RC-I 2025-2024	REDES DE COMPUTADORES I	Projecto #:	1
PROGRAMAÇÃO COM SOCKETS		Data:	07/11/2025
Peer-to-Peer Overlay Network		Prazo:	**21/11/2025**
Implementação de Protocolo na Camada de Aplicação		Versão	1.0

1. Introdução

Pretende-se desenvolver uma aplicação *Peer-to-Peer Overlay Network* (**p2pnet**) com a qual um conjunto de *peers* mantem uma rede sobreposta, construída à base de sessões TCP entre pares deles, sobre a qual podem pesquisar conteúdos. A aplicação compreende assim duas componentes fundamentais: a manutenção da rede sobreposta face à adesão e abandono de *peers*; e a pesquisa de conteúdos na rede sobreposta.

Para auxiliar na manutenção da rede sobreposta existe um servidor de peers (UDP) que guarda a lista actualizada de todos os peers que fazem parte da rede. Para cada um deles, o servidor de peers regista o seu endereço IP, a sua porta TCP onde escuta pedidos de estabelecimento de ligações e o seu número de sequência, correspondendo este último à ordem pela qual o peer aderiu à rede. Uma ligação na rede sobreposta corresponde a uma sessão TCP entre dois pares. Para manter a conectividade da rede sobreposta é importante distinguir qual dos dois peers de uma ligação da rede sobreposta iniciou a sessão TCP e qual aceitou a sessão TCP, isto é, qual foi o cliente e qual o servidor. Se o peer A iniciou a sessão TCP e o peer B aceitou-a, dizemos que B é um vizinho externo de A e A um vizinho interno de B. Dizemos também que A e B são vizinhos. Cada peer pode ter um número máximo $\mathbf{N^+}$ de vizinhos externos e um número máximo $\mathbf{N^-}$ de vizinhos internos. Para além destas restrições, um peer A pode tentar estabelecer uma ligação com um peer B apenas se o número de sequência de B for menor do que o número de sequência de A. O *peer* B pode temporariamente aceitar uma sessão TCP adicional, para além das $\mathbf{N^-}$ máximas, que dure apenas o tempo suficiente para informar o peer A de que não está em condições de estabelecer uma ligação entre A e B.

Quando um *peer* quer pertencer à rede sobreposta, ele regista-se no servidor de peers e pede-lhe os registos de todos os outros peers constantes na rede. Com esta informação, o peer tenta estabelecer ligações com um máximo de $\mathbf{N^+}$ outros peers. Nota-se que, a menos que o peer seja o primeiro na rede, é-lhe sempre possível ligar-se a pelo menos um outro peer, garantido assim que a rede permanece conexa.

Quando um peer quer abandonar a rede, ele apaga o seu registo do servidor de peers e remove todas as ligações a que pertence na rede sobreposta, fechando as sessões TCP correspondentes. Nestas circunstâncias, é possível que um outro peer fique sem vizinhos externos, podendo isto desconectar a rede. Para lidar com esta situação, sempre que um peer B detecta que ficou sem vizinhos externos, ele consulta o servidor de peers para tentar estabelecer novas ligações com um máximo de $\mathbf{N^+}$ outros peers. Se todos os peers com números de sequência inferior ao do peer B já tiverem $\mathbf{N^-}$ vizinhos internos, então o peer B não obterá se quer um vizinho externo por este

processo. No entanto, é dada ao peer B uma possibilidade de estabelecer uma ligação com um peer C, sempre de número de sequência inferior, sendo que o peer C remove uma ligação que tenha com um outro peer D, assim mantendo o seu número de vizinhos internos igual a $\mathbf{N^-}$.Esta troca da ligação, *peer-D-peer-C* por *peer-B-peer-C*, só pode ser efectuada se o número de sequência do peer D for superior ao número de sequência do peer B. Os estudantes devem convencer-se que é sempre possível ao peer B encontrar um peer C ligado a um peer D satisfazendo a condição anterior. Devem também convencer-se que este processo realizado iterativamente conduz sempre a uma rede sobreposta conexa.

A rede sobreposta de peers serve para a partilha de conteúdos entre eles. A cada conteúdo está associado um identificador. Para simplificar, neste mini-projecto considera-se apenas a partilha de conhecimento sobre conteúdos: os identificadores são partilhados, mas os conteúdos propriamente ditos não precisam de ser transferidos. Um peer pode criar um identificador, apagar um identificador que seja do seu conhecimento ou pesquisar um identificador na rede sobreposta. Um peer A que queira pesquisar um identificador do qual não tem conhecimento, pergunta a cada um dos seus vizinhos se conhece o identificador. Cada um desses vizinhos pode, recursivamente, remeter a pergunta a cada um dos seus vizinhos. O número máximo de saltos na rede sobreposta que uma pergunta pode percorrer é um parâmetro dado. Se pelo menos um dos vizinhos do peer A lhe acusa conhecimento do identificador, então também o peer A fica com esse conhecimento. Caso contrário, o peer A declara que não encontrou o identificador na rede. Os peers intermédios usados na pesquisa não guardam conhecimento do indentificador.

2. Especificação

Cada grupo de, no máximo, três estudantes deve concretizar a aplicação **p2pnet** que compreende os elementos seguintes.

- Uma interface de utilizador (secção 2.2).
- O cliente para consulta do servidor de peers (secção 2.3).
- O protocolo para estabelecimento e terminação de ligações na rede sobreposta (secção 2.4).
- O protocolo para pesquisa de identificadores na rede (secção 2.5).

2.1. Invocação da p2pnet

A aplicação p2pnet é invocada com o comando

`p2pnet` `[-s addr] [-p prport] [-l lnkport] [-n neigh] [-h hc]`

em que

- `addr` e `prport` são o endereço IP e a porta UDP bem-conhecido do servidor de peers. Por omissão, `addr` é o endereço IP da máquina **192.168.56.21** e `prport` toma o valor **58000**.
- `lnkport` é o porto TCP no qual o *peer* escuta pedidos para estabelecimento de ligações na rede sobreposta.

- `neigh` é o número máximo de vizinhos internos, $\mathbf{N}^-$, permitidos que assumimos igual ao número máximo de vizinhos externos permitidos, $\mathbf{N}^+$. Este valor tem de ser sempre maior do que zero.
- `hc` é o número máximo de saltos permitidos para a pesquisa de um identificador na rede sobreposta. Este valor tem de ser sempre maior do que zero.

Em resultado da invocação, a aplicação disponibiliza um servidor TCP no porto `lnkport` para recepção de pedidos de estabelecimento de ligações na rede sobreposta.

2.2. Interface de Utilizador

A interface de utilizador responde aos comandos seguintes.

- **join**
  O utilizador pede adesão à rede sobreposta. Em consequência deste pedido, a aplicação realiza a seguinte sequência de acções: (i) regista o peer no servidor de peers, recebendo deste o número de sequência de registo na rede; (ii) solicita ao servidor de peers a lista de todos os peers registados na rede, respondendo este com a dita lista; (iii) tenta estabelecer ligações com um máximo de neigh outros peers. No fim desta sequência de procedimentos, a aplicação informa o utilizador se conseguiu ligar-se à rede.
- **leave**
  O utilizador pretende abandonar a rede sobreposta. Em consequência deste pedido, a aplicação retira o seu registo do servidor de peers e fecha todas as ligações que mantinha com os seus vizinhos. No fim, a aplicação informa o utilizador que abandonou a rede.
- **show neighbors**
  Mostra a lista dos vizinhos internos e a lista dos vizinhos externos do peer. Cada peer é identificado pelo seu endereço IP, a sua porta TCP de escuta de pedidos de ligação e o seu número de sequência.
- **release seqnumber**
  O utilizador solicita a remoção da ligação com o vizinho interno de número se sequência seqnumber. Este comando serve para teste da aplicação.
- **list identifiers**
  Mostra a lista dos identificadores dos quais o peer tem conhecimento.
- **post identifier**
  Adiciona identifier à lista dos identificadores dos quais o peer tem conhecimento.
- **search identifier**
  Pesquisa de identifier na rede sobreposta. Para este efeito, se o peer não tiver já conhecimento de identifier, a aplicação interroga à vez cada um dos seus peers vizinhos. Se pelo menos um peer vizinho lhe der conhecimento de identifier, então a aplicação adiciona identifier à lista dos identificadores que conhece. Caso contrário, se nenhum vizinho lhe der conhecimento de identifier, então identifier não é adicionado à lista de identificadores dos quais o peer tem conhecimento. O utilizador é informado do resultado.
- **unpost identifier**

Apaga identifier da lista dos identificadores na posse do peer.

- **exit**
  Executa as mesmas acções do comado leave e fecha a aplicação.

2.3. Protocolo de comunicação com o servidor de peers

O protocolo de comunicação com o servidor de peers compreende os seguintes comandos emanados do cliente.

- **REG *lnkport***
  A aplicação regista o peer no servidor de peers oferecendo-lhe a porta *lnkport* no qual escutará de pedidos de ligação vindos de outros peers. O servidor de peers responde com a mensagem `SQN seqnumber`, em que `seqnumber` é o número de entrada do peer na rede, atribuído sequencialmente pelo servidor de peers. Se houver erro, o servidor responde com `NOK`.
- **UNR *seqnumber***
  A aplicação apaga o registo do peer no servidor de peers. Note que o peer é univocamente conhecido pelo seu `seqnumber`. O servidor de peers responde com `OK` se a operação é bem sucedida e com `NOK` no caso contrário.
- **PEERS**
  A aplicação solicita ao servidor de peers a lista de todos os peers registados na rede. O servidor de peers responde com a mensagem `LST` numa linha sozinha, seguida de uma sequência de linhas, cada linha respeitando a um peer, terminada numa linha vazia. Uma linha respeitante a um peer tem o formato `IPaddress:lnkport#seqnumber`, em que `IPaddress` e `lnkport` são, respectivamente, o endereço **IP** e porta **TCP** do peer e `seqnumber` é o seu número de sequência.

2.4. Protocolo de estabelecimento e terminação de ligação entre peers

O protocolo para estabelecimento e terminação de ligações na rede redes sobreposta compreende os comandos e respostas seguintes.

- **LNK seqnumber**
  O peer cliente TCP pede ao peer servidor TCP para com ele estabelecer uma ligação da rede sobreposta, indicando-lhe também o seu número de sequência. Se o número de vizinhos internos do peer servidor TCP for inferior ao máximo permitido, então a ligação é efectivamente estabelecida, respondendo o peer servidor TCP com a mensagem `CNF`. Caso contrário, se o número de vizinhos internos do peer servidor TCP for igual ao máximo permitido, então a ligação não é estabelecida, fechando o peer servidor TCP a sessão.
- **FRC seqnumber**
  O peer cliente TCP insiste para que o peer servidor TCP estabeleça com ele uma ligação da rede sobreposta, indicando-lhe também o seu número de sequência. Se o número de vizinhos internos do peer servidor TCP for inferior ao máximo permitido, então o procedimento imita o do comando `LNK`. Se for igual, mas o peer servidor TCP tiver um vizinho interno com número de sequência superior a seqnumber, então o peer servidor TCP

estabelece a ligação com o peer cliente TCP, respondendo com a mensagem `CNF`. Simultaneamente, o peer servidor TCP remove uma ligação com um dos seus vizinhos internos que tenha número de sequência superior a `seqnumber`, fechando a sessão TCP correspondente.

- **CNF**
  O peer servidor TCP confirma que a ligação na rede sobreposta iniciada pelo peer cliente TCP com os comandos `LNK` ou `FRC` se mantem.
- **Fecho de uma sessão TCP**
  O fecho de uma sessão TCP corresponde à remoção de uma ligação na rede sobreposta. Se a remoção da ligação deixar o peer com pelo menos um vizinho externo, então nada mais há a fazer. Pelo contrário, se a remoção da ligação deixar o peer sem vizinhos externos, então a aplicação volta a solicitar ao servidor de peers a lista de todos os peers registados na rede, respondendo este em conformidade. De seguida, a aplicação tenta estabelecer ligações com outros peers de entre aqueles que tenham números de sequência inferiores ao seu, se houver pelo menos um nessas circunstâncias, seguindo a restrição ao número máximo de vizinhos externos. Neste processo, a aplicação pode usar o comando **FRC seqnumber** no máximo uma vez. Se a aplicação não conseguir estabelecer nenhuma ligação nova e o peer tiver vizinhos internos, então a aplicação deve avisar o utilizador de que a rede pode estar desconexa.

2.5. Protocolo para pesquisa de identificadores

O protocolo para pesquisa de um identificador na rede compreende os seguintes comandos e respostas.

- **QRY identifier hopcount**
  Um peer A pergunta a um peer B se este conhece identifier. Em caso afirmativo, o peer B responde com a mensagem `FND`, ficando o peer A a partilhar conhecimento de identifier; em caso negativo, o peer B responde com a mensagem `NOTFND`. Se o peer B não estiver na posse de identifier e hopcount for maior do que um, então ele interroga os seus vizinhos com o comando **QRY identifier hopcount** em que `hopcount` vem decrementado relativamente ao valor recebido do peer A.
- **FND identifier**
  Um peer informa outro de que tem conhecimento de identifier.
- **NOTFND identifier**
  Um peer informa outro de que não tem conhecimento de identifier.

3. Desenvolvimento

Cada grupo de estudantes deve adquirir a destreza necessária sobre programação em redes para realizar a aplicação proposta.

3.1. Estratégia

Para o desenvolvimento global do projecto, sugerem-se os passos seguintes:

1. Implemente o servidor de peers.
2. Implemente o cliente do protocolo de comunicação com o servidor de peers alojado na máquina **192.168.56.21**.
3. Implemente o cliente e o servidor do protocolo para estabelecer uma ligação da rede sobreposta.
4. Implemente os comandos **join** e **show neighbors**.
5. Implemente os comandos **release seqnumber**, **leave** e **exit**.
6. Implemente os comados **post**, **unpost**, **list_id** e **search**.

Comente e teste o seu código à medida que o desenvolve. Note que o mini-projecto será compilado e executado pelo corpo docente apenas no ambiente de desenvolvimento disponível para o laboratório, constituído pelos elementos seguintes:

- Vagrant para infraestrutura e ambiente de execução
- Compilador, depurador e bibliotecas usadas configuradas no vagrant.

Baseie a operação do seu programa no seguinte conjunto de chamadas de sistema, particularmente para linguagem C:

- Leitura de informação do utilizador para a aplicação: `fgets();`
- Decomposição de strings em tipos de dados e vice-versa: `sprintf(), sscanf();`
- Gestão de um cliente UDP: `socket(), close();`
- Comunicação UDP: `sendto(), recvfrom();`
- Gestão de um cliente TCP: `socket(), connect(), close();`
- Gestão de um servidor TCP: `socket(), bind(), listen(), accept(), close();`
- Comunicação TCP: `write(), read();`
- Multiplexagem de informação: `select()`.

Quer os clientes quer os servidores devem terminar graciosamente, pelo menos nas seguintes situações de falha:

- mensagens do protocolo erradas vindas da entidade par correspondente;
- sessão TCP do cliente ou do servidor fechada de forma imprevista;
- condições de erro nas chamadas de sistema.

3.2. Depuração

Desenvolver um cliente e um servidor que supostamente interoperam directamente pode representar um desafio inicialmente, pois só pode testá-los efectivamente quando ambos são desenvolvidos em um grau suficiente. Em muitos casos, uma implementação simples de shim ou mock pode tomar o lugar da coisa completa durante a depuração e o teste. Para este projecto, há uma variedade de ferramentas de rede que podem substituir parcialmente o cliente ou o servidor. A ferramenta `telnet`, já introduzida em aula, funciona de forma semelhante ao cliente e pode ser usada durante o desenvolvimento do servidor:

```
$ telnet <host> <port>
```

Outra ferramenta semelhante - `netcat` (`nc` na linha de comando) - também pode actuar como um servidor TCP trivial:

```
$ nc -l < port>
```

Embora essas ferramentas não implementem a funcionalidade completa do p2pnet, podem oferecer endpoints triviais para interagir durante os estágios iniciais de desenvolvimento.

Outro desafio ao desenvolver software em rede é a necessidade de várias máquinas para formar uma rede. Uma maneira de contornar isso é o uso abundante de virtualização, como já fizemos em aula com o vagrant. Embora ainda deva testar seu projecto usando essas ferramentas, este sistema é simples o suficiente para que possa ser testado em um único host, usando a interface de loopback. Simplesmente fazendo com que os clientes se conectem ao localhost, pode executar o servidor e vários clientes usando apenas janelas de terminal diferentes, em oposição a um ambiente virtualizado mais elaborado.

4. Entrega do Projecto

O código a entregar deve ser armazenado em um repositório GitHub privado partilhado entre os membros do grupo e o docente (username GitHub `joaojdacosta`). No repositório deve conter o código fonte da p2pnet e a respectiva makefile bem como o Vagrantfile e scripts da infraestrutura e ambiente de execução.

A entrega do trabalho é feita pela plataforma Google Classroom ao anexar o relatório no formato PDF dentro do prazo. O modelo de relatório será disponibilizado em breve.

Bom trabalho!