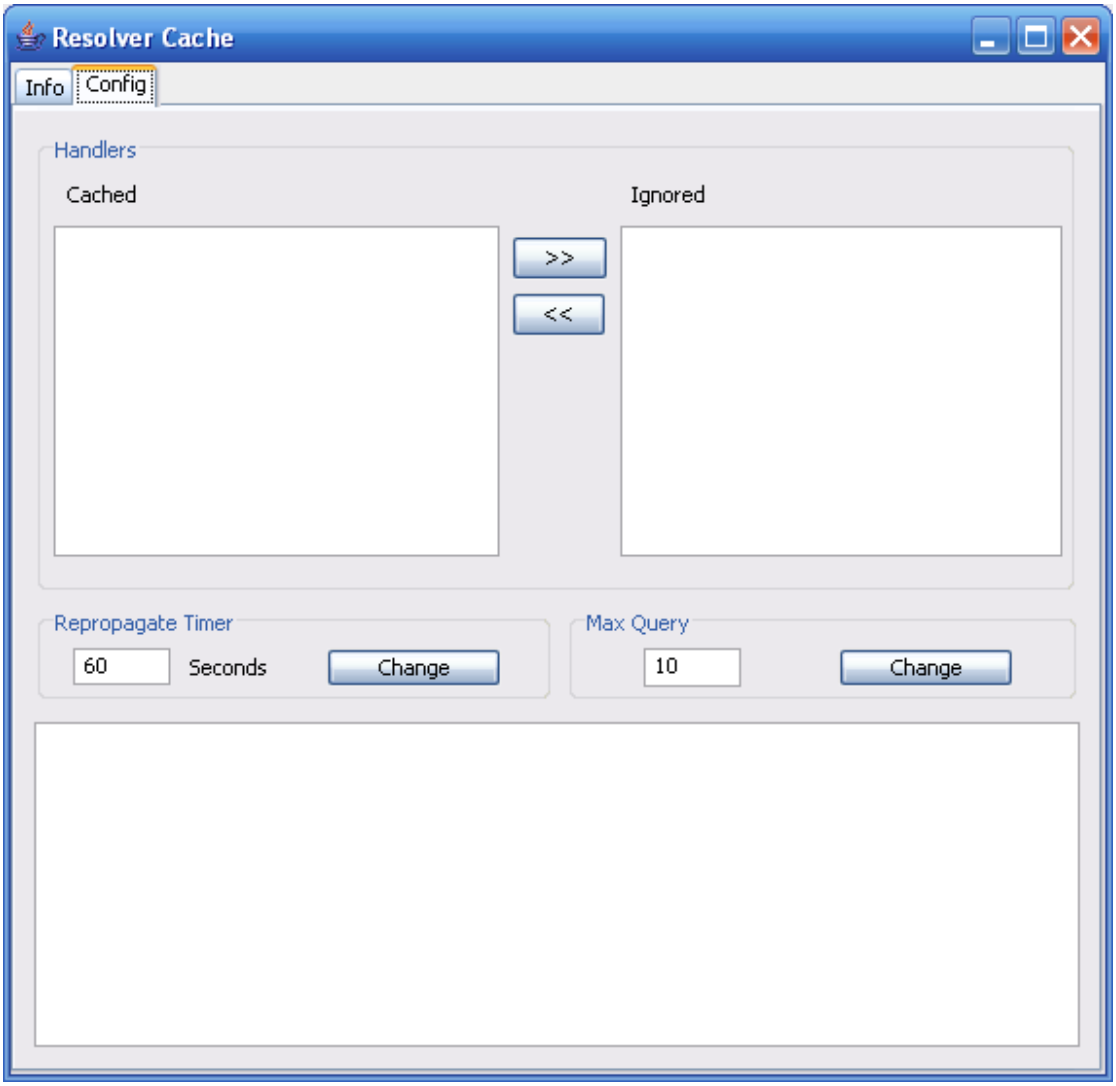

Figura 4.5 – Painel de configuração do Resolver Cache.

4.2 Módulo de Transporte

Como foi anteriormente referido o módulo de transporte da plataforma JXTA não é viável para utilização em redes MANET, devido ao excesso de tamanho resultante de codificar tudo em XML. Para substituir o módulo de transporte da plataforma JXTA foi implementado um novo módulo independente da plataforma.

No desenho deste módulo foi tida em conta a estrutura já implementada na plataforma. Manter a especificação era essencial para facilitar a implementação da aplicação. Ou seja, se o funcionamento do novo módulo implementado e do módulo existente na plataforma fosse semelhante as alterações à estrutura da aplicação seriam mínimas.

De forma a minimizar os dados transmitidos reduziu-se as mensagens ao mínimo, ou seja, a informação extra que a plataforma JXTA adiciona a cada mensagem transmitida foi retirada.

Cada ligação está identificada com um identificador único, construído com o endereço IP e a porta de escuta do destino. As ligações disponíveis são armazenadas numa variável. Sempre que é feito um pedido de estabelecimento de ligação é feita uma validação se já existe uma ligação para o destino pretendido.

As ligações são reaproveitadas sempre que possível. No caso da ligação terminar, ou se não for utilizada há algum tempo, esta é removida das ligações disponíveis.

Uma ligação removida não impossibilita futuros restabelecimentos da mesma. O restabelecimento da ligação é encarado como uma nova ligação. Os dados das ligações são descartados quando estas são eliminadas.

Cada aplicação que pretenda utilizar o módulo tem de conhecer o endereço IP e a porta de escuta do receptor. Este módulo apenas gere as ligações estabelecidas e a comunicação entre os nós. Ao se usar este módulo perde-se parte da flexibilidade que existe nos identificadores de ligações que existem no mecanismo de portos do JXTA – uma ligação só persiste desde que o endereço IP da interface de rede não mude. Desta forma, sacrifica-se flexibilidade para ter melhor desempenho em ligações de curta-duração.

4.2.1 Estrutura do módulo

A constituição do módulo é representada na seguinte figura:

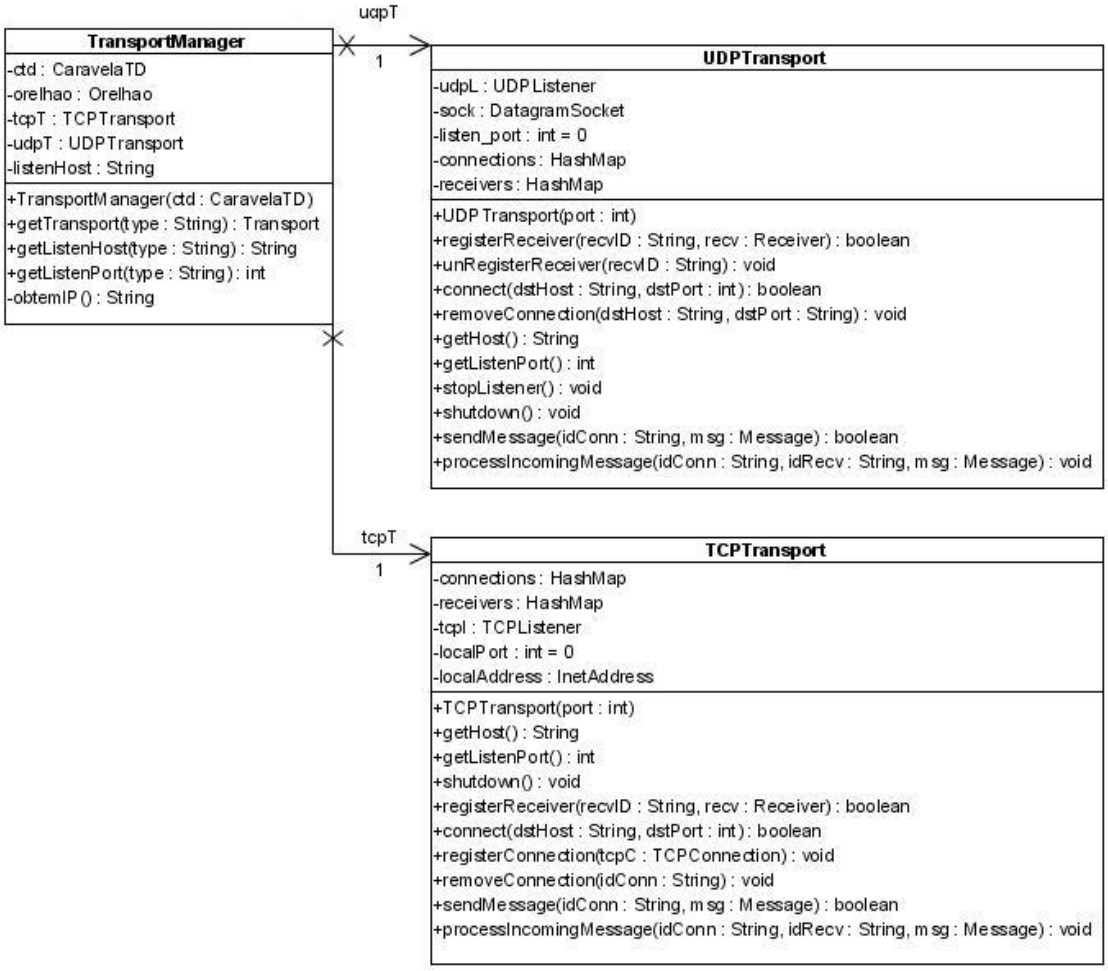

Figura 4.6 – Diagrama do módulo de Transporte

Dois tipos de transporte foram implementados para este módulo: o TCP e UDP. Estes módulos implementam uma interface "*Transport*", ou seja, necessitam de possuir as seguintes funcionalidades:

- *"connect"* – Permite estabelecer uma ligação com um nó de destino. Os parâmetros necessários são o endereço IP e a porta do destino. Antes de estabelecer ligação verifica se já foi estabelecida uma ligação.

- *"registerReceiver"* – Regista um processo *"Receiver"*, capaz de processar as mensagens recebidas. Cada mensagem possui um identificador que determina a que *"Receiver"* a mensagem deve ser associada. Os parâmetros desta função são o identificador e o *"Receiver"*.

- *"sendMessage"* – Envia mensagem para a ligação especificada. As ligações estão identificadas com um identificador único. Caso não exista nenhuma ligação devolve uma falha no envio de mensagem. Antes de tentar enviar a mensagem é necessário estabelecer uma ligação com o nó de destino.

- *"processIncomingMessage"* – Quando uma ligação recebe uma mensagem reencaminha para o transporte para processamento. Cabe ao transporte determinar se existe na lista de *"Receivers"* algum responsável pela mensagem recebida. Senão existir nenhum destinatário a mensagem é descartada.

4.2.1.1 Transporte UDP

Implementar este tipo de transporte requer uma estrutura simples, devido ao facto das comunicações UDP não serem orientadas à ligação.

Para manter a estrutura dos tipos de transporte semelhante, foi criado o conceito de ligação virtual neste módulo.

A utilização de UDP para comunicações tem uma característica que deve ter em conta: as mensagens com recurso a este tipo de transporte podem ser perdidas. No início da comunicação as mensagens são enviadas ao ritmo mais alto. Caso exista congestionamento a perda de mensagens pode ocorrer.

4.2.1.2 Transporte TCP

A implementação deste tipo de transporte é mais elaborada. O facto de ser orientado à ligação possibilitou uma adaptação melhor com a estrutura do módulo idealizada. O tipo de transporte TCP possui ligações reais, ou seja, é estabelecida ligação entre os dois nós. Essa ligação pode ser utilizada em ambos os sentidos. Assim não é necessário estabelecer duas ligações para uma transferência de ficheiros, por exemplo.

A desvantagem deste tipo de comunicação é que estão sujeitas aos mecanismos de controlo de congestão do protocolo TCP, que têm mau desempenho para redes sem fios [Tian05]. Em caso de congestionamento ou perda de pacotes o débito é reduzido, mas não se perdem mensagens.

4.2.2 Classe "*Message*"

Os nós comunicam entre si através de mensagens de uma classe, que estende a classe "*Message*".

Para simplificar e reduzir o número de dados transmitidos apenas foi considerado um único campo na classe "Message": um identificador da mensagem (*"id"*). Este campo deve ser utilizado para identificar a quem se destina a mensagem ou que processo está associado a esta mensagem.

A classe *"Message"* é extensível, sendo possível implementar novas subclasses que herdam as características da classe base. Desta forma é possível construir mensagens mais elaboradas.

Para apoio à aplicação desenvolvida foram implementadas três classes de mensagens orientadas para a transferência de ficheiros, apresentadas mais à frente: a classe *"WelcomeMessage"* para a negociação inicial, e as classes *"RequestMessage"* e *"TransferMessage"* para pedir e transferir ficheiros. Estas três classes são um bom exemplo de como é possível utilizar a estrutura existente no módulo de transporte.

Como referido anteriormente, o módulo de transporte transmite objectos *"Message"*. A classe *"Message"* é *"Serializable"*, ou seja, é possível transformá-la

num conjunto de octetos e o receptor executar a operação inversa. Todas as classes que derivem desta têm de garantir estas propriedades. Esta propriedade é essencial para que os dados sejam correctamente transmitidos pelo módulo de transporte.

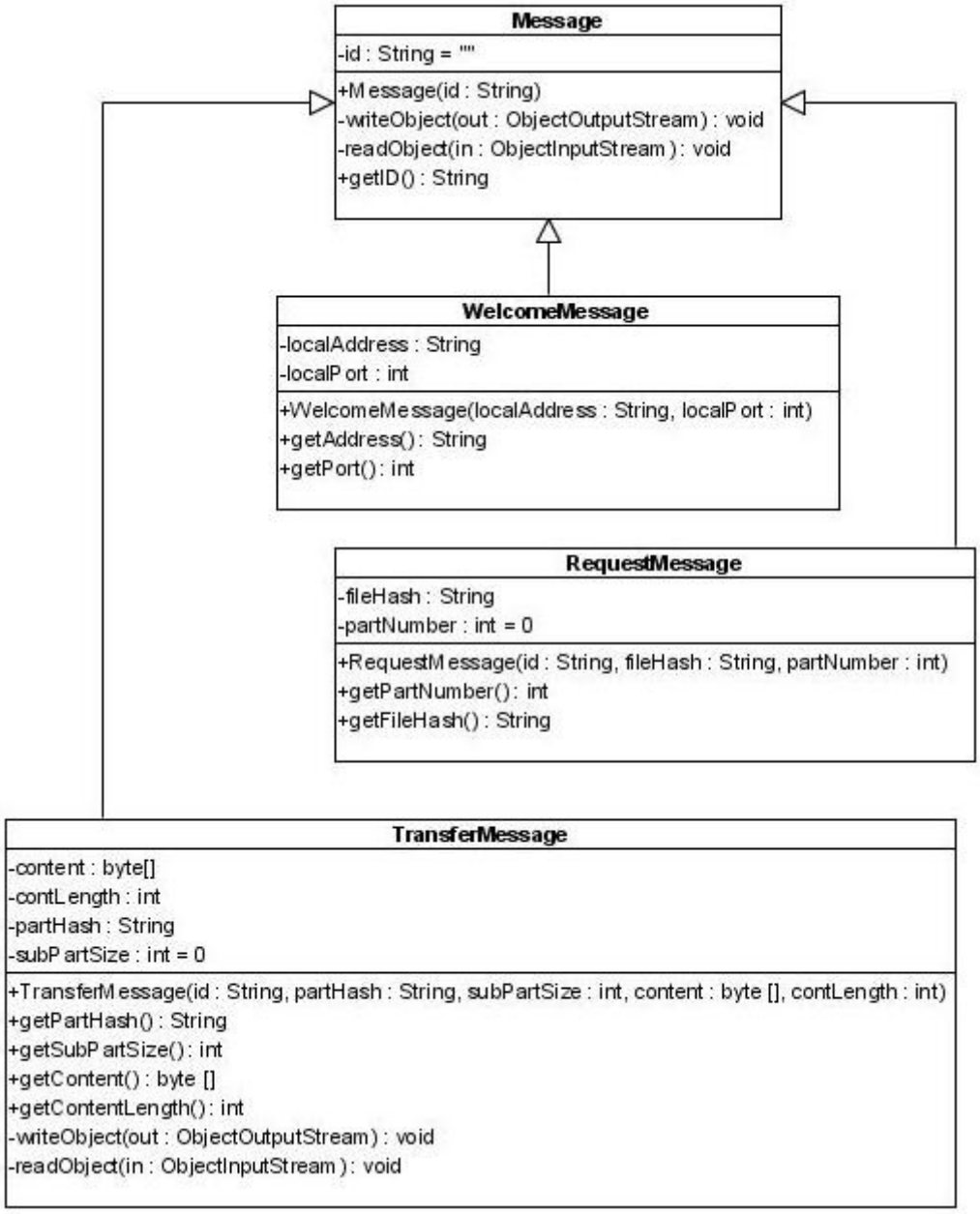

Figura 4.7 – Diagrama dos tipos de mensagem.

4.2.2.1 Classe *"Welcome Message"*

Esta mensagem foi criada para permitir a ambas as partes acordarem os parâmetros da ligação.

Para a estrutura do módulo apenas foram necessários dois campos. O primeiro é um conjunto de caracteres representando o endereço IP de quem enviou a mensagem. No campo seguinte é a porta que o emissor da mensagem está a escuta.

4.2.2.2 Classe *"Request Message"*

A classe *"Request Message"* é um dos exemplos como é possível aproveitar a estrutura existente. Foi criada tendo em conta a transferência de ficheiros, para realizar pedidos de partes dos ficheiros.

Esta classe estende a classe *"message"*. Além do identificador da mensagem adiciona mais dois campos adicionais: o identificador do pedido e o *"part number"*. O identificador do pedido está associado directamente a um identificador único do ficheiro. O campo *"part number"* representa que parte do ficheiro está a ser pedida.

4.2.2.3 Classe *"Transfer Message"*

A classe *Transfer Message* é orientada para as transferências de ficheiros. Este tipo de mensagem possui os seguintes campos:

- Identificador da parte do ficheiro que está a ser transmitida, que contém o seu valor*"MD5 Hash"*.
- Tamanho de cada parte que o ficheiro está dividido.
- Tamanho do bloco de dados.
- Bloco de dados do ficheiro.

Nesta classe mensagem foram implementados métodos de leitura e escrita da mensagem.

4.3 Aplicação Desenvolvida

Para demonstrar o novo serviço e o módulo de transporte foi idealizada e implementada em Java uma aplicação que exemplifica e aproveita as novas funcionalidades adicionadas à plataforma JXTA. Também exemplifica como se pode tirar partido do módulo de transporte implementado. Além de servir para demonstrar, tem também como objectivo ser um exemplo para aplicações futuras que utilizem o novo módulo.

A aplicação desenvolvida possui as seguintes funcionalidades:

- Serviço de gestão de grupos JXTA (criar, juntar, sair)
- Serviço de pesquisas parametrizadas de ficheiros em JXTA
- Partilha e transferência de ficheiros com verificação de integridade com recurso ao MD5.

No decorrer do desenvolvimento desta aplicação foram tomadas algumas opções, tendo sempre em conta que a prioridade era demonstrar o que tinha sido implementado no núcleo da plataforma JXTA. Assim, a parte das pesquisas foi mais cuidada que a parte das transferências. A interface gráfica foi simplificada com a utilização de cinco painéis.

- Painel JXTA (*"JXTA"* - Figura 4.8)
- Painel das Pesquisas (*"Search"* - Figura 4.9)
- Painel das Transferências (*"Transfers"* - Figura 4.12)
- Painel dos Ficheiros Partilhados (*"Files"* - Figura 4.13)
- Painel de *Logs* (Apenas utilizado para depuração)

Os primeiros quatro painéis são descritos ao pormenor mais à frente. O quinto painel contém os diversos registos sobre as operações executadas, erros ocorridos no programa, e avisos de sucesso.

Ao arrancar a aplicação é feita a inicialização da plataforma JXTA, originando assim um nó na rede JXTA. Se for a primeira vez que é inicializada a aplicação, a plataforma ainda não está configurada. Para a configurar esta aparece uma janela de configuração onde é escolhido o nome, palavra de passe, e tipos de transporte (TCP, UDP ou HTTP).

A configuração apenas é feita no primeiro arranque da aplicação. As definições são armazenadas para não voltar a ser necessário configurar. Caso se pretenda alterar parte da configuração existe um botão no painel "JXTA" que quando pressionado altera um registo na plataforma JXTA, indicando que da próxima vez que esta for inicializada deve ser configurado.

Para aceder a esse painel de configuração é necessário reiniciar a aplicação desenvolvida.

4.3.1 Gestão de Grupos na plataforma JXTA

O primeiro painel disponível é referente à plataforma: antes de ser possível utilizar os restantes painéis é necessário criar um novo grupo ou juntar a um existente. Este painel tem dois modos de funcionamento: um para pesquisar e juntar a um grupo, outro para criar.

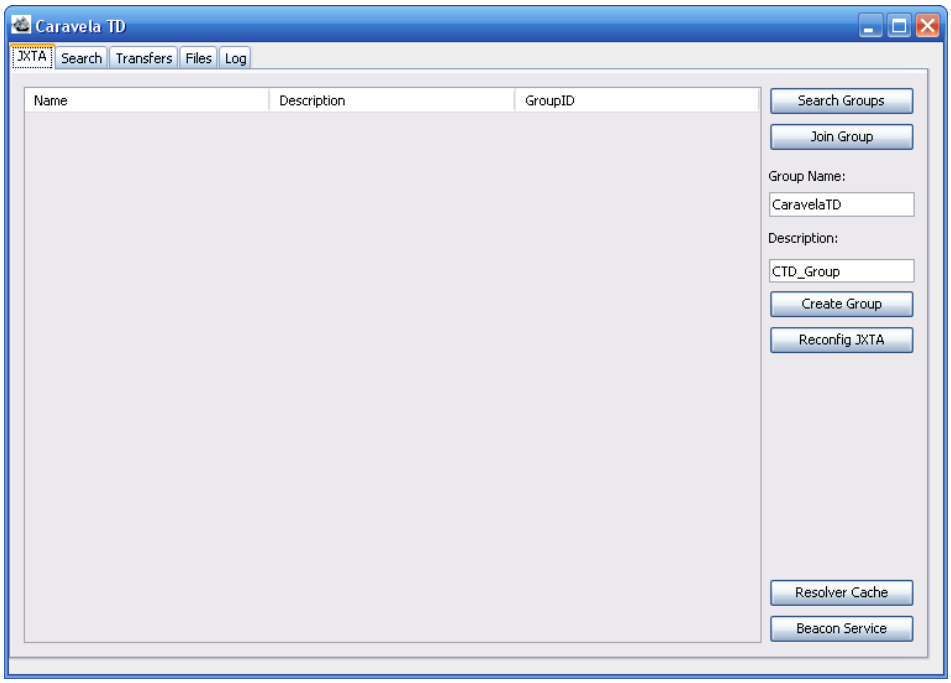

Figura 4.8 – Painel JXTA

No caso de se realizar uma pesquisa por grupos existentes na rede JXTA é necessário utilizar o botão *"Search Groups"*. Esta pesquisa é efectuada recorrendo a um serviço disponibilizado na plataforma JXTA, o *"Discovery Service"*. Este serviço faz um pedido remoto por anúncios, e neste caso é feito um pedido à rede por anúncios de grupos de nós ("*Peergroups"*).

Os anúncios recebidos são armazenados localmente de forma a permitir a sua reutilização mais tarde, ou seja, quando for efectuada uma pesquisa idêntica. Se a aplicação for encerrada este registo não é eliminado. Assim, quando for iniciada a aplicação estes anúncios armazenados são reaproveitados. Este tipo de conceito permite que um grupo que seja regularmente utilizado possua o seu anúncio armazenado localmente, evitando ocupar a largura de banda disponível com nova pesquisa.

Em seguida, selecciona-se um elemento da lista com os grupos descobertos, e com o botão *"Join Group"* o nó fica associado a este. Para um nó pertencer a um grupo terá de cumprir os requisitos especificados no anúncio do grupo.

Caso o objectivo seja criar um grupo, é necessário preencher as duas caixas de texto que correspondem ao nome do grupo e a sua descrição. Com o botão *"Create Group"* é feita a criação do grupo. Para que isto aconteça, a aplicação cria e publica na rede JXTA, e no registo local, um anúncio que representa este novo grupo. Após a criação do anúncio a aplicação associa-se a esse grupo do mesmo modo que anteriormente, para o caso de se juntar a um grupo que tenha sido descoberto.

Neste painel existem dois botões ligados aos serviços adicionados à plataforma JXTA. O primeiro, *"Beacon Service"* foi implementado num projecto anterior [Nelson05]. Enquanto o segundo *"Resolver Cache"* é o novo que se pretende demonstrar.

Os botões apenas permitem mostrar ou ocultar os respectivos painéis desses serviços. No arranque da plataforma JXTA as janelas dos serviços não estão visíveis, mas com esta funcionalidade é possível mostrá-los.

Existe ainda um botão (*"Reconfig JXTA"*) que permite reconfigurar a plataforma JXTA. Como referido anteriormente esta opção só é activada após reiniciar a aplicação.

4.3.2 Pesquisas

Outra funcionalidade da aplicação implementada, talvez a mais sonante, é a possibilidade de realizar pesquisas parametrizadas de ficheiros. Devido ao modo de funcionamento das pesquisas sobre a plataforma, estas estão limitadas aos nós que pertencem ao mesmo grupo da rede JXTA. Com isto, se um nó partilha ficheiros, estes apenas são visíveis para os nós do seu grupo.

Na figura 4.7 é possível visualizar a interface gráfica onde é possível realizar pesquisas. O primeiro passo quando se pretende pesquisar é preencher a caixa de texto com o nome do ficheiro (ou parte do nome do ficheiro) e se necessário definir os parâmetros disponíveis: tipo de extensão, tamanho mínimo e máximo e tempo de validade desta (*"Best Before"*).

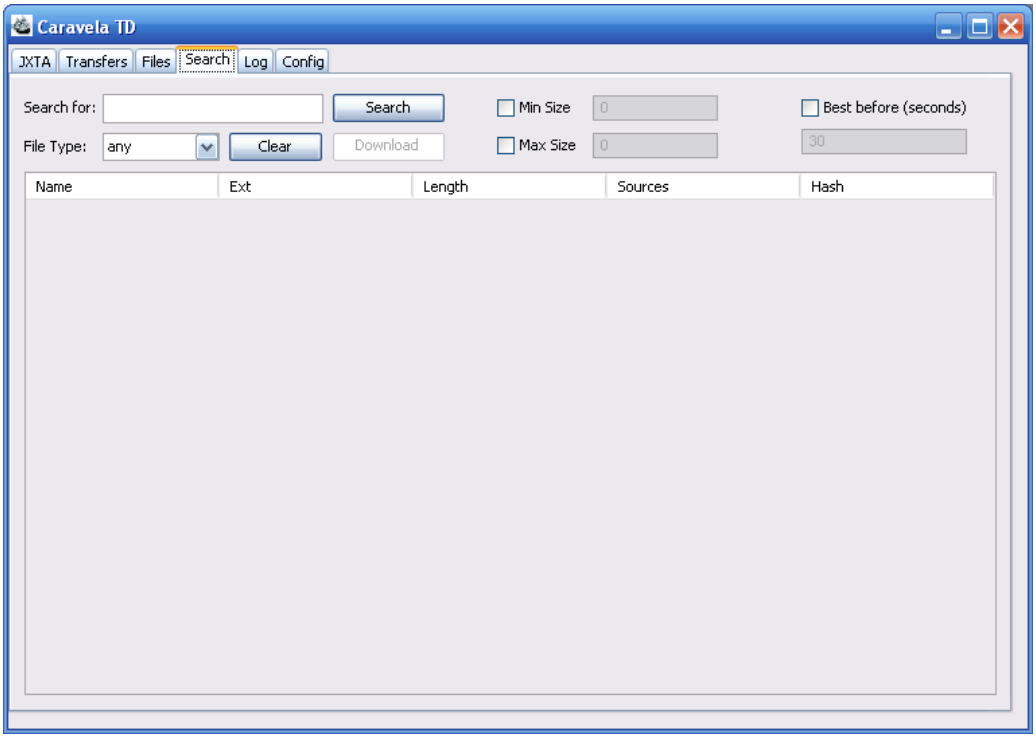

Figura 4.9 – Painel das Pesquisas

Assim que é premido o botão *"Search"* é construída uma *"Query"* com os parâmetros definidos na procura e enviada para o grupo a que o nó pertence. Estas pesquisas estão construídas de modo a tirar proveito do novo serviço implementado na plataforma. O tempo de validade é passado como argumento para o serviço, enquanto os restantes parâmetros são destinados à aplicação existente noutro nó.

Significa que cabe ao serviço de gestão das pesquisas a responsabilidade de verificar e rejeitar se o tempo de validade expirou.

Na aplicação foi implementado um *listener* que é responsável por receber as respostas às pesquisas efectuadas. Estas respostas são compostas por anúncios. Como a rede JXTA se baseia em anúncios foi implementado um novo tipo para esta aplicação: o *FileAdvertisement*. Esta classe é composta pelos seguintes elementos: Message Digest 5 (MD5), nome, extensão e tamanho do ficheiro. Para além destes campos é enviado um anúncio do canal (*PipeAdvertisement*) do nó que possui o ficheiro.

A tabela deste painel contém informações sobre o nome do ficheiro encontrado, o seu tamanho, número de fontes para este ficheiro e por fim o seu *hash* MD5. O cálculo das fontes é realizado à custa do valor de *hash* recebido e do número de respostas dos vários nós.

À medida que são recebidas as respostas à pesquisa efectuada, estas surgem na tabela das respostas. Após a apresentação dos resultados é possível iniciar a transferência de um ficheiro.

Para transferir um ficheiro apenas é necessário seleccionar o item da lista e premir o botão *"Download"*. Em seguida é adicionado à lista de transferências o ficheiro seleccionado.

O mecanismo de pesquisas que foi implementado para a aplicação está representado no seguinte esquema:

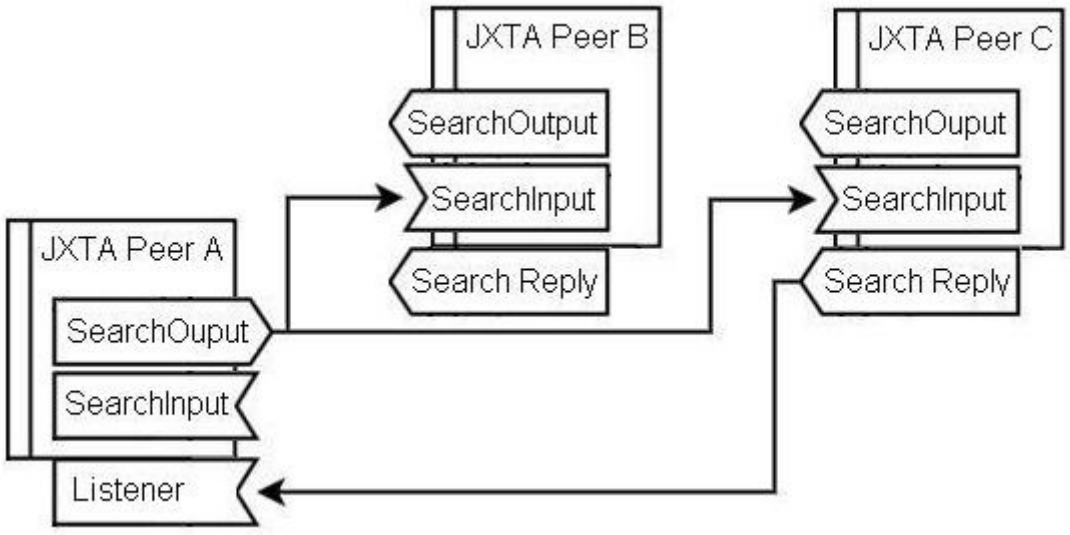

Figura 4.10 – Mecanismo de pesquisa

No esquema anterior é possível visualizar um exemplo envolvendo três nós a correr a aplicação desenvolvida. Para a aplicação não é importante se existe comunicação directa entre os nós, a pesquisa é feita para a rede. A plataforma JXTA é responsável pelo mecanismo de pesquisa.

4.3.3 Transferências de ficheiros

Na aplicação de demonstração foi implementado um mecanismo de transferência de ficheiros entre os nós. Este mecanismo é composto por 3 canais unidireccionais de comunicação, o primeiro é o responsável pelos pedidos, isto significa que todos os nós estão "à escuta" neste canal para detectar novos pedidos de ficheiros. Os restantes dois canais representam uma ligação bidireccional, onde cada canal possui ligação apenas num sentido. Estes são responsáveis pela transferência de ficheiros, um para os dados em si, enquanto o outro é de controlo.

A seguinte figura exemplifica o mecanismo de transferências:

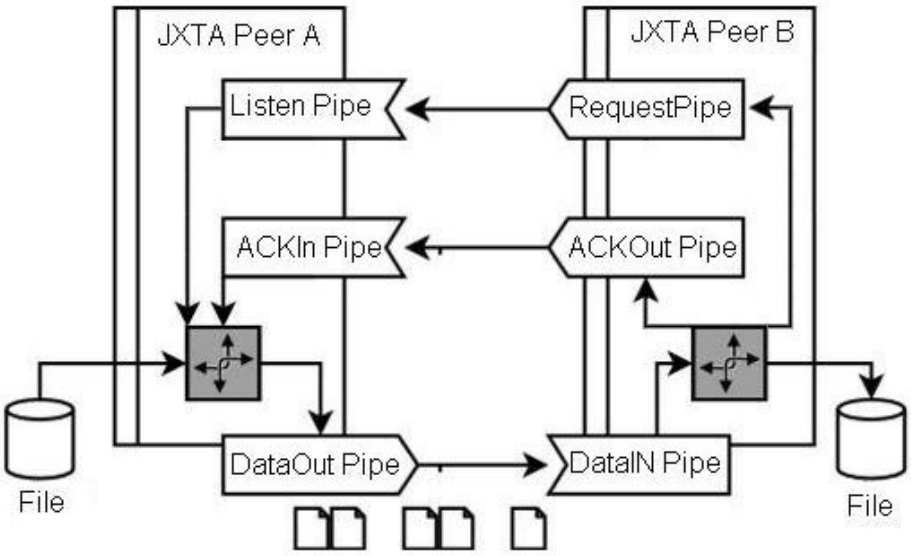

Figura 4.11 – Mecanismo de transferência

Este mecanismo inicialmente era suportado pela plataforma JXTA. Depois de alguns testes verificou-se que era possível melhorar a comunicação nos canais. Na última versão desenvolvida nesta dissertação o mecanismo de transferências mantém

a estrutura inicial, contudo os canais de comunicação utilizam o novo módulo de transporte anteriormente apresentado.

Quando o ficheiro é adicionado à lista aparece no painel da figura 4.12, na lista de *"Downloads"*. Ao iniciar a transferência do ficheiro entra em funcionamento uma classe responsável por gerir as transferências.

Os ficheiros estão divididos em partes, o cálculo do tamanho destas é realizado através da seguinte tabela, definida arbitrariamente através de um conjunto de experiências:

Tamanho da parte	Tamanho do ficheiro
128k	<10485760
256k	<26214400
512k	<52428800
1024k	<104857600
2048k	<209715200
4096k	<419430400
8192k	<838860800
16384k	>=838860800

Tabela 4.1 – Correspondência tamanho do ficheiro com tamanho das partes.

A cada parte está associado um *hash* para verificação da integridade desta. Na primeira mensagem recebida duma parte é fornecido este valor de *hash* MD5. A existência de partes facilita a recuperação de dados: se a recepção de uma parte falhar a verificação é feito um novo pedido apenas para esta parte. A verificação do valor de *hash* MD5 no final da transferência não é necessário porque ao realizar a verificação em todas as partes é garantida a integridade do ficheiro.

A transferência é baseada no envio de confirmações. Sempre que é recebida uma mensagem é enviada uma confirmação. Apesar deste mecanismo produzir mais mensagens, permite um maior controlo das transferências. Por exemplo, o pedido de partes é enviado na confirmação. No caso da confirmação ser igual a -1 significa que o receptor já não deseja pedir mais nenhuma parte desse ficheiro. Caso contrário envia a parte que pretende.

Quando o emissor recebe uma mensagem de confirmação primeiro verifica se o pedido que nela está contido é valido. Caso não seja termina logo a transferência encerrando os canais de comunicação.

Existem sempre dois canais de comunicação para cada transferência, mais um canal comum que permite receber pedidos dos diferentes nós.

Para permitir a transferência simultânea de vários ficheiros é criado uma *thread* para cada transferência. Esta *thread* é responsável por receber os fragmentos do ficheiro, juntá-los, e no fim verificar a integridade do ficheiro.

A outra lista disponível é a lista de ficheiros que estão a ser enviados para outros nós.

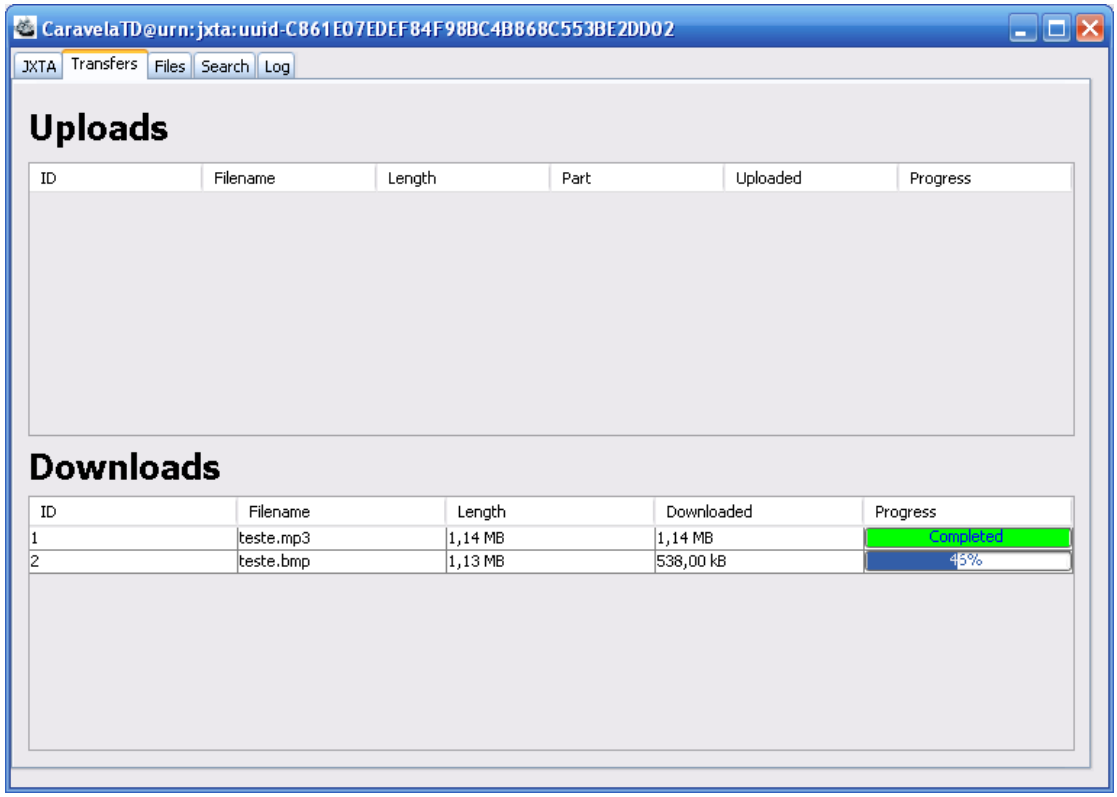

Figura 4.12 – Painel das Transferências

4.3.4 Partilha de Ficheiros

O painel dos ficheiros (representado na figura 4.13) serve para exibir e configurar todos os ficheiros que fazem parte da partilha do nó.

Neste painel existe uma lista à esquerda, onde estão listadas todas as directorias disponíveis no computador. Para uma directoria ficar disponível na partilha é necessário seleccioná-la.

Ao seleccionar uma directoria esta não é automaticamente adicionada à partilha. Para que a aplicação processe as directorias seleccionadas é necessário pressionar o botão *"Update Dirs"*.

Por omissão, se a directoria já foi adicionada anteriormente não é novamente processada. Isto previne a listagem desnecessária de directorias já processadas. Para contornar esta limitação foi incluída a opção *"Force Refresh"*. Com este parâmetro activado todas as directorias seleccionadas são novamente processadas, independentemente se já terem sido adicionadas anteriormente ou não.

Existe uma directoria que é sempre adicionada, a *"./received/"*. Nesta directoria encontram-se armazenados todos os ficheiros recebidos.

Dentro da informação disponível constam o nome do ficheiro, extensão, tamanho e o valor que representa o *hash* MD5 do ficheiro [FastMD5].

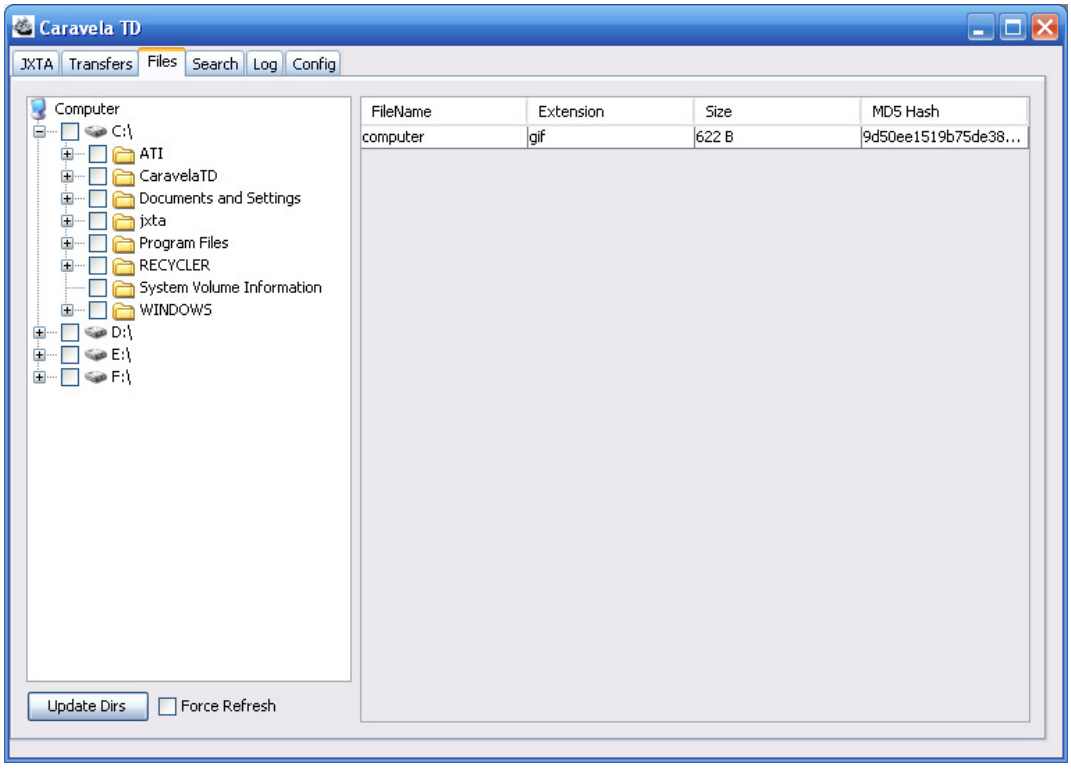

Figura 4.13 – Painel dos Ficheiros Partilhados

Capítulo 5. Resultados

Neste capítulo é feita uma comparação entre os testes realizados para duas situações distintas. No primeiro cenário recorreu-se a plataforma alterada num projecto anterior [Nelson05], no segundo cenário foi utilizada a plataforma desenvolvida neste trabalho, que acrescenta o serviço de gestão de pesquisas à plataforma do primeiro cenário.

Na aplicação realizada foram introduzidas algumas funcionalidades de monitorização da transferência. Os restantes dados foram obtidos com recurso a um analisador de tráfego, o *"Wireshark"* [Wireshark]. Este programa regista todos os octetos que passam pela interface de rede seleccionada. Na versão utilizada existe a possibilidade de filtrar por tipo de tráfego e escolher apenas o tráfego JXTA. No decorrer dos testes verificou-se que após a filtragem dos pacotes observados no canal nem sempre apareciam todos os pacotes pertencentes a transmissão do ficheiro com recurso a rede JXTA. Este problema foi detectado na transmissão dum ficheiro sem compressão, pois foram detectados menos octetos associados ao JXTA que o tamanho do ficheiro.

5.1 Serviço de gestão de pesquisas

O objectivo principal do serviço de gestão de pesquisas é reduzir a troca de mensagens entre nós para manter a informação sobre a rede actualizada. Antes de possuir este serviço, a plataforma tinha de enviar continuamente mensagens de pesquisa que inundam a rede para localizar novos recursos. Ao ter o serviço activo a pesquisa apenas é enviada uma vez, sendo a pesquisa repropagada quando novos vizinhos são detectados. Os nós memorizam as pesquisas durante o tempo de vida especificado, e enviam assincronamente as respostas quando os recursos ficam disponíveis.

5.1.1 Análise de tráfego

Foram escolhidos quatro cenários distintos para testar as novas alterações introduzidas na plataforma. Ambos os cenários possuíam o serviço de *Rendezvous*

activo para manter a informação sobre a rede actualizada, medindo-se os pacotes trocados por este serviço num intervalo de tempo de 180 segundos.

Os dois primeiros cenários utilizam a aplicação desenvolvida num projecto anterior, para avaliar o serviço de *beacons* (Demo1) [Nelson05]. Nos dois últimos cenários é utilizada a aplicação desenvolvida neste projecto (Demo2) para demonstrar o serviço de gestão de pesquisas (SGP).

Na tabela 5.1 estão os resultados obtidos para cada cenário. Nos cenários "Sem SGP" significa que foi utilizada a plataforma alterada num projecto anterior [Nelson05]. Para os cenários "Com SGP" trata-se da plataforma JXTA com o SGP implementado.

	Pacotes	Octetos	Octetos/Pacote
Sem SGP + Demo1 (Cenário 1)	1707	1576470	924
Com SGP + Demo1 (Cenário 2)	1415	1323067	935
Sem SGP + Demo2 (Cenário 3)	526	436197	829
Com SGP + Demo2 (Cenário 4)	431	348595	809

Tabela 5.1 – Total de pacotes e octetos para os quatro cenários testados.

Pela tabela é possível visualizar uma redução elevada nos octetos transmitidos nos cenários que utilizam a Demo2 em relação aos da Demo1. Isto deve-se a que a Demo1 envia periodicamente pacotes para a rede para actualizar a informação sobre os recursos disponíveis. No caso da Demo2, o SGP permite um funcionamento num modo passivo. Apenas reage se surgir uma pesquisa na rede. Este modo de funcionamento foi desenvolvido tendo em conta que o serviço de gestão de pesquisas trata de todas as pesquisas retirando assim esse encargo da aplicação.

Inicialmente foram realizados testes apenas com o cenário 1 e 4, entre estes verificou-se uma redução dos octetos enviados para a rede de 77,89%. Ao verificar isto foram introduzidos os dois cenários intermédios. Deste modo é possível determinar se esta redução foi devida ao SGP ou à aplicação de demonstração. Ao analisar estes cenários intermédios verifica-se que o modo passivo da aplicação é responsável pela maior redução dos octetos transmitidos. No cenário 4 existe uma diminuição no total dos octetos enviados para a rede, cerca de 20% em relação ao cenário 3. A redução entre estes cenários é devida à introdução SGP.

5.2 Transferência de ficheiros

Como a aplicação desenvolvida para analisar o desempenho do SGP envolvia partilha de pesquisas, foram realizados testes ao seu desempenho na transferência de ficheiros.

Foi determinada a carga dos cabeçalhos adicionais que o protocolo JXTA adiciona a cada mensagem que é enviada. Para cada tamanho de bloco existe uma carga associada.

Nos testes realizados foram utilizados tamanhos de blocos de 1024, 2048, 4096, 8196, 16384, 32768 e 65536 para o caso do tipo de transporte ser TCP enquanto que no tipo UDP apenas não foi possível testar todos os tamanhos de blocos. A limitação do transporte UDP está relacionada com a implementação deste, na plataforma JXTA o limite do módulo existente é de 8192 octetos para o conteúdo de cada mensagem. No novo módulo de transporte a limitação está relacionada com o tamanho máximo do datagrama UDP (65536 octetos). Nos testes apenas foram tidos em conta blocos com 32768 octetos. Blocos de 65536 octetos passavam o limite do datagrama devido à informação adicionada a cada mensagem.

As limitações nos blocos devem-se a limitações dos canais de transporte da plataforma JXTA [JXTAPipe].

A aplicação de demonstração permite configurar o tamanho de cada bloco de dados e qual o modo de transporte a ser utilizado (Pipes JXTA, Modulo Transporte UDP/TCP).

O analisador de tráfego capturou o número total dos octetos transmitidos na rede, utilizados para transferir o ficheiro.

5.2.1 Medição da carga do protocolo JXTA

Para determinar a carga do protocolo JXTA sobre a transferência dum ficheiro foram testados dois cenários: no primeiro foi utilizada a plataforma JXTA desenvolvida por Nelson Ruivo com a respectiva aplicação de testes [Nelson05]; no segundo cenário foi usado o serviço de gestão de pesquisas desenvolvido neste trabalho. O ficheiro utilizado nos testes possuía 1198496 octetos.

Neste teste foi utilizado o transporte TCP sem compressão.

A tabela 5.2 contém os resultados obtidos com a plataforma JXTA antes de serem introduzidas as novas alterações (Teste 1). Neste caso foi utilizado a plataforma e aplicação anteriormente desenvolvidas.

Os dados seguintes da tabela 5.2 (Teste 2) representam os resultados obtidos com a plataforma com o serviço de gestão de pesquisas e a aplicação de demonstração

Na tabela seguinte mostram-se os resultados obtidos:

Teste		Tamanho de cada bloco (octetos)						
		1024	2048	4096	8192	16384	32768	65536
1	Enviados	119007	77779	70295	60680	53217	48681	47169
	Recebidos	3331856	2322314	1810809	1548339	1430583	1355013	1366005
	Total	3450863	2400093	1881104	1609019	1483800	1403694	1413174
	"Overhead"	287,9%	200,3%	157,0%	134,3%	123,8%	117,1%	117,9%
2	Enviados	1332050	678977	365995	204557	143922	103669	90075
	Recebidos	2479449	1882312	1565567	1416566	1366151	1320171	1316580
	Total	3811499	2561289	1931562	1621123	1510073	1423840	1406655
	"Overhead"	318,0%	213,7%	161,2%	135,3%	126,0%	118,8%	117,4%

Tabela 5.2 – Comparação entre as duas versões da plataforma JXTA.

Na anterior é possível visualizar o "*overhead*" obtido consoante o tamanho do bloco de dados utilizado. Um resultado esperado era que quanto maior o tamanho do bloco de dados, menor carga do protocolo JXTA na transferência do ficheiro. Como a cada mensagem é adicionada informação de controlo, usar blocos com mais octetos reduz o número total de mensagens enviadas e consequentemente a carga do protocolo.

Ao analisar a tabela verifica-se que a nova aplicação tem mais *overhead* que a aplicação anterior. No bloco 1024 octetos é a situação pior com 30% de agravamento, contudo para valores acima de 4096 a diferença ronda 1%-2%. Para o tamanho máximo é a situação mais favorável com a nova aplicação a ter um ganho de 0,5%.

Estes valores podem ser explicados pelo facto de serem usadas duas abordagens distintas nas duas aplicações. No primeiro caso trata-se duma transferência com atraso fixo entre envio de blocos. Este atraso pode ser configurável mas nunca aproveita totalmente a largura de banda disponível. No segundo caso cada mensagem só é enviada quando a anterior foi recebida correctamente, ou seja, o receptor envia um *acknowledge* e o emissor ao receber este envia a próxima mensagem.

Na tabela 5.3 é possível visualizar os tempos para cada transferência em milisegundos. No primeiro caso foram testados atrasos de 100, 10 e 1ms para cada envio de mensagem. Para a segunda situação o envio de blocos com e sem compressão.

Teste		Tamanho de cada bloco (octetos)						
		1024	2048	4096	8192	16384	32768	65536
1	Atraso 100ms	124422	63141	32484	19547	8922	4937	3859
	Atraso 10ms	18593	9500	5657	3594	3125	2984	3297
	Atraso 1ms	7176	5563	11016	10969	3500	3391	2953
2	Sem Compressão	12734	7125	4391	3109	2531	2171	2016
	Com Compressão	15500	8609	5140	3656	2922	2687	2532

Tabela 5.3 – Tempos de transferência em milisegundos.

Na aplicação desenvolvida para este projecto (Caso 2) existe um mecanismo de confirmação de recepção para cada bloco recebido, enquanto que na aplicação anterior (Caso 1) existe um atraso configurável.

Apenas com tamanhos de blocos pequenos (1024 e 2048 octetos) existe vantagem em utilizar transferência baseada em atraso no envio. Utilizar apenas um milisegundos de atraso nem sempre é vantajoso - para tamanhos de blocos com 4096/8192 octetos o tempo de transferência foi superior devido à necessidade de recuperar pacotes perdidos.

Em relação aos testes realizados com compressão é de notar que o tempo de transferência foi superior aos testes sem compressão. Este atraso é devido ao tempo de processamento, pois cada bloco é comprimido individualmente. No entanto, o valor elevado medido está relacionado com a falta de capacidade de processamento do computador usado nos testes, baseado num processador Intel Pentium III 600MHz e com 320MB de memória.

5.2.2 Medição de carga do módulo de transporte

Para testar o novo módulo de transporte apenas foi usada a aplicação desenvolvida na dissertação, apenas com a diferença do método de transporte. Foram realizados testes para dois cenários: utilizando-se TCP e UDP.

No primeiro cenário foram realizados dois testes distintos com o protocolo de transporte TCP. No primeiro teste (1) foi utilizada a aplicação de demonstração com canais da plataforma JXTA, utilizando o módulo JXTA de transporte TCP. Para o segundo teste (2) foi utilizada a mesma aplicação parametrizada para o módulo de transporte desenvolvido em modo TCP.

		Tamanho de cada bloco (octetos)						
		1024	2048	4096	8192	16384	32768	65536
1	Enviados	1332050	678977	365995	204557	143922	103669	90075
	Recebidos	2479449	1882312	1565567	1416566	1366151	1320171	1316580
	Total	3811499	2561289	1931562	1621123	1510073	1423840	1406655
	"Overhead"	318,0%	213,7%	161,2%	135,3%	126,0%	118,8%	117,4%
2	Enviados	69780	63354	47478	47262	43158	41430	42078
	Recebidos	1377332	1366517	1316108	1299962	1294823	1289263	1289690
	Total	1447112	1429871	1363586	1347224	1337981	1330693	1331768
	"Overhead"	120,7%	119,3%	113,8%	112,4%	111,6%	111,0%	111,1%

Tabela 5.4 – Comparação entre canais JXTA e o novo módulo (TCP)

Ao analisar a tabela anterior é notória a diminuição do *"overhead"* com a utilização do novo módulo. Essa diferença é mais nítida quando foi utilizado o bloco de 1024 octetos, onde a diferença foi de 197,3%. Ao utilizar blocos de 65536 octetos a diferença entre os dois testes é inferior, 6,2%.

Os canais da plataforma à medida que o tamanho do bloco aumenta diminuem drasticamente o *"overhead"* introduzido. No novo módulo de transporte o *"overhead"* introduzido é mais homogéneo, varia entre 120,7% para blocos de 1024 octetos e 111,1% no caso de blocos de 65536, ou seja, varia apenas 9,6%. Nas mesmas condições a utilização de canais da plataforma existe uma diferença de 200,7%.

No segundo cenário repetiram-se os testes do primeiro cenário, com o protocolo de transporte UDP. No primeiro teste foi a aplicação de demonstração com canais da plataforma JXTA mas parametrizada para utilizar UDP. Segundo teste foi alterado o tipo de transporte do módulo para UDP.

Idealmente deveriam ter sido testados ambos os cenários com os mesmos tamanhos de blocos que foram utilizados no cenário anterior, contudo neste cenário existem limitações.

Para os canais JXTA existe uma limitação de 8192 octetos. Ainda foi testada a utilização de blocos de 16384 octetos, mas sem sucesso. O novo módulo de transporte UDP foi limitado a 32768 octetos.

		Tamanho de cada bloco (octetos)					
		1024	2048	4096	8192	16384	32768
1	Enviados	1261323	631660	317857	186493		
	Recebidos	2451606	1851215	1537447	1400889		
	Total	3712929	2482875	1855304	1587382		
	"Overhead"	309,8%	207,2%	154,8%	132,4%		
2	Enviados	334578	168592	138439	47064	24678	10531
	Recebidos	1608166	1425345	1402247	1289666	1257175	1244410
	Total	1942744	1593937	1540686	1336730	1281853	1254941
	"Overhead"	162,1%	133,0%	128,6%	111,5%	107,0%	104,7%

Tabela 5.5 – Comparação entre canais JXTA e o novo módulo (UDP)

Ao analisar os resultados obtidos verifica-se uma redução do *"overhead"* introduzido, à semelhança do que tinha sido verificado no cenário anterior.

Para o tamanho de bloco menor (1024 octetos) a diferença é de 147,7%. Continua a ser uma diferença significativa, mas um pouco menor que a diferença obtida com o tipo de transporte TCP.

Mais uma vez, com o aumentar do tamanho do bloco utilizado os canais de transporte da plataforma obtêm-se melhores prestações. À semelhança do cenário anterior o módulo de transporte UDP introduz um *"overhead"* homogéneo entre os vários tamanhos de blocos.

5.2.3 Ganho obtido com compressão de blocos

A compressão dos blocos de dados tenta compensar a carga que o protocolo JXTA introduz. As experiências seguintes foram realizadas num computador portátil com processador Intel Pentium III 600MHz e 320MB de memória.

Pela tabela 5.6 é possível verificar que a introdução de compressão nos ficheiros tem algum impacto no tempo de transferência de ficheiros. Para o tamanho de bloco 65536 octetos, a compressão introduz um atraso adicional de 500 milisegundos. A

situação mais desfavorável é na utilização de blocos de 1024 octetos, onde o atraso devido à compressão foi de 2766ms.

Nos cenários testados foram utilizados dois ficheiros diferentes: um ficheiro de imagem no formato *"Bitmap"* (1190286 octetos) e um ficheiro de áudio no formato MP3 (1198496 octetos). A realização de testes com dois ficheiros diferentes serve para determinar qual o impacto da existência de compressão.

O ficheiro de imagem foi escolhido por ser um ficheiro que normalmente permite taxas de compressão elevadas. Em relação ao ficheiro de áudio isto já não acontece. O formato MP3 é um formato que já possui compressão de dados.

Na tabela seguinte mostram-se os resultados obtidos:

		Tamanho de cada bloco (octetos)						
		1024	2048	4096	8192	16384	32768	65536
BMP	Com compressão	2612897	1358184	648990	454407	315513	147038	127734
	Sem compressão	3786462	2533807	1977760	1603868	1449601	1381523	1336417
	Ganho	31,0%	46,4%	67,2%	71,7%	78,2%	89,4%	90,4%
MP3	Com compressão	3811499	2534450	1931562	1605861	1456978	1378175	1319755
	Sem compressão	3817717	2561289	1962062	1621123	1510073	1423840	1406655
	Ganho	0,2%	1,0%	1,6%	0,9%	3,5%	3,2%	6,2%

Tabela 5.6 – Total de octetos transmitidos com compressão.

Ao analisar a tabela anterior é possível verificar que para certos tamanhos do bloco de dados o total de octetos transmitidos é inferior ao número total de octetos do ficheiro.

O máximo de compressão que é possível é para o bloco de 65536 onde apenas 9,6% do tamanho do ficheiro são realmente transmitidos para a rede.

Mas a compressão nem sempre tem uma taxa tão elevada. No caso do ficheiro de áudio (MP3) o ganho é mínimo. Comparando o ganho obtido no MP3 na situação mais favorável (bloco 65536 octetos) apenas é possível obter um ganho de 6,2% muito longe dos 90,4% do ficheiro BMP.

Na tabela seguinte mostram-se os ganhos obtidos com a compressão:

		Tamanho de cada bloco (octetos)						
		1024	2048	4096	8192	16384	32768	65536
BMP	Simples	1103078	1112199	1130543	1141891	1149412	1152390	1151419
	Misto	1103072	1112199	1130528	1141891	1149412	1152390	1151419
MP3	Simples	-12502	-6053	4440	15467	20889	22098	22730
	Misto	362	367	4507	15467	20889	22098	22730

Tabela 5.7 – Ganho de compressão obtido em cada bloco de dados.

A tabela anterior mostra o ganho em octetos obtido para cada tamanho do bloco de dados. Foram analisados dois modos de operação: o modo simples e o modo misto. No modo simples são sempre enviados os dados comprimidos, enquanto que no modo misto os dados são enviados sem compressão quando os dados comprimidos tiverem um tamanho superior ao original. O modo misto apenas é possível porque as mensagens são em XML, assim o atributo MIME da etiqueta dos dados indica se estes estão comprimidos ou não.

O ficheiro de imagem tem sempre ganho positivo. O aumento do ganho conforme o aumenta o tamanho do bloco era esperado. Com blocos de 65536 apenas é enviado 3,3% do ficheiro. Neste tipo de ficheiro é normal existirem taxas de compressão tão elevadas. Para este ficheiro não existe diferença entre o modo misto e o modo simples.

No caso do ficheiro de áudio era esperado algum ganho mas pouco. O que foi verificado é que no modo simples, para os blocos de 1024 e 2048 eram enviados mais octetos que o tamanho do ficheiro. Isto é devido ao algoritmo de compressão adicionar informação ao bloco de dados. Surgiu assim a ideia do modo misto. Aconteceu que apenas nos blocos de 1024, 2048 e 4096 ocorreram melhorias. Nos restantes os resultados foram semelhantes aos do modo simples de compressão.

O modo misto foi utilizado na tabela 5.6 quando foi feita a comparação entre o envio com compressão dos blocos e sem compressão.

Capítulo 6. Conclusões

6.1 Síntese do Relatório

Esta dissertação engloba um estudo sobre a plataforma JXTA. O foco principal deste era detectar de que modo poderia ser melhorar o desempenho da pesquisa de recursos da plataforma em redes *ad hoc* móveis e reduzir a carga introduzida pela plataforma JXTA.

Foi desenvolvido um serviço de gestão de pesquisas que permite a realização de pesquisas assíncronas, com um maior controlo sobre as pesquisas.

Uma gestão das pesquisas eficiente reduz a carga do protocolo JXTA nas transferências. Com a redução da carga para a rede é obtido um melhor desempenho, especialmente para redes *ad hoc*.

Também foi implementado o módulo de transporte com vista a substituir o transporte existente na plataforma JXTA.

Uma comunicação eficiente é muito importante para um bom desempenho duma aplicação numa rede *ad hoc*.

6.2 Conclusões

A escolha da plataforma JXTA revelou-se acertada tendo em conta a descoberta de recursos. Este tipo de plataforma serve como base para o desenvolvimento de aplicações distribuídas, facilitando a implementação destas. Uma característica interessante é o facto da sua estrutura ser modular, o que facilita a sua extensão. Um novo módulo foi adicionado para nele serem incluídos todos os serviços destinados a redes *ad hoc*. A este módulo foi adicionado um serviço anteriormente implementado, o serviço de *beacon*. Para além deste serviço foi adicionado um novo serviço implementado de raiz, o serviço de gestão de pesquisas assíncronas.

Os resultados obtidos nos testes realizados revelaram melhorias de desempenho em relação à plataforma JXTA tomada como ponto de partida. Ao introduzir as pesquisas assíncronas elimina-se a necessidade de inundar periodicamente a rede com pesquisas periódicas. As pesquisas são efectuadas uma vez e serão válidas até o seu

tempo de vida expirar. As repropagações quando um novo nó é detectado, representam menor carga que a inundação da rede com pesquisas.

Ao testar a transferência de ficheiros foi possível verificar que o uso do XML nas mensagens pelo protocolo JXTA introduz um custo elevado em largura de banda. O conjunto novo serviço de gestão de pesquisas com a aplicação desenvolvida permitiu reduzir em cerca de 77,89% o tráfego gerado pela plataforma JXTA. Mesmo assim esta carga ainda apresenta um valor relevante – em 180 segundos são enviados para a rede 348595 octetos com apenas dois nós activos.

A aposta em substituir o transporte da plataforma por um novo módulo de transporte independente desta, mostrou ser a uma escolha válida para reduzir a largura de banda. Foi possível reduzir o custo de utilização do protocolo JXTA. Parte da redução é devida ao facto das mensagens, no novo módulo, não serem em XML mas numa forma binária resultante da serialização Java. Esta opção viola uma das premissas da plataforma JXTA: a independência de linguagens. Contudo, o ganho de largura de banda obtido com esta substituição é relevante.

A compressão dos blocos de mensagens permite em alguns casos compensar o aumento de dimensão total de octetos transmitidos com o protocolo, contudo depende muito do tipo de ficheiro que se transfere. Como foi possível verificar, ficheiros que possuam alguma compressão não beneficiam tanto com a compressão dos blocos. No entanto, a compressão induz um aumento no tempo de transmissão devido ao tempo de processamento. Nas experiências realizadas o tempo foi da ordem de grandeza de centenas de milisegundos, mas pode ser reduzido significativamente utilizando-se hardware mais recente, com maior capacidade de cálculo.

Em relação ao transporte *multihop* a plataforma JXTA está em vantagem em relação ao módulo de transporte implementado. A plataforma possui métodos de reencaminhamento de mensagens próprios, o que é uma grande vantagem.

Os canais da plataforma são instáveis. Quando se entra com comunicações com 2 ou mais *hops* essa instabilidade é mais acentuada, e cabe à aplicação estar preparada para estas situações.

O módulo de transporte desenvolvido não permite comunicações *multihop*, mas é útil para comunicações entre nós ao alcance rádio. Caso um nó não esteja ao alcance rádio é possível utilizar canais JXTA. Para determinar se o nó é vizinho pode ser utilizado o serviço de gestão de pesquisas. Como o módulo de transporte

independente desenvolvido mantém praticamente a mesma estrutura dos módulos originais JXTA, a sua integração na aplicação de teste requereu poucas alterações. Desta forma foi possível fazer uma comparação justa entre o desempenho dos dois módulos.

6.3 Perspectivas de Trabalho Futuro

O serviço de *beacon* e o serviço de gestão de pesquisas adaptam uma plataforma destinada a redes com fios para poder funcionar em redes MANET. A descoberta de vizinhos é assegurada pelo serviço de *beacon*, enquanto as pesquisas assíncronas são responsabilidade do serviço de gestão de pesquisas.

O próximo grande passo seria aproveitar o conjunto de ideias que vieram deste a primeira alteração (serviço de *beacon*) até ao módulo de transporte independente, passando pelo serviço de gestão de pesquisas, e iniciar o desenho duma plataforma construída de raiz de forma a ser completamente optimizada para as redes MANET. Para este passo seria muito importante aproveita a experiência obtida com a plataforma JXTA, alguns dos seus conceitos são inovadores.

As actualizações à plataforma JXTA já não são regulares. A integração do módulo MANET em versões mais recentes, deve ser estudada para o caso de existirem alterações no modo de configuração dos módulos na plataforma. No caso de se adicionar apenas um serviço ao módulo MANET apenas é necessário alterar as classes deste, ou seja, o novo serviço fica independente das configurações da plataforma.

É possível reduzir a carga da estrutura de controlo de envio de blocos. Actualmente por cada bloco recebido é enviada uma mensagem de confirmação. Um ficheiro é repartido em partes, estas partes são depois repartidas em blocos para serem enviados na mensagem. Ou seja uma parte do ficheiro de 128kb se for transmitida em blocos de 1kb origina 128 mensagens de confirmação de recepção.

A situação ideal seria o envio de blocos sequencial até enviar o tamanho total da parte do ficheiro e apenas no final ser enviada uma mensagem de confirmação de recepção. Neste caso em vez de enviar 128 mensagens de confirmação de recepção apenas uma mensagem era enviada. Para evitar os problemas de fiabilidade do cenário

anterior poder-se-ia usar um mecanismo de janela com retransmissão selectiva associado a um relógio de retransmissão.

O recurso à compressão mostrou-se viável para reduzir a quantidade de informação transmitida. Uma funcionalidade que poderia reduzir o *overhead* seria a optimização da compressão. Um caso detectado foi que alguns blocos de 65535 octetos quando comprimidos ficavam com cerca de 4000 octetos. Desta forma poder-se-ia adicionar mais octetos até preencher o tamanho disponível na mensagem. Ao recorrer a este tipo de abordagem as mensagens aproveitavam o tamanho do máximo do bloco definido. Optimizando as mensagens o número das mensagens necessárias para transferir o ficheiro decresce e a carga do protocolo JXTA diminui.

Referências

[Bernardo06] L. Bernardo, R. Oliveira, S. Gaspar, D. Paulino, e P. Pinto, *"A Telephony Application for MANETs: Voice over a MANET-extended JXTA Virtual Overlay Network"*, Wireless Information Networks and Systems (WINSYS'06) (part of ICETE'06), ISBN: 972-8865-65-1, pp.227-233. Setúbal, Portugal, Agosto 2006.

[Bernardo08] L. Bernardo, R. Oliveira, S. Gaspar, D. Paulino, e P. Pinto, "A Telephony Application for MANETs: Voice over a MANET-extended JXTA Virtual Overlay Network", e-Business and Telecommunication Networks, CCIS 9, ICETE 2006 Selected Papers, Springer, ISBN: 978-3-540-70759-2, pp. 347-358, 2008.

[Chen01] S. H. Shah, K. Chen, e K. Nahrstedt. "*Cross Layer Design for Data Accessibility in Mobile Ad Hoc Networks*". Proc. of the 5th World Multiconference on Systemics, Cybernetics and Informatics and 7th International Conference on Information Systems Analysis and Synthesis (SCI 2001/ISAS 2001), Orlando, Florida, July 2001.

[Chord] Stoica I., Morris R., Karger D., Kaashoek F. e Hari Balakrishnan. "*Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications*". In Proceedings of the ACM SIGCOMM 2001, San Diego, CA, USA, August 2001.

[FastMD5] Timothy W Macinta's, Fast MD5 Implementation in Java. Retirado em Março 3, 2006 da World Wide Web: http://www.twmacinta.com/myjava/fast_md5.php

[Gnutella04] The Gnutella Protocol Specification v0.4

[JXTAspec] JXTA v2.0 Protocols Specification. Retirado em Setembro 30, 2005 da World Wide Web: https://jxta-spec.dev.java.net/nonav/JXTAProtocols.html

[JXTAb] JXTA v2.3.x: Java Programmer's Guide. Retirado em Setembro 30, 2005 da World Wide Web: http://www.jxta.org/docs/JxtaProgGuide_v2.3.pdf

[JXTAPipe] Tamanho máximo das mensagens. Retirado em Novembro 22, 2005 da World Wide Web: http://www.jxta.org/servlets/ReadMsg?list=discuss&msgNo=13797

[Tariq06] Tariq M., Ammar M e Zegura E., "*Message Ferry Route Design for Sparse Ad hoc Networks with Mobile Nodes*". MobiHoc06, Florença, Itália, Maio 2006.

[Nelson05] Ruivo, N. *"Partilha de ficheiros numa rede ad-hoc"*. Projecto final da Secção de Telecomunicações, Departamento de Engenharia Electrotécnica da Faculdade de Ciências e Tecnologia – Universidade Nova de Lisboa, Portugal. Dezembro 2005.

[Oliveira05a] Oliveira R., Bernardo L. e Pinto P. "*Searching for resources in MANETs: A cluster based flooding approach*". 2nd International Conference on E-business and Telecommunication Networks (ICETE'05).

[Oliveira05b] Oliveira, R., Bernardo, L., Ruivo, N. e Pinto, P. "*Searching for PI resources on MANETs using JXTA"*, IARIA Conferência sobre Service Assurance with Partial and Intermittent Resources (SAPIR'05), Lisboa, Portugal, Julho 2005.

[Penz05] Penz S., "SLP-based Service Management for Dynamic Adhoc Networks". Nov. 2005.

[SLP99] Guttmann E., Perkings C., Veizades J. e Day M., "Service Location Protocol - Version 2" 1999, RFC 2608.

[Srdjan05] Srdjan K., David C. and Daryl P., "*P2P Mobile Sensor Networks*", in the proceedings of the 38th Hawaii International Conference on System Sciences, 2005, Big Island, Hawaii, USA.

[Suri03] Suri, N., Bradshaw, J. M., Carvalho, M., Cowin, T. B., Breedy, M. R., Groth, P. T., & Saavendra, R., "*Agile computing: Bridging the gap between grid computing and ad-hoc peer-to-peer resource sharing*". O. F. Rana (Ed.), Proceedings of the Third International Workshop on Agent-Based Cluster and Grid Computing. Tokyo, Japan, 2003.

[Taylor03] Taylor I., Shields M., Wang I. e Rana O. "Triana Applications within Grid Computing and Peer to Peer Environments" In Journal of Grid Computing, 1(2):199-217. Kluwer Academic Press, 2003.

[Tian05] Ye Tian, Kai Xu, e Nirwan Ansari, "*TCP in Wireless Environments: Problems and Solutions*", IEEE Communications vol. 43 no. 3, pp. S27-S32, Março 2005.

[Tortonesi06] Tortonesi M., Stefanelli C., Suri N., Arguedas M. e Breedy M. "Mockets: A Novel Message-Oriented Communications Middleware for the Wireless Internet" Conferência sobre Wireless Information Networks and Systems (WINSYS'06) Setúbal, Portugal, Agosto 2006.

[Wang05] Wang I. "*P2PS (Peer-to-Peer Simplified)*". 13th Annual Mardi Gras Conference - Frontiers of Grid Applications and Technologies. Louisiana State University, Fevereiro 2005.

[Wireshark] Wireshark – Network Protocol Analyser (Version 0.99.7) http://www.wireshark.org

[80211a] IEEE (Institute of Electrical and Electronics Engineers) Standard 802.11a, 1999.

[80211b] IEEE (Institute of Electrical and Electronics Engineers) Standard 802.11b, 1999.

[80211g] IEEE (Institute of Electrical and Electronics Engineers) Standard 802.11g, 2003.

[80211n] IEEE (Institute of Electrical and Electronics Engineers) Draft 802.11n, Setembro 2007.

Anexo A. Redes sem fios

Uma rede sem fios é constituída por estações. Uma estação pode ser qualquer dispositivo que possui as funcionalidades descritas na norma 802.11, desde um computador portátil até um PDA. Estes dispositivos por serem móveis têm limitações a nível de recursos disponíveis. Normalmente o recurso mais limitado é a bateria. Existem no entanto dispositivos fixos, como os pontos de acesso que permitem, por exemplo, partilhar Internet para as outras estações que a ele estejam ligadas.

A norma define as BBS (*Basic Service Set*) como componente básico para uma rede sem fios. Isto consiste num conjunto de estações que estão todas ao alcance de rádio umas das outras. Dentro duma BBS é possível existirem duas topologias: a primeira é a topologia IBSS (*Independent Basic Service Set*) também denominada por *ad hoc*. A outra topologia é *Infrastructure Basic Service Set* onde existem várias estações ligadas a um ponto de acesso. Esta topologia é utilizada para fazer a ligação entre uma rede sem fios a uma rede fixa, permitindo aos utilizadores ligados a rede sem fios acederem aos serviços disponibilizados na rede fixa.

Numa rede 802.11 em modo *ad hoc* a comunicação entre as estações é directa, isto implica que as estações que pretendam comunicar entre si têm de estar no alcance de rádio mútuo.

No modo *ad hoc* do 802.11b/g é utilizado como mecanismo de acesso ao meio o modo DCF (*Distributed Coordination Function*). Para evitar colisões após cada transmissão é esperado um tempo aleatório. Antes de cada transmissão é feita a verificação se o meio rádio está livre, a primeira estação que inicia uma transmissão toma controlo do meio durante o tempo desta.

Para controlar o acesso ao meio de transmissão o DCF implementa o protocolo CSMA/CA (*Carrier Sense Multiple Access* / C*ollision Avoidance*). Ao contrário das redes *Ethernet* onde é feita a detecção de colisões, o CSMA/CA evita a ocorrência de colisões. O método de acesso ao meio é distribuído. Não existe nenhum sistema central que controle a utilização do meio pelas várias estações. Cada estação utiliza o mesmo método para aceder ao meio.

A especificação da norma 802.11 inclui o sub nível MAC (*Media Access Control*) e LLC (*Logical Link Control*) formando o nível ligação de dados (Data Link), situado entre o nível transporte e nível físico.

O LLC disponibiliza um nível lógico comum a diversos protocolos, este veio assim facilitar a integração da norma 802.11 nos sistemas de comunicação já existentes.

A separação de dados em tramas a partir dos dados entregues pelo LLC é também feita pelo sub nível MAC. Os níveis físicos especificados diferem entre si na tecnologia utilizada para propagar tramas no meio, oferecendo velocidades diferentes de emissão de tramas.

Devido ao meio de comunicação ser aberto este tipo de redes está sujeito a interferências exteriores a rede. Assim as comunicações sem fios possuem uma taxa de erros elevada, ou seja, a fiabilidade destas é reduzida.

A situação mais importante em redes sem fios é a da estação escondida. Nesta situação existem três estações onde apenas uma consegue comunicar com as restantes (estação central). Como nesta situação existem duas estações que não conseguem comunicar entre si, se ambas tentarem utilizarem o meio simultaneamente a estação que comunica com ambos transmite com uma taxa elevada de erros devido à colisão de pacotes. Isto acontece porque nesta situação o mecanismo CSMA/CA não funciona. Para evitar que ocorra uma elevada taxa de colisões nas transmissões da estação central é utilizado um mecanismo VCS (*Virtual Carrier Sense*). Este consiste no envio de mensagens RTS (*Request To Send*) e CTS (*Clear To Send*). Se uma estação pretende transmitir envia um RTS para reservar o acesso ao meio, assim as restantes estações são informadas da sua intenção. Cabe à estação destinatária da mensagem enviar um CTS, informando as estações presentes no seu raio de recepção que também não podem emitir. Com a troca de RTS e CTS uma estação está na realidade a efectuar uma reserva de um canal virtual no nível MAC.

Recorrer ao envio de mensagens do tipo RTS e CTS introduz atraso na mensagem que se pretende enviar, portanto apenas é utilizado este sistema quando o tamanho da mensagem a enviar for elevado. Se o tamanho da mensagem for pequeno, é mais eficiente recorrer à sua retransmissão caso seja necessário.

Ao nível virtual é utilizado um NAV (*Network Allocation Vector*). O NAV é um temporizador que indica o intervalo de tempo que o meio vai estar ocupado. Nas tramas existe um campo que define o tempo de duração de cada trama, com isto as outras estações tomam conhecimento do tempo de NAV de cada transmissão. Quando uma estação transmite, configura o NAV para o tempo que espera utilizar o meio e

envia o tempo de NAV juntamente com as mensagens. As restantes estações vão actualizando e decrementando o seu NAV com os tempos de NAVs que receberem até este chegar a zero. Isto significa que enquanto o NAV for diferente de zero o meio está ocupado. Ao chegar a zero as estações ficam com a informação que o meio está virtualmente livre.

Apenas é tentada a recuperação de erros caso ocorra uma falha numa trama de sinalização ou quando falha o acesso ao meio. A recuperação de erros é baseada na retransmissão das mensagens, caso o limite de retransmissões seja ultrapassado é sinalizada a falha do envio aos níveis superiores.

Nas mensagens para vários receptores (*multicast*) não existe recuperação de erros devido a estas não possuírem confirmação. Este tipo de mensagens é menos fiável.

2.1.7 Nível transporte

O nível de transporte mais utilizado actualmente na Internet é o TCP. Mas nem sempre este transporte é o mais indicado. Como é o caso de aplicações que possuam tráfego multimédia em tempo real.

No TCP existe um sistema de janelas de congestão que permite controlar o fluxo de dados. Uma ligação TCP é caracterizada no início por um arranque lento (*Slow Start*), ou seja, o tamanho da janela de congestão é reduzido e vai aumentando exponencialmente até chegar ao tamanho definido dinamicamente. Após atingir este valor o aumento continua duma forma linear para evitar o congestionamento da rede.

Como referido o tamanho da janela é definido dinamicamente consoante as perdas de pacotes verificadas: quando ocorre uma perda de pacotes esta é sub-entendida como sinal de congestão da rede. Caso o tempo de espera do pacote de sinalização for excedido o tamanho da janela é reduzido para o seu valor inicial; por outro lado se for recebida a indicação de perda de pacotes antes do temporizador expirar, o tamanho da janela é reduzido para metade.

O algoritmo de controlo de congestão utilizado no protocolo TCP foi desenhado para comunicações em redes fixas. É assumido que a perda de um pacote é sempre causada por congestão na rede e não por erros na transmissão, ou seja, foi tido em conta que a probabilidade de erro em comunicações por fios é bastante baixa.

Contudo nas redes sem fios a probabilidade de erro em transmissões isoladas é significativa.

Numa rede *ad-hoc*, por vezes ocorrem perdas de pacotes devido a falhas nas rotas provocadas pela movimentação das estações na rede. Actualizar as rotas e a adaptação das tabelas de encaminhamento introduz atraso na entrega de pacotes, independentemente do algoritmo utilizado. O algoritmo de controlo de congestão do TCP quando ocorre perda de pacotes ou expirem os temporizadores de sinalização, baixa o ritmo de transmissão. Contudo a causa das perdas pode ser temporária e não significa congestão na rede.