

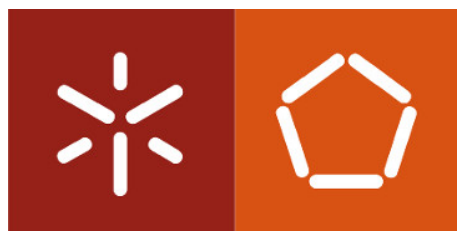
Arquiteturas Emergentes de Rede

4 de Junho de 2021

Grupo 3

a83899	André Morais
A85367	Francisco Lopes
A84485	Tiago Magalhães

Entrega de Conteúdos P2P em Redes Móveis Espontâneas



Mestrado Integrado em Engenharia Informática
Universidade do Minho

Conteúdo

1	Introdução	2
2	Rede P2P	3
2.1	Concepção	3
2.2	Especificação dos protocolos	3
2.2.1	Primitivas de comunicação	3
2.2.2	Interações	4
2.2.3	Formato das mensagens protocolares (PDU)	4
3	Rede de Suporte	5
3.1	Concepção	5
3.2	Especificação dos protocolos	5
3.2.1	Primitivas de comunicação	5
3.2.2	Formato das mensagens protocolares (PDU)	5
3.2.3	Interações	6
4	Testes e Resultados	6
5	Conclusão e Trabalho futuro	9

Resumo

Neste relatório explicamos de que forma é que realizamos as diferentes partes do trabalho prático. Na primeira fase do trabalho foi construída uma rede P2P com descargas de conteúdos. Numa segunda parte foi-nos pedido uma construção uma rede de suporte baseada em nomes e tolerante a atrasos. Explicaremos quais os protocolos que utilizamos, de que forma os utilizamos, como foram projetados e qual a estratégia utilizada.

Keywords: P2P, Multicast, DTN, NDN, UDP, Sockets, CORE, IP

1 Introdução

No âmbito da Unidade Curricular de Arquiteturas Emergentes de Redes, foi-nos proposto numa primeira fase a construção de uma rede *Peer-to-Peer* (P2P) com descarga de conteúdos e na fase seguinte a implementação uma rede de suporte baseada em nomes e tolerante a atrasos.

2 Rede P2P

2.1 Concepção

A abordagem para construção da rede P2P, foi a de tornar uma rede estruturada de forma a simplificar a localização de recursos e diminuir o *flooding*, utilizando uma heurística de proximidade, baseada na identificação de recursos e nós através de *id*'s.

Esta rede tem como funcionalidade a possibilidade de guardar e encontrar ficheiros, assim como transferi-los.

Uma vez que nos encontramos numa rede móvel sem infraestrutura no processo de *bootstrap* recorremos ao uso de *multicast*.

2.2 Especificação dos protocolos

A rede principal foi baseada no protocolo **Kademlia**, em que há uma tabela de *hash* distribuída para rede de computadores *peer-to-peer* descentralizada. As regras definidas para a topologia são que cada nó irá ter um **id** associado, tal como cada recurso e a localização destes está pendente da distância através da métrica *XOR*. A rede tem um parâmetro K que indica o número de vizinhos que um nó deverá ter e um parâmetro X para escolher apenas um determinado número de nós no processo de procura que irá ser descrito posteriormente.

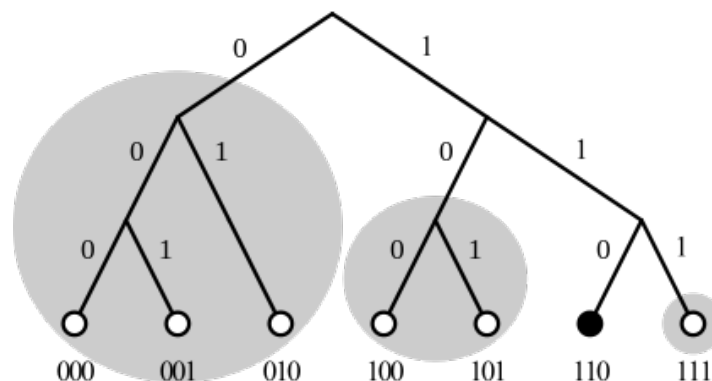


Figura 1: Hash-Table Distribuída P2P

2.2.1 Primitivas de comunicação

As primitivas de comunicação utilizadas na rede P2P são as seguintes:

- **PING** - Verifica se um nó se encontra disponível;
- **STORE** - Indica um nó a guardar um par *key,value*, de acordo com a politica definida pela rede;
- **FIND_NODE** - O recetor da primitiva devolve os vizinhos mais próximos de um determinado **id**;

- **FIND_VALUE** - Retorna os k nós mais próximos ao valor de identificação do recurso, exceto se o nó receptor conter informação acerca do recurso, nesse caso retorna o valor associado a este;
- **BS** - Devolve um nó que responda ao *multicast*.

2.2.2 Interações

Primeiramente um nó ao juntar-se à rede irá passar por um processo de *bootstrap*, a fim de encontrar um determinado número de vizinhos, assim, o nó irá mandar uma mensagem *multicast* de forma a encontrar um primeiro vizinho.

Após a descoberta, irá pedir a esse vizinho outros nós que possa conhecer e irá adicioná-los à sua lista, caso ainda não tenha vizinhos suficientes. No fim, com entradas de outros nós, ele poderá conhecer estes potenciais vizinhos ao receber pedidos deles no processo de *bootstrap*. A cada pedido que receba, só irá atualizar a sua lista de vizinhos se esta não se encontrar completa.

Posteriormente ao processo de *bootstrap* um nó poderá realizar as operações de armazenamento, localização e transferência de um recurso.

Para armazenar um recurso, o nó que o pretenda guardar irá descobrir os nós da rede com **id** mais próximo ao do recurso, através da primitiva **FIND_NODE** aplicada aos seus vizinhos, de uma forma recursiva, isto é, vai sendo chamada a cada lista de vizinhos que receba, selecionando apenas os X mais próximos, sendo cada lista recebida mais próxima do *id* pesquisado, parando as iterações quando receber nós com proximidade pior aos recebidos anteriormente. O nó mais próximo vai armazenar numa tabela com pares *Key-Value*, sendo a *Key* o **id** do recurso e o *Value* o **id** onde se encontra o recurso.

De modo a transferir um recurso basta indicar o **nome do recurso** e o **id** do nó que o tenha.

Para localizar um recurso, a primitiva **FIND_VALUE** irá ser chamada de forma recursiva num processo semelhante ao descrito no processo de armazenamento.

2.2.3 Formato das mensagens protocolares (PDU)

O PDU utilizado na rede P2P é o seguinte:

- *Seq_num* - Número de sequência;
- *Total* - Número total de pacotes;
- *Last* - Indica se é o último pacote;
- *Method* - Indica a primitiva;
- *Payload* - Dados.

3 Rede de Suporte

Na segunda fase, foi nos pedido para a construir uma rede de suporte para ajudar a anterior, com as seguintes funcionalidades específicas, tais como a tolerância a atrasos e baseada em nomes em alternativa à rede que usava endereços IP.

3.1 Conceção

Na comunicação baseada em nomes, foram utilizados *id's*, uma vez que garantem a interoperabilidade com a rede P2P que já os utilizava.

Para obtermos um tolerância a atrasos, utilizamos como encaminhamento o reenvio em árvore, de maneira a reduzir o *flooding*.

3.2 Especificação dos protocolos

3.2.1 Primitivas de comunicação

Além das primitivas utilizadas na rede **P2P**, é utilizada mais um primitiva **CONTACT** para saber quais o nós no raio de alcance.

3.2.2 Formato das mensagens protocolares (PDU)

Foram utilizados dois tipos de PDU, baseados nas redes NDN, sendo um deles um pacote de interesse com os campos:

- *Content-Name*
- *Nounce*

E o pacote de dados:

- *Content-Name*
- *Nounce*
- *Dados*

Nesta rede a procura não é orientada aos dados mas sim ao *id* do *host*, usando-se assim o *content-name* com o seguinte formato:

PRIMITIVA/ID_DESTINO/ID_ORIGEM/NMR_REPLICAS/.

3.2.3 Interações

A rede irá receber primitivas da rede P2P, no entanto no formato protocolar definido anteriormente, com base no *id* destino que se encontra no *content-name* irá descobrir por *multicast* o seu recetor, mas seguindo uma estratégia de reenvio em árvore. Esta estratégia funciona da seguinte maneira: O nó que envia a mensagem escolhe um determinado número de cópias, supondo que são X , que por *multicast* com 1 salto no máximo de distância, envia no máximo a X seus vizinhos, ficando cada vizinho com $(X - 1)/2$ cópias, caso não tenha X vizinhos ao alcance irá guardar as cópias que faltam e esperar por contacto de novos vizinhos para enviar essas cópias. Se algum nó com este pacote não tenha cópias irá esperar por um contacto com o destinatário do pacote para o enviar.

4 Testes e Resultados

Testamos as duas redes com a ajuda do **CORE**. Apresentamos uma disposição de *routers* ligados a uma rede wireless (como apresenta a Figura 2) com um script de mobilidade.

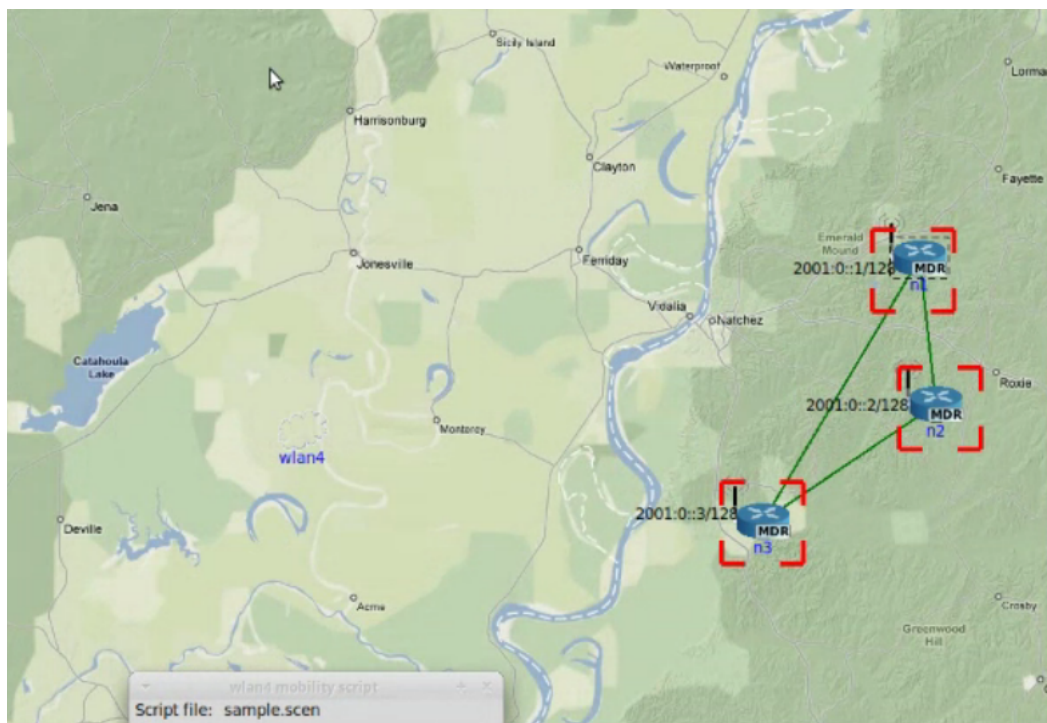
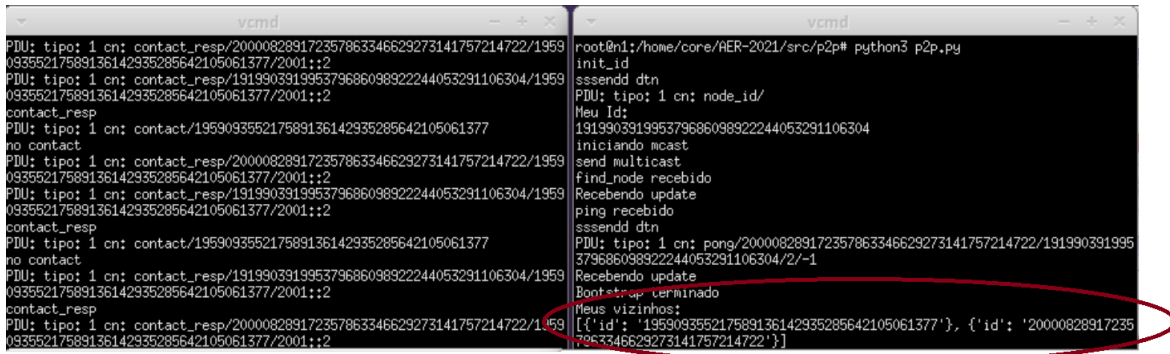


Figura 2: CORE

Primeiramente os *routers* ligam-se à rede de suporte separadamente da *P2P*, nesta ocorre um processo de bootstrap para descoberta de vizinhos, após isto os pacotes P2P são passados para rede de suporte e casos estes sejam dirigidos ao seu id, são processados e enviados de volta à rede *P2P*, caso contrário irão ser enviadas cópias para os *routers* vizinhos que através

de *polling* da primitiva CONTACT vão sabendo que *routers* se encontram no alcance, caso não tenha cópias esperam por um contacto do destinatário do pacote.



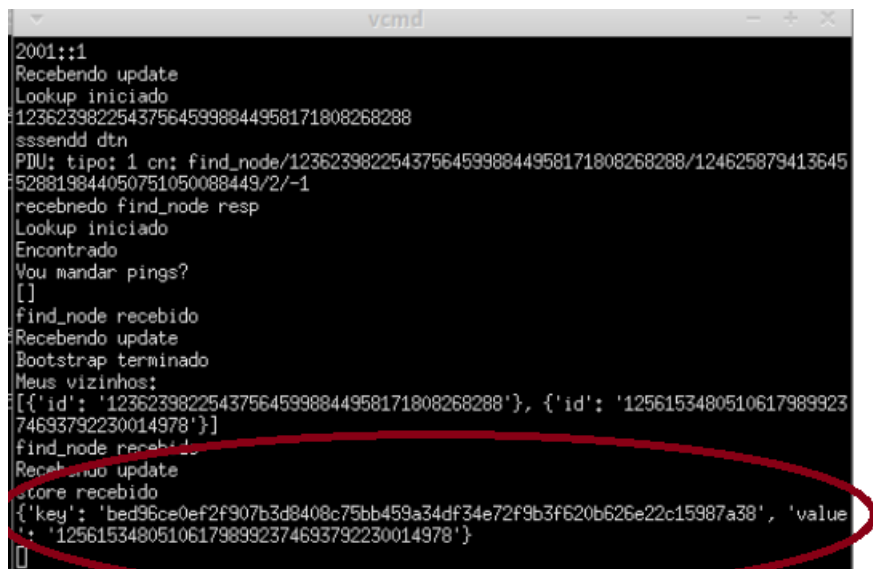
```

vcmd                                     vcmd
PDU: tipo: 1 cn: contact_resp/200008289172357863346629273141757214722/1959
09355217589136142935285642105061377/2001::2
PDU: tipo: 1 cn: contact_resp/191990391995379686098922244053291106304/1959
09355217589136142935285642105061377/2001::2
contact_resp
PDU: tipo: 1 cn: contact/195909355217589136142935285642105061377
no contact
PDU: tipo: 1 cn: contact_resp/200008289172357863346629273141757214722/1959
09355217589136142935285642105061377/2001::2
PDU: tipo: 1 cn: contact_resp/191990391995379686098922244053291106304/1959
09355217589136142935285642105061377/2001::2
contact_resp
PDU: tipo: 1 cn: contact/195909355217589136142935285642105061377
no contact
PDU: tipo: 1 cn: contact_resp/191990391995379686098922244053291106304/1959
09355217589136142935285642105061377/2001::2
contact_resp
PDU: tipo: 1 cn: contact/200008289172357863346629273141757214722/1959
09355217589136142935285642105061377/2001::2

root@r1:~/home/core/HER-2021/src/p2p# python3 p2p.py
init_id
sssendd dtn
PDU: tipo: 1 cn: node_id/
New Id:
191990391995379686098922244053291106304
iniciando mcast
send multicast
find_node recebido
Recebendo update
ping recebido
sssendd dtn
PDU: tipo: 1 cn: pong/200008289172357863346629273141757214722/191990391995
379686098922244053291106304/2/-1
Recebendo update
Bootstrap terminado
Meus vizinhos:
[{'id': '195909355217589136142935285642105061377'}, {'id': '20000828917235
7863346629273141757214722'}]
  
```

Figura 3: Consola da esquerda a correr a rede NDN e o da direita, a rede P2P

Ao fazer o *store* de um produto, ele guarda como key o *id* do recurso e o value o *id* do nodo, como podemos verificar na imagem seguinte:



```

vcmd
2001::1
Recebendo update
Lookup iniciado
123623982254375645998844958171808268288
sssendd dtn
PDU: tipo: 1 cn: find_node/123623982254375645998844958171808268288/124625879413645
528819844050751050088449/2/-1
recebendo find_node resp
Lookup iniciado
Encontrado
Vou mandar pings?
[]
find_node recebido
Recebendo update
Bootstrap terminado
Meus vizinhos:
[{'id': '123623982254375645998844958171808268288'}, {'id': '1256153480510617989923
74693792230014978'}]
find_node recebido
Recebendo update
store recebido
{'key': 'bed96ce0ef2f907b3d8408c75bb459a34df34e72f9b3f620b626e22c15987a38', 'value':
'125615348051061798992374693792230014978'}
  
```

Figura 4: Store feito no nodo

Quando queremos transferir de um nodo para o outro, usamos o comando **find <nome do ficheiro>**. Se o nodo que contem o recurso estiver na rede, a transferência ocorre com sucesso


```
vcmd
Find value iniciado
sssendd dtn
PDU: tipo: 1 cn: find_value/124625879413645528819844050751050088449/123623
982254375645998844958171808268288/2/bed96ce0ef2f907b3d8408c75bb459a34df34e
72f9b3f620b626e22c15987a38/-1
find_value_resp recebido
{'key': True, 'value': '125615348051061798992374693792230014978'}
Find value iniciado
Encontrado valor em:
Resultado find value
125615348051061798992374693792230014978
sssendd dtn
PDU: tipo: 1 cn: download/125615348051061798992374693792230014978/12362398
2254375645998844958171808268288/2/t.md/-1
A fazer um transfer!
1
A fazer um transfer!
2
Transferencia terminada!
```

Figura 5: Transferência feita

Por outro lado, se o nodo que tiver o recurso que está a ser procurado mas não estiver na rede, aparece a mensagem '*UNREACHABLE*' na linha de comandos como podemos verificar na Figura 6

```
vcmd
no contact
PDU: tipo: 1 cn: find_value/124625879413645528819844050751050088449/123623
982254375645998844958171808268288/0/bed96ce0ef2f907b3d8408c75bb459a34df34e
72f9b3f620b626e22c15987a38/228884277606109716388876734842352041984
no find value
bed96ce0ef2f907b3d8408c75bb459a34df34e72f9b3f620b626e22c15987a38
enviando para app
2001::2
115
PDU: tipo: 0 cn: find_value_resp/123623982254375645998844958171808268288/1
24625879413645528819844050751050088449/3/-1
Pacote de dados tipo 0
adicionando pacote à cache
Valor da cache: 3
Com copias
Pacote não destinado tipo 1
no send copias
Mandando copias
para, 123623982254375645998844958171808268288
vou manda do send
Mandando copias
para, 125615348051061798992374693792.30014978
vou manda do send
UNREACHABLE!!!
```

Figura 6: Nodo inalcançável

5 Conclusão e Trabalho futuro

Com este trabalho podemos perceber melhor o funcionamento de redes **P2P**, **DTN** e **NDN**, especialmente as diferenças que existem entre este tipo de redes *Adhoc*. De manifestar, a necessidade de armazenar o pacote quer de interesse, quer de dados na rede, para garantir tolerância a atrasos. Utilizamos o encaminhamento em árvores de modo a reduzir o *flooding*, que é o principal problema do encaminhamento epidémico.

Pensamos que atingimos os objetivos propostos e num trabalho futuro, podíamos tornar esta rede mais eficiente, evitando ainda mais os pacotes desnecessários a circular na rede e também na rede P2P quando um nó receber uma comunicação, se a sua lista de vizinhos estiver completa, este optar por verificar se já perdeu conexão com alguns dos vizinhos e substituir pelo *id* do nó do qual recebeu uma comunicação mais recente.

Referências

- [1] JONES, E. e WARD, P. (2006). Routing strategies for delay-tolerant networks.
- [2] Named-Data-Network. Acedido em Maio, em: <https://named-data.net/>.
- [3] Kademlia. Acedido em Março, em: <https://pt.wikipedia.org/wiki/Kademlia>.