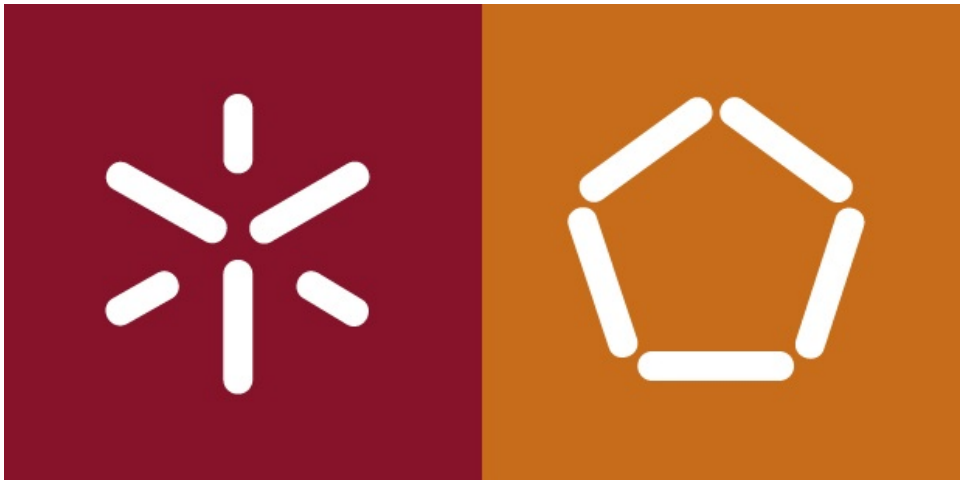

Arquiteturas Emergentes de Redes TP1

Grupo 9
Bruno Chianca Ferreira PG33878
João Rui de Sousa Miguel A74237
Paulo Jorge Machado Guedes A74411

Universidade do Minho

Resumo Este artigo discorre sobre o projeto e prototipagem de um algoritmo de roteamento para redes *Adhoc* baseando-se no paradigma da espontaneidade onde cada nodo anuncia sua presença na rede, ao mesmo tempo que recebe os anúncios de seus vizinhos de rede para a construção de sua própria tabela de roteamento. Para exemplificar o funcionamento, foi implementada uma aplicação de difusão de notícias.

Keywords: adhoc, encaminhamento, UDP, TPC, python, IPV6

1 Introdução

Numa rede Adhoc, cada utilizador é um *router*, que serve para redirecionar pacotes. Para tal, é necessário conhecer um número mínimo de utilizadores na rede, de modo a transmitir os pacotes para esses mesmos, com o intuito que o pacote chegue ao utilizador destinado.

Neste projeto, cada utilizador conhece os utilizadores da rede até nível dois, ou seja, conhece os seus vizinhos diretos, e os vizinhos diretos desses seus mesmos vizinhos diretos, guardando essas informações numa tabela de encaminhamento.

Para tal, foi criado um protocolo "Hello", onde se enviará para vizinhos diretos do utilizador, um pacote que contêm os seus próprios vizinhos diretos. Caso necessite de comunicar com um utilizador que não conheca, foi criado um novo protocolo, "Route Request", que procura, com um número limite de saltos e com um limite de tempo até dar *timeout*, um caminho até esse mesmo utilizador.

Por fim, foi criado um protocolo aplicacional, que testa a tabela de encaminhamento, procurando as noticias (através de um pedido tcp feito pela parte aplicacional, para o nível de rede) de um utilizador na rede.

2 Definição do Protocolo de Encaminhamento

De modo a conhecer os vizinhos na rede, foi criada uma tabela de encaminhamento, que guarda as informações desses vizinhos, mais especificamente:

1. Nome do nodo
2. Nome do vizinho para o próximo salto de modo a chegar ao nodo pretendido
3. Endereço IPv6 até o vizinho para o próximo salto
4. *Timestamp* de quando foi atualizado o caminho para o nodo pretendido
5. *Round Trip Time* (RTT) até ao vizinho do próximo salto

Deste modo, é possível guardar as informações necessárias para reencaminhar pacotes até ao nodo pretendido (nome do nodo), e é possível atualizar a tabela de encaminhamento, escolhendo os caminhos com menor RTT. É possível também manter a tabela atualizada, removendo todos os caminhos que não possuam registos recentes, removendo todos os caminhos cujo *timestamp* seja de à mais de duas vezes o tempo do protocolo hello.

2.1 Protocolo Hello

Este protocolo é responsável por manter a tabela de encaminhamento atualizada com os vizinhos até nível 2. Envia aos seus vizinhos diretos os seus próprios vizinhos diretos, notificando que caso necessite de comunicar com algum deles, este está disponível para encaminhar os pacotes. Cabe a estes vizinhos decidir qual caminho guardar, para cada utilizador, dependendo de várias informações sobre a rede.

Para enviar aos seus vizinhos, é necessário que todos os utilizadores da rede Adhoc, estejam no mesmo grupo IPv6, nomeadamente no grupo ff02::1. Assim, os pacotes são enviados em *multicast* para este grupo. Este protocolo apenas irá comunicar por UDP com outros utilizadores, através da porta 9999, e será enviado periodicamente para a rede Adhoc.

Este protocolo terá o seguinte PDU:

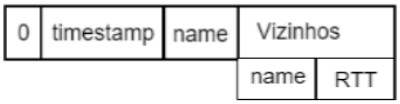

Figura 1. PDU do pacote Hello

1. O primeiro parâmetro servirá para identificar o tipo de protocolo da mensagem, permitindo um melhor *parse*;
2. Deverá ser enviado um *timestamp* de quando o pacote foi enviado, de modo a calcular o RTT da conexão entre o utilizador e o vizinho
3. Nome do utilizador que enviou o pacote
4. Lista dos vizinhos diretos do utilizador:
   (a) Nome do vizinho do utilizador
   (b) RTT da conexão entre o utilizador e o vizinho

2.2 Protocolo Route Request

Para que o utilizador possa comunicar com um utilizador que não esteja na sua tabela de encaminhamento (a uma distância maior que 2 saltos), foi criado um novo protocolo, protocolo "Route Request".

Este protocolo possui um número de saltos limite, e um tempo de *timeout*, em que após esse período de tempo, o utilizador já não está à espera de resposta, nem esta será retransmitida.

Este protocolo apenas comunicará através de UDP com os outros utilizadores, através da porta 9999, e deverá ter o seguinte PDU:

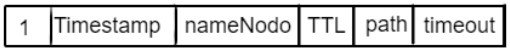

Figura 2. PDU do pacote Route Request

1. O primeiro parâmetro servirá para identificar o tipo de protocolo da mensagem, permitindo um melhor *parse*;
2. Deverá ser enviado um *timestamp* de quando o pacote foi enviado, de modo a verificar se o pacote já expirou o tempo de *timeout*;
3. Nome do utilizador que deseja conhecer o caminho para;
4. Número de saltos máximo até atingir o utilizador destino;
5. Caminho percorrido até ao momento;
6. Periodo de tempo pelo qual o utilizador que enviou o pacote "Route Request"irá esperar pela resposta.

Caso encontre o utilizador desejado, este deverá responder ao pacote "Route Request"com um novo protocolo "Route Reply".

2.3 Protocolo Route Reply

Este protocolo apenas será utilizado caso um pacote "Route Request"seja bem sucedido. Assim sendo, será criado um pacote "Route Reply"no nodo que conhecer o utilizador pretendido, que é enviado por UDP, através da porta 9999, realizando o caminho inverso ao "Route Request".

Sendo assim, o PDU do protocolo "Route Reply"será:

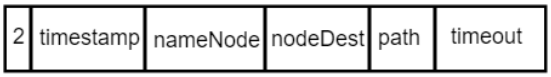

Figura 3. PDU do pacote Route Reply

1. O primeiro parâmetro servirá para identificar o tipo de protocolo da mensagem, permitindo um melhor *parse*;
2. Deverá ser enviado um *timestamp* de quando o pacote foi enviado, de modo a verificar se o pacote já expirou o tempo de *timeout*;
3. Nome do utilizador que irá enviar o "Route Reply";
4. Nomde do utilizador que o pacote "Route Request"desejava conhecer;
5. Caminho a percorrer;
6. Periodo de tempo pelo qual o utilizador que enviou o pacote "Route Request"irá esperar pela resposta.

2.4 Protocolo Aplicacional

Com o intuito de simular a camada de aplicação na pilha de rede, foi implementado um protocolo para difusão de notícias de um nodo para outro. O servidor de notícias pode ser qualquer um dos nodos e o seu funcionamento é independente do protocolo de roteamento.

Para simular a camada de aplicação é criado um socket *TCP* na porta 9999. Este socket deve receber os pacotes do **cliente** e interpretar as requisições. Caso receba uma requisição inválida deve imediatamente descartar o pacote. Para já, duas primitivas válidas estão disponíveis: **GET e NEWS**.

O cliente, quando deseja requisitar notícias de um nodo, deve então enviar à porta *TCP:9999* um pacote no formato **GET([nodo origem],[nodo destino])**, onde o nodo de origem é o próprio nodo onde o cliente esta a correr e nodo destino é o nodo de onde o cliente deseja receber as notícias.

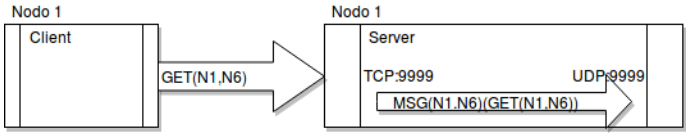

Figura 4. Comunicação entre Cliente e Servidor

O programa responsável pela gestão da camada de aplicação deverá então receber o pacote, identificar a diretiva **GET** válida e adicionar um cabeçalho do tipo **MSG** referente a sua própria camada e de seguida enviar o pacote à próxima camada de encaminhamento, que aqui está representada como um socket *UDP:9999*. O protocolo de encaminhamento por sua vez ao receber o pacote válido verifica se é um pacote do tipo GET ou NEWS.

GET

No caso de uma mensagem do tipo **GET**, o agente verifica quem é o destinatário e caso seja ele mesmo requisita ao servidor de notícias as notícias disponíveis para envio. A partir daí o servidor cria um pacote novo do tipo NEWS formato **NEWS([nodo origem],[nodo destino]),(news)**, onde o nodo de origem é o próprio nodo onde o servidor de notícias está a correr e nodo destino é o nodo do cliente que requisitou as notícias, concatenado com as notícias atuais do nodo servidor.

Caso não seja ele o destinatário final, ele acede à tabela de encaminhamento e reencaminha o pacote recebido para o nodo que consta como *NEXT HOP* referente ao destinatário. Caso não consiga encontrar o destinatário na tabela de encaminhamento, efetua um comando *ROUTE REQUEST* e tenta novamente encaminhar o pacote.

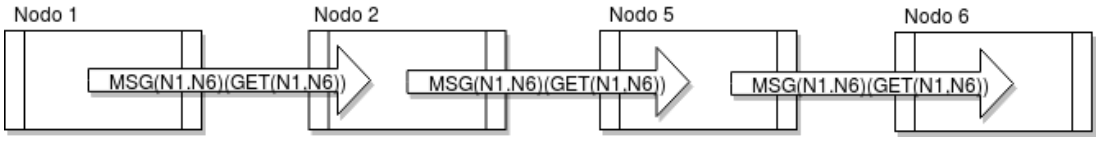

Figura 5. Reencaminhamento dos pacotes até o destino

Quando o pacote finalmente chega ao servidor, o mesmo procedimento previamente descrito é executado e o novo pacote segue de volta ao cliente junto com as notícias.

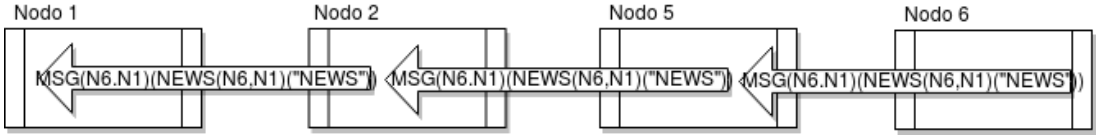

Figura 6. Pacote de NEWS agora com a direção invertida

NEWS

No caso de uma mensagem do tipo **NEWS**, o agente verifica quem é o destinatário e caso seja ele mesmo, envia a mensagem à camada de aplicação via socket *TCP:9999*. O agente então identifica que é um pacote do tipo NEWS e encaminha ao cliente requisitante fazendo uso do mesmo socket previamente aberto.

Caso não seja ele o destinatário final, ele acede à tabela de encaminhamento e reencaminha o pacote recebido para o nodo que consta como *NEXT HOP* referente ao destinatário. Caso não consiga encontrar o destinatário na tabela de encaminhamento, efetua um comando *ROUTE REQUEST* e tenta novamente encaminhar o pacote.

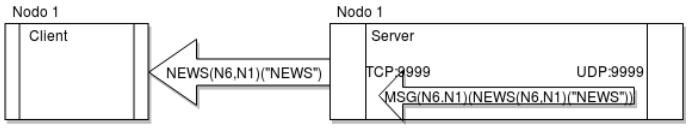

Figura 7. Pacote de NEWS retorna ao cliente

3 Implementação do Protocolo

3.1 Protocolo Hello

Este protocolo terá de ser enviado periodicamente, mas deverá ser estabelecido um balanço de modo a não fazer um *flood* da rede com pacotes "Hello", prejudicando a qualidade da rede, mas de modo a manter as conexões minimamente atualizadas, de modo a manter registadas as mudanças na rede. Decidiu-se que seria enviado a cada 10 segundos, um pacote deste protocolo em *multicast*, visto tratar-se de um modelo experimental de pequenas dimensões.

Assim, quando um utilizador necessita enviar um pacote "Hello", verifica quais as conexões na tabela de encaminhamento são antigas, removendo-as, e construindo de seguida o pacote com os seus vizinhos diretos, baseado na tabela de encaminhamento. É declarada uma conexão antiga caso tenha sido atualizada apenas à dois pacotes "Hello"atrás. Envia também, juntamente com os seus vizinhos diretos, o RTT associado da conexão com cada vizinho, assim como um *timestamp* de quando enviou o pacote, de modo a que os utilizadores que recebam o pacote, possam calcular o RTT da conexão.

Quando um utilizador recebe um pacote "Hello", é calculado o *timestamp* atual, e comparado com o *timestamp* enviado no pacote. Juntamente com o RTT de cada vizinho, enviado no pacote "Hello", e o RTT da conexão, calculado com ambos os *timestamps*, é calculado o RTT final até ao vizinho de nível 2.

Deste modo, para cada vizinho no pacote, é verificado se já existe um registo na sua tabela de encaminhamento, e caso não exista, é adicionado um novo registo com o *timestamp* de quando recebeu o pacote, o RTT calculado acima, o nome do vizinho de nível 2, e o nome e endereço do próximo salto (que é o nome e endereço do vizinho que enviou o pacote). Caso já exista um registo na tabela, é atualizada a tabela em dois casos:

1. Se o nome do vizinho do próximo salto for igual ao que enviou o pacote;
2. Se o RTT for menor ou igual à conexão que está registada na tabela de encaminhamento;

São guardadas também as informações do vizinho que enviou o pacote, guardando assim as conexões dos vizinhos diretos. Neste caso, o nome do nodo e o nome do vizinho do próximo salto são iguais.

3.2 Protocolo Route Request e Route Reply

Para que o protocolo "Route Request"seja bem sucedido, é necessário enviar um *timestamp* de quando o pacote foi enviado inicialmente, o nome do utilizador que deseja conhecer, e o caminho percorrido até esse utilizador, de modo a que possa ser efetuada a resposta do "route request"pelo caminho inverso, guardando nas tabelas de encaminhamento de todos os utilizadores que façam parte desse caminho, um registo com as informações para comunicar com o utilizador pretendido.

Inicialmente, o caminho inicia-se com o nome do utilizador que iniciou o "route request", e é enviado em *multicast*, para o grupo IPv6 "ff02::1".

Caso os utilizadores que recebam esse "route request"possuam na sua tabela de encaminhamento, o utilizador com quem se pretende comunicar, é enviado um pacote "route reply", com o percurso inverso ao caminho enviado no "route request". Os utilizadores que receberem um pacote "route reply", guardam na sua tabela de encaminhamento um registo, guardando o nome do utilizador com quem se pretende comunicar (nameNode), e o nome e endereço IPv6 de quem enviou o pacote, visto que esse é o próximo salto a tomar caso queira comunicar com esse mesmo utilizador.

Caso os utilizadores que recebam um "route request"não possuam na sua tabela de encaminhamento o utilizador com quem se pretende comunicar, o utilizador adiciona o seu nome ao caminho do pacote, e reenvia em *multicast* para o mesmo grupo. Para evitar *loops*, este processo apenas acontece caso o seu nome não esteja já no caminho, caso contrário, não retransmite o pacote. É verificado, após receber o pacote, se o tempo de timeout já não excedeu, comparando o *timestamp* enviado no pacote "route request"ou "route reply"com um *timestamp* atual.

Assim que o caminho do pacote "route reply"acabe, significa que um caminho foi encontrado. Após receber um "route reply", todos os outros "route reply"dentro do mesmo periodo de tempo (timeout) são ignorados, porque fazem referência a um caminho que o utilizador já conhece, mas que tem um RTT maior.

3.3 Protocolo Aplicacional

Para implementação do protocolo aplicacional foi criada uma função em *python* que corre como um *thread* da aplicação "adhoc_app.py"previamente descrita. A função abre um socket TCP:9999 e começa a escutar esperando por conexões em *loop* infinito. Quando recebe uma conexão, aceita-a, e começa a esperar que dados lhe sejam enviados. Quando recebe os dados, extrai a primitiva(GET ou NEWS). Caso seja do tipo GET, primeiramente guarda o objeto da conexão aberta com o cliente para posterior uso e de seguida forma um novo pacote da seguinte forma:

[3,"MSG", (meu_nodo), (dest_nodo), (pacote_recebido)], onde:

1. 3 - Identificador de pacote do tipo 3 (Mensagem de aplicação).
2. MSG - Identicador do nível aplicacional.
3. (meu_nodo) - Nome do nodo que recebeu o pacote TCP.
4. (dest_nodo) - Nome do nodo destinatário do pacote.
5. (pacote_recebido) - O pacote previamente recebido via TCP.

Após formado, o pacote é então enviado para o nível de encaminhamento fazendo-se uso de um socket UDP:9999 para o próprio nó, onde a mensagem será devidamente encaminhada. Caso a primitiva seja do tipo NEWS, o pacote TCP é extraido, e enviado ao cliente fazendo uso da mesma conexão previamente aberta. O cliente irá extrair as noticias do pacote TCP recebido, após o qual as conexões TCP são finalizadas.

Caso o pacote TCP recebido seja vazio, significa que não foi encontrado o utilizador. O socket UDP:9999 recebe o pacote e identifica o tipo de pacote 3(Mensagem de aplicação). Quando identifica o cabeçalho válido do tipo MSG, verifica o destinatário do pacote. Caso seja ele mesmo, tenta identificar o tipo de pacote a nível de aplicação, procurando as primitivas válidas GET ou NEWS.

GET

Quando identifica a primitiva do tipo GET, o primeiro passo é procurar as notícias a serem enviadas fazendo uso da função *get.news()*. A função então retorna as notícias a serem enviadas. De posse das notícias, o servidor cria então um novo tipo de pacote, trocado a primitiva GET por NEWS, invertendo (meu_nodo) por (dest_nodo) e concatenando as notícias a serem enviadas. Após formar a nova mensagem, um novo socket UDP é aberto para envio da mensagem novamente ao protocolo de encaminhamento através da porta 9999.

NEWS

Quando identifica a primitiva do tipo NEWS, o primeiro passo é abrir um socket TCP na porta 9999 da camada de aplicação e de seguida enviar o pacote de notícias removendo o cabeçalho MSG. Após o envio o socket TCP é então fechado. Caso o destinatário não seja ele, a mensagem deve ser então encaminhada. Para isso, busca em sua tabela de encaminhamento corrente pelo destinatário, caso encontre, abre um socket UDP:9999 envia a mensagem inalterada para o *NEXT HOP* identificado na tabela. Se o destinatário não estiver na tabela, faz uso da função *route_request()* para que a tabela de encaminhamento seja atualizada com o destinatário e então reenvia a mensagem inalterada para o protocolo de encaminhamento, repetindo o processo.

4 Testes e Resultados

Para efeitos de teste, utilizou-se a topologia *core* fornecida pela equipa docente "**Topo-AdHoc-v6.imn**". Simulou-se então a ligação entre um cliente em **A0** com um servidor em **A14**. Inicialmente verificou-se o conteúdo dos dicionários de *hello* e *routing*, onde se observa os nodos que dizem respeito a vizinhos diretos em *hello* e os nodos com $ttl \leq 2$. As figuras seguintes dizem respeito à topologia utilizada e ao conteúdo inicial destes mesmos dicionários nos nodos utilizados (**A0**, **A2**, **A8** e **A14**).

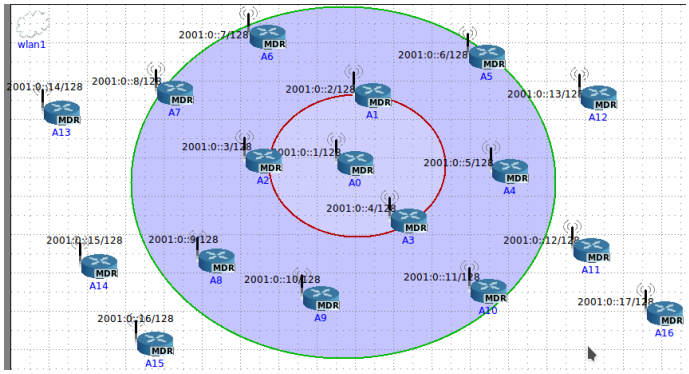

Figura 8. Modelo usado para testes no CORE

```
A0#>hello
Current hello table:
{'A2': 0, 'A3': 0}
A0#>route
Current routing table:
Node Name       | Next hop      | IPV6 address          | Timestamp     | RTT
A2              | A2            | fe80::200:ff:feaa:2   | 1522681868    | 0
A3              | A3            | fe80::200:ff:feaa:3   | 1522681861    | 0
A8              | A2            | fe80::200:ff:feaa:2   | 1522681868    | 0
A9              | A3            | fe80::200:ff:feaa:3   | 1522681861    | 0
```

Figura 9. Conteúdo inicial dos dicionários de *hello* e *routing* do nodo **A0**

```
A2#>hello
Current hello table:
{'A8': 0, 'A0': 0}
A2#>route
Current routing table:
Node Name       | Next hop      | IPV6 address          | Timestamp     | RTT
A14             | A8            | fe80::200:ff:feaa:8   | 1522681868    | 0
A9              | A8            | fe80::200:ff:feaa:8   | 1522681868    | 0
A8              | A8            | fe80::200:ff:feaa:8   | 1522681868    | 0
A3              | A0            | fe80::200:ff:feaa:0   | 1522681866    | 0
A0              | A0            | fe80::200:ff:feaa:0   | 1522681866    | 0
```

Figura 10. Conteúdo inicial dos dicionários de *hello* e *routing* do nodo **A2**

```
A8#>hello
Current hello table:
{'A2': 0, 'A9': 0, 'A14': 0}
A8#>route
Current routing table:
Node Name       | Next hop      | IPV6 address          | Timestamp     | RTT
A2              | A2            | fe80::200:ff:feaa:2   | 1522681868    | 0
A0              | A2            | fe80::200:ff:feaa:2   | 1522681868    | 0
A3              | A9            | fe80::200:ff:feaa:9   | 1522681869    | 0
A9              | A9            | fe80::200:ff:feaa:9   | 1522681869    | 0
A14             | A14           | fe80::200:ff:feaa:e   | 1522681874    | 0
```

Figura 11. Conteúdo inicial dos dicionários de *hello* e *routing* do nodo **A8**

```
A14#>hello
Current hello table:
{'A8': 0}
A14#>route
Current routing table:
Node Name       | Next hop      | IPV6 address          | Timestamp     | RTT
A9              | A8            | fe80::200:ff:feaa:8   | 1522681878    | 0
A8              | A8            | fe80::200:ff:feaa:8   | 1522681878    | 0
A2              | A8            | fe80::200:ff:feaa:8   | 1522681878    | 0
```

Figura 12. Conteúdo inicial dos dicionários de *hello* e *routing* do nodo **A14**

Para testar a camada de rede *UDP* utilizou-se o comando *route request* no nodo **A0**. Este comando permite descobrir a rota até um nodo destino (neste caso selecionou-se **A14**), sendo que as tabelas de encaminhamento dos nodos intermédios serão atualizadas. De seguida apresenta-se precisamente, não só a instrução executada, como também o resultado das tabelas de todos os nodos.

```
A0#>route request A14 2 5
A0#>Encontrei caminho para o nodo
route
Current routing table:
Node Name       | Next hop      | IPV6 address          | Timestamp     | RTT
A2              | A2            | fe80::200:ff:feaa:2   | 1522681994    | 0
A14             | A2            | fe80::200:ff:feaa:2   | 1522681999    | 0
A3              | A3            | fe80::200:ff:feaa:3   | 1522681997    | 0
A8              | A2            | fe80::200:ff:feaa:2   | 1522681994    | 0
A9              | A3            | fe80::200:ff:feaa:3   | 1522681997    | 0
```

Figura 13. Envio de *route request* de **A0** para **A14**

```
A8#>route
Current routing table:
Node Name       | Next hop      | IPV6 address          | Timestamp     | RTT
A2              | A2            | fe80::200:ff:feaa:2   | 1522682005    | 0
A0              | A2            | fe80::200:ff:feaa:2   | 1522682005    | 0
A3              | A9            | fe80::200:ff:feaa:9   | 1522682014    | 0
A9              | A9            | fe80::200:ff:feaa:9   | 1522682014    | 0
A14             | A14           | fe80::200:ff:feaa:e   | 1522682010    | 0
```

Figura 15. Tabela de encaminhamento de **A8**

```
A2#>route
Current routing table:
Node Name       | Next hop      | IPV6 address          | Timestamp     | RTT
A14             | A8            | fe80::200:ff:feaa:8   | 1522682012    | 0
A9              | A8            | fe80::200:ff:feaa:8   | 1522682012    | 0
A8              | A8            | fe80::200:ff:feaa:8   | 1522682012    | 0
A3              | A0            | fe80::200:ff:feaa:0   | 1522682001    | 0
A0              | A0            | fe80::200:ff:feaa:0   | 1522682001    | 0
```

Figura 14. Tabela de encaminhamento de **A2**

```
A14#>route
Current routing table:
Node Name       | Next hop      | IPV6 address          | Timestamp     | RTT
A9              | A8            | fe80::200:ff:feaa:8   | 1522682012    | 0
A8              | A8            | fe80::200:ff:feaa:8   | 1522682012    | 0
A2              | A8            | fe80::200:ff:feaa:8   | 1522682012    | 0
```

Figura 16. Tabela de encaminhamento de **A14**

Tal como esperado, não se verificam alterações nas tabelas de encaminhamento dos nodos (**A2**, **A8** e **A14**) isto porque todos estes nodos conseguem chegar até a **A14** com um $ttl \leq 2$. A única exceção será o nodo **A0** que passa agora a conter informações sobre *routing* para o nodo **A14**. Esta informação será mantida até que ocorra um *timeout* por não receber mensagens de *hello* periodicamente.

Para testar o protocolo aplicacional foi elaborada uma aplicação simples para enviar pacotes de requisição de notícias. A aplicação *news_agent.py* deve ser iniciada com o parâmetro *client* para que se comporte como um cliente no nodo onde está a correr. O utilizador pode então no prompt de comando executar o comando *get [nodo_destino]* e o programa irá conectar-se à porta *TCP:9999* para ter acesso à camada de aplicação. A figura seguinte exemplifica um teste do nodo **A0** requisitando notícias ao nodo **A12**.

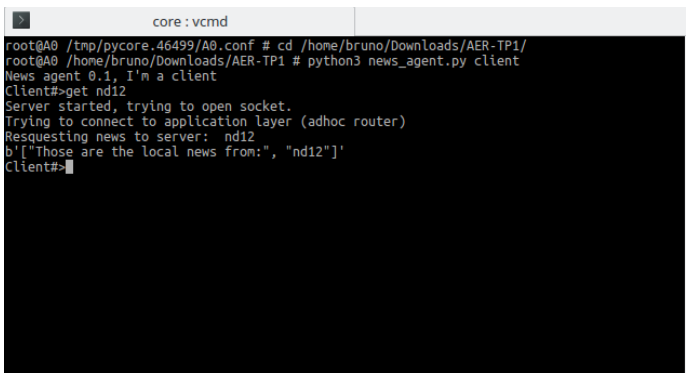

Figura 17. Modelo usado para testes no CORE

5 Conclusões e Trabalhos Futuros

Deste trabalho, retira-se que as redes *adhoc* baseiam-se no reencaminhamento de mensagens por parte de todos os nodos da rede. Neste tipo de topologias, um equipamento que pretenda comunicar terá de servir de intermediário a mensagens que não digam respeito a si próprio.

Facilmente se percebe que esta abordagem poderá apresentar alguns problemas a nível de segurança, deste modo, futuramente seria interessante tentar implementar algum tipo de mecanismos de cifragem de dados para não comprometer a confidencialidade dos dados.

Este trabalho ajudou ainda a perceber os mecanismos envoltos na troca de mensagens numa topologia de redes *adhoc*, não só a nível teórico como a nível de implementação. O grupo considera que conseguiu efetuar todos os objetivos propostos pela equipa docente.

Caso a rede fosse implementada a uma maior escala, o tempo entre pacotes de protocolo "Hello"teria de ser maior, e o número de saltos e tempo de *timeout* nos pacotes de protocolo "Route Request"e "Route Reply"teriam de se adaptar às condições da rede, testando vários valores de saltos e *timeouts*, antes de declarar um utilizador como inalcançável.