

Universidade Federal do Amazonas – UFAM
Instituto de Computação – ICOMP

Felipe Matheus e Victor Roque

Trabalho de Simulação

Introdução

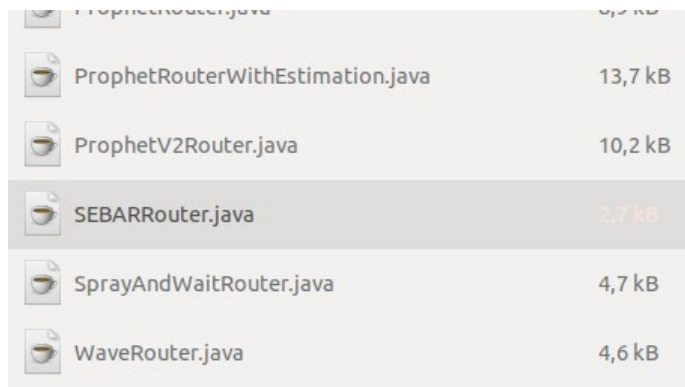
Este relatório expõe as etapas do processo de implementação do algoritmo de roteamento SEBAR no ambiente simulado ONE (Opportunistic Network Environment).

O SEBAR [1] é um algoritmo de roteamento para redes de sensores sem fio que usa o conceito de feromônio, muito comum em algoritmos da classe ACO (Ant Colony Optimization). O SEBAR aplica essa estratégia para encontrar a melhor rota entre dois nós de uma rede oportunista.

Implementação


As etapas abaixo relatam o processo de implementação do algoritmo SEBAR. Todas as alterações a seguir foram incorporadas à branch *‘developer’* do repositório. Também é possível acompanhar estas etapas através do histórico de commits.

1. **Criação da nova classe `SEBARRouter.java` dentro do módulo *‘routing’*.** Essa classe guarda os métodos e atributos que implementam o SEBAR.

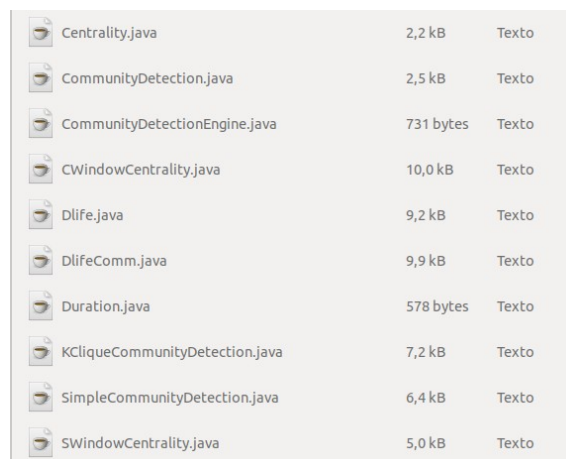


ProphetRouterWithEstimation.java	13,7 kB
ProphetV2Router.java	10,2 kB
SEBARRouter.java	2,7 kB
SprayAndWaitRouter.java	4,7 kB
WaveRouter.java	4,6 kB

2. **Criação do submódulo *‘community’* dentro de *‘routing’*.** Esse submódulo contém classes que auxiliam na detecção de comunidades e no cálculo da centralidade usados no SEBAR. Todas as classes desse novo módulo foram retiradas de [2].



community	10 itens	Pasta
maxprop	2 itens	Pasta
schedule	3 itens	Pasta
util	3 itens	Pasta
ActiveRouter.java	19,7 kB	Texto
DirectDeliveryRouter.java	914 bytes	Texto



Centrality.java	2,2 kB	Texto
CommunityDetection.java	2,5 kB	Texto
CommunityDetectionEngine.java	731 bytes	Texto
CWindowCentrality.java	10,0 kB	Texto
Dlife.java	9,2 kB	Texto
DlifeComm.java	9,9 kB	Texto
Duration.java	578 bytes	Texto
KCliqueCommunityDetection.java	7,2 kB	Texto
SimpleCommunityDetection.java	6,4 kB	Texto
SWindowCentrality.java	5,0 kB	Texto

3. **Implementação das funções de cálculo do SEBAR como métodos da classe SEBARRouter.** Os métodos `calculoEk()` e `calculoEck()` implementam as formulas descritas em [3]. Novas constantes `ENERGIA_CONSUMIDA` e `PORC_ENERGIA` também foram incorporadas à classe.

4. **Inclusão dos atributos ‘comunidade’ e ‘centralidade’ na classe SEBARRouter.** Estes atributos registram a formação de comunidades e valores de centralidade que serão consultados pelo algoritmo de roteamento.

5. **Inclusão dos parâmetros ‘Group.K’ e ‘Group.familiarThreshold’ no arquivo ‘default_settings.txt’.** Estes parâmetros são requisitados pela classe `KcliqueCommunityDetection`.

```
# Kclique
Group.K = 2
Group.familiarThreshold = 2
```

6. **Sobrecarga do método `update` dentro da classe SEBARRouter para implementar o fluxo principal do algoritmo SEBAR.**

```
for (Connection c : getConnections()) {
    for (Message m : getMessageCollection()) {
        if (m.getTo() == c.getOtherNode(getHost())) {
            startTransfer(m, c);
            this.deleteMessage(c.getMessage().getId(), false);
        } else if (m.getHops().contains(m.getTo())) {
            continue;
        } else if (comunidade.isHostInCommunity(m.getTo()) &&
            calculoEk(getHost()) < calculoEk(m.getTo())) {
            startTransfer(m, c);
        } else if (comunidade.isHostInCommunity(getHost()) &&
            comunidade.isHostInCommunity(m.getTo()) &&
            calculoEck(getHost()) < calculoEck(m.getTo())) {
            startTransfer(m, c);
        }
    }
}
```

O trecho de código acima está implementado dentro do método `update`. Nele iteramos por todas as conexões atuais e, para cada conexão, percorremos o buffer de mensagens para fazer as verificações descritas em [3].

7. **Sobrecarga do método `messageTransferred` dentro da classe SEBARRouter.**

```
public Message messageTransferred(String id, DTNHost from) {
    Message m = super.messageTransferred(id, from);
    comunidade.newConnection(getHost(), m.getTo(), this.comunidade);
    return m;
}
```

A intenção é forçar que atributo `comunidade` registre uma nova conexão a cada transferência de mensagem.

```
comunidade.newConnection(getHost(), m.getTo(), this.comunidade);
```

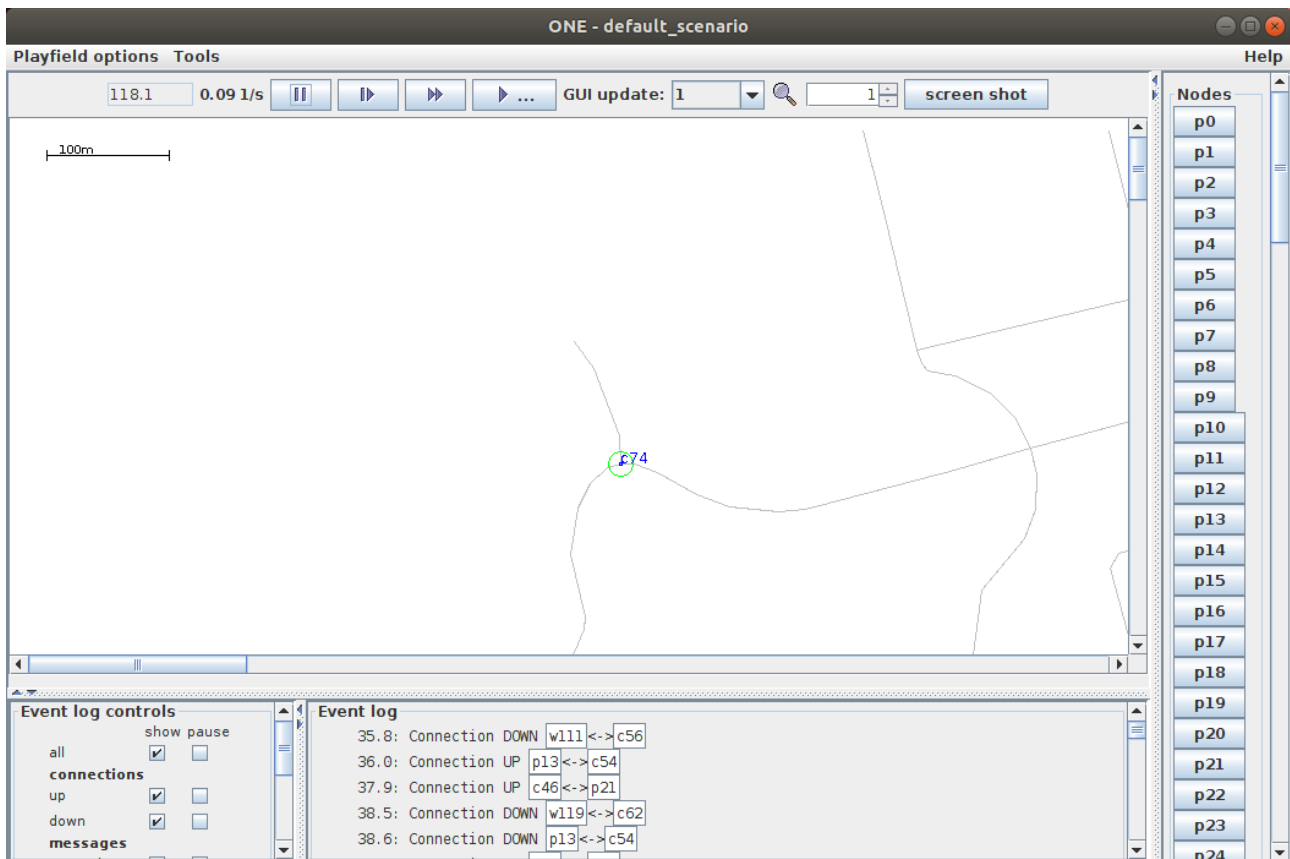
Esse registro é importante para que tenhamos conhecimento das comunidades que se formam durante as conexões.

8. Alteração do parametro 'Group.router' no arquivo 'default_settings.txt' para executar o novo algoritmo de roteamento no ONE.

```
Group.router = SEBARRouter|
```

Execução

Compile o código do ONE através do comando `./compile.sh`. Em seguida, execute-o através do comando `./one.sh`. Para rodar a simulação utilize o painel de controle na parte superior da janela.



Referências

1. <https://pdfs.semanticscholar.org/ad21/1ba42dc8dbd139ba75c01d7c1482b0c57cf2.pdf>
2. https://www.netlab.tkk.fi/tutkimus/dtn/theone/contrib/dLife_v1.0.zip
3. <https://github.com/diogosm/AP1-avaliacao-de-desempenho/blob/master/materialApoio/enunciados/Quest%C3%A3o%2016.pdf>