Relatório do segundo Trabalho Prático (Avaliação de Desempenho 2019/1) - Questão 12

Alunos: Renato Lousan e Oscar Othon

O nosso trabalho consistiu em implementar os egoísmos individual e social para uma rede DTN (Delay-tolerant network) no simulador The One

Estratégias de implementação

Egoísmo individual

Para implementarmos o egoísmo individual, tivemos que mudar as classes `DTNHost.java` e `SimScenario.java`. Na classe `SimScenario.java`, adicionamos um novo parâmetro de entrada no arquivo de configuração, esse parâmetro é chamado de `porcentagem`, onde o mesmo pode ser usado para definir a porcentagem de nós individualmente egoístas na rede

```
public static final String EGO_PERCENTAGE = "porcentagem";
 // linha 73 do SimScenario.java
```

Além disso, na `SimScenario.java` mudamos o método `createHosts()` que define a criação dos hospedeiros da rede através da criação de um `ArrayList` com tipo da classe `DTNHost.java` (que mudaremos mais a frente). Logo, a única coisa que nos restou a fazer nesse método foi a adição de um parâmetro no construtor dos hospedeiros e a separação da adição de hospedeiros egoístas e não-egoístas.

```
double nrofIndEgos = s.getDouble(EGO_PERCENTAGE); // Obten
ção do valor da porcentagem de nós egoístas no arquivo de
configuração (linha 333 de SimScenario.java)
```

```
// PARTE MUDADA DO CÓDIGO original (final do método create
Hosts() em SimScenario.java)
// creates hosts of ith group
        for (int j=0; j<nrofHosts; j++) {
            ModuleCommunicationBus comBus = new ModuleComm
unicationBus();
            double qtdEgoistas = Math.floor(nrofIndEgos *
(double) nrofHosts);

            if(j <= qtdEgoistas){
                DTNHost host = new DTNHost(this.messageLis
teners,
                this.movementListeners, gid, interfaces, c
omBus,
```

```
                mmProto, mRouterProto, true); // true sign
ifica que o hospedeiro é egoísta
                hosts.add(host);
                } else {
                    DTNHost host = new DTNHost(this.messag
eListeners,
                        this.movementListeners, gid, inter
faces, comBus,
                        mmProto, mRouterProto, false);
                hosts.add(host);
                }

            }
            }
```

Na classe `DTNHost.java`, adicionamos um novo atributo do tipo `Boolean`, que especifica se o nó é um hospedeiro egoísta ou não, além disso adicionamos esse atributo ao construtor da classe.

```
public DTNHost(List<MessageListener> msgLs,
            List<MovementListener> movLs,
            String groupId, List<NetworkInterface> interf,
            ModuleCommunicationBus comBus,
            MovementModel mmProto, MessageRouter mRouterPr
oto, Boolean isEgoista) {
        this.comBus = comBus;
        this.location = new Coord(0,0);
```

```
        this.address = getNextAddress();
        this.name = groupId+address;
        this.net = new ArrayList<NetworkInterface>();
        this.isEgoistaBoolean = isEgoista;

        for (NetworkInterface i : interf) {
            NetworkInterface ni = i.replicate();
            ni.setHost(this);
            net.add(ni);
        }
```

Até aqui, fizemos apenas sinalizações para saber quais nós são egoístas. Agora, para implementar o egoísmo em si, tivemos que criar um novo *return value* na classe `MessageRouter.java` para sabermos que a conexão foi rejeitada porque o nó é egoísta.

```
/** Receive return value for node is egoist */
    public static final int DENIED_EGOIST = -100; // (linh
as 86 e 87 da MessageRouter.java )
```

Agora, na classe `ActiveRouter.java` (extensão da classe `MessageRouter.java`), modificamos o método `checkReceiving()` que checa se o nó quer ou não começar a receber a mensagem. No caso, adicionamos uma condição na checagem: se o nó for egoísta, ele recebe a mensagem, senão, ele não recebe a mensagem

```
    protected int checkReceiving(Message m, DTNHost from)
```

```
{
        if (isTransferring()) {
            return TRY_LATER_BUSY; // only one connection
at a time
        }

        if ( hasMessage(m.getId()) || isDeliveredMessage(m
) ||
                super.isBlacklistedMessage(m.getId())) {
            return DENIED_OLD; // already seen this messag
e -> reject it
        }

        if (m.getTtl() <= 0 && m.getTo() != getHost()) {
            /* TTL has expired and this host is not the fi
nal recipient */
            return DENIED_TTL;
        }

        if (energy != null && energy.getEnergy() <= 0) {
            return MessageRouter.DENIED_LOW_RESOURCES;
        }

        if (!policy.acceptReceiving(from, getHost(), m)) {
            return MessageRouter.DENIED_POLICY;
        }

        /* remove oldest messages but not the ones being s
```

```
ent */
        if (!makeRoomForMessage(m.getSize())) {
            return DENIED_NO_SPACE; // couldn't fit into b
uffer -> reject
        }
        if (getHost().isEgoistaBoolean == true) { //condiç
ão adicionada
            return DENIED_EGOIST;
        }

        return RCV_OK;
    }
```

Egoísmo social