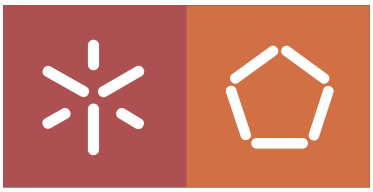

Universidade do Minho
Escola de Engenharia

Redes Móveis

Trabalho Prático
MiEI - 4º Ano - 1º Semestre

A85308 Filipe Miguel Teixeira Freitas Guimarães
A85242 Maria Miguel Albuquerque Regueiras
A86271 Renata Gomes Dias Ribeiro

Braga,
17 de janeiro de 2021

Conteúdo

Lista de Figuras

1 Introdução

Neste relatório encontra-se uma descrição detalhada não só dos resultados obtidos como também do processo em si da realização do projeto da cadeira de Redes Móveis.

Como proposto pela equipa docente, este projeto tem como tema central a criação de uma **rede oportunista** com base na cidade de Braga. Entende-se por rede oportunista uma rede que toma partido de nodos independentes capazes de transmitir informação entre si temporariamente. A comunicação *wireless* entre si é de curto alcance sendo que estas redes tentam facilitar problemas ,como por exemplo, *bottleneck*. No caso em estudo, estes nodos, para além destas características, apresentam **mobilidade** tornando esta rede mais dinâmica.

Primeiramente apresenta-se os requisitos do enunciado de uma forma sucinta para melhor compreensão dos objetivos do projeto. É também mencionado como se selecionou e processou o mapa da cidade de Braga. De seguida, refere-se a abordagem tomada pelo grupo e como se realizaram as simulações no software recomendado. Por fim, são explicadas as configurações finais escolhidas de acordo com os testes realizados e análise das métricas em causa assim como o que retiramos da resolução deste projeto.

2 Requisitos

Como forma de melhor contextualizar o problema, resume-se de seguida os objetivos, restrições e pontos principais do enunciado.

Foi proposto implementar com base na cidade de Braga uma rede oportunista onde existem 100 sensores que recolhem informação sobre os níveis de poluição da cidade. Estes sensores podem gerar 10kB de informação por minuto, informação esta que deve ser depositada em 5 portais existentes que se encontram espalhados pela cidade e permanentemente conectados à internet.

Para a informação ser transmitida dos sensores aos portais, é aproveitado o movimento dos carros que circulam nas vias de trânsito:

- Ao aproximar-se de um sensor recolhe o pacote de dados gerado por este;
- Quando se cruza com outros carros pelo seu percurso pode partilhar esses dados;
- Ao aproximar-se de um portal, deposita esse pacote.

Todas estas comunicações são *wireless* e todos os intervenientes tiram partido de interfaces **Bluetooth** apresentando um comportamento volátil, ou por outras palavras, temporário e consequentemente de curto alcance. As interações entre nodos intermediários (carros) dizem-se de *store-carry-and-forward* (guardar, transportar e encaminhar).

Assim, o objetivo principal do projeto passa por estimar um número ótimo de carros que devem existir em circulação para que os pacotes de dados gerados pelos sensores apenas apresentem um atraso de, no máximo, uma hora até chegar a um portal.

3 Resolução

Nesta secção é ser abordado todo o processo de resolução do problema justificando cada decisão de forma clara e detalhada.

3.1 Definição e Processamento do Mapa

Para obter o modelo do mapa da cidade de Braga recorreu-se à ferramenta *JOSM* (Java OpenStreetMap Editor).

Decidiu-se optar por um mapa que cobrisse todo o centro de Braga assim como a sua periferia. Como se pode ver na figura 2 (apenas uma porção do mapa) este está repleto de conteúdo que não interessa para este projeto (prédios, árvores, entre outros).

Figura 1: Mapa original de Braga.

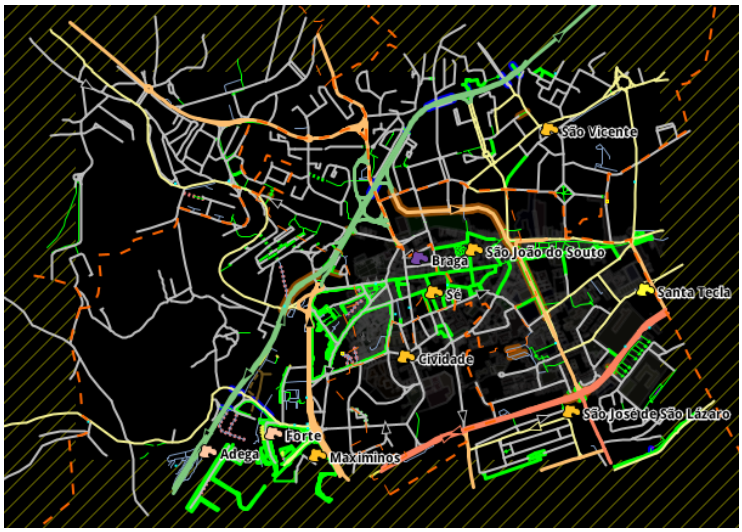

Figura 2: Mapa simplificado de Braga.

Do mapa obtido apenas são necessárias as estradas e equiparáveis para simular a circulação dos carros. Para isto recorreu-se, mais uma vez, ao *JOSM* para apagar as estruturas (e.g. edifícios) resultando no mapa da figura 3.

Para conseguir adicionar o mapa criado ao software de simulação *The ONE* foi preciso convertê-lo para um formato que o simulador conseguisse processar. Para isso recorreu-se a uma ferramenta disponível chamada *OSM2WKT* para transformar o ficheiro .osm em .wkt.

Figura 3: Mapa final carregado no *The one*.

3.2 Realização de Simulações e Abordagem Tomada

Os cenários de simulação no software *The ONE* são construídos a partir de grupos de nodos. Cada grupo é configurado com recursos diferentes e cada nodo possui um conjunto de recursos básicos que podem ser manipulados tais como a interface de rádio, o armazenamento, o tipo de movimento, o protocolo de routing das mensagens, entre outros.

Neste ponto fala-se de como se alteraram estas características de modo a cumprir os objetivos mencionados anteriormente.

3.2.1 Definições da simulação

As configurações da simulação que está a ser desenvolvida estão presentes no ficheiro **default_settings.txt**. Esta simulação contém a duração de 4200 segundos ,*endTime*, e é divido em 10 minutos (600 segundos) para a transmissão de mensagens e uma hora (3600 segundos) para distribuição e entrega das mensagens geradas. Por fim é definido o número de grupos, *nrofHostGroups*, que vão constituir esta simulação (Sensores, Portais e Carros) e que serão abordados no capítulo seguinte. Os restantos elementos já se encontravam pré-definidos.

```
Scenario.name = redes_moveis
Scenario.simulateConnections = true
Scenario.updateInterval = 0.1
Scenario.endTime = 4200
# 3 tipos de grupos: sensores, carros e portais
Scenario.nrofHostGroups = 3
```

3.2.2 Sensores, Portais e Carros

Deste modo, o *The ONE* trata os elementos sensores, portais e carros como grupos. As configurações dos mesmos estão presentes no ficheiro **groups.txt** que será de seguida analisado.

Neste ficheiro definiram-se os três grupos que compõe esta simulação:

- Grupo 1: grupo dos **sensores** com 100 hosts;

```
# Grupo 1 - Sensores #
Group1.nrofHosts = 100
Group1.groupID = S
```

- Grupo 2: grupo dos **portais**, com 5 hosts;

```
# Grupo 2 - Portais #
Group2.nrofHosts = 5
Group2.groupID = P
```

- Grupo 3: grupo dos **carros**. Neste grupo existem algumas características específicas. Primeiro foi necessário definir o tipo de movimento que os nodos deste grupo teriam, neste caso, *CarMovement* para simular o movimento de um carro na vida real. De seguida, definiu-se que os carros apenas podem circular em estradas e um intervalo de velocidades às quais o podem fazer.

```
# Grupo 3 - Carros #
Group3.movementModel = CarMovement
Group3.groupID = C
Group3.okMaps = 1       # carros só circulam em estradas
Group3.speed = 2.7, 20
```

Existem ainda configurações gerais sobre a simulação como um todo como observamos no excerto de código seguinte. Aqui é definido que protocolo de encaminhamento de mensagens é utilizado, os tamanhos dos buffers, o número e o tipo das interfaces dos nodos (Bluetooth), *time to live* das mensagens, entre outros.

```
Group.movementModel = ShortestPathMapBasedMovement
Group.router = EpidemicRouter #Protocolo de encaminhamento de mensagens
Group.bufferSize = 100M
Group.waitTime = 0, 120
# All nodes have the bluetooth interface
Group.nrofInterfaces = 1
Group.interface1 = btInterface
# Walking speeds
Group.speed = 0.0, 0.0
# Message TTL
Group.msgTtl = 300
```

Decidiu-se não posicionar nenhum destes grupos manualmente. Entendeu-se que, mesmo de forma aleatória, ao longo das diferentes simulações, estes continuam sempre na mesma posição não afetando os resultados.

Compreendemos que seria benéfico colocar mais portais perto da zonas assinalada na figura 4, porém, não teríamos os meios para descobrir se, na realidade, seria possível colocar fisicamente este tipo de equipamento. Teria de ser feito um estudo de campo para ver quais dos melhores locais conseguem oferecer as condições necessárias para a colocação.

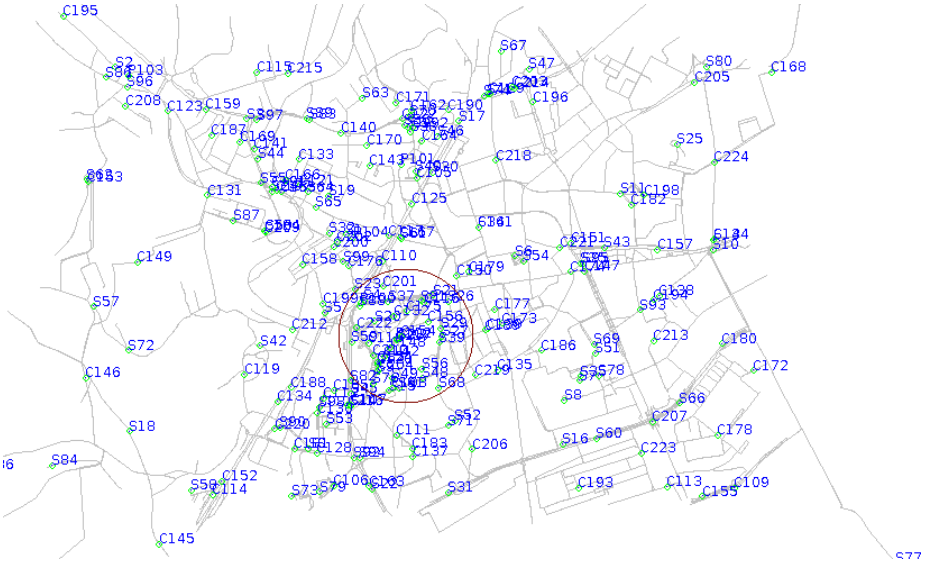

Figura 4: Zona mais movimentada

3.2.3 Eventos e Mensagens

O *The One* trata as mensagens como eventos (*Event*). Primeiro tentamos criar apenas um evento de maneira a que cada sensor gerasse 10 *kbytes* de dados por minuto. Mas verificou-se, desta forma, que o simulador escolhia apenas um sensor aleatório por minuto.

Para contornar esta dificuldade verificada no simulador decidimos fazer um pequeno programa em *Java* para criar as definições necessárias para os eventos.

A essência do programa está representada nas linhas de código em baixo. Escrevemos, iterativamente, para o ficheiro *events.txt* os 100 eventos necessários para serem gerados os 10 *kbytes* por sensor.

```
FileWriter myWriter = new FileWriter("events.txt");
myWriter.write("Events.nrof = 100\n");
myWriter.write("#####################################\n");
int k = 0;
for (int i = 0; i<100; i++){
```

```
    myWriter.write("Events"+(i+1)+".class = MessageEventGenerator\n");
    myWriter.write("Events"+(i+1)+".interval = 60,60\n");
    myWriter.write("Events"+(i+1)+".size = 10k,10k\n");
    myWriter.write("Events"+(i+1)+".hosts = "+i+","+i+"\n");
    myWriter.write("Events"+(i+1)+".tohosts = 100,105\n");
    myWriter.write("Events"+(i+1)+".prefix = M"+k+"\n");
    myWriter.write("Events"+(i+1)+".time = 0, 600\n");
    myWriter.write("###################################\n");
    k++;
}
```

Depois de correr o programa temos como output o ficheiro *events.txt*, em baixo respresentado.

```
Events.nrof = 100
###################################
Events1.class = MessageEventGenerator
Events1.interval = 60,60
Events1.size = 10k,10k
Events1.hosts = 0,0
Events1.tohosts = 100,105
Events1.prefix = M0
Events1.time = 0, 600
###################################
...
###################################
Events100.class = MessageEventGenerator
Events100.interval = 60,60
Events100.size = 10k,10k
Events100.hosts = 99,99
Events100.tohosts = 100,105
Events100.prefix = M99
Events100.time = 0, 600
###################################
```

Nas configurações conseguimos ver que:

- São criados 100 eventos.
- A classe usada é a *MessageEventGenerator*
- Cada evento é gerado de 60 em 60 segundos
- Todos têm o mesmo tamanho, 10kb
- Cada evento tem o host que o vai gerar, bem como para quem vai entregar a mensagem (*tohosts*), que neste caso são os endereços dos portais.
- Todos os eventos têm nomes diferêntes
- Colocamos, também, estes eventos a serem gerados durante apenas 10 min (600 s)

4 Testes e Análise dos Resultados

Por fim, uma vez entendido o funcionamento das ferramentas e como manipular as variáveis de forma a otimizar certos parâmetros, foi preciso definir que métricas se iam avaliar e realizar assim várias simulações teste. No final, recolheram-se os resultados e tiraram-se as respetivas conclusões sobre eles.

As variáveis que foram alteradas ao longo dos testes foram os **protocolos de encaminhamento de mensagens** entre os dispositivos e o **número de carros** em circulação nas vias. Desta forma é possível comparar não só qual o melhor protocolo como também o número ótimo de carros para cada um.

Dentro dos protocolos existentes, existem dois tipos principais: os que replicam mensagens (*replication-based*) e os que não replicam (*forwarding-based*). Ambas são aplicáveis a redes deste género onde é tolerado um certo *delay* na transmissão das mensagens.

Nos protocolos *replication-based* demonstram maiores taxas de entrega de mensagens visto existirem várias cópias na rede onde uma será entregue no destino. Contudo apresentam um maior desaproveitamento de recursos no geral e pode levar a um maior nível de congestionamento da rede em certos sítios.

Por outro lado, os protocolos *forwarding-based* aproveitam melhor os recursos existentes na rede pois existe apenas uma cópia da mensagem em circulação. No entanto apresentam taxas de entrega menores em comparação aos anteriores.

Os protocolos que se escolheram para testar são na sua maioria do tipo *replication-based* à exceção de alguns uma vez que existem vários tipos dentro destas duas categorias incluindo protocolos híbridos que combinam características dos dois:

- **Epidemic Router** e **EpidemicOracleRouter**: baseado em mecanismos de *flooding*, as mensagens de um nodo são sempre replicadas e transferidas se o nodo pelo qual se cruza ainda não possuír uma cópia delas. As mensagens mais antigas são apagadas se a memória não for suficiente. Neste protocolo em específico há uma confirmação por parte do recetor final que permite apagar em todos os nodos a mensagem que já foi entregue.
- **MaxPropRouter**: semelhante ao protocolo Epidemic mas onde é necessária uma ordem de transferência para existir prioridade (uma fila de espera) para que as mensagens que aparentem ter uma maior facilidade em chegar ao destino sejam tratadas primeiro.
- **ProphetRouter** e **ProphetRouterWithEstimation**: o protocolo *PRoPHET* (Probabilistic Routing Protocol using History of Encounters and Transitivity) tem a mesma base que o protocolo Epidemic com a vantagem de tirar partido do facto que os cruzamentos entre nodos não são 100% aleatórios na realidade. Assim, usa um algoritmo que recorre a probabilidades sobre destinos mais prováveis dos nodos em circulação escolhendo replicar as mensagens para nodos com mais probabilidade de as entregar com sucesso.

- **SprayAndWaitRouter**: uma das exceções dentro destes protocolos uma vez que combina *forwarding* com *replication*. Neste limita-se o número de cópias de mensagens e estas são transmitidas até ser atingido um número máximo de cópias em circulação, limitando também o número de comunicações existentes na rede.
- **FirstContactRouter**: neste caso existe apenas uma cópia da mensagem em circulação na rede e os nodos partilham a mensagem com o primeiro nodo que encontram, apagando-a da sua própria fila.

Para cada um destes realizaram-se testes com valores diferentes de número de carros até se obter a maior probabilidade de entrega com sucesso possível. Esta é encontrada nos relatórios produzidos pelo software uma vez corrida a simulação no tempo estipulado.

4.1 Testes e Relatórios

Para realizar os testes foi necessário passar como argumento ao software os seguintes ficheiros de configuração:

```
settings/groups.txt
settings/events.txt
settings/reports.txt
settings/EpidemicOracle_120.txt
```

Os dois primeiros são relativos as configurações anteriormente mencionadas dos grupos e dos eventos. O ficheiro *EpidemicOracle_120.txt* contém as configurações de qual o protocolo de encaminhamento em causa (abordado mais à frente). Por fim, o ficheiro *reports.txt* contém as configurações dos relatórios que serão gerados no final do teste (quantos e quais os relatórios que vão ser gerados):

```
Report.nrofReports = 4
Report.warmup = 0
Report.report1 = MessageStatsReport
Report.report2 = ContactTimesReport
Report.report3 = DeliveredMessagesReport
Report.report4 = MessageDelayReport
```

Os relatórios mais úteis e que foram mais analisados são os seguintes:

- **MessageStatsReport**: indica as estatísticas relacionadas com as mensagens no geral. Neste obtiveram-se os valores finais que se iam analisar especialmente o número de mensagens criadas, o número de mensagens entregues e a probabilidade de entrega.
- **DeliveredMessagesReport**: indica informações sobre o processo de entrega das mensagens nomeadamente o percurso que uma determinada mensagem percorreu e por que nodos passou. Foi utilizado para garantir que percorriam caminhos válidos.

4.2 Resultados

De seguida são apresentados os testes efetuados com os diferentes protocolos de encaminhamento e variando o número de carros presentes na simulação. Para efetuar os testes e a respetiva análise, foram criados diferentes ficheiros de configuração para os diferentes protocolos com a seguinte estrutura:

```
Group.router = (Nome do Protocolo de Encaminhamento)
Group3.nrofHosts = (Número de carros na simulação)
Report.reportDir = (Diretório da pasta onde serão guardadas os reports)
```

4.2.1 EpidemicRouter

O primeiro protocolo que se analisou foi o EpidemicRouter e é com base nos resultados obtidos, fazendo variar o número de carros, que se partiu para a análise dos restantes protocolos. Numa primeira tentativa, utilizou-se um valor simbólico de 50 carros. Rapidamente se tornou evidente que o valor de 50 era demasiado pequeno pelo que, após tentativa-erro, chegou-se a um valor ótimo de 150 carros com um percentagem de entrega de 95.70 %.

```
Message stats for scenario redes_moveis
sim_time: 4200.1000
created: 1000
started: 171161
relayed: 171158
aborted: 0
dropped: 0
removed: 0
delivered: 816
delivery_prob: 0.8160
response_prob: 0.0000
overhead_ratio: 208.7525
latency_avg: 1815.1608
latency_med: 1894.2000
hopcount_avg: 5.2426
hopcount_med: 5
```

4.2.2 EpidemicOracleRouter

Passando para o protocolo EpidemicOracle, um primeiro teste foi efetuado com o valor ótimo obtido no protocolo anterior, 150 carros. Com este valor atingiu-se uma taxa de entrega de mensagens de 100%. No entanto, para tentar obter o menor número possível de carros na simulação, era necessário testar o protocolo com valores abaixo dos 150. Após algumas tentativas, demonstradas no gráfico da figura 5, chegou-se a um valor ótimo de 130 carros, com uma taxa de entrega de mensagens de 100% e com uma média de 7 saltos entre sensor e carros até atingir o portal.

```
Message stats for scenario redes_moveis
sim_time: 4200.1000
created: 1000
```

```
started: 282714
relayed: 282714
aborted: 0
dropped: 0
removed: 93909
delivered: 991
delivery_prob: 0.9910
response_prob: 0.0000
overhead_ratio: 284.2815
latency_avg: 1248.5626
latency_med: 933.0000
hopcount_avg: 7.1968
hopcount_med: 7
buffertime_avg: 768.5054
buffertime_med: 530.3000
```

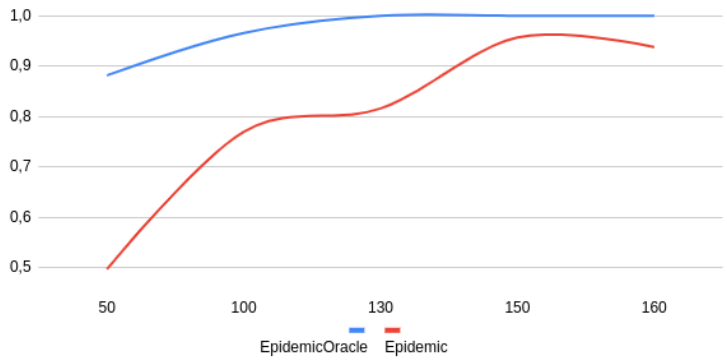

Figura 5: Relação entre os protocolos EpidemicOracle e Epidemic e probabilidade de entrega para diferentes números de carros.

4.2.3 MaxPropRouter

Com este protocolo, a taxa máxima que foi atingida foi de 81%, com 811 mensagens entregues e com 100 carros na simulação. Este protocolo parecia de facto ser promissor, no entanto, não foi possível aumentar o número de carros devido ao peso computacional que estava a ser exigido por parte do software. Assim, este foi o valor máximo obtido com este protocolo.

```
Message stats for scenario redes_moveis
sim_time: 4200.1000
created: 1000
started: 99814
relayed: 99810
aborted: 2
dropped: 0
removed: 48351
delivered: 811
delivery_prob: 0.8110
```

```
response_prob: 0.0000
overhead_ratio: 122.0703
latency_avg: 1794.5982
latency_med: 1897.2000
hopcount_avg: 4.5919
hopcount_med: 4
buffertime_avg: 1085.3019
buffertime_med: 886.3000
```

4.2.4 ProphetRouter e ProphetRouterWithEstimation

Tendo em que conta que com o EpidemicRouter obtiveram-se excelentes resultados com o valor de 130 carros, utilizou-se esse mesmo valor como ponto de partida para a execução deste protocolo. Infelizmente, com 130 carros na simulação, os resultados estavam muito abaixo do esperado com apenas 60% e 81% das mensagens entregues nos protocolos ProphetRouter e ProphetRouterWithEstimation respetivamente. Assim, como se pretendia atingir o número mínimo de carros possível, e para obter melhores resultados seria necessário aumentar o número de carros, não se realizou mais nenhum teste com outro valor mais alto.

```
Message stats for scenario redes_moveis
sim_time: 4200.1000
created: 1000
started: 71333
relayed: 71331
aborted: 0
dropped: 0
removed: 0
delivered: 603
delivery_prob: 0.6030
response_prob: 0.0000
overhead_ratio: 117.2935
latency_avg: 1937.2118
latency_med: 2130.4000
hopcount_avg: 4.1078
hopcount_med: 4
```

```
Message stats for scenario redes_moveis
sim_time: 4200.1000
created: 1000
started: 110889
relayed: 110888
aborted: 0
dropped: 0
removed: 0
delivered: 814
delivery_prob: 0.8140
response_prob: 0.0000
overhead_ratio: 135.2260
latency_avg: 1837.9248
```

```
latency_med: 1868.5000
hopcount_avg: 5.0700
hopcount_med: 5
```

4.2.5 SprayAndWaitRouter

Seguindo a mesma lógica do protocolo anterior, iniciou-se a execução deste protocolo com o 130 carros. No entanto, como em casos anteriores, este número de carros não foi o suficiente para atingir uma taxa de entrega ótima rondando apenas os 72% e com uma média de 4 saltos.

```
Message stats for scenario redes_moveis
sim_time: 4200.1000
created: 1000
started: 67260
relayed: 67260
aborted: 0
dropped: 0
removed: 0
delivered: 729
delivery_prob: 0.7290
response_prob: 0.0000
overhead_ratio: 91.2634
latency_avg: 1803.3864
latency_med: 1816.2000
hopcount_avg: 3.9163
hopcount_med: 4
```

4.2.6 FirstContactRouter

Para o protocolo *FirstContactRouter* voltou-se a executar com o melhor número de carros obtidos até ao momento, 130, e obtiveram-se valores muito abaixo do esperado rondando a ordem dos 0.08%.

```
Message stats for scenario redes_moveis
sim_time: 4200.1000
created: 1000
started: 26661
relayed: 26661
aborted: 0
dropped: 0
removed: 26661
delivered: 86
delivery_prob: 0.0860
response_prob: 0.0000
overhead_ratio: 309.0116
latency_avg: 1467.7012
latency_med: 1205.5000
hopcount_avg: 16.0814
hopcount_med: 13
```

```
buffertime_avg: 128.0402
buffertime_med: 62.5000
```

No gráfico apresentado na figura 6 podemos observar os valores obtidos resumidos para uma fácil visualização destes.

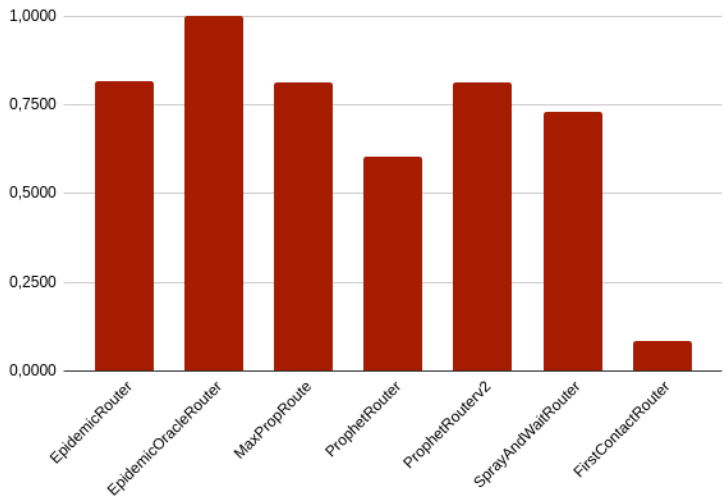

Figura 6: Relação entre protocolos e probabilidade de entrega com 130 carros.

Podemos observar que o **EpidemicOracleRouter** obteve a melhor taxa de entrega de mensagens como mencionado anteriormente , uma taxa de 100%. Para além disso, o **EpidemicRouter**, **MaxPropRouter** e **ProphetRouterWithEstimation** obtiveram valores semelhantes e bastante aceitáveis. O pior protocolo é claramente o **FirstContactRouter** com uma taxa abaixo dos 10%.

5 Conclusão

Neste projeto fomos confrontados com o conceito de redes oportunistas e como aplicá-lo para simular um ambiente onde agentes móveis transmitem mensagens entre si criando uma rede dinâmica.

Para atingir a melhor solução do problema em mãos, foram testados os vários protocolos de encaminhamento de mensagens que se encontravam implementados na ferramenta *The ONE*. Para cada protocolo, foi analisado o número de carros que a simulação necessitava para obter a melhor taxa de sucesso possível.

Como pudemos observar, o protocolo que melhor se adequou ao problema em causa foi o EpidemicOracle. Seguindo uma política de *flooding*, num ambiente pouco congestionado tendo em conta a dimensão do nosso mapa, verificou-se que todas as mensagens geradas foram entregues na totalidade com uma média de 7 saltos entre sensores e portais e com um total de 130 carros.

É importante referir que durante a realização deste projeto foram encontradas algumas dificuldades em compreender o funcionamento da ferramenta *The ONE* especialmente porque não existe grande variedade de documentação disponível, o que dificultou o suave avanço deste projeto.

Por fim, podemos afirmar que a realização deste projeto aprensetou uma perspetiva diferente sobre o tópico de redes oportunistas e como estas podem ser úteis na transmissão de informação utilizando agentes móveis. Futuramente, poderia ser investigada de uma forma mais detalhada como distribuir os portais e sensores de formas mais estratégica e tentar otimizar ainda mais o processo.