UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

MAURÍCIO LEITE DAU

Comunicação entre MANETs com protocolo de roteamento baseado na fonte

Trabalho de Graduação.

Prof. Dr. João Netto
Orientador

Porto Alegre, novembro de 2013.

Agradecimentos

Eu dedico a realização deste trabalho aos meus pais, que nunca pouparam esforços para fornecer a mim e aos meus irmãos uma educação de qualidade. Foram esses esforços que me permitiram ingressar na universidade e que me deram coragem para acreditar nos meus sonhos. Agradeço à minha maravilhosa companheira Gisele Medeiros pelo apoio durante todo o decorrer do ano e por, muitas vezes, me ouvir pacientemente explicar sobre redes de computadores e me devolver um sorriso encorajador. Foi com o apoio desse amor que este trabalho foi realizado.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

AODV	Ad-Hoc On-demand Distance Vector
DSR	Dynamic Source Routing
FIFO	First In, First Out
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
MAC	Media Access Control
MANET	Mobile Ad-Hoc Network
RREQ	Route Request
RREP	Route Reply
RERR	Route Error
SSID	Service Set Identifier
TTL	Time to live
UUID	Universal Unique Identifier
VANET	Vehicular Ad-Hoc Network

LISTA DE FIGURAS

RESUMO

O mercado de dispositivos móveis apresenta uma grande heterogeneidade, sendo composto por *smartphones*, *tablets*, *notebooks*, *netbooks*, sensores e outros dispositivos com diferentes interfaces de comunicação sem fio. Este trabalho propõe uma alteração na estrutura do protocolo de roteamento *Dynamic Source Routing* (DSR) para que seja possível que esses dispositivos formem uma rede de comunicação. Essa alteração compreende a troca do modo de endereçamento IP para o endereçamento UUID. Essa alteração possibilitará a abstração dos meios de transmissão de dados, permitindo que diferentes dispositivos forme uma rede ad hoc. A comunicação entre dispositivos com diferentes tecnologias de transmissão de dados ocorre por meio de um dispositivo intermediário que possui ambas tecnologias de transmissão.

A fim de analisar o comportamento do protocolo implementado são realizadas transmissões de dados em uma rede Wi-Fi operando em conjunto a uma rede emulada, que utiliza a estrutura *FakeInterface*. Essa rede emulada foi construída para possibilitar a execução de testes durante o desenvolvimento do trabalho e para confirmar o funcionamento do protocolo com múltiplas interfaces de rede.

Palavras-Chave: DSR, AODV, redes ad hoc, comunicação.

Communication between MANETs with source based routing protocol

ABSTRACT

The mobile market has a great heterogeneity, being compound by smartphones, tablets, notebooks, netbooks, sensors and others devices with different wireless network interfaces. This work proposes a modification in the structure of Dynamic Source Routing (DSR) protocol, to allow different devices to establish a communications network. This modification consist in changing the addressing mode from IP to UUID, enabling the abstraction of the transmission media and allowing devices with different data transmission technology to create an ad hoc network. The communication between this devices take place via an intermediary device with both data transmission technology.

To analyze the behavior of the implemented protocol data transmission in a Wi-Fi network operating with an emulated network that uses FakeInterface structure were analyzed. The emulated network was constructed to allow the execution of tests during the development and to confirm the operation of the protocol with multiple network interfaces.

Keywords: DSR, AODV, ad hoc networks, communications.

1 INTRODUÇÃO

O número de aplicações móveis está cada vez maior, impulsionado pelo avanço dos smartphones e pelos elementos de computação distribuída. A característica comum de todos os dispositivos é a comunicação sem fio, seja por Wi-Fi, Bluetooth ou outras tecnologias de transmissão de dados. Entretanto, cada tecnologia possui suas características específicas e dois dispositivos só conseguem se comunicar se possuírem o mesmo hardware. Com isso, muitos usuários não conseguem realizar algumas atividades, pois ficam restritos ao hardware que possuem. Este trabalho propõe uma arquitetura de comunicação entre dispositivos móveis que é capaz de transpor essa barreira, utilizando técnicas de roteamento em redes ad hoc.

É uma situação comum dois usuários de smartphones desejarem trocar arquivos e não possuírem nenhum cartão de memória ou cabo para transferir os dados de um aparelho para o outro. Nesse caso, os usuários optam por realizar a transferência por meio de uma rede sem fio. Uma rede sem fio criada por dispositivos móveis é chamada de MANET (Mobile Ad Hoc Network) e possui a característica de ser uma rede descentralizada e temporária, exatamente como no caso da transferência de arquivos entre dois smartphones.

Contudo, pode acontecer de os dispositivos possuírem configurações de hardware diferentes, o que impede a comunicação entre eles. Caso haja um terceiro dispositivo que possua o hardware de ambos, a transferência do arquivo pode acontecer por meio deste novo dispositivo, utilizando-o como uma ponte.

Outra aplicação dessa arquitetura é no mercado de jogos digitais, no qual pode-se ter um conjunto de usuários que possuem dispositivos móveis distintos com o interesse de jogar entre si por meio de uma rede sem fio. É comum os jogos serem desenvolvidos para suportarem uma determinada tecnologia de comunicação (i.e. Wi-Fi), excluindo todos os aparelhos que não a possuem. Porém, utilizando os elementos que possuem a tecnologia do jogo e a dos outros aparelhos, a comunicação pode acontecer por meio desses dispositivos.

O ambiente de troca de arquivos entre dois dispositivos que não possuem o mesmo hardware de comunicação pode ser visto na Figura X, na qual temos o nodo A tentando enviar um arquivo para o nodo B. As duas redes sem fio até podem se atingir, mas na prática são duas redes totalmente independentes.

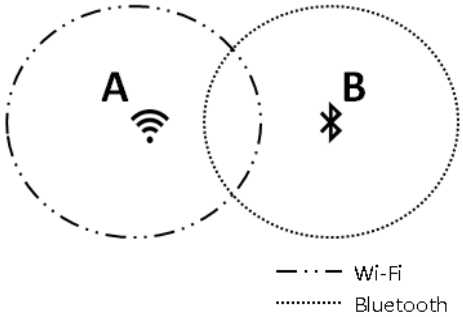


Figura 1 Duas redes sem fio de tecnologias diferentes se sobrepondo

Neste caso, adicionando um terceiro dispositivo, o nodo C, que suporte as duas redes sem fio, podemos transferir o arquivo do nodo A para o B por meio do C, de forma transparente. Pois quando o nodo A procurar os dispositivos que estão ao seu alcance, o nodo C vai responder ele mesmo (C) e todos a quem ele alcança (B). Assim, teremos uma transferência de arquivos entre A e B de forma transparente, mesmo eles possuindo diferentes tipos de redes sem fio.

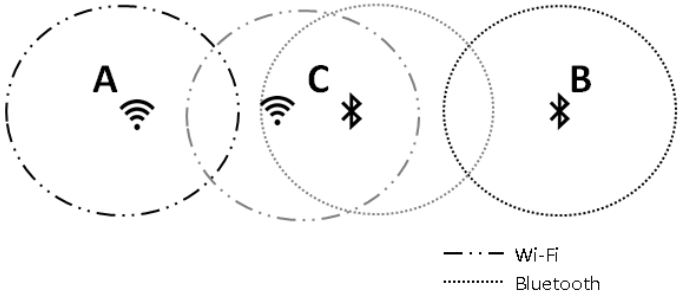


Figura 2 Nodo C operando como ponte para comunicação de A com B

A seção 2 deste trabalho define as características da comunicação ad hoc de dispositivos móveis sem fio e apresenta o problema de comunicação entre diferentes redes sem fio. A seção 3 apresenta as soluções existentes para realizar a comunicação dentro de uma rede ad hoc sem fio, explicando os protocolos que servem de base para a arquitetura proposta.

O cenário básico deste trabalho é uma MANET com poucos nós, baixa mobilidade e com grande heterogeneidade de hardware, desde smartphones até sensores sem fio. Considera-se que os usuários desejam trocar arquivos ou rodar uma aplicação comum sobre todos eles e, com isso, seus dispositivos participam ativamente da rede, realizando a retransmissão de pacotes quando solicitado.

Todos os dispositivos que tiverem interesse em participar da rede deverão seguir as regras de roteamento definidas pela arquitetura, evitando assim, que ocorram loops nos trajetos e falsas indicações de roteamento.

A tradução literal da expressão "Ad hoc" é: algo com uma finalidade específica. Isso significa que os serviços ad hoc têm um objetivo a ser alcançado . Após alcançado o objetivo inicial, específico, o serviço deve ser considerado concluído. Um exemplo que ilustra o termo ad hoc é o trabalho efetuado pelos avaliadores ou revisores ad hoc, os quais possuem a função de avaliar ou revisar textos sem possuírem vínculo com os seus produtores e que, ao concluírem a avaliação ou a revisão do seu objeto de trabalho, têm por encerrada a sua tarefa.

Uma rede Ad hoc é criada por um conjunto de nodos independentes com o interesse de realizar uma atividade. Esses nodos não necessitam de outros para conseguirem se comunicar. Essa rede tem a duração da atividade proposta e, com o término desta, a rede deixa de ser necessária e é desmanchada. Uma rede Ad hoc criada por dispositivos móveis é chamada de MANET (Mobile Ad Hoc Networking) e adiciona à rede ad hoc as características de operar sobre redes sem fio e depender de baterias. A comunicação numa MANET deve ser simples e consumir pouca energia, pois os dispositivos utilizam o mesmo meio, o ar, para transmitir as suas informações e possuem uma fonte finita de energia, a bateria.

O objetivo deste trabalho é desenvolver uma arquitetura que, implementada em diferentes dispositivos, seja capaz de estabelecer uma rede ad hoc e de prover envio/recebimento de informações por meio da mesma. Para tal, será implementado um protocolo de roteamento para redes ad hoc com uma alteração no modo de endereçamento. O endereçamento utilizado será o Universal Unique Identifier (UUID), capaz de identificar uma grande quantidade de dispositivos unicamente. Esse endereçamento é utilizado a fim de abstrair a tecnologia de transmissão, possibilitando que diferentes dispositivos consigam se comunicar por meio de um dispositivo intermediário que possua ambas tecnologias de transmissão.

2 PADRÕES DE REDES AD HOC

Uma rede ad hoc sem fio é um conjunto de dispositivos móveis que formam uma rede temporária e não dependem de uma infraestrutura centralizada [2]. Pelo fato de essa rede não possuir nodos exclusivos para manutenção da conectividade, todos os nodos devem participar do gerenciamento da rede, ou seja, além de gerenciar seus próprios fluxos de dados, eles também devem auxiliar na manutenção dos fluxos dos outros nodos.

Os nodos que estão no alcance direto da transmissão sem fio são denominados vizinhos. Os vizinhos de um nodo mudam constantemente durante o funcionamento de uma rede ad hoc, e o grau de alteração dos vizinhos define qual a mobilidade da rede. Uma rede altamente móvel possui uma grande variação de vizinhos, como as redes veiculares, i.e. *Veicular Ad Hoc Network* (VANET) [25], nas quais os nodos se locomovem com alta velocidade ficando pouco tempo com os mesmos vizinhos.

A alteração dos vizinhos reflete diretamente na alteração da topologia da rede, obrigando o nodo que ainda queira se comunicar com um antigo vizinho a descobrir uma nova rota para enviar seus dados. Uma rota é uma sequência de nodos não repetidos por meio da qual um nodo de origem S consegue enviar dados para um nodo de destino D.

O consumo de energia dos dispositivos que compõem uma rede ad hoc sem fio está relacionado não só com as operações particulares dos nodos, mas também com as operações de gerenciamento da rede, pois quando um nodo não está transferindo dados próprios ainda assim estará retransmitindo dados dos outros nodos. O consumo de energia é mais sensível para as redes ad hoc de dispositivos móveis (MANET), nas quais os nodos operam conectados a uma bateria, e o tempo entre recargas deve ser estendido ao máximo. A maior parcela do consumo de energia dos dispositivos móveis é gasta na comunicação sem fio. Assim, diminuindo o tempo de transmissão de dados, aumenta-se a vida útil da bateria do dispositivo [9], [10].

A configuração de uma rede ad hoc difere das redes infraestruturadas, desde a placa de rede sem fio, que em alguns casos deve ser configurada de forma diferente, como no padrão IEEE 802.11, no qual o modo de transmissão deve ser alterado para ad hoc, até os protocolos de roteamento, que devem considerar a variação da topologia durante a descoberta e a manutenção de rotas. Em seguida, serão apresentados três padrões de comunicação sem fio e o seu uso nas redes ad hoc. Após, serão analisados dois protocolos de roteamento para redes ad hoc.

2.1Padrões de Rede Sem Fio

2.1.1 Bluetooth

O padrão de comunicação Bluetooth [1], [11], [12] opera na faixa de 2.4MHz a 2.48MHz e foi desenvolvido para prover conectividade local entre um pequeno conjunto de dispositivos. Uma rede Bluetooth é composta de dois a oito dispositivos, conectados na forma de mestre e escravos. A topologia mínima do Bluetooth é denominada *piconet*, formada por um dispositivo mestre controlando até sete dispositivos escravos. Diversas *piconets* podem ser conectadas entre si, utilizando os dispositivos mestres para controlar essas conexões. Uma topologia com *piconets* interconectadas é denominada *scatternet*. As conexões em uma *piconet* são sempre entre dois pontos que operam de forma estruturada, ou seja, existe um nodo centralizador que controla as transmissões e serve de ponte para as comunicações entre os escravos. Na figura a seguir, pode-se verificar a interligação de três *piconets* (redes 1, 2 e 3) por meio dos notebooks A, mestre da *piconet* 1, B, mestre da *piconet* 2, e C, mestre da *piconet* 3.

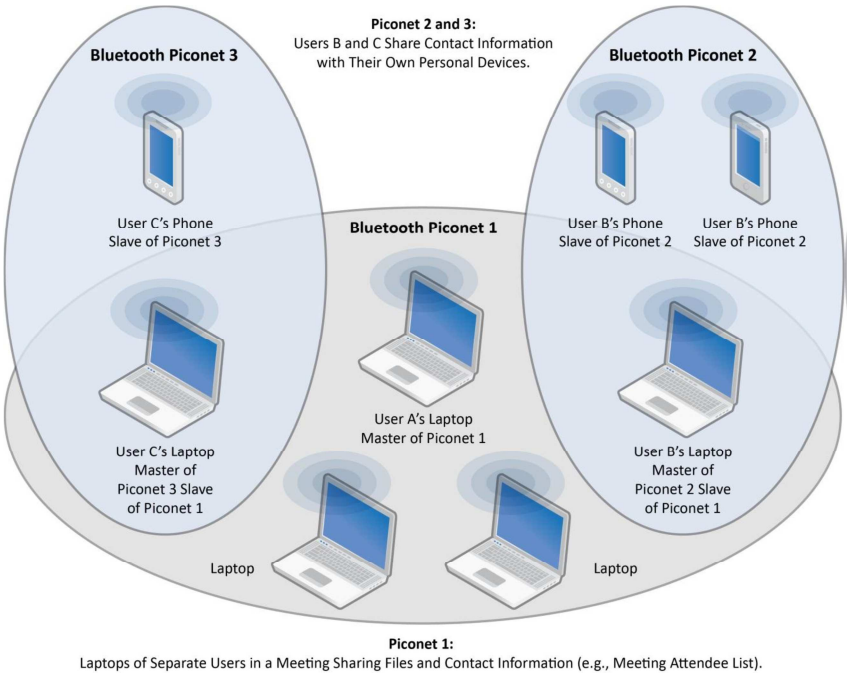


Figura *3Piconets* e *scatternets*[1]

O uso do meio físico no Bluetooth ocorre de forma dividida no tempo e utiliza salto de frequência (*frequency hopping*). O salto de frequência é utilizado de forma a diminuir a interferência entre redes Bluetooth. A divisão no tempo é definida pelo dispositivo mestre, que orienta os escravos a enviarem os dados durante um certo período e a escutarem por dados durante outro período de tempo [1].

O exemplo de funcionamento mais simples do Bluetooth é uma *piconet* composta por dois nodos: um mestre e outro escravo. O salto de frequência, por padrão, ocorre

[1] Padgette & Scarfone, 2011, p. 6-8.

1600 vezes por segundo dentro de uma faixa de 80MHz, definindo intervalos de tempo de 625µ segundos. Assim, o dispositivo mestre transmite, por exemplo, pacotes nos intervalos ímpares de tempo e o escravo transmite pacotes nos intervalos pares.

A padronização do Bluetooth é controlada pelo *Bluetooth Special Interest Group* (Bluetooth SIG), composto principalmente por Intel, Sony e IBM. O IEEE realizou a padronização do Bluetooth como o padrão IEEE 802.15.1, porém não realizou a manutenção do mesmo.

O Bluetooth pode ser classificado pelo seu alcance e pela sua potência de operação. A classe 1 opera com 100mW de potência e possui alcance de até 100m. A classe dois, mais comum entre os dispositivos comerciais, opera com 2.5mW de potência e possui alcance de até 10m. A terceira classe é a menos comum e opera com 1mW de potência a distâncias de até 1m. As taxas de transmissão podem chegar a 24Mbits/s na versão 3.0, no entanto, a taxa de transmissão mais comum é a de até 3Mbits/s, definida na versão 2.0.

Uma rede ad hoc multinível pode ser construída com a utilização das *scatternets* do Bluetooth, ligando *piconets* entre si. Diversos trabalhos já propuseram maneiras de conectar *piconets* a fim de construir redes multinível mais complexas[13]. Para a construção dessas redes, mais um tipo de nodo é definido, o mestre-escravo. Uma *scatternet* é uma *piconet* de mestres de outras *piconets* e também possui um mestre definido. Um escravo de uma *scatternet* é o mestre de uma *piconet*, também denominado mestre-escravo.

2.1.2 Zigbee

O Zigbee[15] é um padrão baseado na IEEE 802.15.4, voltado para o mercado de baixo consumo e de redes de sensores. Possui um alcance de, aproximadamente, 100 metros, dependendo da quantidade de obstáculos, e uma taxa de transmissão de até 250 kbits/s. O Zigbee se assemelha ao Bluetooth por possuir, em cada rede, um dispositivo que coordena a criação e a manutenção da rede.

A camada física e a de controle de acesso ao meio (MAC) foram definidas na IEEE 802.15.4, regulando que a faixa de frequência utilizada é a de 2.4GHz, para todo o mundo, e 915MHz e 868MHz para as Américas e a Europa, respectivamente. A taxa de transmissão nas faixas de 2.4GHz, 915MHz e 868MHz pode chegar a 250kbits/s, 40 kbits/s e 20kbits/s, respectivamente.

Existem três tipos de dispositivos Zigbee diferentes, o coordenador, o roteador e o dispositivo final. O coordenador (*Zigbee coordinator*) é o dispositivo que inicia e armazena as informações da rede, funcionando também como centro de confiança (*trust center*), ou seja, como nodo confiável. A conexão entre duas redes Zigbee é controlada pelos dispositivos coordenadores.

O roteador (*Zigbee router*), além de executar aplicações de usuário, é capaz de agir como um nodo intermediário retransmitindo dados de outros dispositivos. O dispositivo final (*Zigbee end device*) é o mais simples de todos, contendo somente a estrutura necessária para se comunicar com o coordenador ou com o roteador. Esse comportamento aumenta a vida útil da bateria do dispositivo final, pois permite que ele permaneça no modo de hibernação por mais tempo.

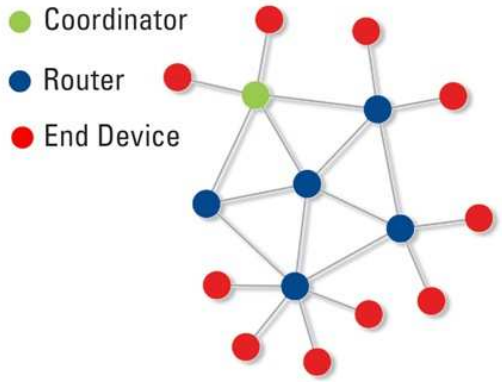


Figura 4 Topologia de uma rede mesh ZigBee.[2]

A especificação do Zigbee, regulada pela *Zigbee Alliance*, define três camadas na estrutura do protocolo. A camada superior é a camada de rede, construída sobre a camada MAC, ela realiza o roteamento e provê o acesso à rede para a camada superior.

As redes Zigbee são utilizadas para auxiliar no controle e na automação de plantas industriais e de casas inteligentes, substituindo as redes cabeadas. Porém, são pouco usadas para construção de redes de propósito geral, pois possuem baixa taxa de transferência. Além disso, o tempo para a construção e a organização da rede é muito extenso.[14]

2.1.3 802.11

A família de padrões IEEE 802.11 define as estruturas básicas para a construção de uma rede local sem fio (WLAN). A estrutura básica do protocolo define políticas de controle e de acesso ao meio físico, o ar, e corresponde às camadas física e de enlace do modelo OSI. As redes Wi-Fi são dividas em: infraestruturadas e ad hoc. As redes infraestruturadas são construídas baseadas em pontos de acesso como gerenciadores da comunicação e reguladores da rede, enquanto que nas redes ad hoc todos os dispositivos podem operar como pontos de acesso, provendo a comunicação com a rede.

O Wi-Fi opera na faixa de frequência entre 2.4GHz a 2.5GHz, definindo 14 canais de operação, espaçados de 5MHz, sendo o primeiro canal centrado em 2.412GHz. As transmissões são multiplexadas no tempo e, quando uma colisão é detectada, ambos colisores realizam um algoritmo de *backoff*, esperando um tempo randômico antes de transmitir novamente. Diversas redes podem coexistir operando sobre o mesmo canal, realizando uma divisão entre o tempo de transmissão de ambas as redes.

No nível de enlace, um frame é definido para a transmissão de dados. Os frames são divididos em frames de dados e de gerenciamento. As informações que caracterizam uma rede sem fio são transmitidas nos frames de gerenciamento, sendo estes transmitidos pelo ponto de acesso. Uma rede 802.11 possui um canal de operação, um identificador de rede (SSID) e um ponto de acesso. Nas redes sem infraestrutura, todas as estações podem servir de ponto de acesso, neste caso o identificador de rede é compartilhado entre as estações.

[2]Imagem retirada de Iboun Sylla: http://www.eetimes.com/document.asp?doc_id=1276404. Acesso em: 12/11/2013

2.2Protocolos de Roteamento para redes Ad Hoc

Os protocolos de roteamento desenvolvidos especificamente para as redes ad hoc possuem a mobilidade e o consumo de energia como pontos principais. Os protocolos são classificados, basicamente, como proativos ou reativos [3]. Os protocolos reativos operam de acordo com a necessidade de transmissão, aguardando até que existam dados a serem enviados para então começarem a operar, enquanto os protocolos proativos verificam constantemente alterações na topologia da rede para que sempre exista uma rota válida disponível para transmitir os dados.

2.2.1 Classificação

2.2.1.1 Protocolos Proativos

Os protocolos de roteamento para redes cabeadas verificam constantemente alterações na topologia da rede e atualizam seu estado interno para refletir o estado atual da topologia. O estado interno desses protocolos é, geralmente, representado por tabelas, contendo, para cada destino, o próximo vizinho que deve ser visitado. Da mesma forma, os protocolos proativos de redes ad hoc mantêm as informações atualizadas da rede, trocando mensagens mesmo quando não há necessidade de comunicação entre os nodos.

Os protocolos proativos mantêm tabelas com a informação de roteamento para cada par de nodos da rede. A atualização da tabela de roteamento, bem como a atualização do estado da rede, ocorre em intervalos de tempo pré-definidos, ocasionando um consumo constante da banda disponível. Os nodos podem manter um grande conjunto de rotas na tabela de roteamento, mas só utilizar uma pequena parcela, tornando desnecessária a energia e a banda consumida para a aquisição dessas rotas.

Pelo fato de todos os nodos possuírem rotas para todos os destinos da rede antes de o usuário solicitar uma comunicação, o atraso para estabelecimento da comunicação é baixo. Porém, o consumo de energia de cada nodo é maior, pois mesmo que as aplicações do usuário estejam ociosas, a tarefa de descoberta de rede continuará ativa. A quantidade de banda da rede consumida para manutenção do protocolo, mesmo que constante, é maior que das outras classes de protocolos, e esse consumo está presente em todas as etapas do protocolo.

2.2.1.2 Protocolos Reativos

Os protocolos reativos caracterizam-se por não iniciarem o roteamento enquanto não for necessário, ou seja, as rotas só serão descobertas quando o usuário necessitar. O funcionamento desses protocolos é dividido em duas etapas: a descoberta de rota e a manutenção da rota. A descoberta de rota só é iniciada quando o usuário tenta enviar dados para um destino. Até então, o nodo participante da rede ad hoc estaria somente escutando o tráfego da rede e respondendo a solicitações de rota ou reencaminhando pacotes.

O consumo de energia e de banda da rede tendem a ser menores nos protocolos reativos, pelo fato de que eles não realizam envios repetidamente e podem ficar longos períodos sem enviar pacotes para a rede, dependendo do comportamento do usuário e dos vizinhos. Porém, o tempo entre o início de uma descoberta e o início do envio dos dados é maior que o dos protocolos proativos, pois toda a rede pode ser visitada antes de uma rota ser definida.

Uma descoberta de rota pode ocasionar uma procura global na rede, atingindo todos os nodos. Alguns protocolos implementam uma busca em expansão, incrementando o *time to live* (TTL) do pacote IP até atingir o destino.

Para evitar o armazenamento de rotas inválidas, que não refletem mais o estado da topologia, as rotas são descartadas após um tempo sem uso. Assim, as rotas podem ser invalidadas por timeout ou por erros na transmissão.

2.2.1.3 Outras classes de protocolos

Existem diversas classes de protocolos para rede ad hoc sem fio, como os híbridos e os hierárquicos, contudo essas classes são mais utilizadas para a solução de problemas específicos, nos quais os nodos não suportam atualizações, como as redes de sensores. Outras classes de protocolos são: híbridos, geográficos e hierárquicos.

Os protocolos híbridos utilizam as características dos protocolos proativos e dos reativos, dividindo a rede em duas partes: zona local e não local. Dentro da zona local, o protocolo opera de forma proativa, mantendo atualizadas as informações sobre os seus vizinhos e os vizinhos deles. A comunicação entre zonas ocorre de forma reativa e só é disparada quando existe a necessidade de comunicar diferentes zonas entre si.

2.2.2 Ad Hoc On-demand Distance Vector (AODV)

O AODV é um protocolo de roteamento para redes ad hoc multissaltos que opera de forma reativa por meio de tabelas de roteamento [8], [16]. Essas tabelas contêm as informações dos destinos conhecidos, juntamente com o número de saltos até os destinos e um identificador único para cada rota. Para realizar a descoberta de novas rotas, o protocolo define as mensagens de solicitação de rota (RREQ), de resposta de rota (RREP) e de erro na rota (RERR).

O AODV não realiza nenhuma busca por rotas enquanto não existir um fluxo de dados a ser enviado para um destino não conhecido, caracterizando um protocolo sob demanda. Um nodo procurando uma rota para um destino espalha mensagens de RREQ na rede e espera o recebimento de um RREP para essa solicitação de rota. Quando um nodo recebe um RREQ, este pode reencaminhar o pacote, caso não seja o destino da solicitação, ou responder um RREP, caso saiba uma rota para o destino ou caso ele mesmo seja o destino da solicitação.

A manutenção das rotas acontece durante a troca de dados entre os nodos. Caso algum pacote não possa ser enviado para um próximo nodo, o nodo que está tentando enviar o pacote retorna uma mensagem de RERR para os nodos precursores dessa entrada da tabela de roteamento. Essa entrada contém o endereço e o número de sequência da fonte e do destino da rota. Também contém o próximo nodo para atingir o destino e uma lista de nodos que utilizam essa entrada da tabela, denominados nodos precursores (precursor nodes).

2.2.2.1 Estruturas Básicas

Para o correto funcionamento do AODV, um nodo deve possuir um número de sequência, uma tabela de roteamento e uma lista de nodos precursores. Além dessas estruturas, um nodo deve ser capaz de receber e transmitir mensagens do tipo RREQ, RREP, RERR e RREP-ACK.

Uma mensagem do tipo *route request* (RREQ) contém o identificador da solicitação, o endereço do originador e do destino e o número de sequência do destino. O identificador da solicitação é composto pelo número de sequência e pelo endereço do

originador. O campo de *time to live* (TTL) da mensagem é utilizado nas procuras do tipo *expanding ring search*. Esse tipo de procura espalha RREQ, incrementando o TTL até atingir o nodo destino.

A mensagem de resposta de um RREQ é um *route reply* (RREP), que contém o total de saltos entre o nodo originador do RREQ e o nodo destino, o endereço do originador e do destino, o número de sequência do destino e o intervalo de tempo durante o qual esta rota deve ser considerada válida. Por fim, quando uma ligação entre dois nodos de uma rota deixa de funcionar, uma mensagem de *route error* (RERR) é gerada contendo o número de saltos que não serão atingidos, o endereço do nodo não atingível e o número de sequência do destino da rota.

Todo nodo mantém um número de sequência (*sequence number*) utilizado para controlar a quantidade de mensagens de RREQ e RREP que circulam pela rede e para evitar *loops* em uma rota. Um nodo deve incrementar seu *sequence number* antes de realizar uma nova descoberta de rota e, quando originando um RREP, deve escolher o maior *sequence number* entre o seu e do RREQ relacionado.

Uma tabela de roteamento deve existir em todos os nodos da rede, contendo, para cada rota registrada, o endereço do destino da rota, o *sequence number* do destino, o total de saltos entre este nodo e o destino, o próximo salto, a lista de precursores e o tempo de vida que esta rota possui. Para cada destino, só deve existir uma rota na tabela de roteamento, ou seja, a tabela de roteamento do AODV não armazena múltiplas rotas para um destino. Todas as entradas da tabela possuem um tempo de expiração relacionado. Cada vez que uma entrada é utilizada, o seu time out é atualizado com o instante de tempo atual mais um *offset*. O *offset* pode ser estático, definido por uma constante em tempo de compilação, ou dinâmico, calculado durante a execução do protocolo.

2.2.2.2 Descoberta de Rota

A descoberta de rota envolve a criação e disseminação de *route requests* e a criação de *route replies*. A criação de RREQ só acontece quando um nodo possui dados a serem enviados para um destino e não possui nenhuma rota válida na sua tabela de roteamento. Antes de disseminar o RREQ, o nodo deve armazenar a solicitação com um *time out*, que é o tempo máximo que uma descoberta de rota pode durar. Assim, o nodo que originou um RREQ não propaga a requisição quando recebê-la dos seus vizinhos e reinicia a descoberta quando passar o tempo limite de uma descoberta de rota.

O mecanismo de disseminação de RREQ padrão no AODV é a pesquisa em expansão de anel (*expanding ring search*). O limite de tentativas de descoberta pode ser estático, definido em tempo de compilação, ou dinâmico, definido durante a execução do protocolo.

Um nodo, ao receber um RREQ, adiciona uma rota para o emissor do RREQ, não necessariamente a origem do RREQ, se ainda não possui uma rota válida para esse nodo. Em seguida, analisa se já processou esse RREQ, comparando com as últimas requisições armazenadas. O nodo descarta o pacote caso já tenha processado a requisição.

Ao receber um RREQ válido, o nodo atualiza o *sequence number* da fonte e do destino da requisição, caso sejam maiores do que os armazenados na tabela de roteamento, e verifica se é o destino, ou se conhece uma rota para o destino da

requisição. Não sendo o destino da requisição, o nodo atualiza o *hop count* e realiza o *broadcast* da requisição.

Uma mensagem de RREP é enviada quando o nodo é o destino de um RREQ ou possui uma rota para o destino. Caso o nodo que esteja respondendo ao RREQ seja um nodo intermediário, esse nodo deve verificar se a *flag* de *Gratuitous RREP* (GRREP) está ligada. O *Gratuitous RREP* é uma mensagem de RREP enviada automaticamente para o destino de um RREQ, para diminuir o excesso de comunicação quando o destino nunca recebe um RREP pois os nodos intermediários respondem por ele. O uso do GRREP garante que o destino de um RREQ possua uma rota para o originador do RREQ.

2.2.2.3 Manutenção de Rota

Os nodos podem utilizar mensagens de HELLO para confirmarem a ligação com seus vizinhos. Essas mensagens são utilizadas quando o nodo possui rotas ativas na sua tabela de roteamento. Assim, o nodo pode garantir que a sua ligação com o vizinho não irá expirar ou descobre antecipadamente se perdeu conexão com algum vizinho.

Uma alternativa ao uso de mensagens de HELLO que possui o mesmo efeito é enviar mensagens de RREQ para o vizinho solicitando uma rota para esse vizinho. Caso o vizinho não responda ao RREQ após o time out da descoberta de rota, é porque esse vizinho não está mais acessível e sua entrada deve ser removida da tabela de roteamento.

Além dessas opções, um nodo pode receber confirmações de envio das camadas de comunicação mais inferiores. Independentemente da forma em que a conectividade com os vizinhos (conectividade local) é confirmada, todos os nodos devem manter atualizadas essas informações na tabela de roteamento.

Quando uma falha na conexão local é verificada, uma mensagem de erro deve ser enviada para todos os nodos registrados como precursores dessa entrada. Ao receber uma mensagem de erro, um nodo deve atualizar a sua tabela de roteamento de acordo com os dados recebidos na mensagem e retransmitir a mensagem, caso possua nodos precursores nas entradas atualizadas.

2.2.3 Dynamic Source Routing (DSR)

O Dynamic Source Routing [2], [6], ou roteamento dinâmico baseado na fonte, é um protocolo de roteamento para redes ad hoc reativo e iniciado na fonte. No contexto do DSR, a expressão rede ad hoc está relacionada a uma rede não centralizada, não dependente de infraestrutura e de multissaltos. O seu funcionamento está relacionado à atividade da origem, pois enquanto não transmitir pacotes não irá enviar dados para a rede. No momento em que a origem transmitir dados para um determinado destino, o protocolo iniciará a busca por uma rota para esse destino e, em seguida, encaminhará os dados para o destino utilizando a rota descoberta.

O DSR possui duas etapas de funcionamento: a descoberta e a manutenção da rota. A descoberta da rota só é iniciada quando surge um novo conjunto de dados para serem transmitidos a um destino para o qual não existe uma rota conhecida. Assim, quando uma descoberta de rota é iniciada, todos os pacotes transmitidos pela origem são armazenados até uma nova rota estar disponível. Esses pacotes são descartados caso não sejam transmitidos até um determinado tempo limite.

Durante a transmissão dos dados por meio de uma rota, o protocolo realiza a manutenção dessa rota, para evitar que um novo processo de descoberta de rota seja iniciado por falhas na transmissão. Para isso, cada nodo que encaminha pacotes de uma comunicação é responsável por confirmar a transmissão desses pacotes entre ele e o próximo nodo, reenviando os pacotes até um número máximo de vezes ou encaminhando uma mensagem de erro para a origem da comunicação.

A descoberta de rota utiliza dois tipos de mensagens, o pedido de rota (*route request*) e a resposta de rota (*route reply*), referenciadas como RREQ e RREP, respectivamente. Para a manutenção de rota, existem as mensagens de erro de percurso (*route error*) e de confirmação de recebimento (*acknowledgment*), definidas, respectivamente, para divulgar problemas nas rotas e para confirmar o recebimento de dados.

Uma implementação simplificada do DSR pode conter somente essas duas etapas, de descoberta e de manutenção, e iniciar uma descoberta de rota para cada novo fluxo de dados a ser transmitido, e confirmar a retransmissão dos dados a cada salto. Nessa implementação, o atraso no início da transmissão dos fluxos será constante durante todo o tempo de execução da aplicação, pois para cada novo fluxo todas as etapas do DSR deverão ser executadas. Para uma implementação mais complexa e otimizada, o protocolo define estruturas de dados auxiliares e outras etapas entre a descoberta da rede e a manutenção da rede. A definição do protocolo prevê otimizações no comportamento básico do DSR, alterando a maneira como responde a determinados pacotes e adicionando novas etapas no funcionamento.

2.2.3.1 Estruturas de dados auxiliares

Para cada etapa do protocolo DSR existe uma ou mais estruturas de dados definidas para auxiliar e otimizar a sua execução. A etapa de descoberta de rota utiliza um *buffer* de envio (*Send Buffer*), para armazenar os pacotes que ainda não podem ser enviados por não existir uma rota para o destino, e uma tabela de *route request*, para controlar o início e o recebimento de solicitações de rota. Durante a manutenção da rota, os pacotes que aguardam a confirmação de recebimento do próximo nodo são armazenados no buffer de manutenção (*Maintenance Buffer*) e, dependendo da implementação do protocolo, pode ser utilizada uma fila para enviar os pacotes para a interface de rede (*Network Interface Queue*).

2.2.3.1.1. Send Buffer

Todo nodo que implementa o DSR possui um *send buffer*, que é uma fila de pacotes que não podem ser enviados pelo nodo, pois este ainda não possui uma rota para o destino do pacote. Cada pacote adicionado ao *send buffer* está relacionado com o momento no qual foi adicionado, e existe um tempo máximo em que o pacote pode permanecer no *buffer* e, após esse tempo, o pacote deve ser removido. O *send buffer* é implementado como uma fila FIFO (*first in, first out*), que remove as entradas mais antigas para liberar espaço para as entradas mais recentes. Assim, previne-se que ocorra um estouro no *send buffer*.

2.2.3.1.2. Route Cache

A cache de rotas armazena as rotas aprendidas por um nodo, por meio de um processo de descoberta de rota ou de captura de pacotes e leitura dos cabeçalhos. O nodo remove as rotas aprendidas após um determinado tempo limite ou quando recebe mensagens de erro informando um defeito em alguma rota. A estrutura interna da cache

de rota pode ser definida de maneiras diferentes, como apresentado na seção 4.1 da sua RFC.

No contexto deste trabalho, considera-se que a *route cache* armazena como rota a sequência de nodos entre a fonte e o destino. Também considera-se que a cache possui mais de uma rota para cada destino e um número máximo de destinos armazenados, removendo sempre a rota que está há mais tempo sem ser utilizada.

2.2.3.1.3. Route Request Table

A tabela de RREQ armazena três diferentes informações: os RREQ realizados pelos vizinhos, os RREQ realizados pelo nodo e o contador de RREQ do nodo. Os *requests* provenientes dos vizinhos são armazenados numa fila FIFO de tamanho fixo que contém o endereço do originador e o identificador único do RREQ.

São armazenados o TTL do RREQ iniciado pelo nodo, o instante de tempo que o nodo enviou um RREQ para este destino pela última vez, o número de tentativas consecutivas de realizar uma descoberta de rota para este destino e o tempo restante até que o nodo comece uma nova descoberta de rota. Utilizando as informações armazenadas nesta tabela, um nodo implementa um mecanismo de afastamento (*back-off*) que limita o número de descoberta de rotas iniciadas por um nodo durante um intervalo de tempo.

2.2.3.1.4. Maintenance Buffer

O *buffer* de manutenção é composto por uma fila de pacotes que aguardam confirmação de recebimento do próximo salto. Para cada pacote são armazenados o número de retransmissões e o instante de tempo da última retransmissão. Um nodo adiciona os pacotes no *buffer* de manutenção depois da primeira tentativa de transmissão do pacote.

Depois de um número máximo de retransmissões, o pacote deve ser descartado. Após o descarte do pacote, um nodo pode optar por enviar uma mensagem de erro para a origem, informando a falha na rota, ou tentar salvar o pacote.

O salvamento de pacote no *buffer* de manutenção consiste em procurar a existência de outra rota para esse destino na *route cache* do nodo e retransmitir o pacote usando uma outra rota. O salvamento de pacotes no *buffer* de manutenção não inicia uma nova descoberta de rede.

2.2.3.2 Descoberta de Rota

A etapa de descoberta de rota é realizada quando um nodo necessita enviar um ou mais pacotes para um destino que não está armazenado na *route cache*. Assim, o nodo armazena os pacotes no *send buffer* e prepara um pacote de RREQ, contendo o endereço do nodo de origem e de destino e um identificador único gerado pela *route request table*. Em seguida, o nodo envia o RREQ para todas as suas interfaces de rede e armazena o RREQ na *route request table*. O nodo retransmite o RREQ de acordo com o limite de *requests* por intervalo de tempo que a *route request table* impõe.

Quando um nodo recebe um RREQ, ele deve analisar a validade do identificador para, em seguida, armazená-lo na *route request table*. Caso seja o destino do RREQ, o nodo deve responder a um RREP com uma rota válida. A rota de RREP pode ser a rota inversa do RREQ ou o nodo destino pode iniciar uma nova descoberta de rede caso os enlaces de comunicação não sejam bidirecionais. No contexto deste trabalho, considera-se que os enlaces são bidirecionais, ou seja, se uma rota da origem para o destino

funciona, a mesma rota do destino para a origem também deve funcionar. Se o nodo não for o destino do RREQ, ele deve inserir seu endereço na lista de nodos percorridos e retransmitir o RREQ para todas as suas interfaces de rede.

Ao receber um RREP, o nodo deve verificar se ele é o destino do RREP. Se não for o destino, deve retransmitir o RREP utilizando a lista de nodos do pacote. Se o nodo for o destino do RREP, ele deve atualizar a sua *route cache* e tentar retransmitir os pacotes do *send buffer*.

2.2.3.3 Manutenção da Rota

A manutenção de rota ocorre durante a transmissão dos pacotes entre dois nodos. A confirmação do funcionamento da rota é feita nodo a nodo. No momento em que um nodo descobre que o próximo *hop* está inatingível, ele envia uma mensagem de *route error* para o nodo originador da transmissão. Todos os nodos que receberem o pacote de *route error* atualizam a sua *cache* de rotas. A transmissão do pacote de erro pode ser feita por meio de uma rota para a origem presente na *route cache* ou revertendo o caminho do pacote que se tentou transmitir.

A confirmação de recebimento pelo próximo nodo pode acontecer das seguintes formas: a camada de enlace do nodo que está enviando os dados pode repassar para o nível do DSR uma confirmação de envio, ou o nodo pode operar em modo promíscuo, escutando todos os pacotes que estão circulando na rede, inclusive se o próximo nodo reenviou o pacote, ou o nodo pode enviar e receber confirmações no nível do DSR, utilizando os pacotes de confirmação (*acknowledgement*). A confirmação pela camada de enlace deve ser utilizada prioritariamente, caso esta forma não esteja disponível na implementação do DSR, deve-se tentar usar o modo promíscuo da aquisição de pacotes. Em último caso, a confirmação de envio no nível do DSR deve ser utilizada, por ser a menos eficiente.

2.2.4 Definição da rede ad hoc implementada

A capacidade de uma rede ad hoc sem fio está diretamente relacionada com a capacidade dos nodos que vão operar nessa rede. Uma rede local formada por computadores *desktop* suporta altas velocidades de transmissão e é facilmente roteada, porém os nodos são estáticos e não possuem restrições de consumo de energia. Normalmente, os nodos de uma rede ad hoc possuem diversas restrições, sejam de energia ou de capacidade de processamento.

Uma rede de sensores criada por meio de uma rede ad hoc possui uma séria restrição de consumo de energia, uma vez que não serão recarregadas após serem distribuídas no ambiente. No entanto, uma rede de *smartphones* possui uma restrição de consumo reduzida, porque os proprietários dos dispositivos podem recarregá-los diariamente. Assim, o consumo pode ser calculado considerando uma vida útil de uma dia, ao invés de vários anos, como no caso da rede de sensores.

Considera-se dispositivo móvel sem fio todo aparelho que dependa de bateria e consiga se comunicar por meio de alguma interface de rede sem fio. Um notebook que possui uma interface de rede Wi-Fi pode ser considerado um dispositivo móvel. Contudo, exemplos mais cotidianos de dispositivos móveis são os *smartphones* e os *tablets*.

As interfaces de rede sem fio dos *smartphones* e dos *tablets* são, geralmente, Wi-Fi e Bluetooth. Ambas interfaces de rede possuem suporte nativo para a construção de redes

ad hoc de um único nível, sendo necessária a implementação de um protocolo de roteamento para a construção de uma rede ad hoc multinível.

Microcontroladores do tipo Arduino também podem compor a rede ad hoc, pois existe suporte para interfaces de rede sem fio por meio de *shields*, placas de hardware projetadas especialmente para fornecer um serviço a uma placa Arduino. Existem *shields* de, praticamente, todas as tecnologias de transmissão existentes no mercado, principalmente Wi-Fi e Bluetooth.

O endereçamento por meio de UUID, realizado no nível de rede, permite que dispositivos com diferentes interfaces de rede consigam comunicar-se por meio de outro dispositivo que possua ambas interfaces. Essa abordagem permitirá, por exemplo, a comunicação entre um microcontrolador Arduino equipado com Bluetooth e um notebook equipado com Wi-Fi, por meio de um smartphone equipado com ambas tecnologias.

3 IMPLEMENTAÇÃO DO PROTOCOLO

Para a realização deste trabalho serão implementadas duas interfaces de rede, uma Wi-Fi e outra virtual. A interface virtual foi desenvolvida para realizar simulações e testes da implementação e será explicada no próximo capítulo. Assim, a mesma estrutura do exemplo anterior do Arduino pode ser repetida utilizando dois notebooks, um com interface Wi-Fi e outro com interface Wi-Fi e interface virtual.

O DSR será utilizado neste trabalho por possuir muitas estruturas opcionais na sua definição, permitindo o desenvolvimento do trabalho por etapas, com cada etapa gerando uma versão funcional do protocolo. O fato de o DSR armazenar mais de uma rota para cada destino permite que implementações futuras sejam capazes de transmitir dados entre dois nodos utilizando duas interfaces de rede diferentes. Assim, a escolha do protocolo a ser implementado não considera eventuais diferenças de desempenho existentes entre as opções apresentadas no capítulo anterior.

Com isso, na construção da rede ad hoc, utilizou-se o protocolo *Dynamic Source Routing* para realizar o roteamento e abstraiu-se o meio de transmissão. A implementação do protocolo foi realizada utilizando a linguagem de programação Java e o ambiente de programação Eclipse.

Por ser uma implementação voltada para o mercado de dispositivos móveis, a linguagem de programação escolhida deve compilar para diferentes plataformas, principalmente para *smartphones* e notebooks, e deve suportar o uso de orientação a objetos, pois o protocolo será implementado de forma didática, para que seja possível utilizar o código fonte para aprofundar os estudos sobre a estrutura do protocolo. E, para permitir o desenvolvimento do protocolo por etapas, a linguagem de programação utilizada também deve suportar o uso de testes automatizados.

Neste trabalho, foi utilizada a linguagem de programação Java, por ser compilável para os sistemas operacionais dos dispositivos de teste Android e Windows e por ser uma linguagem orientada a objetos, que permite a construção do protocolo por meio de módulos testáveis e didáticos. Além disso, existe o *framework* JUnit [17] para testes automatizados do código que será executado nos *notebooks* e o AndroidJUnit [18], que permite a realização de testes automatizados nos *smartphones* Android.

DSR é o protocolo de roteamento utilizado por permitir a abstração das camadas inferiores da rede, facilitando a criação de um endereçamento no nível de roteamento, pois a sua estrutura não depende de características ou serviço providos pelas camadas inferiores. Os protocolos que dependem diretamente do uso do endereçamento IP, por exemplo, obrigam que as outras interfaces de rede também suportem o endereçamento IP.

As características do comportamento do DSR também foram consideradas durante a escolha. Por ser um protocolo reativo e sob demanda, espera-se que durante a sua execução o consumo de energia permaneça baixo, pois o protocolo só será executado nos momentos em que existirem demanda, sendo capaz de entrar em longos períodos de hibernação.

O funcionamento do DSR depende de estruturas auxiliares, permitindo ao programador construí-lo de forma iterativa, implementando e realizando testes unitários com uma estrutura de cada vez. Assim, durante a construção da estrutura principal do protocolo, sabe-se que as estruturas auxiliares estão operando de acordo com a especificação.

Espera-se que, durante a execução do protocolo, exista uma grande troca de mensagens nos períodos de configuração da topologia, com dispersão de mensagens de RREQ e RREP e baixa troca de mensagens de configuração durante a transmissão dos dados de usuário, pelo fato de a rota ser enviada com os dados e por não necessitar transmitir mensagens de manutenção das rotas periodicamente.

A capacidade do protocolo de suportar o roteamento em diferentes tecnologias de transmissão de dados aumenta o número de dispositivos capazes de executar a aplicação e também provê ao protocolo a capacidade de distribuir a carga da transmissão de dados por meio de diferentes interfaces de redes, permitindo diminuir o congestionamento de uma interface.

A abstração do meio de transmissão possibilitou implementar interfaces virtuais para executar o protocolo em uma única máquina, permitindo simular o mesmo código que será executado na rede real. Muitas ferramentas de simulação necessitam de grandes trechos de códigos específicos da ferramenta ou definem um ambiente de simulação que difere do ambiente real no qual o protocolo será executado. Assim, aproveitou-se a facilidade de implementação de interfaces do protocolo para construir um ambiente de simulação em uma única máquina.

Na primeira seção do capítulo, é abordada a alteração realizada no protocolo a fim de permitir o endereçamento no nível de rede. Em seguida, é apresentada a implementação do protocolo, mostrando como as classes foram construídas e conectadas e de que maneira as estruturas auxiliares foram implementadas e testadas. Na terceira seção, é apresentado o ambiente de testes construído.

Na quarta seção é explicado como a interface do protocolo para redes Wi-Fi foi construída e como implementar outras interfaces para o protocolo. Ao final do capítulo, é analisada a estrutura final construída para implementar o protocolo DSR.

A rede ad hoc implementada é composta por um conjunto de dispositivos que implementam o protocolo DSR. Cada dispositivo da rede pode possuir um ou mais nodos, representados pela estrutura *INode* e possui um identificador único (UUID). Um dispositivo pode ter uma ou mais interfaces, todas conectadas ao roteador do dispositivo.

3.1Adaptação do Dynamic Source Routing

O DSR implementado neste trabalho possui algumas características diferentes da especificação original, definida em [2]. Foi alterado o modo de endereçamento dos nodos, antes usava-se o endereço IPv4 da interface de rede, agora usa-se o *Universal*

Unique Identificator (UUID) do nodo e foi alterado o nível da camada OSI na qual o protocolo será executado.

O UUID, definido em [5], possui 128 bits (16 *bytes*) e é amplamente difundido entre diversas plataformas, desde microcontroladores Arduino até notebooks. O UUID gerado randomicamente é utilizado para identificar de forma única recursos ou dispositivos. Alguns produtos de bancos de dados utilizam implementações específicas do UUID como identificadores únicos das entradas no banco de dados.

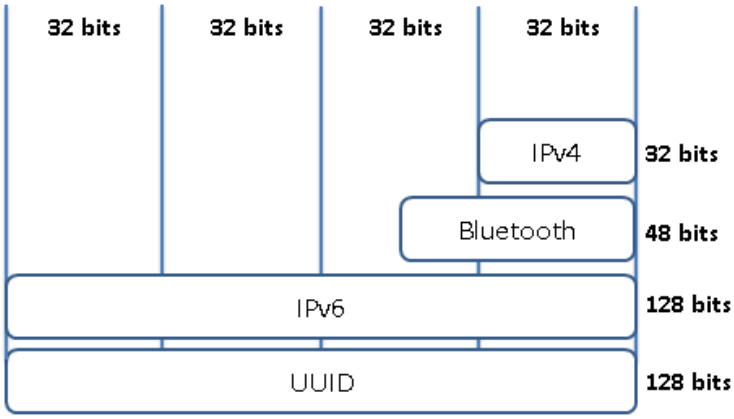


Figura 5 Tamanho, em bits, dos diferentes modos de endereçamento

Cada elemento da rede possui um identificador único que pode ser definido de forma dinâmica ou estática. A identificação dinâmica provê uma baixa probabilidade de dois elementos de uma mesma rede possuírem o mesmo identificador. No estado atual da implementação não existe um mecanismo de controle de conflitos de identificação, porém, a probabilidade de conflitos de endereçamento é baixa o suficiente para não interferir no funcionamento do protocolo.

O usuário que tiver interesse em configurar a sua rede e souber antecipadamente quantos dispositivos formarão a rede, poderá endereçar os dispositivos de forma estática, contudo estará assumindo a responsabilidade de controlar os conflitos de endereçamento que poderão ocorrer.

A especificação original do DSR define que o protocolo seja utilizado no mesmo nível do endereçamento IP, nível de rede do modelo OSI, utilizando os endereços para realizar o roteamento dos pacotes. Assim, uma implementação completa do DSR poderia adicionar ao nível IP as funcionalidades do protocolo. Contudo, seria necessário implementar *drivers* das interfaces de rede e alterar o funcionamento dos dispositivos para suportarem o protocolo. O objetivo do trabalho é a análise e a implementação do protocolo DSR assim, o protocolo será executado no nível aplicação, permitindo que o código gerado possa executar em diferentes dispositivos. A fim de realizar testes em ambiente real, o protocolo foi executado sobre a pilha UDP/IP, utilizando endereços de broadcast em redes infraestruturadas.

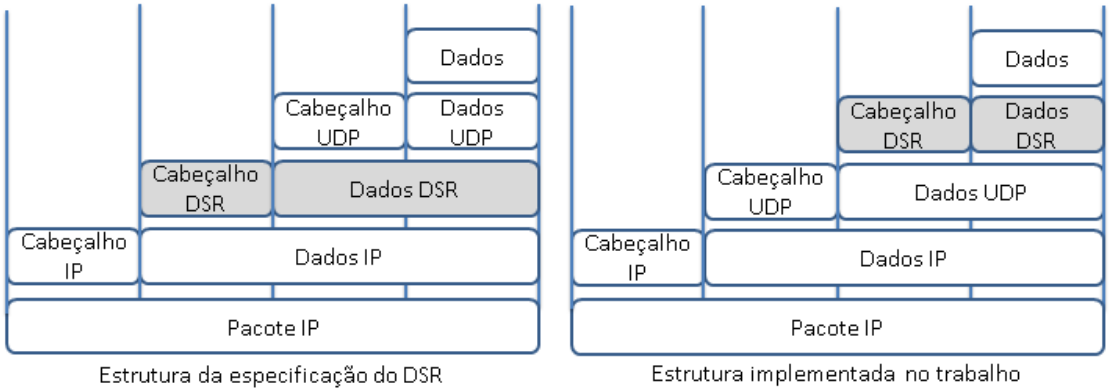


Figura 6 Composição de um pacote IP

3.2Estrutura dos Dispositivos

Cada dispositivo participante da rede ad hoc sem fio possui um roteador que implementa o protocolo DSR, um conjunto de interfaces do protocolo, que realizam a transferência dos dados do roteador para o meio físico, e zero ou mais nodos, utilizados pelos usuários da aplicação. Existe um UUID definido para cada nodo e interface do protocolo existente no dispositivo. Esse identificador é utilizado como endereço na rede ad hoc. Um conjunto não repetido de identificadores define uma rota.

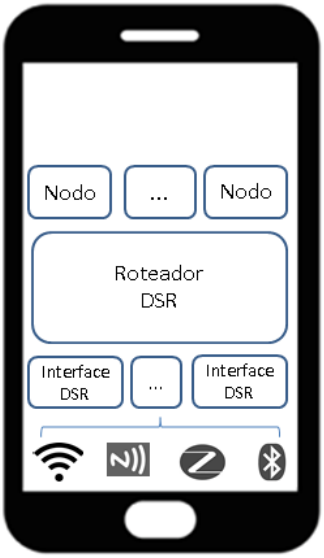


Figura 7 Detalhe de um dispositivo que implementa o protocolo DSR

3.3Implementação das Estruturas

Para a implementação do DSR foram construídas as estruturas de dados auxiliares, seguidas dos conjuntos de cabeçalhos. Após, implementou-se o módulo de roteamento, que faz uso das estruturas implementadas.

Tendo em vista que a aplicação de roteamento será executada em diferentes plataformas, possuindo diferentes características de hardware, decidiu-se construí-la de forma modular, para facilitar a troca de módulos dependendo da arquitetura alvo. A aplicação de roteamento foi dividida em três níveis: usuário, roteamento e transmissão. O nível de usuário possui as estruturas que são acessadas pelos usuários da aplicação. A

lógica de controle e descoberta de rotas está no nível de roteamento e, no nível de transmissão, está o controle de transmissão específico para cada tecnologia sem fio.

No nível de usuário foi definido o tipo *INode*, que provê a funcionalidade de enviar e receber dados na forma de *array* de *bytes*. Assim, o usuário que utilizar esta implementação deverá instanciar objetos do tipo *INode* e enviar e receber dados por meio dos seus métodos. A definição estática do identificador também é realizada usando os métodos de configuração do *INode*.

Todas as classes das estruturas auxiliares e dos cabeçalhos do protocolo estão definidas no nível de roteamento junto com a classe Router, que é responsável pela descoberta e manutenção de rotas. A comunicação do nível de roteamento com os outros dois níveis ocorre por meio da interface *IDSRInterface*, que define quais métodos uma interface de transmissão do protocolo deve implementar. Com o intuito de facilitar o desenvolvimento, considerou-se que o *INode* seria, do ponto de vista do Router, uma *IDSRInterface*, normalizando o comportamento do *Router* para que toda a sua comunicação seja feita por meio de uma única interface.

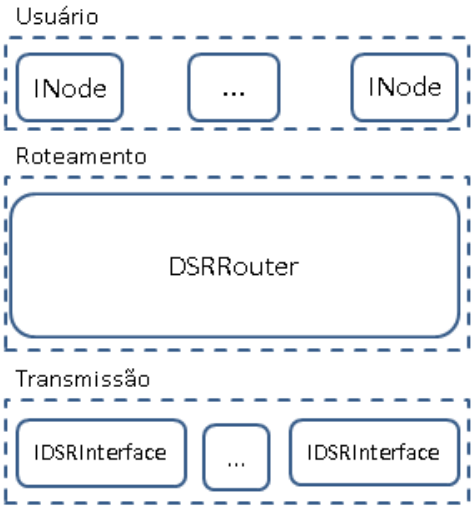


Figura 8 Níveis da implementação do protocolo DSR

A implementação do protocolo começou pelas classes das estruturas auxiliares e dos cabeçalhos. Em seguida, foram implementadas as classes que respondem à interface *IDSRInterface* e, então, foi desenvolvida a classe de roteamento Router.

3.3.1 Cabeçalhos

O DSR utiliza dois tipos de cabeçalhos: o fixo e o de opções. O cabeçalho fixo contém o tipo do próximo cabeçalho (campo *next header*), o tamanho total do cabeçalho DSR e a lista de cabeçalhos de opções. Os cabeçalhos de opção são separados pelo seu tipo, podendo ser de solicitação de rota (*route request option*), de resposta de rota (*route reply option*), de erro na rota (*route error option*), de solicitação de recebimento (*acknowledgement request option*), de confirmação de recebimento (*acknowledgement option*) e de rota da fonte (*source route option*).

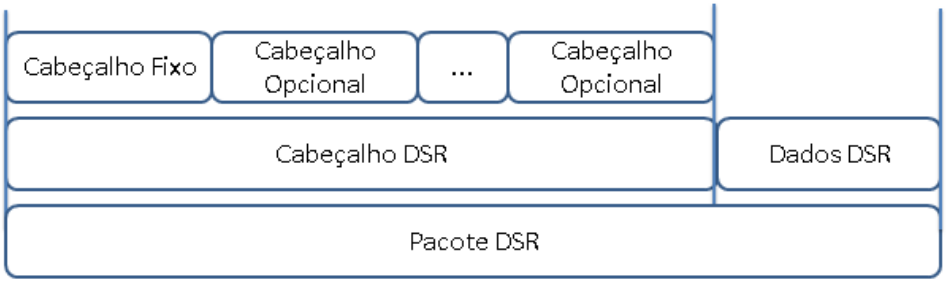


Figura 9 Estrutura de um pacote do DSR

Um pacote do DSR contém os dados que devem ser transmitidos, um cabeçalho que contém as opções e a origem e o destino da transmissão. Para que seja possível transferir um pacote DSR pela rede e para que o receptor do pacote saiba interpretar os dados recebidos criou-se uma interface de serialização, o *IDSRSerializable*. Toda classe que implementa esta interface é capaz de transformar seus dados em um *array* de *bytes* e de preencher os campos da classe a partir de um *array* de *bytes*.

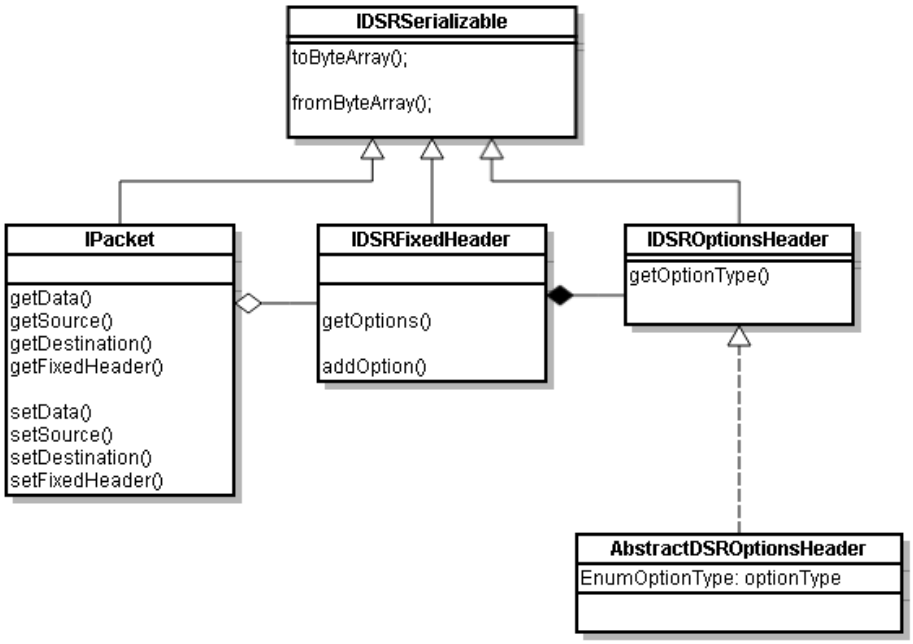


Figura 10 Diagrama de classes para representar um pacote DSR

As implementações específicas do *IDSROptionsHeader* são os tipos de opções disponíveis do DSR. As representações gráficas dos dados dos cabeçalhos estão representadas por meio da seguinte notação:

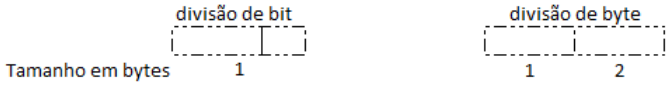


Figura 11 Notação da representação dos cabeçalhos

3.3.1.1 Fixed Header

A porção fixa do cabeçalho do DSR contém as informações necessárias para iniciar o processamento das opções, são elas: próximo cabeçalho, tamanho do cabeçalho e a lista de opções. A informação de próximo cabeçalho, neste trabalho, é sempre 59 [19]

(*No next header*), pois não há outros cabeçalhos após o DSR. A disposição dos campos do cabeçalho fixo é melhor visualizada na imagem a seguir, representando o tamanho, em *bytes*, de cada campo.

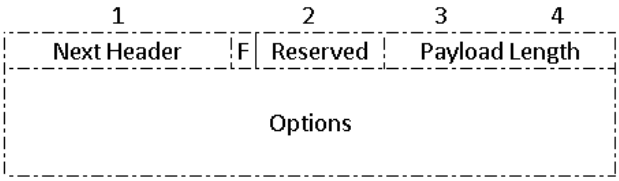


Figura 12 Cabeçalho fixo do DSR

3.3.1.2 Route Request Option

A opção de solicitação de rota é utilizada quando um nodo deseja enviar dados para outro nodo na rede, mas não possui uma rota para esse nodo. Assim, o nodo que deseja enviar os dados, denominado originador, para um destino deve enviar esta opção para todos os seus vizinhos.

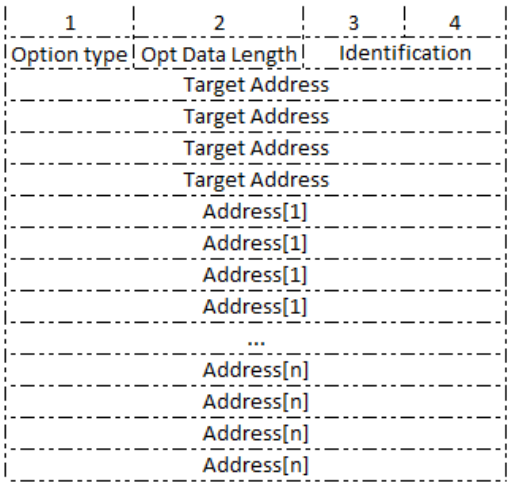


Figura 13 Opção de solicitação de rota do DSR

O campo *option type* é um identificador inteiro único da opção. O campo *identification* é um identificador inteiro único desta solicitação de rota, gerado pelo originador com base ‘em um algoritmo gerador de valores incrementais. O campo *target address* (16 *bytes*) contém o endereço do nodo que se deseja descobrir uma rota. O *Address* contém o identificador (UUID) dos pontos pelo qual o pacote já passou, construindo uma rota da fonte para o destino. O UUID da fonte e do destino não é listado no campo *Address*.

3.3.1.3 Route Reply Option

A opção de *route reply* é utilizada para devolver uma rota válida para o originador de um route request. O campo de *Address* possui o identificador do primeiro *hop* depois do originador até o destino.

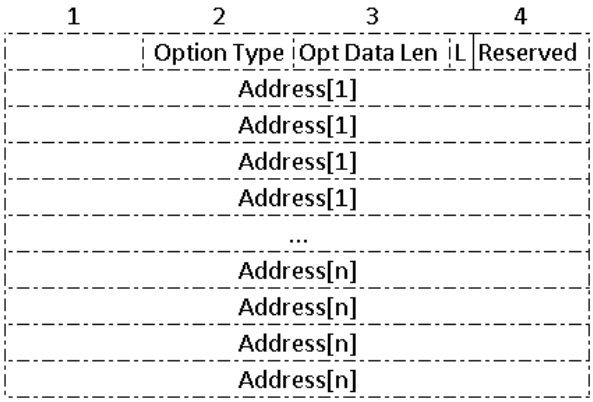


Figura 14 Opção de resposta de rota do DSR

3.3.1.4 Route Error Option

A opção de *route error* é utilizada para informar erros em uma parte da rota ou para informar a incapacidade de processar determinada opção do cabeçalho. Esta opção ainda contém a informação da fonte e do destino da transmissão e um identificador do erro.

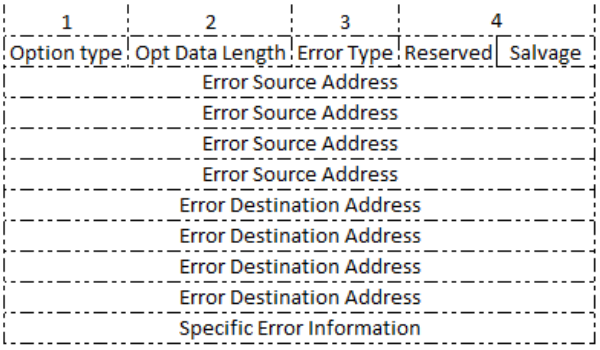


Figura 15 Opção de erro na rota

Se o erro for de vizinho não atingível, é adicionado à opção o identificador do vizinho. Caso o erro seja de opção desconhecida, adiciona-se o código dessa opção.

3.3.1.5 Acknowledgement Option

A confirmação de recebimento é utilizada para ter um controle no nível de roteamento da transmissão do pacotes para o próximo *hop*. Esta opção possui o identificador da solicitação de rota (2 *bytes*) e o identificador do gerador do *acknowledgement* (16 *bytes*) e do destino do *acknowledgement* (16 bytes).

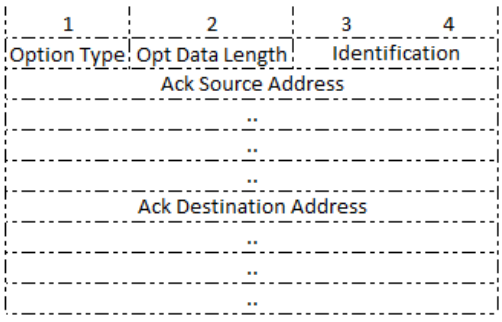


Figura 16 Opção de confirmação de recebimento

3.3.1.6 Acknowledgement Request Option

Opção utilizada para solicitar que o próximo *hop* responda com uma opção de *Acknowledgement*. Possui o identificador da solicitação de rota.

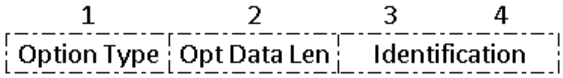


Figura 17 Opção de solicitação de confirmação de recebimento

3.3.1.7 Source Route Option

Esta opção é utilizada para informar qual a rota que este pacote deve percorrer para chegar ao destino. Contém a informação de quantos hops já foram visitados, no campo de *segments left*, e a lista de identificadores que constituem a rota atual.

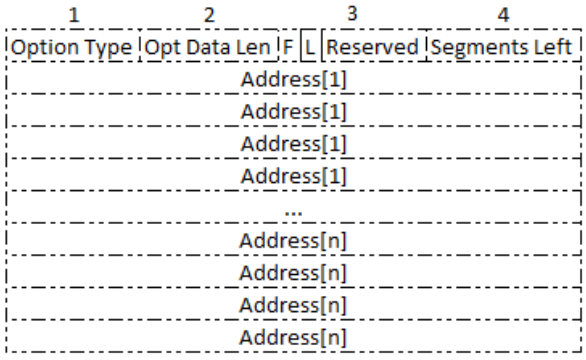


Figura 18 Opção de rota da origem

3.3.2 Estruturas Auxiliares

As estruturas auxiliares do protocolo são tabelas e buffers utilizados para armazenar informações sobre a rede de modo a diminuir o atraso nas operações de transmissão e descoberta de rota. As estruturas foram implementadas independentemente das outras, utilizando testes unitários para confirmar seu funcionamento.

3.3.2.1 Route Cache

A cache de rotas é utilizada por todos os roteadores a fim de armazenar as rotas conhecidas. As rotas são aprendidas sempre que o roteador uma nova mensagem. Nas mensagens de *route request*, o roteador pode aprender rota para todos os nodos que a mensagem já percorreu invertendo o campo de *address* , e nas mensagens de *route error* o roteador pode corrigir eventuais erros descobertos nas rotas. Cada entrada na cache de rotas possui um *timeout* que é atualizado toda vez que essa entrada é acessada e, quando a entrada expira, ela é removida da cache.

Para cada destino na *route cache* existe uma lista de *route cache entry*, que contém a rota até o destino e o *timestamp* de quando essa entrada foi acessada pela última vez. As entradas são removidas quando o *timestamp* mais o *timeout* das rotas é menor que o tempo atual. Para controlar o *timeout* das entradas, a *route cache* possui um verificador de *timeout* que é executado a cada *RouteCachePurgePeriod* período de tempo.

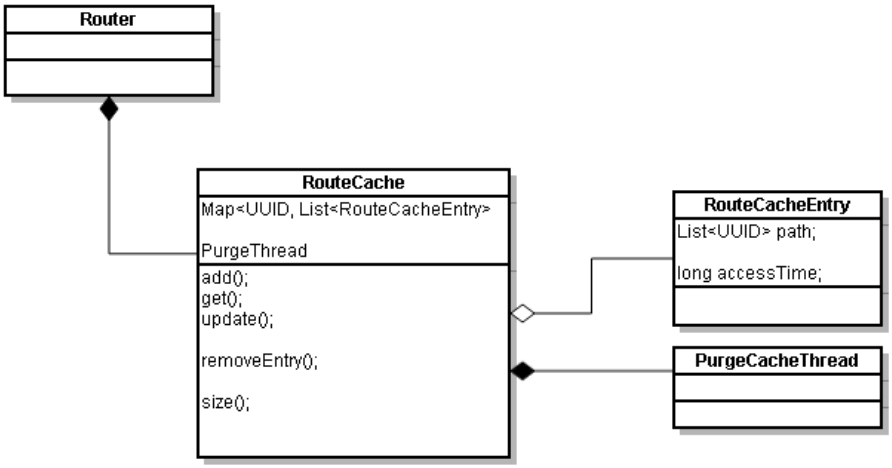


Figura 19 Diagrama de classes da estrutura RouteCache

A rota armazenada na cache é uma lista de UUID cujo primeiro elemento é o primeiro *hop* após o roteador e o último elemento é o último *hop* antes do destino. Assim, para utilizar a rota deve-se adicionar o UUID destino no final da lista. A rota retornada pela cache é a primeira rota, índice zero, na lista de entradas de um destino. O número de entradas na route cache é limitado e igual a *MaxCacheLength*, para evitar que seja consumida muita memória no dispositivo. O número de rotas que a cache pode conter para um destino não é limitado, para que o nodo tenha mais alternativas caso uma rota falhe e, caso uma rota aprendida não seja utilizada ela será removida. Foram criados casos de teste para confirmar o funcionamento da tabela de rotas, verificando a instanciação correta do objeto, o limite de rotas armazenadas e a remoção de rotas expiradas.

3.3.2.1.1. Send Buffer

O *SendBuffer* é uma fila utilizada para armazenar os pacotes cujo destino é desconhecido e uma descoberta de rede foi iniciada. Assim que o roteador descobrir uma rota para o destino, o pacote armazenado no *Send Buffer* deve ser removido e enviado. Para cada pacote adicionado no *send buffer* um *send buffer entry* é criado, contendo o pacote armazenado e o instante de tempo em que foi armazenado.

O *SendBuffer* possui uma restrição de tamanho, para evitar o consumo excessivo de memória do dispositivo, definida por *MaxSendBufferLength*. As entradas do *SendBuffer* são removidas após um *MaxSendBufferTime*, que define o tempo no qual uma entrada pode ficar no *SendBuffer* antes de ser removida. As entradas também são removidas por ordem de entrada, quando é atingido o limite de armazenamento do buffer, seguindo a regra de *first-in first-out* (FIFO).

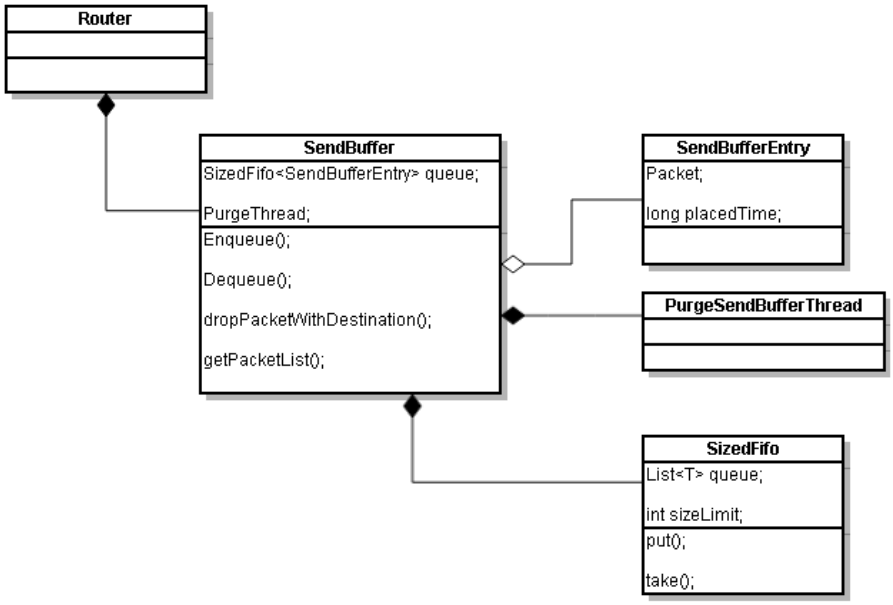


Figura 20 Diagrama de classes do Send Buffer

A classe *SendBuffer* possui uma thread responsável por remover as entradas expiradas, executando a cada *SendBufferPurgeInterval*. O teste do *SendBuffer* verifica se a estrutura respeita o tamanho limite, o tempo máximo que uma entrada pode ser armazenada e se os pacotes marcados como enviados são removidos do *buffer*.

3.3.2.1.2. Route Request Table

A tabela de solicitações de rota armazena informações referentes as requisições de rota realizadas e encaminhadas pelo roteador. Os dados são armazenados em um mapa, relacionando um UUID destino com uma *RoutedRequestTableEntry*, que contém o número de solicitações de rota que foram iniciadas pelo roteador para o destino, o *timestamp* da última solicitação e no *timestamp* de quando outra solicitação de rota pode ser feita para esse destino, para controlar a taxa de solicitações que são geradas pelo roteador.

A *Route Request Table* possui também um contador de solicitações, utilizado para identificar unicamente cada solicitação iniciada por um roteador. Esse contador é um inteiro que é inicializado com um valor randômico e incrementado a cada nova solicitação de rota. O contador é inicializado com um valor randômico para evitar que outros roteadores ignorem as requisições geradas por um roteador que tenha sido reinicializado, pois, caso um roteador sempre inicialize o contador com zero, as solicitações serão ignoradas pelos vizinhos até que o contador atinja o valor superior a última solicitação.

Os identificadores das solicitações geradas por outros roteadores são armazenados em um *buffer* do tipo FIFO, contendo o identificador do originador e do destino da solicitação. Esses dados são armazenados na *RouteRequestIDEntry* que compõe o *RouteRequestIDFifo*.

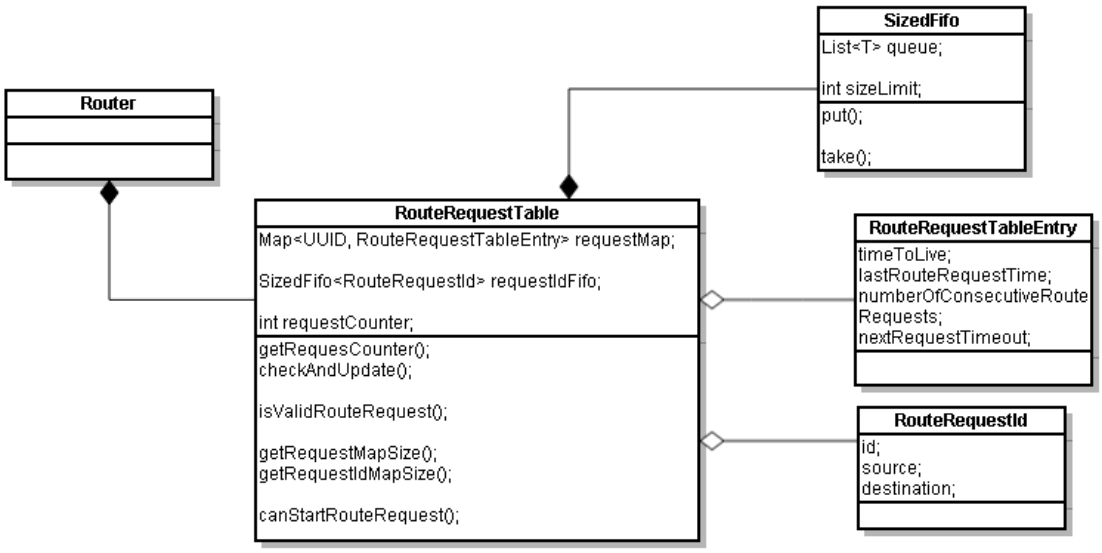


Figura 21 Diagrama de classes da RouteRequestTable

Os testes unitários do *RouteRequestTable* foram divididos entre as estruturas que compõe essa classe. Um conjunto de testes foram criados para verificar que o *requestCounter* gerava identificadores incrementais e que o valor inicial era um número randômico. Para o *requestMap*, testou-se a sua inicialização, o tamanho máximo da tabela e a remoção de entradas expiradas. Verificou-se também, se o *RouteRequestIDFifo* respeitava o tamanho máximo, se removia as entradas expiradas e se era capaz de validar corretamente um identificar de solicitação.

3.3.2.1.3. Maintenance Buffer

O *Maintenance Buffer* é utilizado pelas interfaces do DSR para confirmar que o próximo *hop* recebeu os dados enviados. O *buffer* é um mapa de UUID e *MaintenanceBufferEntry*. A entrada do *Maintenance Buffer* possui o número de retransmissões que já foram realizadas, o *timestamp* de quando que a entrada foi criada e de quando foi realizada a última retransmissão, além do *array* de *bytes* do pacote a ser transferido. Para controlar o tamanho do *buffer* e verificar erros na transmissão, as entradas são removidas após um *timeout*, um máximo de retransmissões ou se novas entradas são adicionadas.

As operações sobre os pacotes armazenados no *buffer* são de responsabilidade de quem estiver utilizando o *buffer*. Assim, o *buffer* não realiza nenhuma operação sobre os dados armazenados, ao contrário dos *buffers* já apresentados, pois cada interface DSR pode ter um comportamento específico na transmissão dos pacotes.

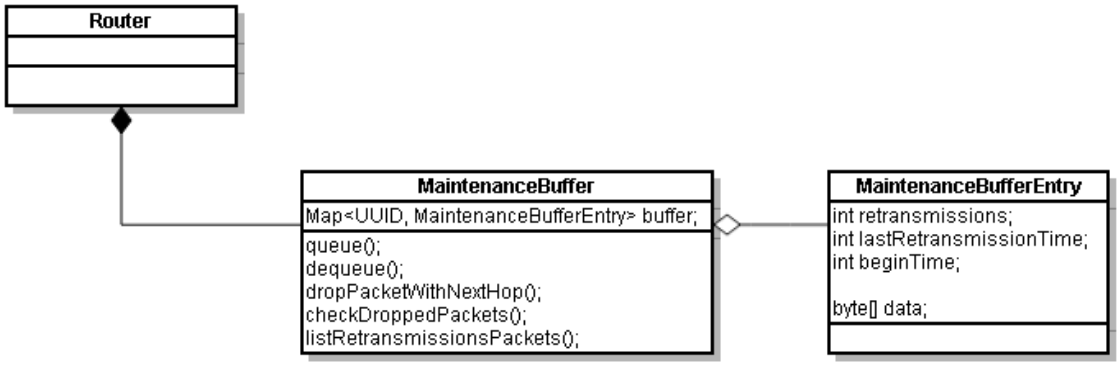


Figura 22 Diagrama de classes do MaintenanceBuffer

Para confirmar o funcionamento do *Maintenance Buffer* foram criados testes unitários que verificam o tamanho máximo do *buffer*, a inserção e remoção de pacotes e a listagem de pacotes que podem ser retransmitidos é respeitada. Pelo fato de o *Maintenance Buffer* ser utilizado preferencialmente pelas interfaces DSR, os pacotes são salvos na forma de *array* de *bytes*.

3.3.3 Interfaces DSR

A abstração do meio de transmissão é feita utilizando a interface *IDSRInterface*, que define o comportamento padrão de comunicação de uma interface DSR para o roteador. Toda classe que implementa essa interface é considerada pelo roteador como uma interface de rede, permitindo encapsular as tecnologias de transmissão e a emulação em classes de comportamento específico e operando de forma única sob o ponto de vista do roteador.

Dois grupos de classes implementam a interface DSR: classes da camada superior, como aplicação e transporte, e da camada inferior, de enlace, transporte ou emulação. A camada superior, neste trabalho, é a camada de aplicação, ou seja o usuário. Para tal, criou-se a interface *INode* que representa um nodo da rede. A camada inferior pode ser a camada de enlace ou a emulação.

3.3.3.1 Camada Inferior

As implementações da interface DSR para camadas inferiores foram a classes *FakeInterface*, utilizada para fins de emulação e testes, e a *WifiInterface*, que realiza a comunicação utilizando a tecnologia 802.11. Para realizar uma extensão ao protocolo, basta implementar a interface *IDSRInterface* e conectar essa implementação no roteador.

A classe *FakeInterface* realiza a emulação de uma interface real do ponto de vista do roteador, porém ela está conectada diretamente a outras *FakeInterface*, possibilitando a simulação da rede em um único computador. A criação de uma rede virtual é realizada por meio do método *addFakeInterface()*, listando quais são as conexões possíveis para essa interface. Por exemplo, para criar a seguinte topologia de rede usando *FakeInterface*, basta realizar as seguintes chamadas:

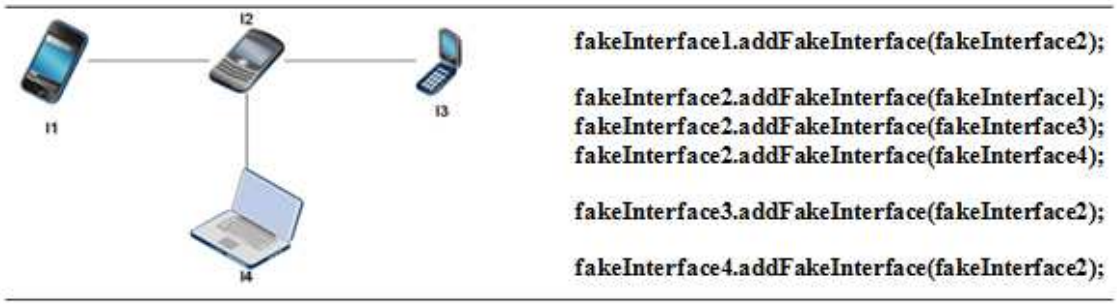


Figura 23 Exemplo de conexão de FakeInterfaces

Cada chamada de *addFakeInterface()* realiza conexão em único sentido, sendo necessário realizar a chamada nas duas *FakeInterface*. Com essa estrutura é possível, então, emular o comportamento do protocolo em diferentes topologias de rede e também pode-se analisar o comportamento com a variação das conexões, realizando a chamada de *removeFakeInterface()*.

A conexão das interfaces DSR com o roteador é feita por meio da chamada *setRouter()*, na qual a interface se configura no roteador e armazena uma referência ao roteador para transferir os pacotes recebidos.

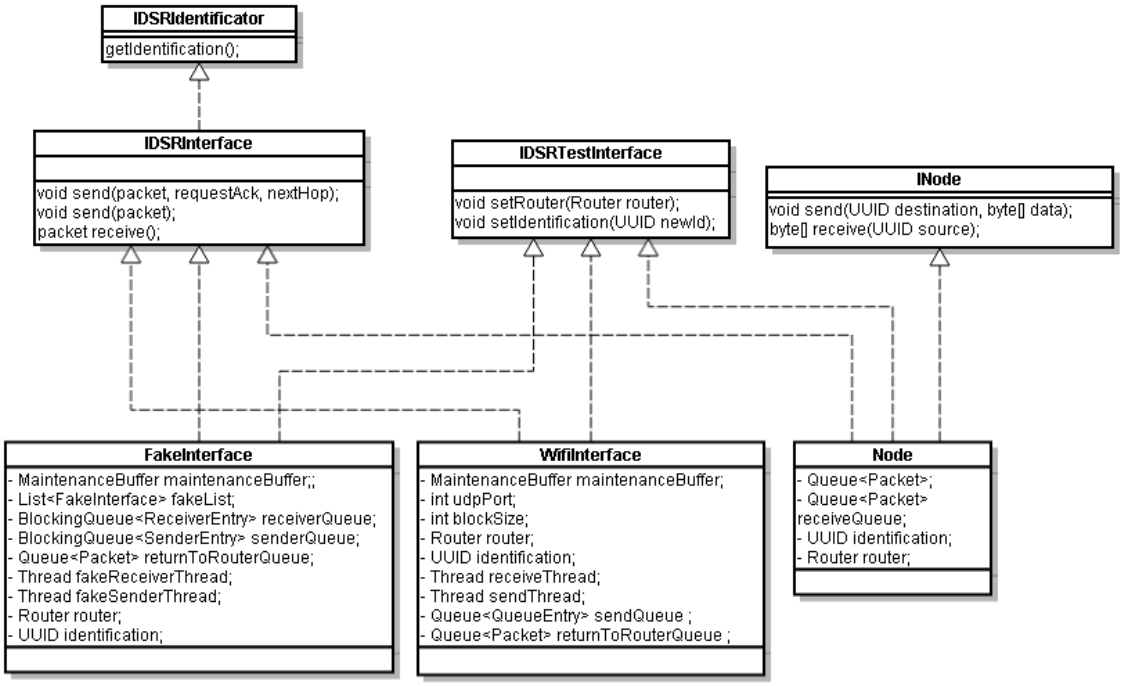


Figura 24Diagrama de classes das interfaces DSR

A classe *WifiInterface* realiza a comunicação utilizando o 802.11, enviando os dados para o endereço de broadcast. O envio dos dados pode ser otimizado, realizando o *broadcast* somente das mensagens de RREQ e enviando as outras mensagens diretamente para o próximo vizinho. Para isso, é necessário manter um mapeamento do UUID do vizinho com o seu endereço IP. Neste trabalho todas as mensagens são transmitidas em broadcast.

3.3.3.2 Camada Superior

A comunicação com a camada de aplicação é realizada por meio da interface *INode*, que possui os métodos *send* e *receive*. A implementação de *INode* deve responder também a interface *IDSRInterface*, para que seja possível a comunicação com o roteador. As chamadas de *send* não são bloqueantes, enquanto a chamada de *receive* é bloqueante. O *INode* não utiliza os pacotes DSR, enviando e recebendo *array* de *bytes*.

3.3.4 Roteador

O roteador é o elemento responsável pela lógica do protocolo DSR, utilizando todas as estruturas apresentadas anteriormente. Cada dispositivo da rede ad hoc sem fio possui uma ou mais interfaces DSR, das camadas superior e inferior, e possui somente um roteador. O roteador não possui identificador próprio, utilizando os identificadores das interfaces DSR para realizar o roteamento.

Cada roteador possui uma *RequestTable*, *RouteCache* e um *SendBuffer* para auxiliar o roteamento dos pacotes. Além disso, possui uma lista de todas as interfaces cadastradas, utilizadas para enviar e receber pacotes. As interfaces são cadastradas utilizando o método *registerInterface()* e são descadastradas utilizando *unregisterInterface()*. Pode ser atribuído um nome para o roteador, que, neste caso, foi utilizado na depuração. O roteamento é realizado acessando todas as interfaces e buscando um pacote de cada. Para cada pacote recebido, todos os cabeçalhos do pacote são processados.

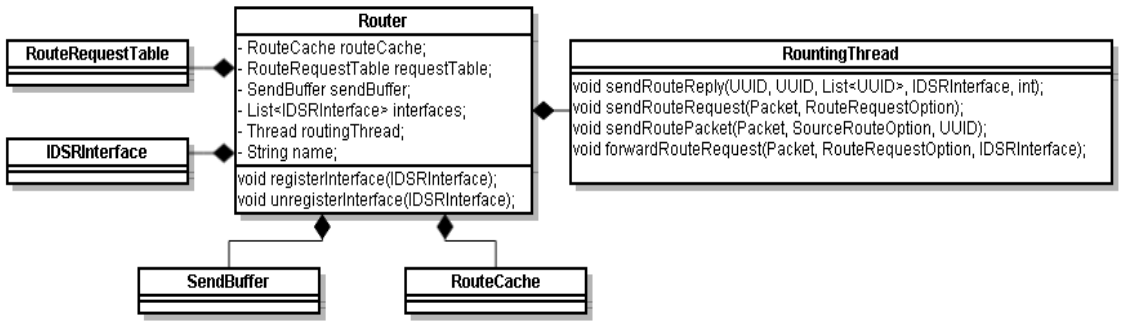


Figura 25Diagrama de classes do Router

Inicialmente, os testes do roteador foram realizados construindo uma topologia de *FakeInterface* de dois nodos ligados diretamente. Após a verificação de funcionamento, topologias de rede mais complexas foram criadas para validar o funcionamento. Os testes criados enviavam pacotes de um nodo origem para outro nodo destino, e do destino para a origem.

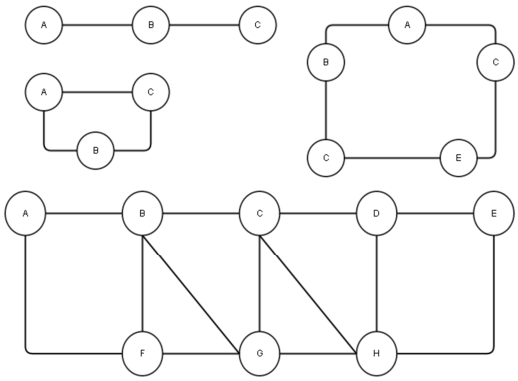


Figura 26 Topologias criadas para testar o Router

3.3.5 Constantes

O DSR define um conjunto de constantes que devem ser utilizadas na implementação. Essas constantes representam os tamanhos máximos das estruturas auxiliares, a quantidade de tentativas de envio de pacotes antes de ser considerado um erro na rota, o tempo entre descobertas de rota iniciadas por um nodo, entre outros valores.

```
6  /**
7   * @author Mauricio L. Dau <mauricioldau@gmail.com>
8   *
9   */
10 public class DSRAttributes {
11
12     public static boolean isRunning = true;
13
14     // ROUTE CACHE ATTRIBUTES
15
16     /**
17      * Maximum number of route entries that can be stored in route cache
18      */
19     public static final int MaxCacheLen = 64;
20
21     /**
22      * Maximum time routes can be queued in route cache, in miliseconds
23      */
24     public static final long RouteCacheTimeout = 30 * 1000;
25
26     public static final long RouteCachePurgePeriod = 5 * 1000;
27
28     // ROUTE REQUEST TABLE ATTRIBUTES
29
30     /**
31      *  Maximum number of retransmissions for route request discovery
32      */
33     public static final int RouteRequestRetries = 8;
34
35     /**
36      *  Maximum number of request entries in the request table
37      */
38     public static final int RequestTableSize = 64;
```

Figura 27 Classe contendo as constantes do protocolo

4 ANÁLISE DO COMPORTAMENTO

Neste capítulo é realizada uma análise do comportamento do protocolo implementado, apresentando duas execuções em dois ambientes diferentes, um real e outro simulado. O objetivo deste capítulo é confirmar o funcionamento do código implementado e verificar o comportamento da implementação, obtendo informações e tempo de execução e quantidades de pacotes transmitidos. Análise de desempenho e comparações com outras implementações são propostas de trabalhos futuros.

Para cada caso, as transmissões foram repetidas apenas uma vez e os dados adquiridos representam uma execução. Não foram realizadas mais repetições pois o objetivo é avaliar o comportamento, sendo assim, variações nos dados referentes a tempo de execução do sistema operacional ou outras interferências externas não influenciam na avaliação dos dados.

A fim de analisar o comportamento do protocolo implementado, criou-se a topologia da figura 28, contendo oito dispositivos. Cada dispositivo possui um roteador DSR e uma interface de rede *FakeInterface*. Os dispositivos um e oito foram o transmissor e o receptor da transmissão, respectivamente.

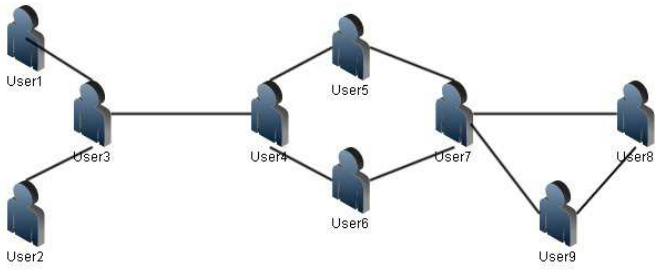


Figura 28 Topologia de teste com FakeInterface

A aplicação criada para analisar o comportamento transmite números sequenciais, de 0 a 120, a cada 50 milissegundos. O objetivo dessa execução é analisar o comportamento das duas etapas do protocolo, a descoberta e a manutenção de rotas.

O resultado esperado, de acordo com as referências [6], [20] e [21], é um grande fluxo de mensagens de descoberta de rota no início da execução e o declínio desse fluxo após a rota ser descoberta. Também é esperado que as mensagens de descoberta de rota sejam transmitidas para toda a rede, pois todo nodo que recebe uma solicitação de rota e não é o destino da solicitação reenvia a solicitação para todos os seus vizinhos.

O resultado da descoberta de rede pode ser visto a figura 30, que apresenta a quantidade de cada tipo de pacote que foi transmitido em cada instante de tempo. Esses

valores representam o início da comunicação entre os dispositivos 1 e 8. A execução inicia com a transmissão de dois pacotes de RREQ.

Foram transmitidos dois pacotes de RREQ pelo dispositivo 1, pois os nodos são, do ponto de vista do roteador, interfaces de rede. Assim, o roteador, tentando descobrir uma rota para o destino, envia um RREQ para todas as suas interfaces. Na figura 29 pode-se verificar a retransmissão de um RREQ recebido em uma interface para as outras interfaces.

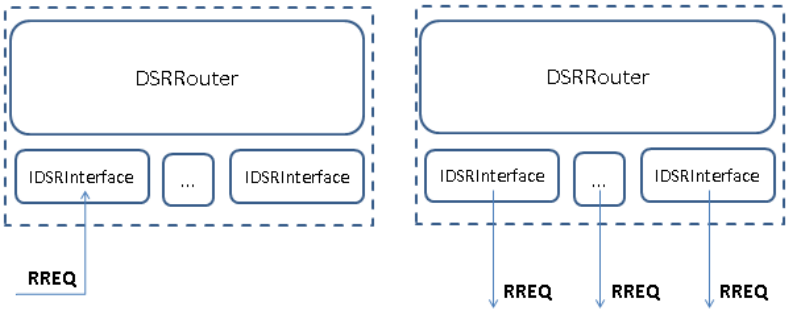


Figura 29 Recebimento e transmissão de RREQ

Em seguida, o RREQ é retransmitido pela rede até atingir o nodo destino que, no instante de tempo de 762 milissegundos, retorna um RREP para fonte assim que processa o RREQ. A retransmissão do RREQ é feita juntamente de um *Source Route*, que contém o caminho que o RREP deve seguir para atingir a origem do RREQ. Com isso, a retransmissão de um RREP acontece da mesma forma que a retransmissão de um pacote de dados.

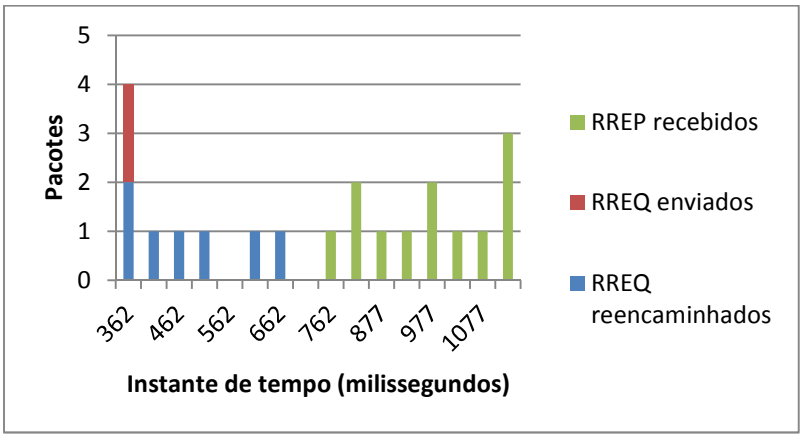


Figura 30 Fluxo de mensagens durante descoberta de rota

A grande quantidade de RREP recebidos, a partir de 827 milissegundos, é reflexo dessa característica, pois todos os nodos recebem o pacote contendo o RREP, e alguns nodos, como o nodo 9, recebem mais de uma vez. Neste caso, o nodo 9 recebe o RREP gerado pelo nodo 8 e o retransmitido pelo nodo 7. No instante 1127 milissegundos, o nodo originador do RREQ recebe o RREP e inicia a transmissão.

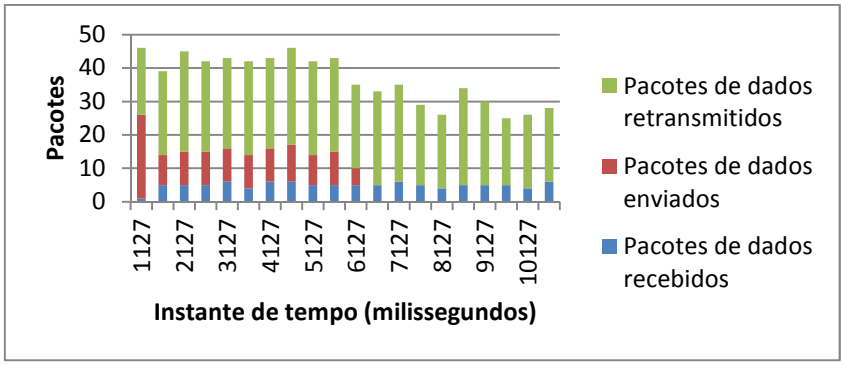


Figura 31 Fluxo de mensagens durante a manutenção da rota

A figura 31 apresenta a quantidade de pacotes que circularam na rede durante um intervalo de 500 milissegundos. Durante o primeiro intervalo, de 1127 milissegundos até 1627 milissegundos, o nodo origem insere 24 pacotes de dados na rede que são retransmitidos pelos nodos vizinhos até atingir o dispositivo final, que os recebe e entrega para o nodo destino. Pode-se verificar que a taxa média de transmissão de pacotes é de aproximadamente 10 pacotes por segundo. A figura a seguir apresenta a composição do pacote de dados da transmissão e o tamanho de cada porção do pacote.

Porção	Tamanho (bytes)
Source Route	180
Cabeçalho fixo	4
Cabeçalho do pacote	36
Dados	4
Total	224

Figura 32 Composição de um pacote de dados

No caso desta transmissão, os dados de configuração da rede representaram 98% do tamanho dos pacotes de dados transmitidos e a rota representou 80% do tamanho dos pacotes. O tamanho da rota, em *bytes*, enviada junto a todos os pacotes de dados, nesta implementação do DSR, é:

$$Tamanho_{rota} = 2 * número\ de\ saltos * 16$$

Cada salto é endereçado pelo UUID da interface, que possui 16 *bytes*, e cada salto possui duas interfaces: a de entrada e a de saída. Por isso, o tamanho do pacotes de dados cresce linearmente com o aumento do número de *hops*. Porém, é necessário diferenciar as interfaces de entrada e saída para permitir que o roteador transmita os dados por diferentes interfaces de rede.

A implementação do DSR não define uma restrição ao tamanho dos pacotes transmitidos. Essa tarefa é repassada para as camadas inferiores da rede.

O tempo de descoberta de rota, nesta topologia, foi de aproximadamente 800 milissegundos, e o tempo total da transmissão dos 120 pacotes foi de aproximadamente 10 segundos.

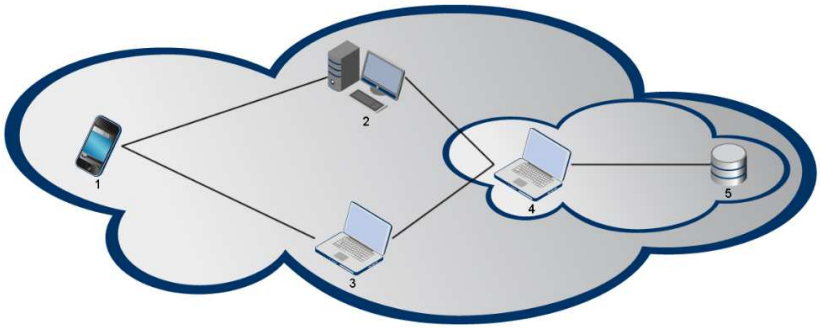


Figura 33 Topologia para teste em rede real

Para a execução do protocolo em um ambiente real, criou-se a topologia da figura 33, contendo diferentes dispositivos móveis e não móveis. O objetivo desta execução é confirmar a capacidade da implementação de suportar o roteamento utilizando diferentes interfaces de rede. Assim, existe a rede 802.11, composta pelos dispositivos de 1 a 4, e existe a rede de *FakeInterface* representada pelos dispositivos 4 e 5.

Neste ambiente, o nodo transmissor é o dispositivo 1, um *smartphone* Android [21], versão 2.3.6 (*Gingerbread*), utilizando a interface de rede 802.11. O dispositivo 2 é um *desktop* com Windows XP SP3 [23] conectado a rede Wi-Fi utilizando uma placa de rede sem fio. Os dispositivos 2 e 3 são um *notebook* e um *netbook*, respectivamente, também conectados a rede 802.11 por meio das suas placas de rede sem fio. O dispositivo 5 é um roteador virtual, executando no dispositivo 4 e conectado a ele por meio de uma rede de *FakeInterface*.

A rede 802.11 utilizada neste ambiente é infraestruturada, existindo um ponto de acesso não representado pela figura, provendo a conexão Wi-Fi entre os dispositivos 1 a 4. Optou-se por utilizar uma rede infraestruturada para a análise do comportamento, pois a rede ad hoc gerada pelo Windows não é uma rede *multihop*, ou seja, todos os nodos estariam conectados em um único dispositivo, que atuaria como ponto de acesso, comportamento semelhante a usar um roteador como ponto de acesso.

Outro motivo pelo qual optou-se pelo uso de rede infraestruturada é o fato de os dispositivos Android por padrão não conectarem a redes ad hoc sem fio, sendo necessário alterar o arquivo *wpa_supplicant* do sistema, liberando esse tipo de rede no menu de escolha de redes. Para realizar a alteração desse arquivo de sistema é necessário ter os direitos de superusuário, que não são disponíveis por padrão.

A obtenção dos direitos de super usuário nos dispositivos Android é uma ação que pode causar danos irreversíveis ao aparelho ou adicionar falhas de segurança no dispositivo [24]. Pelo fato de o único aparelho disponível para testes ser de uso pessoal, optou-se por não alterar as configurações do *smartphone*.

A figura a seguir apresenta o ambiente de testes real construído para analisar o funcionamento da implementação. Para possibilitar a realização dos testes, os dispositivos 1 e 4 ignoram os pacotes um do outro, utilizando o endereço IP para tal.

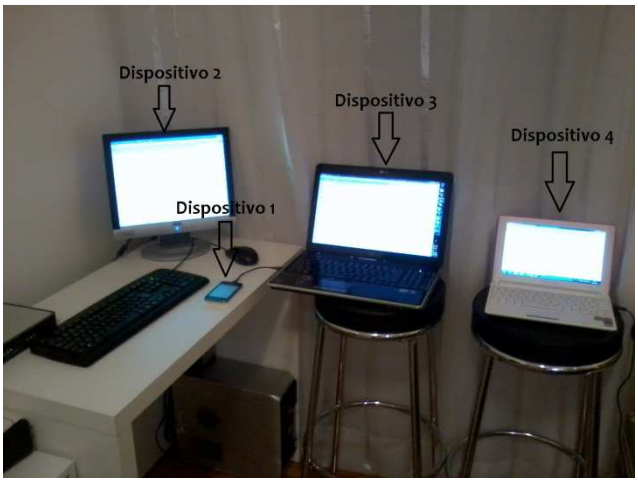


Figura 34 Ambiente de testes com quatro dispositivos

A aplicação executada sobre a topologia real foi a mesma utilizada na topologia de *FakeInterfaces*, trocando o nodo receptor para o dispositivo 5. Os gráficos 33, 34 e 35 apresentam o fluxo de pacotes dos dispositivos 3, 2 e 4, respectivamente. A captura dos pacotes do dispositivo 1, o *smartphone* Android, não foi realizada por limitações no desenvolvimento da aplicação. Ao término da execução da aplicação, é salvo um arquivo no diretório "*C:\Simulacao\Global.csv*" contendo o somatório dos pacotes a cada instante de tempo. Pelo fato dos dispositivos Android possuírem um sistema de arquivos diferenciado, operando com SD *card*, o método de salvamento dos arquivos é diferenciado. Pela limitação de tempo para execução do trabalho, optou-se por não implementar o mecanismo de salvamento no *smartphone*. Usou-se o gráfico do dispositivo 3 para analisar o comportamento da transmissão.

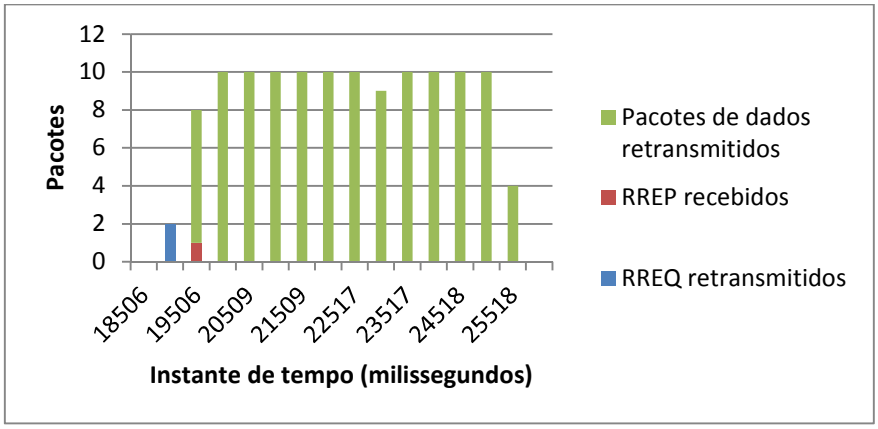


Figura 35 Resultado da execução do dispositivo 3

Após a execução da aplicação, verificou-se que a rota criada entre os dispositivos 1 e 5 foi 1,3,4 e 5. Os gráficos estão relacionados com o tempo de execução da aplicação em cada dispositivo, assim, os valores de tempo são utilizados para verificar o tempo entre cada conjunto de mensagens.

A fase de descoberta de rede está nos intervalos iniciais dos três gráficos, envolvendo a transmissão dos pacotes de RREQ e RREP. Em seguida, acontece a transmissão dos dados, por meio dos pacotes de dados. O tempo para realizar a descoberta da rede é de aproximadamente 500 milissegundos.

O tempo entre a descoberta da rede e o término da transmissão é de, aproximadamente, 7 segundos. Esse tempo é menor que o da transmissão na rede de *FakeInterfaces*, pelo fato de a rede real possuir menos saltos entre a origem e o destino.

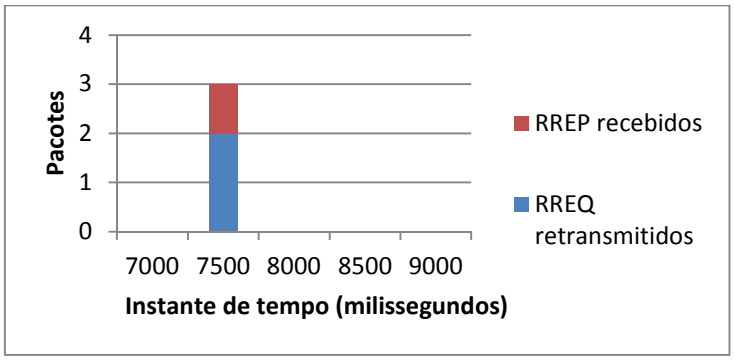


Figura 36 Resultado da execução do dispositivo 2

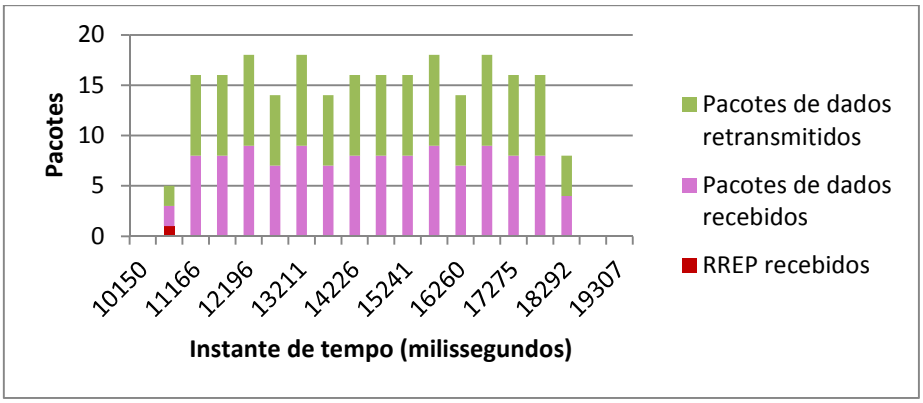


Figura 37 Resultado da execução do dispositivo 4

5 CONCLUSÃO

Este trabalho objetivou-se a implementar uma rede ad hoc utilizando o protocolo de roteamento *Dynamic Source Routing* e abstraindo as tecnologias de transmissão de dados. Essa abstração foi possível por meio do uso de UUID como endereço de rede.

A fim de motivar a implementação da rede ad hoc com endereçamento UUID, foram analisados diferentes padrões de comunicação sem fio que podem ser utilizados para construir redes de dispositivos móveis. Após a análise dos padrões, foi escolhido o padrão 802.11 para construir a rede ad hoc utilizada e analisar o comportamento da implementação.

Para facilitar o desenvolvimento do trabalho e validar as etapas desenvolvidas, criou-se a interface de rede *FakeInterface*, que possibilitou a construção de topologias virtuais em um único computador. O uso dessa interface de rede facilitou a implementação, pois pode-se construir testes unitários com o intuito de, automaticamente, validar as estruturas implementadas.

Após a análise dos padrões de rede sem fio, foi realizado um estudo dos protocolos de roteamento de redes ad hoc, apresentando as classificações dos protocolos. Também foi realizada uma análise dos protocolos *Ad-Hoc On-demand Distance Vector* e *Dynamic Source Routing*. Essa análise fundamentou a escolha do DSR como protocolo de roteamento a ser alterado, pois possui baixa dependência com as estruturas das camadas inferiores.

Em seguida, foi apresentada a construção das estruturas do DSR e de que modo elas se relacionam. O uso da linguagem Java possibilitou a execução do DSR em diferentes dispositivos com poucas alterações do código fonte.

A análise do comportamento permitiu verificar que as alterações realizadas no DSR permitiram criar uma rede ad hoc com diferentes interfaces de rede. O tempo de execução, entre descoberta e manutenção de rota, se mostrou aceitável, necessitando menos de um segundo para realizar uma descoberta de rota em uma topologia real de 5 elementos.

A realização deste trabalho propiciou a análise e a implementação de um protocolo de roteamento para redes ad hoc, bem como a verificação da capacidade dos dispositivos comerciais de suportarem essas redes. Verificou-se que existe uma limitação dos *smartphones* Android, que não permitem a conexão a redes em modo ad hoc.

O alto custo para implementar as interfaces DSR específicas para cada padrão de rede sem fio analisado impediu o seu desenvolvimento durante o trabalho. Porém, permitiu desenvolver a estrutura *FakeInterface*, que, por sua vez, facilitou a implementação do protocolo.

Os mecanismos de salvamento de pacotes e de distribuição de mensagens de erro na rota foram implementados parcialmente, por limitações de tempo para realização do trabalho. Assim, melhorar a manutenção de rotas da implementação é a próxima atividade a ser realizada no âmbito do trabalho.

A implementação de outras interfaces DSR, como *Bluetooth*, *ZigBee* ou NFC, são oportunidades de trabalhos a serem realizados futuramente. Outra oportunidade que surge deste trabalho é a construção de um mecanismo de distribuição de carga por diferentes interfaces de rede, ou seja, um dispositivo que percebe o congestionamento em uma interface de rede e que pode optar por transferir novos fluxos de dados por outras interfaces de rede, possibilitando uma maior vazão de dados pela rede.

REFERÊNCIAS

[1] **Specification of the Bluetooth System**, disponível em <https://www.bluetooth.org/en-us/specification/adopted-specifications>. Acesso em novembro 2013.

[2] JOHNSON, D.; HU, Y.; MALTZ, D. **The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4**: RFC 4728. [S.l.]: Internet Engineering Task Force, Network Working Group, 2007.

[3] CONTI, M.; GIORDANO, S. Multihop Ad Hoc Networking: The Theory. **Communications Magazine, IEEE.** [S.l.], v.45, n.4, p. 78-86, abril 2007.

[4] CONTI, M.; GIORDANO, S. Multihop Ad Hoc Networking: The Reality. **Communications Magazine, IEEE.** [S.l.], v.45, n.4, p. 78-86, abril 2007.

[5] LEACH, P.; MEALLING, M.; SALZ, R. **A Universally Unique Identifier (UUID) URN Namespace**: RFC 4122. [S.l.]: Internet Engineering Task Force, Network Working Group, 2005.

[6] JOHNSON, D.; MALTZ, D. **Dynamic Source Routing in Ad Hoc Wireless Networks**. Ad Hoc Networking, cap. 5, p. 139-167, 2000.

[7] Artigo de alguem sobre o AODV

[8] PERKINS, C.; BELDING-ROYER, E.; DAS, S. **Ad hoc On-Demand Distance Vector (AODV) Routing**: RFC 3561. [S.l.]: Internet Engineering Task Force, Network Working Group, 2005.

[9] FRIEDMAN, R.; KOGAN, A.; YEVGENY, K. On Power and Throughput Tradeoffs of WiFi and Bluetooth in Smartphones. **IEEE Transaction on Mobile Computing**. [S.l], v. 12, n. 7, p. 1363-1376, julho 2013.

[10] FEENEY, L.; NILSSON, M**. Investigating the Energy Consumption of a Wireless Network Interface in an Ad Hoc Networking Enviroment**. IEEE INFOCOM. Alaska, abril 2001.

[11] FERRO, E.; POTORTI, F. **Bluetooth and WiFi Wireless Protocols: A Survey and a Comparison**. IEEE Wireless Communications, fevereiro 2005.

[12] BISDIKIAN, C. **An Overview of the Bluetooth Wireless Technology**. IEEE Communications Maganize, p. 86-94, dezembro 2001.

[13] YU, C.; HUNG, S.; CHEN, Y. **Forming Mesh Topology for Bluetooth Ad Hoc Networks**. IEEE 17th Internation Symposium on Consumer Electronics, Hsinchu, p. 123-124, junho 2013.

[14] BAKER, N. **ZigBee and Bluetooth strengths and weaknesses for industrial applications**. IEEE Computing & Control Engineering Journal, [S.l.], v. 16, n. 2, p. 20-25, abril 2005.

[15] KINNEY, P. **ZigBee Technology: Wireless Control that Simply Works**. IEEE Communications Design Conference, outubro 2003.

[16] CHAKERES, I.; BELDING-ROYER, E. **AODV Routing Protocol Implementation Design**. Internation Conference on Distributed Computing Systems Workshops, [S.l.], p. 698-703, março 2004.

[17] KENT, B. **JUnit Pocket Guide**. 1ª ed. Sebastopol: O'Reilly, 2004.

[18] MILANO, D. **Android Application Testing Guide**. 1ª ed. [S.l.]: Packet Publishing, junho 2011.

[19] DEERING, S.; HINDEN, S. **Internet Protocol, Version 6 Specification**: RFC 2460. [S.l.]: Internet Engineering Task Force, Network Working Group, 1998.

[20] CHENG, Y.; ÇETINKAYA, E.; STERBENZ, J. **Dynamic Source Routing (DSR) Protocol Implementation in ns-3**. Desenzano, WNS3, março 2012.

[21] PEREIRA, I.; PEDROZA, A. **Redes Móveis Ad Hoc Aplicadas a Cenários Militares**. Florianópolis, I Workshop de Computação da Região Sul, maio 2004.

[22] Android Developers reference: http://developer.android.com/index.html Acessado em novembro de 2013.

[23] Visão global do Windows XP: http://en.wikipedia.org/wiki/Windows_XP Acessado em novembro de 2013.

[24] SHABTAI, A.; FLEDEL, Y.; KANONOV, U. **Google Android: A Comprehensive Security Assessment**. Security & Privacy IEEE, [S.l.], v. 8, n. 2, p. 35-44, março 2010.

[25] DJENOURI, D.; SOUALHI, W.; NEKKA, E. **VANET's Mobility Models and Overtaking: An Overview**. ICTTA, Damascus, p. 1-6, 2008.

ANEXO A CÓDIGO FONTE

Durante a realização deste trabalho diversos projetos na linguagem Java para computadores e *smartphones* Android foram desenvolvidos. Com o intuito de facilitar trabalhos futuros, o código fonte desses projetos estão disponíveis no site GitHub. O acesso aos arquivos é liberado para todos e os projetos são disponibilizados por meio da licença Creative Commons BY.

O endereço para acesso aos projetos é:

- https://github.com/mldau

Os projetos relacionados a este trabalho são:

- DSRFramework
- DSRFramework-test
- DSRFramework-Android

ANEXO B TRABALHO DE GRADUAÇÃO 1

Arquitetura de comunicação entre diferentes MANETs utilizando protocolos de roteamento baseado na fonte

Maurício L. Dau[1], João Netto[2]

[1 2]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS) – Porto Alegre – RS – Brazil

{ mldau, netto} @inf.ufrgs.br

Abstract. *The expansion of intelligent devices and smartphones has increased the use of wireless Technologies. However, the daily devices uses different data transmission technology can block the communication between some devices. This work purposes a communication model between wireless ad hoc mobile networks that provides to users the capability of communicate over different data transmission technology, using reactive and source based routing protocols.*

Resumo. *A expansão do mercado de smartphones e de dispositivos inteligentes impulsionou o uso de tecnologias sem fio. Contudo, inúmeras tecnologias de transmissão de dados são utilizadas no dia a dia, o que pode levar a situações de incompatibilidade entre alguns dispositivos. Este trabalho propõe uma arquitetura de comunicação entre redes ad hoc de dispositivos móveis que propicia aos usuários a liberdade de se comunicarem com outros dispositivos independente da tecnologia de transmissão, fazendo uso de protocolos de roteamento reativos e baseados na fonte.*

1. Introdução

O número de aplicações móveis está cada vez maior, impulsionado pelo avanço dos smartphones e pelos elementos de computação distribuída. A característica comum de todos os dispositivos é a comunicação sem fio, seja por Wi-Fi, Bluetooth ou outras tecnologias de transmissão de dados. Entretanto, cada tecnologia possui suas características específicas e dois dispositivos só conseguem se comunicar se possuírem o mesmo hardware. Com isso, muitos usuários não conseguem realizar algumas atividades, pois ficam restritos ao hardware que possuem. Este trabalho propõe uma arquitetura de comunicação entre dispositivos móveis que é capaz de transpor essa barreira, utilizando técnicas de roteamento em redes ad hoc.

É uma situação comum dois usuários de smartphones desejarem trocar arquivos e não possuírem nenhum cartão de memória ou cabo para transferir os dados de um aparelho para o outro. Nesse caso, os usuários optam por realizar a transferência por

meio de uma rede sem fio. Uma rede sem fio criada por dispositivos móveis é chamada de MANET (Mobile Ad Hoc Network) e possui a característica de ser uma rede descentralizada e temporária, exatamente como no caso da transferência de arquivos entre dois smartphones.

Contudo, pode acontecer de os dispositivos possuírem configurações de hardware diferentes, o que impede a comunicação entre eles. Caso haja um terceiro dispositivo que possua o hardware de ambos, a transferência do arquivo pode acontecer por meio deste novo dispositivo, utilizando-o como uma ponte.

Outra aplicação dessa arquitetura é no mercado de jogos digitais, no qual pode-se ter um conjunto de usuários que possuem dispositivos móveis distintos com o interesse de jogar entre si por meio de uma rede sem fio. É comum os jogos serem desenvolvidos para suportarem uma determinada tecnologia de comunicação (i.e. Wi-Fi), excluindo todos os aparelhos que não a possuem. Porém, utilizando os elementos que possuem a tecnologia do jogo e a dos outros aparelhos, a comunicação pode acontecer por meio desses dispositivos.

O ambiente de troca de arquivos entre dois dispositivos que não possuem o mesmo hardware de comunicação pode ser visto na Figura X, na qual temos o nodo A tentando enviar um arquivo para o nodo B. As duas redes sem fio até podem se atingir, mas na prática são duas redes totalmente independentes.

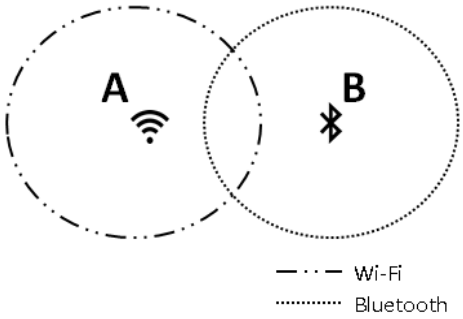


Figura 38 Duas redes sem fio de tecnologias diferentes se sobrepondo

Neste caso, adicionando um terceiro dispositivo, o nodo C, que suporte as duas redes sem fio, podemos transferir o arquivo do nodo A para o B por meio do C, de forma transparente. Pois quando o nodo A procurar os dispositivos que estão ao seu alcance, o nodo C vai responder ele mesmo (C) e todos a quem ele alcança (B). Assim, teremos uma transferência de arquivos entre A e B de forma transparente, mesmo eles possuindo diferentes tipos de redes sem fio.

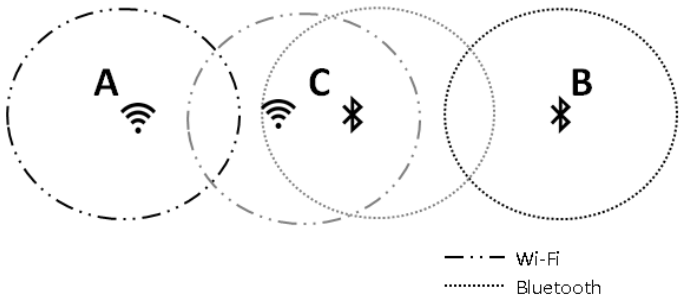


Figura 39 Nodo C operando como ponte para comunicação de A com B

A seção 2 deste trabalho define as características da comunicação ad hoc de dispositivos móveis sem fio e apresenta o problema de comunicação entre diferentes redes sem fio. A seção 3 apresenta as soluções existentes para realizar a comunicação dentro de uma rede ad hoc sem fio, explicando os protocolos que servem de base para a arquitetura proposta. Na seção

4.1. é apresentada a arquitetura de comunicação, com todos os seus componentes e protocolos de roteamento. A seção 5 apresenta quais são os componentes que serão implementados neste trabalho. A maneira pela qual será conduzida a prova de conceito da arquitetura é apresentada na seção 6. E, por fim, a seção 7 contém a avaliação do trabalho até o momento e o cronograma dos trabalhos que serão realizados.

2. Definição do Problema

O cenário básico deste trabalho é uma MANET com poucos nós, baixa mobilidade e com grande heterogeneidade de hardware, desde smartphones até sensores sem fio. Considera-se que os usuários desejam trocar arquivos ou rodar uma aplicação comum sobre todos eles e, com isso, seus dispositivos participam ativamente da rede, realizando a retransmissão de pacotes quando solicitado.

Todos os dispositivos que tiverem interesse em participar da rede deverão seguir as regras de roteamento definidas pela arquitetura, evitando assim, que ocorram loops nos trajetos e falsas indicações de roteamento.

A tradução literal da expressão "Ad hoc" é: algo com uma finalidade específica. Isso significa que os serviços ad hoc têm um objetivo a ser alcançado . Após alcançado o objetivo inicial, específico, o serviço deve ser considerado concluído. Um exemplo que ilustra o termo ad hoc é o trabalho efetuado pelos avaliadores ou revisores ad hoc, os quais possuem a função de avaliar ou revisar textos sem possuírem vínculo com os seus produtores e que, ao concluírem a avaliação ou a revisão do seu objeto de trabalho, têm por encerrada a sua tarefa.

Uma rede Ad hoc é criada por um conjunto de nodos independentes com o interesse de realizar uma atividade. Esses nodos não necessitam de outros para conseguirem se comunicar. Essa rede tem a duração da atividade proposta e, com o

término desta, a rede deixa de ser necessária e é desmanchada. Uma rede Ad hoc criada por dispositivos móveis é chamada de MANET (Mobile Ad Hoc Networking) e adiciona à rede ad hoc as características de operar sobre redes sem fio e depender de baterias. A comunicação numa MANET deve ser simples e consumir pouca energia, pois os dispositivos utilizam o mesmo meio, o ar, para transmitir as suas informações e possuem uma fonte finita de energia, a bateria.

Um conjunto de protocolos de comunicação e de hardware de transmissão de dados definem uma rede, como no caso dos protocolos TCP/IP que operam sobre redes Ethernet. Neste caso, para um novo dispositivo participar da rede ele deve implementar os protocolos TCP/IP e possuir interfaces de rede Ethernet, caso contrário, não será capaz de se comunicar. As redes ad hoc sem fio possuem a mesma característica, impedindo novos dispositivos de participar da rede se não implementarem os mesmos protocolos e interfaces de rede.

O objetivo deste trabalho é desenvolver uma arquitetura que, implementada em diferentes dispositivos, seja capaz de estabelecer uma rede ad hoc e de prover envio/recebimento de informações por meio da mesma. Para tal fim, será escolhido um protocolo de roteamento para redes sem fio ad hoc e será implementada a arquitetura proposta na seção 4.

3. Soluções Existentes

Os protocolos de roteamento tradicionais, tais como RIP (*Routing Information Protocol*) e OSPF (*Open Shortest Path First*), são boas escolhas para redes cabeadas e que possuem pouca alteração na suas topologias. Contudo, as MANETs possuem grande variação nas suas topologias e não têm suporte de uma boa infraestrutura, que forneça qualidade nas transmissões. Assim, as abordagens tradicionais não são uma boa escolha no ambiente de redes ad hoc sem fio.

Os protocolos de roteamento podem ser classificados como proativos, reativos e híbridos, relacionados com a maneira que o protocolo descobre a topologia da rede. Os protocolos proativos analisam a topologia da rede constantemente, verificando a existência de novos caminhos e rompimento de caminhos antigos. Porém, essa análise consome uma porção fixa da banda da rede, pois os nodos trocam informações incessantemente e, com o aumento do número de nodos, a análise consome uma porção maior da banda da rede. No caso deste trabalho, os protocolos proativos não serão abordados, exatamente por inundarem a rede com informações que, em muitos momentos, é inútil. No caso de um smartphone estar com a rede ad hoc sem fio ativada e o protocolo for proativo, ele ficará constantemente enviando mensagens mesmo não existindo nenhum nodo próximo para recebe-las. Mesmo existindo a possibilidade de otimização, essa abordagem não será utilizada no trabalho.

Operando de forma reativa, existem os protocolos que só realizam comunicação quando existe alguma informação para ser transmitida, seja do próprio dispositivo ou um roteamento para outro dispositivo. Os protocolos reativos descobrem a rota por demanda, assim, quando há informação para ser transmitida o roteador reativo inicia o processo de descoberta da rede e, quando descobrir o caminho de roteamento, irá transmitir a informação. Esta abordagem possui menos tráfego de configuração que a proativa, consumindo menos energia do dispositivo, mas possui uma maior latência para transmissão de dados, pois necessita descobrir a rota até o nodo destino.

A seguir serão apresentados três protocolos de roteamento em MANETs, o *Dynamic Source Routing* (DSR), o *Ad hoc On-demand Distance Vector* (AODV), e o *Lightweight Mobile Routing* (LMR) todos reativos e baseados na fonte. Roteamento baseado na fonte é aquele no qual o nodo origem, ou fonte, da transmissão é quem define qual a rota, ou caminho, pela qual a informação deve ser transmitida.

3.1. *Dynamic Source Routing* (DSR)

O roteamento dinâmico baseado na fonte (DSR) funciona de forma reativa, sendo ativado quando a origem da transmissão inicia a buscar pelo nodo destino. O DSR possui duas etapas de funcionamento: descoberta da rota e manutenção da rota. Os nodos da rede devem possuir uma lista temporária com as rotas aprendidas durante a execução. Cada rota aprendida possui um prazo de validade, e deve ser descoberta novamente após expirada a sua validade. Para realizar a descoberta da rede o protocolo define duas mensagens, a solicitação de rota (*route request*) e a resposta de rota (route reply).

A descoberta de rede é iniciada pelo nodo fonte da transmissão, que envia uma mensagem de solicitação de rota para todos os nodos no seu alcance. A mensagem de solicitação de rota contém o endereço do destino, o seu próprio endereço, um identificador único da mensagem e uma lista de nodos que compõe a rota para o destino. Todos os nodos que receberem esse pedido de rota verificam se possuem uma rota válida para o destino. Se o nodo não possui uma rota válida, deve inserir seu endereço na lista de nodos da mensagem e enviar a solicitação de rota para todos os nós no seu alcance. Utilizando o identificador único da mensagem, os nodos verificam se já receberam a mensagem, e, analisando a lista de nodos da mensagem, verificam se já constam na lista de roteamento, a fim de evitar retransmissões .

Quando a solicitação de rota atingir o destino ou um nodo que contenha uma rota válida para o destinou uma mensagem de resposta de rota será criada. Neste momento, a mensagem de solicitação de rota contém a lista de nodos que foram percorridos da fonte até o nodo atual e essa lista é inserida na ordem invertida na mensagem de resposta de rota. Se a mensagem de resposta for criada por um nodo intermediário, este irá inserir na lista a rota dele até o destino.

O retorno de um route reply depende da característica das ligações (links) entre os nodos. Considerando que todas as ligações são simétricas, que suportam transmissões em ambas direções, a resposta é enviada utilizando a lista gerada na mensagem de route request. Se for numa rede assimétrica, o nodo que estiver enviando ou retransmitindo um *route reply* deverá realizar uma descoberta de rota e enviar o route reply em seguida.

O protocolo prevê o uso de uma mensagem de erro na rota (route error), que informa um erro de transmissão para um determinado nodo. Ao receber uma mensagem de *route error*, o nodo deve remover de sua tabela a posição com falha de transmissão, truncando as rotas existentes nessa posição. Situações de validação da comunicação podem acontecer por meio da transmissão de confirmações de transmissão (*acknowledgements*) para o nodo anterior. No caso das redes sem fio, um nodo pode analisar as trocas de mensagens e verificar que a mensagem foi retransmitida.

3.2. *Ad hoc On-demand Distance Vector* (AODV)

A solicitação de um vetor de distância ocorre de maneira semelhante ao DSR, quando uma fonte se comunicar com um destino e não possuir uma rota válida, por ela ter expirado ou não existir, ela iniciará o processo de descoberta de caminho (path

discovery) enviando uma mensagem de solicitação de rota (RREQ) para todos os seus vizinhos. A mensagem de RREQ será retransmitida por todos os nodos até atingir o destino ou um nodo intermediário que possua uma rota válida.

Todos os nodos que utilizam o AODV possuem um número de sequência e um identificador de broadcast (broadcast ID) que são enviados, juntamente com seu IP e o número de sequência do nodo destino, na mensagem de RREQ, tornando-a única. Quando um nodo realiza um RREQ ele incrementa o seu broadcast ID. Com o uso desses identificadores, o protocolo AODV consegue diminuir o número de respostas de rota, pois um nodo intermediário só pode responder a rota que possui se possui um número de sequência do destino maior ou igual ao enviado pela fonte.

Ao receber um primeiro RREQ, um nodo intermediário armazena o endereço de origem da mensagem e retransmite o pedido. As próximas mensagens de RREQ idênticas que receber são descartadas. Quando o RREQ atinge o nodo de destino um nodo intermediário que possui uma rota válida, este responde com uma mensagem de resposta de rota (RREP) atráves de quem recebeu o RREQ. A reposta é retransmitida pelos nodos intermediário, que armazeram o endereço do nodo de origem. Em seguida, o nodo fonte pode enviar os dados para o nodo destino.

Para cada rota ou entrada adicionada na lista de roteamento, um prazo de validade é definido (semelhante ao DSR) e, quando o prazo expira, essa entrada deve ser removida da tabela. O nodo fonte pode solicitar uma nova descoberta de caminho caso se desloque fisicamente, alterando a topologia original. Caso um nodo intermediário note qualquer alteração na topologia estabelecida, ele deve enviar uma notificação de falha no ligação para o inverso da rota. Quando essa notificação atingir a fonte, esta pode decidir entre descobrir um novo caminho ou finalizar a comunicação.

O protocolo AODV considera que todas as ligações entre os nodos são simétricas, pois quando descobre uma rede armazena o endereço do nodo fonte para realizar a transmissão na ordem reversa do caminho. Assim, ao escolher o AODV como protocolo de roteamento de uma rede, deve-se verificar se a topologia da mesma é simétrica.

3.3. *Lightweight Mobile Routing* (LMR)

Os protocolos descritos anteriormente consideram que o tempo de vida de uma topologia é bem menor que o tempo necessário para descobrir a rota necessária para a comunicação. O LMR foi desenvolvido para ser utilizado em redes que possuem grande mobilidade, o que faz a topologia ser alterada com maior frequência.

Este protocolo possui as seguintes mensagens: *Query* (QRY), *Reply* (RPY) e *Failure Query* (FQ). A mensagem QRY contém o identificador do nodo de origem, ou fonte, o identificador do nodo destino, um número de sequência e o identificador do nodo que está retransmitindo a mensagem. As mensagens de RPY e FQ contêm o identificador do destino e do retransmissor.

Cada nodo da rede mantém uma tabela com o estado da ligação entre os seus vizinhos (link status table). Essa tabela é criada para cada destino que o nodo desejar se comunicar, informando se o nodo possui alguma ligação com o destino.

No início da operação da rede, todos os estados são não-assinalados, significando que o nodo não possui nenhuma informação de ligação o destino. Em seguida, um nodo realiza um query e todos os outros retransmitem essa mensagem. Se o

query atingir o destino, este irá transmitir uma mensagem de RPY que também será retransmitida por todos os nodos, até atingir o nodo de origem. Quando um nodo recebe uma mensagem de RPY ele marca o estado da ligação como *downstream*, anotando que é capaz de enviar dados para o destino. Caso depois de algum tempo a fonte não receba uma mensagem de RPY pode decidir entre fazer uma nova solicitação ou cancelar a comunicação.

As mensagens são retransmitidas analisando o estado da ligação em cada nodo, resultado em caminhos sem ciclos. A comunicação neste protocolo pode ser simétrica ou assimétrica, pois o estado de cada ligação é armazenado e atualizado durante a execução do protocolo.

4. Definição da Arquitetura

A arquitetura de comunicação é formada por 4 componentes: a Aplicação, o Controlador, o Roteador e as Interfaces de rede. Os componentes serão explicados de forma detalhada na seção 4.1. O protocolo de roteamento utilizado é explicado na seção de Roteamento. O Roteador pode possuir zero ou mais interfaces de rede diferentes, transferindo a informação pelas interfaces ou passando para o Controlador.

A comunicação acontece de forma reativa e baseada na fonte, ou seja, quando um usuário realizar uma transferência, este deve solicitar a topologia da rede, verificar se o destino está na topologia, caso contrário deve solicitar uma nova verificação da topologia, e enviar os dados para o destino. Na figura 3 percebe-se que a arquitetura suporta a comunicação entre as aplicações de um mesmo nodo, utilizando o Controlador.

Cada aplicação que desejar participar da rede por meio da Arquitetura, deverá se registrar no Controlador, para que este adicione a aplicação na tabela de roteamento e a torne visível para outras aplicações. No momento em que uma aplicação é registrada no Controlador, ela recebe um identificador único, baseado na RFC4122, chamado de *Universally Unique Identifier* (UUID). Um UUID é um identificador de 128 bits gerado randomicamente que auxilia na identificação única de uma aplicação na rede. A maior parte dos sistemas operacionais de smartphones atuais consegue gerar UUIDs rapidamente, de forma que não influencia no tempo de execução do algoritmo. A estrutura do UUID é explicada na seção Estruturas.

De posse do seu UUID uma aplicação consegue solicitar a lista de nodos atingíveis, receber dados do Controlador e, ao término da sua execução, remover seu registro da tabela de roteamento. Os UUIDs não são utilizados pelas Interfaces de rede, que operam sobre as suas tecnologias específicas, implementando os protocolos necessários para prover a comunicação sobre essa rede específica.

Um nodo pode conter diversas Interfaces de rede, inclusive adicioná-las e removê-las do roteador dinamicamente. O custo do roteamento num dispositivo é diretamente proporcional ao número de interfaces que possui e a volatilidade das mesmas, pois com mais interfaces maior é o número de nodos na tabela de roteamento.

4.1. Estrutura

Esta arquitetura foi desenvolvida sem uma tecnologia alvo, sendo possível implementá-la em qualquer plataforma (Arduino, Android, iOS, ...). A arquitetura pode ser implementada em um dispositivo sem interfaces de rede, para uso na comunicação interna. A arquitetura é composta pelo Controlador, Roteador, Aplicação e pelas

Interfaces de rede. O Controlador e o Roteador são componentes essenciais e as Interfaces de rede não.

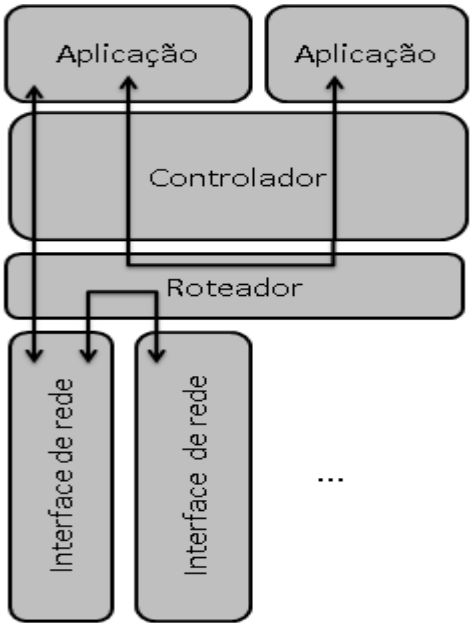


Figura 40 Estrutura da Arquitetura

A Aplicação é quem irá usufruir do ambiente de rede desenvolvido, utilizando uma interface bem definida para comunicação com a Arquitetura. As aplicações se comunicam diretamente com o Controlador, que realiza a atividade requisitada ou reencaminha as chamadas para o roteador. Para cada aplicação registrada no Controlador é associada um identificador único (UUID).

O Controlador é o componente que contém as filas de mensagens de cada aplicação, que retransmite as chamadas de transmissão de dados, que atende as chamadas de recebimento de dados pelas aplicações e quem gera os identificadores. O Controlador comunica-se diretamente com o Roteador e com as aplicações. Cada aplicação possui uma única fila de mensagens do tipo First In First Out (FIFO) armazenadas e administradas pelo Controlador.

O Roteador realiza o envio e o recebimento das mensagens, gerencia as Interfaces de rede disponíveis e auxilia na manutenção da rede. O Roteador possui uma tabela contendo todos os destinos acessíveis pelo nodo e todas as aplicações cadastradas no nodo. Quando não há nenhuma aplicação cadastrada, o Roteador funciona exclusivamente como uma ponte. Associado a cada UUID na tabela de roteamento está o seu endereço real da Interface de rede, assim, a Interface de rede não opera sobre os UUIDs, apenas sobre os endereços definidos pela sua pilha de protocolos.

A Interfaces de redes operam como repetidores dos dados pelo, ou para o, Roteador. As Interfaces possuem o conhecimento *single-hop* da rede, enquanto que o Roteador detém o conhecimento dos próximos hops disponíveis.

4.2. Roteamento

A transmissão dos dados entre os nodos acontecerá de forma reativa e baseada na fonte, pois os dispositivos, em sua maioria, são dependentes de bateria, com isso, transmissões e roteamentos só acontecerão quando necessário. O protocolo de roteamento utilizado

levará em conta a necessidade de interconectar diferentes redes entre si, ultrapassando a barreira das características específicas das tecnologias utilizadas.

Inicialmente, o roteador terá conhecimento de quantas interfaces de rede estão conectadas a ele, mas a ideia final do trabalho é desenvolver o roteador de maneira a suportar a inclusão e remoção dinâmica das interfaces de rede.

5. Implementação

A implementação deste trabalho será realizada para dispositivos móveis do tipo *tablet*, com sistema operacional Android. Por limitação dos equipamentos disponíveis, será utilizada a API 12 do Android, lançada na versão *Gingerbread* (2.3). O desenvolvimento do código fonte será na linguagem Java, padrão para desenvolvimento Android, e na linguagem XML, padrão para definição de interface gráfica no Android.

A aplicação apresentada na figura 3 é um programa com interface gráfica (*Guided user interface*, GUI) que será capaz de enviar e receber arquivos por meio da arquitetura proposta. Para realizar a transmissão de um arquivo deve-se rodar a mesma aplicação em dois nodos diferentes da rede ou no mesmo nodo, para usar a capacidade de comunicação local (semelhante ao IP de *localhost* 127.0.0.1).

O Controlador e o Roteador, apresentados na seção 4.1, serão implementados utilizando somente as bibliotecas padrão do Java, a fim de evitar o acoplamento desses componentes no ambiente Android e facilitar a implementação da Arquitetura em outros ambientes. O Controlador é o componente responsável por gerar os identificadores únicos (UUID) para os nodos da rede, com isso, utilizará a classe *UUID* do pacote *java.utils* para a geração desses identificadores.

As interfaces de rede, o controlador e o roteador serão utilizados por meio de serviços do ambiente Android, que possibilitará a existência de uma instância única de cada componente, facilitando a comunicação de diversas aplicações com o controlador e entre os componentes.

Duas tecnologias de transmissão de dados serão implementadas, o Wi-Fi e o Bluetooth. Ambas tecnologias estão disponíveis nos *tablets* e podem ser dinamicamente ativadas e desativadas. Assim, cada nodo da arquitetura possuirá duas interfaces de rede, que serão ativadas e desativadas a fim de simular nodos com diferentes *hardwares*.

A implementação das interfaces de rede engloba o envio e recebimento de dados na tecnologia específica da interface e encaminhamento dos mesmos para o roteador. Com isso, as interfaces de rede possuem um conhecimento *single-hop* da rede, ou seja, só possuem informações dos seus vizinhos diretos.

A lógica do roteamento, elaborada por meio do protocolo de roteamento, está centralizada no roteador, que detêm o conhecimento *multi-hop* da topologia. E é por meio desse conhecimento que ele é capaz de encaminhar os dados para os destinatários.

6. Casos de Teste

A arquitetura proposta será implementada utilizando a combinação da linguagem Java com a plataforma Android. O ambiente de testes será um conjunto de *tablets* executando uma aplicação de transferência de arquivos.

A topologia de teste inicial é a da figura 4, utilizando a capacidade do Android de desligar as interfaces de rede. Outra finalidade da capacidade de desligamento de interfaces de rede do Android será a de testar o roteador e sua habilidade de lidar com interfaces de rede incluídas dinamicamente.

Outras topologias também serão utilizadas para testar a arquitetura no caso de existirem múltiplos caminhos e laços na rede. Exemplos de topologias de teste são apresentadas na **Figura 41** e na **Figura 42**.

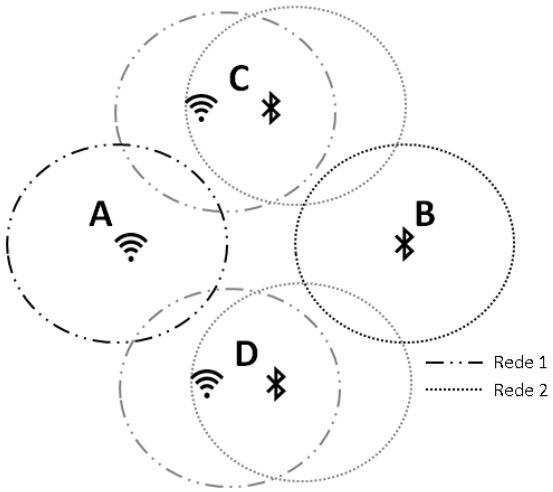
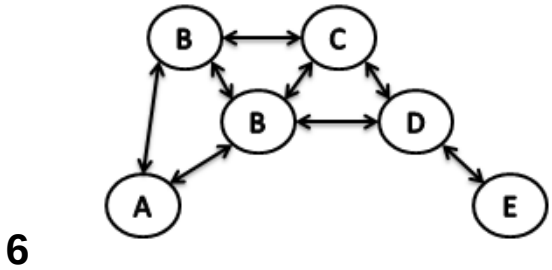


Figura 41 Topologia com mais de um caminho entre A e B



6

Figura 42 Caminho de A até E com possíveis laços

7. Cronograma

Dando continuidade ao trabalho realizado até o momento, será definido o protocolo de roteamento baseado na fonte utilizado na arquitetura proposta neste artigo. Após selecionado, será acrescentada no protocolo a habilidade de operar sobre diferentes redes e utilizando o identificador UUID definido neste artigo.

Com o protocolo de roteamento estruturado será atribuindo a cada par (Aplicação/Controlador, Controlador/Roteador, Roteador/Interfaces de Rede) um conjunto de operações para fornecer ao usuário a capacidade de descobrir os nodos da rede e de enviar/receber dados. Serão definidas as estruturas de dados utilizadas na interface de programação da aplicação (API).

A proposta final da arquitetura será analisada do ponto de vista teórico, a fim de eliminar incoerências que possam evitar o funcionamento correto da arquitetura. A arquitetura final será implementada em dispositivos móveis e testado em ambiente real para, além de validar a proposta de trabalho, analisar a viabilidade comercial do mesmo.

As atividades da segunda etapa deste trabalho de graduação estão definidas semanalmente no seguinte cronograma:

Tarefas\Mês		AGOSTO				SETEMBRO				OUTUBRO				NOVEMBRO			
		1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
1	Escolha e validação do protocolo de roteamento	x	x	x													
2	Definição das APIs		x	x	x												
3	Configuração do ambiente de desenvolvimento e de testes				x												
4	Implementação da Aplicação do usuário					x	x			x				x			
5	Implementação do Controlador					x	x	x	x								
6	Implementação do Roteador						x	x	x	x							
7	Implementação da interface de rede para Wi-Fi										x	x					
8	Implementação da interface de rede para Bluetooth										x	x					
9	Testes da arquitetura							x	x	x	x	x	x	x			
10	Aquisição de dados de análise de desempenho da arquitetura													x	x		
11	Escrever a monografia											x	x	x	x	x	x

Como apresentado no início desta seção, os itens 1 e 2 serão os primeiros realizados. Em seguida, será configurado o ambiente de desenvolvimento, com a instalação e configuração do Android Software Development Kit (SDK) na máquina de desenvolvimento. Com o ambiente de desenvolvimento pronto, será desenvolvida a aplicação no nível do usuário sucedida da implementação do Controlador e do Roteador.

No mês de outubro serão desenvolvidas as interfaces Wi-Fi e Bluetooth e começará os testes da aplicação. No final desse mês inicia a escrita da monografia.

Para o mês de novembro, fica reservada a escrita da monografia e a aquisição de dados de desempenho da arquitetura. Também considera-se que eventuais atrasos de desenvolvimento podem ser resolvidos no início do mês de novembro.

Referências

Kuo, Y. (2009) “A Lightweight Routing Protocol for Mobile Target Detection in Wireless Sensor Networks”, Yeh W. e Chen C., Department of Computation Science & Informatics, Soochow University.

Network Working Group (2005) “A Universaly Unique Identifier (UUID) URN Namespace”, http://www.ietf.org/rfc/rfc4122.txt, Junho.

Brignoni, G. V. (2005) “Estudo de protocolos de roteamento em Redes Ad hoc”, Departamento de Informática e Estatística, Universidade Federal de Santa Catarina.

Tannenbaum, A. S. (2003), Computer Networks, Pearson Education/Prentice Hall, 4ª edição.

Perkins, E. Royer e S. DAS. “Ad hoc On Demand Distance Vector”, http://moment.cs.ucsb.edu/AODV/aodv.html, Junho.

Johnson, Malz (1996) “Dynamic Source Routing in ad hoc wireless networks”, Mobile Computing, Editado por Tomasz Imielinski e Hank Korth, Kluwer Academic Publishers, US.