Optimal Journeys and Trade-offs on DTNs

Alfredo Goldman
Instituto de Matemática e Estatística
Universidade de São Paulo
São Paulo - SP - Brazil
gold@ime.usp.br

Paulo Henrique Floriano
Instituto de Matemática e Estatística
Universidade de São Paulo
São Paulo - SP - Brazil
floriano@ime.usp.br

César Gamboa Machado
Instituto de Matemática e Estatística
Universidade de São Paulo
São Paulo - SP - Brazil
csrgm@ime.usp.br

ABSTRACT

This article describes algorithms to compute optimal journeys in predictable DTNs. We combine the algorithms to compute the shortest, fastest and foremost journeys in order to provide intermediate optimal journeys which can be used according to the environment characteristics. We also propose a preliminary empirical study on the number of data mules needed to provide connectivity in sparse scenarios on several environments. We present in the article several experiments that show the importance of the proposed approach.

1. INTRODUCTION

Delay Tolerant Networks (DTNs) are dynamic networks, with mobile nodes, transient connections and transmission delays. Despite of this fact, in some cases, it is possible to know the whole topology in advance. Using this information, optimal routings can be provided. We are not particularly interested on the routes provided by these optimal routings, indeed the topology complete knowledge can be unrealistic. However, these optimal values can be used as bounds to provide useful information on how close to the optimal a routing algorithm is.

To model a DTN, a structure capable of representing the transient connections is needed. We use evolving graphs, which were initially proposed in [Ferreira 2002]. In this approach, each edge has a list of activity intervals. In an evolving graph, finding a route between two nodes corresponds to finding a path with non-decreasing and valid time instants.

The size of a journey can be measured by three different parameters: The instant on which the message is delivered, the number of hops and the transit time (the difference between the arrival time and the starting time). The journeys which optimize these parameters are called, respectivelly, **foremost**, **shortest** and **fastest**.

The algorithms to compute these three optimal journeys were proposed in [Xuan et al. 2003]. However, maybe this is not enough. Imagine an example where the foremost journey takes too many hops, on the other side the shortest journey takes too much time. So, there was a need to provide bounds for other situations. The insight for these problems was first presented on [Monteiro et al. 2007].

The simultaneous optimization of independent parameters of a given problem was already well explored on Game Theory. In this context, we search for trade offs where it is not possible to improve a parameter without degrading the other. We will show how to use this techniques to build intermediate journeys.

We also present in this work a preliminary study on the influence of the number of data mules on the connectivity and on the optimal journeys on sparse scenarios. We hope to provide insights on the minimal number of mules to provide reasonable connectivity.

The remaining of the text is organized in the following way, on Section 2 we briefly present the evolving graphs model, we also present the basis used on the next section. On Section 3 we show how to optimize more than one parameter simultaneously. We provide some experiments using the ONE simulator on Section 4.

2. JOURNEYS AND EVOLVING GRAPHS

We formally define an evolving graph in the following way:

Definition 1 (Evolving Graph)**.** *Let $G = (V_G, E_G)$ be a graph and $S_G = G_0, G_1, \ldots, G_\tau$ ($\tau \in \mathbb{N}$) an ordered set of subgraphs of G such that $\bigcup_{i=1}^{\tau} G_i = G$. The system $\mathcal{G} = (G, S_G)$ is denoted evolving graph.*

Let $V_\mathcal{G} = \bigcup V_i$ and $E_\mathcal{G} = \bigcup E_i$. We define $N = |V_\mathcal{G}|$ and $M = |E_\mathcal{G}|$.

Two vertices are adjacent in $\mathcal{G}$ if, and only if, they are adjacent in any G_i. The time spent to send a packet through an edge e of the graph is given by $\zeta(e)$.

Additionally, for each edge e of $E_\mathcal{G}$, we can define its activity period $P(e)$ which corresponds to the time intervals when the edge can be traversed.

We define $\delta_E = max\{|P(e)|, e \in E_\mathcal{G}\}$, the greater among the interval sets on the graph and $\mathcal{M} = \sum_{e \in E_\mathcal{G}} |P(e)|$, the number of intervals on the graph. We assume that all intervals in $P(e)$ are greater than $\zeta(e)$.

In the implementation, and also on the algorithm analysis, it is convenient to define $f(e,t)$, with $e \in E_G$ and $t \in [1, \tau]$, as the function that, given an edge and a time instant, returns the next time instant in which the edge can be traversed ($\min\{t' \mid t' \in P(e) \text{ e } t' \geq t\}$) or ∞, if this edge does not exist. Using binary search on the time intervals it is possible to find the value of $F(e,t)$ in time $O(\log \delta_E)$.

ExtremeCom 2010 Dharamsala, India

Definition 2 (Journey). *Let $R = e_1, e_2, \ldots, e_k$ ($e_i \in E_G$) be a path in G, and $R_\sigma = \sigma_1, \sigma_2, \ldots, \sigma_k$ ($\sigma_i \in [1, \tau]$, $\sigma_i \leq \sigma_j$ $\forall i < j$) a valid schedule which says when each edge of R is traversed. We define $\mathcal{J} = (R, R_\sigma)$ as a journey in $\mathcal{G}$.*

We also define, for a journey $\mathcal{J}$, $arrival(\mathcal{J})$ as the arrival time instant on the destination ($\sigma_{|R|}+\zeta(e_{|R|})$), $departure(\mathcal{J})$ as the time the first edge is used (σ_1) and $transit(\mathcal{J})$ as the time in transit of the Journey ($arrival(\mathcal{J}) - departure(\mathcal{J})$).

For each intermediary node in a journey, we define as waiting time, the time between the arrival at the node and the instant when the next edge is used ($\sigma_i - \sigma_{i-1}$).

2.1 Optimal Journeys

We can define three different kinds of journeys from a node s to a node t in an evolving graph, considering the following parameters: the arrival time, the number of hops and the time in transit.

2.1.1 Foremost journey

We denote as **foremost** journey the one where the arrival time ($arrival(\mathcal{J})$) is minimal. To compute this journeys we consider that even if the prefix of a foremost journey is not always a foremost journey it is possible to find a prefix which respects this property.

We can proceed in a similar way as Dijkstra's algorithm to compute minimum cost paths in usual graphs. We construct the set C of vertices for which the foremost journey was already computed, and the set Q of vertices already visited but whose foremost journey was not yet computed.

At each step we find a vertex $u \in Q$, such that its arrival time is minimum. Then we remove u from Q and add it to C, all the neighbours of u are added to Q and their arrival time is updated. When Q becomes empty all the reachable vertices of V will be in C. This algorithm was proposed in [Xuan et al. 2003] and has complexity $O(M \log \delta_E + N \log N)$.

2.1.2 Shortest journey

We name **shortest** journey the one with the minimal number of edges ($|R|$). To compute the shortest journeys from a node s, we compute at each iteration, the paths with smaller arrival time with k hops from the paths of size $k-1$. Doing that, we obtain a prefix tree where each node can appear more than one time, but with no duplications on the same level of the tree. This algorithm was proposed in [Xuan et al. 2003] and has complexity $O(N(M \log \delta_E + N))$.

2.1.3 Fastest journey

We call **fastest** journey the one that has minimal transit time ($transit(\mathcal{J})$). In this case, the journey's prefix is not relevant for the problem.

To compute these journeys we use the fact that a fastest journey with starting time t is always a foremost journey among the journeys that begin at time t. So, we have to compute all the relevant foremost journeys. This algorithm was also proposed in [Xuan et al. 2003] and has complexity $O(\mathcal{M}(M \log \delta_E + N^2))$.

From these ideas it is possible to consider the Space-Time network $\mathcal{R}$ [Pallottino and Scutella 1998] given by the evolving graph $\mathcal{G}$.

We stress that in this network it is possible to compute different optimal journeys just providing weights for the edges in an appropriate way. However, the complexity is not interesting when we look for trade offs between shortest and foremost journeys. So, we introduce another way to compute trade offs in the next section.

3. HOW TO COMPUTE INTERMEDIARY OPTIMAL JOURNEYS

We now discuss how to implement the Pareto points based on the shortest and the foremost journeys.

We use the algorithm for shortest journeys proposed in [Xuan et al. 2003] as a base to compute Pareto optimal solutions. On the original algorithm, on iteration k, all the journeys with origin s and at most k hops are computed, when a node u is found for the first time a foremost journey between s and u is found. However, for the original algorithm only the ones that arrived earlier were used.

From this step, it is easy to see that, all the intermediary optimal journeys with up to k edges can be found at each iteration. The algorithm follows:

```
OptimalJourneys(G, s)
 1  for v ∈ V_G
 2       do t_LBD[v] ← ∞
 3          J[v] ← ∅
 4          J_optimal[v] ← ∅
 5  t_LBD[s] ← 0
 6  k ← 0
 7  while k < N
 8       do k ← k + 1
 9          (e_min, t_min) ← Edges-Times-Selection(G, t_LBD)
10          for v ∈ V_G such that e_min[v] ≠ nil
11               do Let (u, v) = e_min[v]
12                  Let (R, R_σ) = J[u]
13                  J[v] ← (R ∪ e_min[v], R_σ ∪ t_min[v])
14                  J_optimal[v] ← (J_optimal[v] ∪ J[v])
15                  t_LBD[v] ← t_min[v] + ζ(e_min[v])
16  return J_optimal
```

```
Edges-Times-Selection(G, t_LBD)
 1  for v ∈ V_G
 2       do e_min[v] ← nil
 3          t_min[v] ← ∞
 4          t_arrival[v] ← t_LBD[v]
 5  for (u, v) ∈ E_G
 6       do t ← f((u, v), t_LBD[u])
 7          if t + ζ(u, v) < t_arrival[v]
 8             then e_min[v] ← (u, v)
 9                  t_min[v] ← t
10                  t_arrival[v] ← t + ζ(u, v)
11  return (e_min, t_min)
```

From the previous algorithm we can also compute the optimal journeys with weights on the parameters. We just have to iterate the list of optimal journeys looking for the one with the minimal cost.

3.1 Experiments

We divide the experiments in two, first we study the behavior of the trade off journeys and then we provide experiments on the number of data mules.

3.2 Tradeoff Experiments

We use the simulator *Opportunistic Network Environment* (ONE) [Keränen et al. 2009]. We use the optimal journeys and the optimal intermediary journeys. On the figures we can see the MiddleHop routing which uses the average of the number of hops on the shortest and foremost journeys. Similarly, QuarterHop and 3QuarterHop use the first and third quarter, respectively. On the MiddleRnd, one of the intermediate optimal journeys is randomly chose [1].

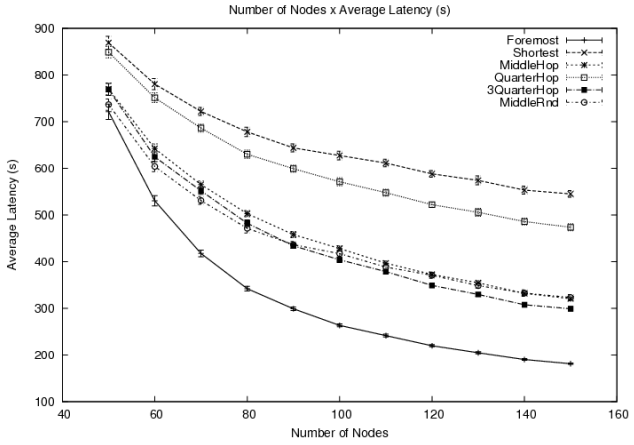

Figure 1: Number of Nodes x Average Latency in seconds

The nodes' transmission radius was fixed in 15m, their speed in between 20 and 100m/s. The packet size was fixed in 5kb, the transmission speed in 250kbps and the nodes' buffer size in 100kb. The transmission speed is much larger than the packet size to ensure that each transmission lasts less than 1s. Since ONE only uses discrete time intervals, each transmission actually lasts exactly 1s in the simulations. Besides that, each simulation lasts 3000s and a new message is created every 10s. The nodes' movement is determined by shortest paths in the map of Helsinki.

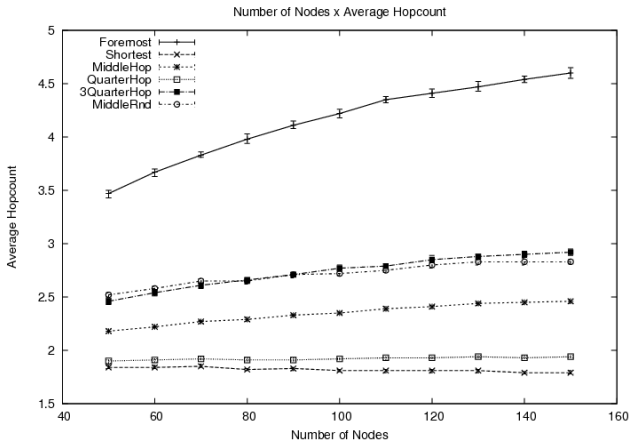

Figure 2: Number of Nodes x Average Hopcount

As we can see, on Figures 1 and 2 the intermediary optimal journeys provide a clear trade off between the shortest and foremost journeys.

[1]More details can be found in `http://www.linux.ime.usp.br/~catita/dtn/`

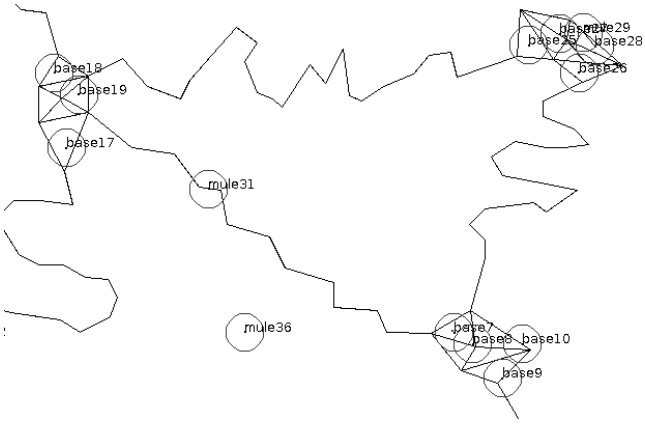

Figure 3: ONE representation of the map used in the scenario described

3.3 Mules experiments

We also conducted some experiments on an isolated environment with several fixed nodes and we varied the number of data mules. The representation of the scenario can be seen in Figure 3. In this scenario, there are 29 fixed nodes scattered in a pattern that resembles villages connected by roads in a remote place. We ran experiments in which the mules only move through the roads (simulating cars or buses), some experiments where the mules may move anywhere, with the Random Waypoint model, (simulating helicopters) and mixed scenarios.

In Figure 4 we can see the influence of the number of mules on the message delivery probability when the mules move through the roads only. As expected, the delivery probability grows with the number of mules, but when we have over 10 mules, the growth rate falls drastically and the value barely exceeds 85% after that. These results show that we have little almost no gain if we add even more mules. The different routing strategies wield similar results in this scenario.

So, interestingly, the routing algorithms somehow followed the behavior of the bounds.

In Figure 5 we see a scenario in which the mules move randomly in the map, the delivery probability is significantly smaller.

Additionally, the probability grows steadily as the number of mules increase, even when 20 mules are reached. This suggests that if more mules are added, the probability may grow further. Also, the routing protocols with knowledge of the network seem to do better in this scenario than the traditional algorithms, which is expected.

Comparing the previous figures we can notice that when the mules have to follow the roads, the delivery probability increases rapidly, even with few mules (Figure 4), however after a certain point, the additional mules do not provide further connectivity. On the other side, when the mules do not follow any specific route, we have a slow increase on the delivery probability when more mules are added and this tendency continues beyond 20 mules.

In the next experiment we try a mixed approach, with two kinds of mules.

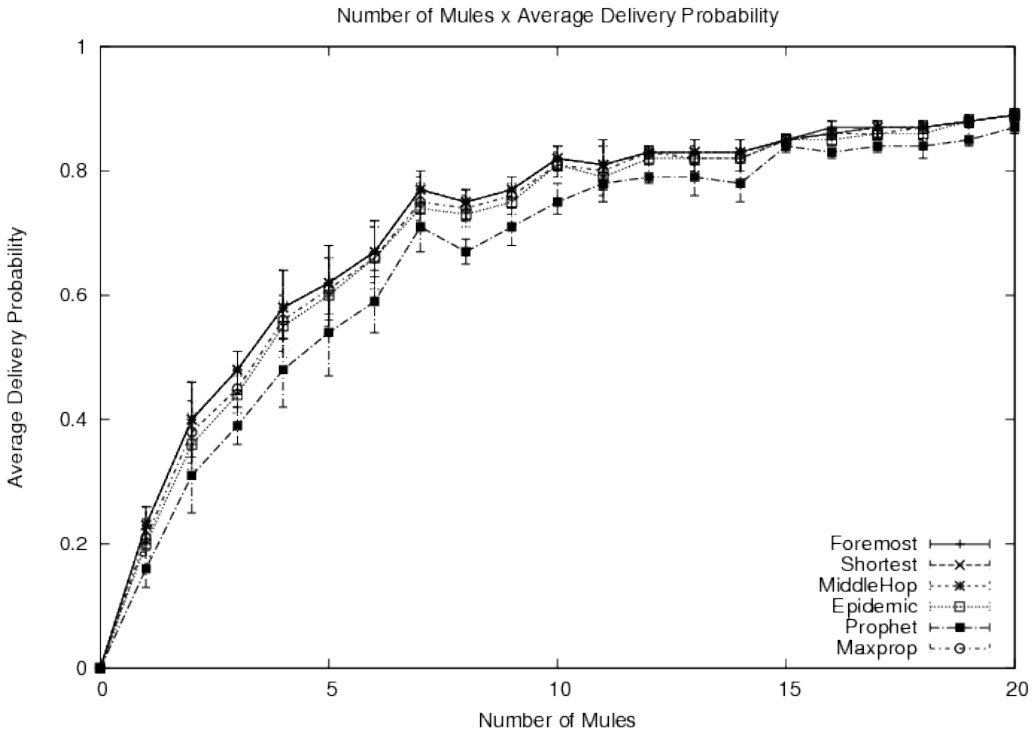

Figure 4: Influence of number of mules on the delivery probability.

Figure 6 shows another experiment in which 2 mules move through the roads, 2 move in RWP and the transmit radius varies. We can see that even with a small transmit radius, the delivery probability is above 40%. This value is equivalent to having over 8 RWP mules or over 3 map mules. So, the mixed scenario can achieve better results than only one kind of mule.

On the Figure 7 we can see the influence of the number of mules on the number of hops found for different routing algorithms in a scenario where the mules may only move on the roads of the map. In the Figure we also show the classical routing algorithms such as epidemic, Maxprop and Prophet.

For the shortest and MiddleHop journeys the average number of hops is always close to 2, even when there are only few mules. For the other algorithms the number of hop counts grows proportionally to the number of mules. However, on the experiment with latency (Figure **??**) we notice that with more data mules, the Epidemic, Maxprop and foremost routings obtained better performance. We can also notice that the latency decreases as the number of mules grow. With zero mules, no messages arrive at their destination, so the average latency is shown as zero.

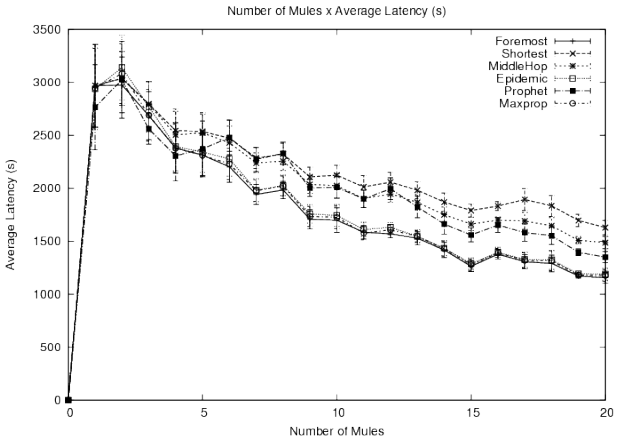

Figure 8: Influence of number of mules on the average latency.

4. ACKNOWLEDGEMENT

The authors would like to thank CNPq (*Conselho Nacional de Desenvolvimento Científico e Tecnológico*) and FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo) for the financial support.

5. CONCLUSION

In this paper we could see the practical issues involving the implementation of trade off optimal journeys. We presented an easy way to implement trade offs between the shortest and the foremost journeys.

The intermediary journeys can be used in situations where both the shortest and foremost journeys do not provide satisfactory results, on the journey length or in the journey duration.

We also presented some initial results on the influence of the number of mules on the routing algorithms. We presented results evidencing the diference between using mules

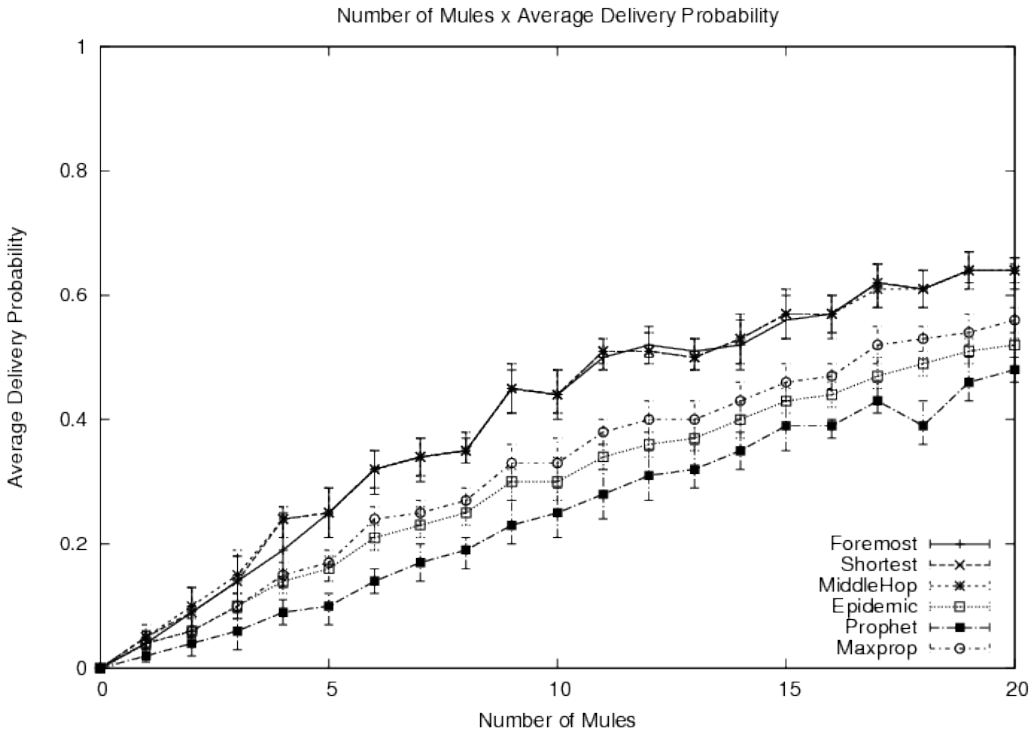

Figure 5: Influence of number of mules on the delivery probability.

restricted to the paths or mules free to move using the RWP model. An experiment with a combination of both kind of mules were also presented. We also studied the impact of the number of mules on the average hop count and on the latency.

With this work we provided more tools to analyse the quality of routing protocols and some insights on the number of data mules needed. As future research, we intend to better understand the influence of the number and type of the mules on different scenarios. We also intend to change the mules speed and storage capacity.

6. REFERENCES

[Ferreira 2002] Ferreira, A. (2002). On models and algorithms for dynamic communication networks: the case for evolving graphs. In *In Proc. ALGOTEL.*

[Keränen et al. 2009] Keränen, A., Ott, J., and Kärkkäinen, T. (2009). The ONE Simulator for DTN Protocol Evaluation. In *SIMUTools '09: Proceeding of the 2nd International Conference on Simulation Tools and Techniques*, New York, NY, USA. ICST.

[Monteiro et al. 2007] Monteiro, J., Goldman, A., and Ferreira, A. (2007). Using Evolving Graphs Foremost Journey to Evaluate Ad-Hoc Routing Protocols. In *In Procedings of 25th Brazilian Symposium on Computer Networks (SBRC'07), Belem, Brazil.*

[Pallottino and Scutella 1998] Pallottino, S. and Scutella, M. (1998). Shortest path algorithms in transportation models: classical and innovative aspects. *Equilibrium and advanced transportation modelling*, pages 245–281.

[Xuan et al. 2003] Xuan, B., Ferreira, A., and Jarry, A. (2003). Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14:267–285.

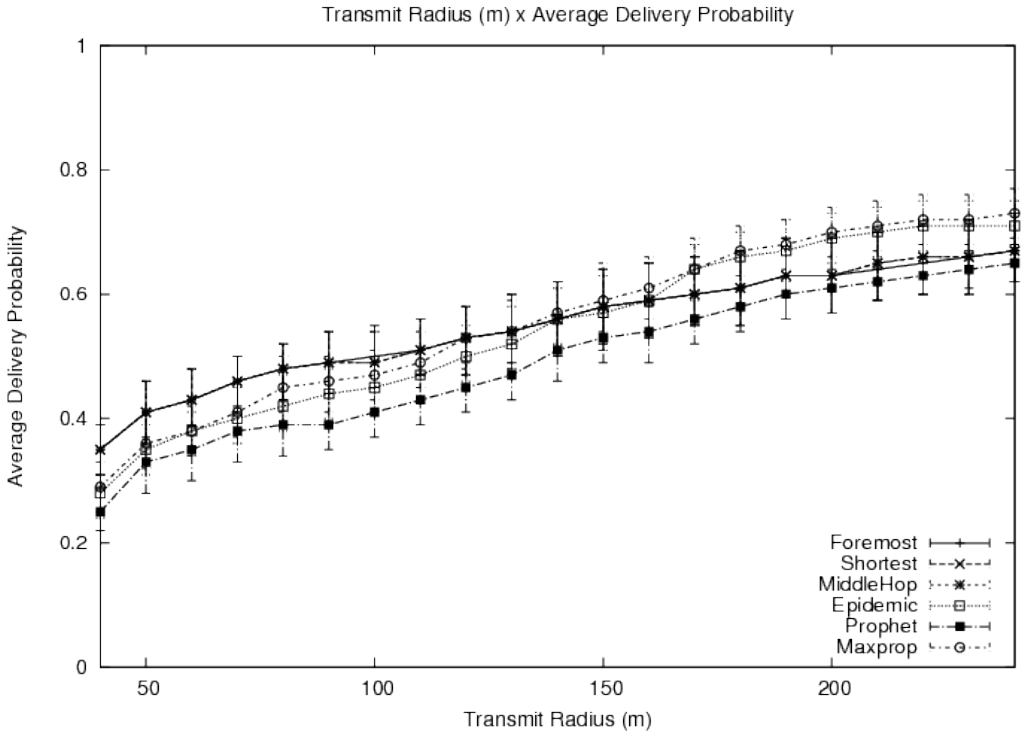

Figure 6: Mixed scenario with 2 RWP mules and 2 map mules.

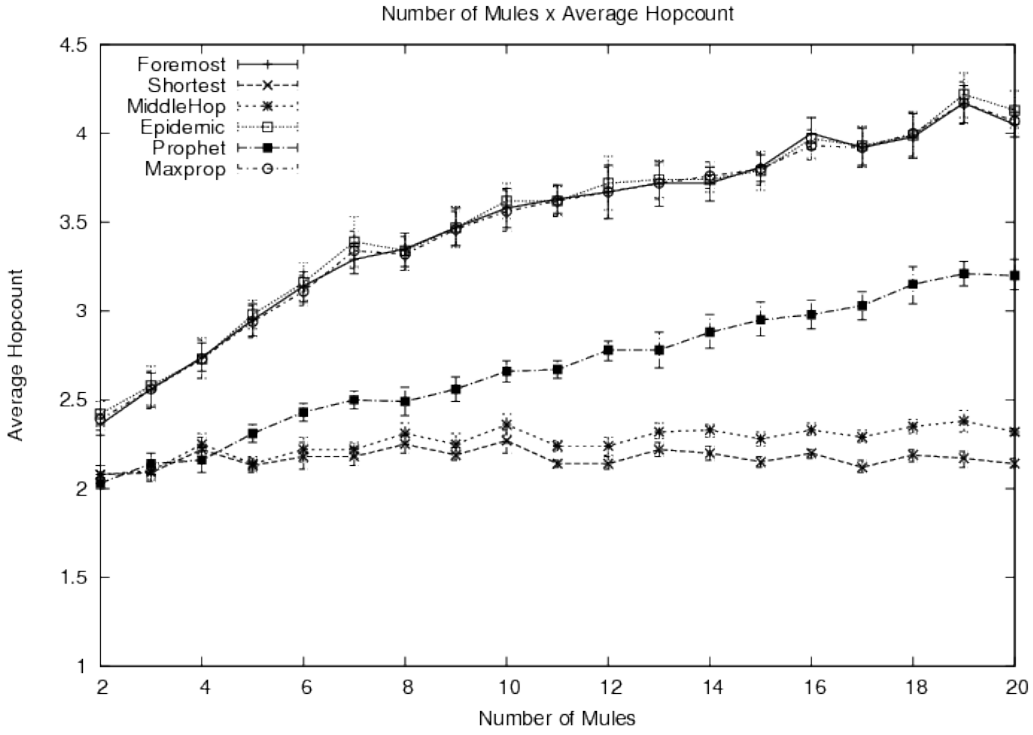

Figure 7: Influence of number of mules on the number of hops.