Universidade do Minho
Escola de Engenharia
Departamento de Informática
Licenciatura em Engenharia Informática

Comunicações por Computador

"Desenho e implementação de um serviço de mensagens em rede AdHoc"



Pedro Faria - A60998
Mariana Medeiros - A61041
Miguel Pinto - A61049

1 Introdução

Foi proposto aos alunos inscritos da Unidade Curricular de Comunicações por Computador no presente ano letivo 2013/2014 que desenhassem e implementassem um serviço de mensagens em rede AdHoc, sobre IPv6.

As redes AdHoc são diferentes de todos os outros tipos de redes porque, ao contrário do que é comum, as redes AdHoc não possuem uma topologia predeterminada e não possuem um só ponto de acesso. Por isso, numa rede AdHoc, de formação espontânea, os nós têm de ser capazes de descobrir rotas para outros nós com ajuda dos seus vizinhos, criando assim uma rede.

Para além disso, era necessário o uso dos dois protocolos estudados nas aulas teóricas e práticas: UDP e TCP. O primeiro usado para descobrir rotas e o segundo para enviar e receber mensagens.

Neste relatório, e com o objetivo de que seja o mais elucidativo possível, são explicadas detalhadamente todas as tomadas de decisões e o desenvolvimento do projeto. No final, é também elaborada uma análise dos resultados obtidos e as respetivas conclusões.

2 Especificação do Protocolo

2.1 Primitivas de Comunicação e Formato das Mensagens Protocolares (PDU)

Para garantir a comunicação na topologia, foram criadas as seguintes primitivas:
- Protocolo Hello;
- Protocolo Route Request;
- Protocolo Route Reply;
- Protocolo Message.

Já para a comunicação entre o cliente e um host específico da topologia, foi criado o protocolo TW.

2.1.1 Protocolo Protocol («abstract»)

Este protocolo é extendido por todos os outros protocolos existentes na topologia e tem como função principal armazenar os dados do host emissor antes de ele enviar um pacote.

Para armazenar este dados, recorremos à classe *InetAddress* que se refere ao IP do host que vai enviar o pacote (host emissor) e uma *String* que armazena o nome do host da topologia.

Este trabalho prático assenta essencialmente sobre o protoclo UDP, o que não nos dá a garantia de que a mensagem foi entregue.

2.1.2 Protocolo Hello

As mensagens de Hello estão constantemente a ser enviadas para os seus vizinhos diretos, por multicast. Estes, por sua vez, quando acordam enviam também uma mensagem aos seus vizinhos diretos pelo mesmo método. Este processo é repetido o tempo todo.

No que diz respeito às mensagens de Hello, estas têm um formato simples e bem definido:

Contêm um texto que identifica a mensagem do protocolo e também uma lista dos vizinhos diretos (N=1) que conhece até ao momento. Quando o pacote chega a um dos vizinhos, este retira da lista o próprio nó e identifica os hosts que são sub-vizinhos do host, ficando a conhecer os vizinhos e sub-vizinhos num raio N=2.

Um pacote Hello é enviado periodicamente de 5 em 5 segundos (hello interval) para os vizinhos diretos. Sempre que chega um pacote Hello ao host, é verficado o tempo da última escuta de todos os vizinhos; se ultrapassar o tempo definido - 20 segundos (dead interval) - esse vizinho é declarado como inacessível e desativado da topologia.

Estas duas váriaveis (hello interval e dead interval) são definidas estaticamente na classe própia deste protocolo.

2.1.3 Protocolo Route Request e Route Reply

As mensagens do Route Request são despoletadas sempre que o host destino da mensagem está fora da vizinhança conhecida do host recetor da mensagem. Neste momento, é realizada a descoberta de uma rota para o destino. O nó de destino, ou um nó com uma rota válida na tabela de vizinhança N=2, pode responder ao pedido com uma mensagem Route Reply que contém o caminho que o host emissor tem de percorrer até conseguir chegar ao destino da mensagem.

Estas mensagens também devem ser enviadas e recebidas na porta 9999 (UDP).

Ao longo da descoberta de rota para um determinado destino é preenchida uma lista com os hosts visitados até ao momento para se evitar ciclos infinitos.

No protocolo Route Request estão definidos o nome do host destinatário deste pacote e uma lista com o caminho percorrido neste processo de busca. Além destas variavéis, estão definidos mais dois inteiros que indicam o método de busca deste protocolo (não implementado): o Timeout (tempo máximo de espera por um pedido de rota) e o Radius (raio máximo para procurar um destino).

O protocolo Route Reply é definido com uma lista de hosts que terão de ser percorridos para se conseguir entregar uma mensagem caso o host não esteja dentro da vizinhança. Este caminho depois de ser encontrado para o host destino, deve ser percorrido inversamente até ao nodo emissor do pedido Route Request. Para o ajudar nesta tarefa está definido um inteiro que controla em que estado do caminho se encontra.

2.1.4 Protocolo Message

O protocolo Message é criado depois de se ter recebido um pacote TW, onde são armazenados os nodos de emissão e de destino da mensagem e o texto que o utilizador efetuou no *tweet* anterior.

Assim que é o recebido este protocolo, via UDP na porta 9999, é desencadeado um processo que verifica se é possível entregar a mensagem diretamente ao destinatário sem ser necessário recorrer ao protocolo Route Request. Caso o destino deste pacote seja um vizinho ou sub-vizinho do próprio a mensagem, é enviada

2.1.5 Protocolo TW

Este protocolo é usado sempre que surge um *tweet* através do telnet. Nesse momento, é criado um pacote TW que contém a mensagem, o host destino e algumas informações referentes ao host emissor. Caso o tamanho da mensagem seja superior a 500 bytes (valor definido no objeto pela variável *size*) esta não é enviada para o localhost, para o tratamento do pacote.

2.2 Interações

No que diz respeito às interações da nossa topologia, elas variam de protocolo para protocolo sendo que cada nó inicialmente interage apenas com os seus vizinhos diretos.

No protocolo Hello um nó interage com todos os seus vizinhos diretos através do envio de uma mensagem Hello.

No protocolo Route Request um nó (nó emissor) interage com os seus vizinhos através de uma mensagem Route Request, onde verifica se estes contêm nos seus sub-vizinhos o host que procura ou uma rota válida para o destino (nó emissor –> nó destino).

Já no protocolo Route Reply, este interage de modo inverso ao protocolo Route Request, como já possuí um caminho válido para o host destino, percorre inversamente está lista para devolver essa informação ao nó emissor do pedido.

No protocolo Message, um nó interage com os seus vizinhos diretos e com os seus sub-vizinhos. Após o tratamento e devida separação dos dados da mensagem, este envia a mensagem de duas formas distintas: caso o nó destino se encontre na sua vizinhança (vizinhos e sub-vizinhos), envia a mensagem diretamente. Caso contrário, é então necessário recorrer ao protocolo Route Request.

Por fim, no protocolo TW um *handler* recebe uma mensagem do utilizador onde posteriormente é convertido em TW contendo a mensagem, o nó destino e informações do nó emissor e a mensagem, procedendo desta forma ao seu envio para o localhost caso o seu tamanho seja válido.

3 Implementação

De modo a conseguir responder a todos os objetivos deste trabalho prático foram ainda definidos 6 objetos para a estruturação do projeto.

A classe Utilities contém funções estáticas que permitem serializar/desserializar um objeto em Java e descobrir o IP e nome associado a um host na topologia.

O objeto Host contém as seguintes variáveis de instância:
private InetAddress ip;
private String name;
private int porta;
private GregorianCalendar lastResponse;
private int status;
private HashMap<String,InetAddress> vizinhos;

As variáveis *ip* e *name* armazenam informação relativa ao ip e ao nome do host. O *lastResponse* serve para verificar se host ainda se encontra ativo no sistema, guardando o valor da última vez que o host recebeu um pacote Hello. Quando este host fica inativo, o estado dele é mudado e a variável status é alterada para 0. O HashMap guarda informação relativa aos vizinhos diretos do host que serão depois enviados num protocolo Hello a estes vizinhos diretos para o conhecimento de vizinhos nível 2.

O HandlerMessage é uma thread que é iniciada em cada host para ficar à escuta de pedidos TCP feitos pelo telnet quando um utilizador pretender fazer um *tweet*, sempre que surge um pacote em formato de texto este é dividido para se criar o pacote TW que será enviado para o HandlerReceive.

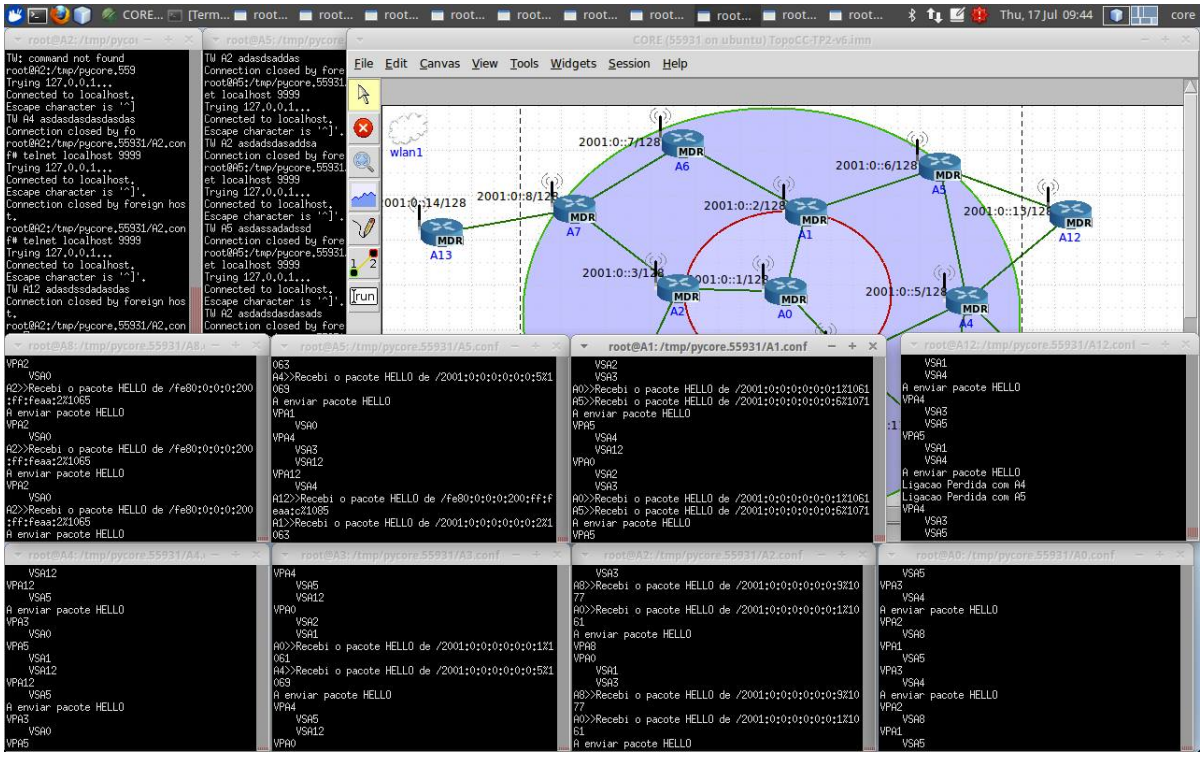
O HandlerSend é o responsável por enviar periodicamente os pacotes Hello para os vizinhos diretos de cada Host. É aqui que é criado esse pacote e fornecido os dados necessários da vizinhança de segundo nível.

O HandlerReceive recebe o pedido de vários tipos de protocolos e define para um dado tipo a tarefa que irá desenvolver. Como a lista de vizinhos pode ser alterada de um momento para o outro e para evitar perdas de dados, os métodos foram definidos concorrentemente recorrendo à sicronização.

AdhocApp é o objeto responsável por começar a execução das tarefas para comunicação entre nodos.

4 Testes e Resultados

No final do desenvolvimento do projeto, procedeu-se a alguns testes. De seguida são apresentados alguns dos resultados obtidos:



5 Conclusões e Trabalho Futuro

Em modo de conclusão, vemos este projeto como mais uma forma de pôr em prática os conhecimentos adquiridos e as ferramentas disponibilizadas na Unidade Curricular de Comunicações por Computador.

O projeto foi desenvolvido de maneira a conseguir dar resposta a tudo o que nos foi solicitado. No entanto, algumas das funcionalidades não chegaram a ser totalmente implementadas. Por exemplo, no protocolo Route Request, deveria ser possível indicar o método de procura do destino (através da variável Timeout ou da variável Radius) mas tal não foi conseguido.

De uma forma geral, e apesar de algumas falhas, conseguimos focar-nos numa das funcionalidades essenciais do projeto que seria o envio de mensagens que o utilizador inseriu para os vizinhos diretos e sub-vizinhos.

No futuro e com o objetivo de aprendermos mais coisas, podemos continuar a desenvolver as funcionalidades que ficaram agora pelo caminho.