

Lecture 20: Tuesday, 3rd April, 2007

Presented by: Aditya Dhananjay  
Instructor: Soma Chaudhuri

Scribe: Aditya Dhananjay

## 1 Introduction

This lecture covers Location-Based Routing. The presentation was given by Aditya Dhananjay.

Note: The figures shown in these scribe notes are color coded. Therefore, please take a color print out if you prefer a hard copy of these notes.

In computer networks, the term routing refers to selecting paths in a computer network along which to send data. Routing directs forwarding, the passing of logically addressed packets from their source, toward their ultimate destination through intermediary nodes. We first take a look at traditional routing methods like Distance Vector routing and Link State routing before moving on to routing in mobile ad-hoc networks.

### 1.1 Distance Vector Routing [1]

Distance vector algorithms use the Bellman-Ford algorithm. This approach assigns a number, the cost, to each of the links between each node in the network. Nodes will send information from point A to point B via the path that results in the lowest total cost (i.e. the sum of the costs of the links between the nodes used).

The algorithm operates in a very simple manner. When a node first starts, it only knows of its immediate neighbors, and the direct cost involved in reaching them. (This information, the list of destinations, the total cost to each, and the next hop to send data to get there, makes up the routing table, or distance table.) Each node, on a regular basis, sends to each neighbor its own current idea of the total cost to get to all the destinations it knows of. The neighboring node(s) examine this information, and compare it to what they already 'know'; anything which represents an improvement on what they already have, they insert in their own routing table(s). Over time, all the nodes in the network will discover the best next hop for all destinations, and the best total cost.

When one of the nodes involved goes down, those nodes which used it as their next hop for certain destinations discard those entries, and create new routing-table information. They then pass this information to all adjacent nodes, which then repeat the process. Eventually all the nodes in the network receive the updated information, and will then discover new paths to all the destinations which they can still *reach*.

## 1.2 Link State Routing [2]

When applying link-state algorithms, each node uses as its fundamental data a map of the network in the form of a graph. To produce this, each node floods the entire network with information about what other nodes it can connect to, and each node then independently assembles this information into a map. Using this map, each router then independently determines the best route from itself to every other node.

The algorithm used to do this, Dijkstra's algorithm, does this by building another data structure, a tree, with the current node itself as the root, and containing every other node in the network. It starts with a tree containing only itself. Then, one at a time, from the set of nodes which it has not yet added to the tree, it adds the node which has the lowest cost to reach an adjacent node which already appears in the tree. This continues until every node appears in the tree.

This tree then serves to construct the routing table, giving the best next hop, etc, to get from the node itself to any other network.

## 1.3 Flooding

This is a solution, where every packet is sent to every node in the network. Each packet has a unique ID, so nodes do not forward the same packet twice. Flooding based routing works as follows: Let us consider the example shown in figure 1. Consider what happens when node  $A$  wishes to talk with node  $G$ .

The color conventions in figure 1 are the following: Nodes that are transmitting *Route Request* packets are shown in orange. The route request transmissions of a particular iteration are shown by orange arrows. Old *Route Request* packet transmissions are shown using blue arrows. *Route Reply* packets are shown using green arrows. Nodes that are transmitting *Route Reply* packets in a particular iteration are shown in green. Finally, the red edges represent the *finalized* forward direction links to get from the source  $A$  to the destination  $G$ . The working of the flooding algorithm is easy to understand by looking at figure 1.

## 2 Motivation

We need to keep in mind that mobile ad-hoc networks are highly dynamic; Meaning that the links could forged and broken almost constantly. Distance vector routing suffers from the *count to infinity* problem. As distance vector routing could have a large convergence time once a link fails, it is not the best choice for mobile ad-hoc networks. Moreover, the routing overhead is very high. Similarly with Link state routing, the number of routing packets sent will be prohibitively high; This is because a large number of link state packets have to be flooded through the network. Since these problems greatly increase the battery power spent by the mobile nodes, they are infeasible.

### 3 Location Awareness

We have just seen that traditional routing protocols do not scale well in mobile networks. In the paper *Location Aided Routing in MANETs* by Young-Bae Ko and Nitin Vaidya, the authors make use of a node's physical location to aid in routing. Each node knows its physical  $(x, y)$  location. This location awareness is possible by using GPS [3].

Assume that at time  $t_0$ , node  $A$  and  $B$  were communicating. At time  $t_0$ , node  $A$  *knows* the location of node  $B$ , which is  $x_{b_0}, y_{b_0}$ . At some later point of time  $t_1$ , node  $A$  wants to talk with node  $B$ . Therefore,  $A$  can *estimate* the current location of  $B$  using its previous location  $x_{b_0}, y_{b_0}$ , as well as the time that has elapsed since they last communicated. For example, let us assume that the velocity of node  $B$  is  $v$ . Therefore, the distance that  $B$  could have traveled in the time period of  $t_0$  to  $t_1$  is equal to:

$$Distance = v \times (t_1 - t_0)$$

#### 3.1 Expected Zone

Therefore, at time  $t_1$ , node  $B$  can be anywhere in a circular area of radius  $Distance$ , centered at  $x_{b_0}, y_{b_0}$ ; assuming that the direction of  $B$ 's motion is unknown. Using this information,  $A$  can *geographically forward* the packet to the current location of  $B$ . Sometimes, there might be additional information available. For example, node  $A$  might know that under no circumstances will node  $B$  travel southward. In this case, the current location of  $B$  can be more accurately estimated as being a semicircle of radius  $Distance$  centered at  $x_{b_0}, y_{b_0}$ . These two situations are shown in figure 2. This area is known as the *expected zone*.

#### 3.2 Request Zone

When a node  $A$  wants to talk with a node  $B$ , the data has to be sent to the expected zone. In order to do this, the data should be flooded in a rectangular area, which includes the expected zone. This area is known as the request zone. It is the smallest rectangle containing the source and the expected zone. There are two cases that we need to consider. First, the case when the source is outside the expected zone. Second, the case when the source is within the expected zone. These two cases are shown in figure 3.

#### 3.3 The Algorithm - LAR 1

The algorithm works as follows: The source node  $A$  calculates the *request zone* by the mechanism mentioned above. It then broadcasts the packet into the wireless channel. The packet contains the coordinates of the request zone, as well as the address of the destination node  $B$ . Now, all nodes receiving this broadcast check to see if they lie within the request zone. If they do, then they forward the packet by broadcasting it again. However, if a node outside the request zone receives the packet, it is discarded. Care is taken to ensure that duplicates are identified and not forwarded. This way, the packet ultimately reaches the expected zone and then the destination  $B$ .

### 3.3.1 Is it Perfect?

No. Sometimes, even though there is a path from the source  $A$  to the destination  $B$ , the path might not belong to the request zone. In this case, the packet will fail to reach  $B$  even though a path exists. This scenario is shown in figure 4. The algorithm works around *holes* in the following manner: Suppose the transmission from source node  $A$  to the destination node  $B$  fails, then  $A$  retransmits the packet; but this time, it specifies a larger request zone. Another alternative is that if the first transmission fails, then the packet is simply flooded. If this second phase also fails, node  $A$  assumes that node  $B$  has died.

## 3.4 The Algorithm - LAR 2

Recall that in LAR 1, every packet contained the coordinates of the request zone, as well as the destination address. But this might be inefficient, because unnecessary packet transmissions might take place. For example, intermediate node  $I_1$  might broadcast the packet, and this broadcast is received by intermediate node  $I_2$ . If  $I_2$  also lies in the request zone,  $I_2$  broadcasts it again. The problem arises when  $I_2$  is further away from the destination  $B$  than  $I_1$  is. Ideally in this case,  $I_2$  should not forward the packet. To implement this algorithm, the piggybacking of information is modified. Suppose a node  $I_1$  is forwarding a packet, in addition to the request zone coordinates and the destination address, it also contains a value which indicates the distance from itself to the destination. Therefore when another intermediate node  $I_2$  hears this broadcast, it broadcasts it further only if it is closer to the destination than  $I_1$ . This way, in every step, the request zone keeps getting smaller and smaller. This mechanism is shown in figure 5.

To make this more precise, consider the two intermediate nodes  $I_1$  and  $I_2$ . Let the distance from  $I_1$  to the destination be  $Distance_{I_1}$ ; Similarly, let the distance from  $I_2$  to the destination be  $Distance_{I_2}$ . For constants  $\alpha$  and  $\beta$ , node  $I_2$  will forward the data it received from  $I_1$  only if the following condition holds good.

$$\alpha \times Distance_{I_1} + \beta \geq Distance_{I_2}$$

As mentioned in the previous paragraph, this mechanism can be imagined to be such that the request zone gets smaller with every packet that is forwarded. This is shown in figure 6.

## 4 Simulation

By extensive simulations, the authors argue that their protocols work better than conventional flooding. Given below are a few key points about the simulation.

- Maryland Routing Simulator
- $1000^2$  units square area
- 15, 30 or 50 stations

- All stations move continuously ( $1.5 \text{ ms}^{-1}$  to  $32.5 \text{ ms}^{-1}$ )
- Radio range 200, 300, 400 and 500 units
- 10 packets per second (Exponential inter arrival times)
- Ignore collisions, congestion and transmission errors
- Results averaged over 30 runs
- *DP*: Number of data packets received at the destination
- *RP*: The number of routing packets received by all nodes

## 4.1 Performance

The results show the following facts about the performance of LAR-1 and LAR-2. Let  $\eta$  be the number of routing packets per data packet.

- In all cases, LAR-2 has a lower  $\eta$  than LAR-1, hence is more efficient
- In all cases, Flooding has the highest  $\eta$  and hence is the most inefficient
- As the speed of the nodes increase,  $\eta$  increases for all three schemes. This is obvious, because higher speed implies that links are broken faster, and the request zone gets larger.
- As the transmission range increases,  $\eta$  decreases for all three schemes. This is because a smaller number of hops have to be traversed to reach the destination - Hence the routing is more efficient
- For LAR-2, routing is most efficient when  $\alpha = 1$ . Also,  $\eta$  shows very little variation with changes in  $\beta$ .

All the performance graphs are in the paper, hence have been omitted from the scribe notes.

## 5 Optimizations

### 5.1 More efficient request zone

Previously, we have seen that the edges of the request zone were parallel to the  $X$  and  $Y$  axes. This might sometimes lead to unnecessarily large request zones. The optimization drops the requirement that the edges have to be parallel to the  $X$  and  $Y$  axes. From figure 7, we see that the size of the request zone (and hence, the scope of flooding) can be substantially reduced.

## 5.2 LAR-2 like behavior of LAR-1

The size of the request zone can be reduced with each round of forwarding, leading it to behave like LAR-2. Every intermediate forwarding node changes the size of the request zone, making it progressively smaller. The flip side of this approach is that the protocol becomes more vulnerable to routing failure because of *holes*. Refer to figure 8 for a clearer understanding.

## 5.3 Conical Request Zone

Instead of having rectangular request zones, the scope of flooding can be further reduced if the request zone is conical. Refer to figure 9 for a clearer understanding.

## 5.4 Directional Antenna

In the previous discussion, we assumed that all nodes had omnidirectional antennas. However, if nodes are equipped with directional antennas, then packets can trivially be forwarded only along a certain direction. This way, the scope of flooding is greatly reduced. However the drawbacks are that a node might now know its *orientation*, and moreover, the accuracy of the directional antenna might be low. As a result, this scheme might not work very well. Finally, this increases the cost of all nodes. refer to figure 10 for a clearer understanding.

# 6 Conclusion

- Conventional routing protocols do not scale well with mobile nodes
- Location awareness can restrict flooding
- Selective forwarding is possible
- Saves energy, reduces latency
- Many optimizations are possible

# References

- [1] [http://en.wikipedia.org/wiki/Distance-vector\\_routing\\_protocol](http://en.wikipedia.org/wiki/Distance-vector_routing_protocol)
- [2] [http://en.wikipedia.org/wiki/Link-state\\_routing\\_protocol](http://en.wikipedia.org/wiki/Link-state_routing_protocol)
- [3] [http://en.wikipedia.org/wiki/Global\\_Positioning\\_System](http://en.wikipedia.org/wiki/Global_Positioning_System)

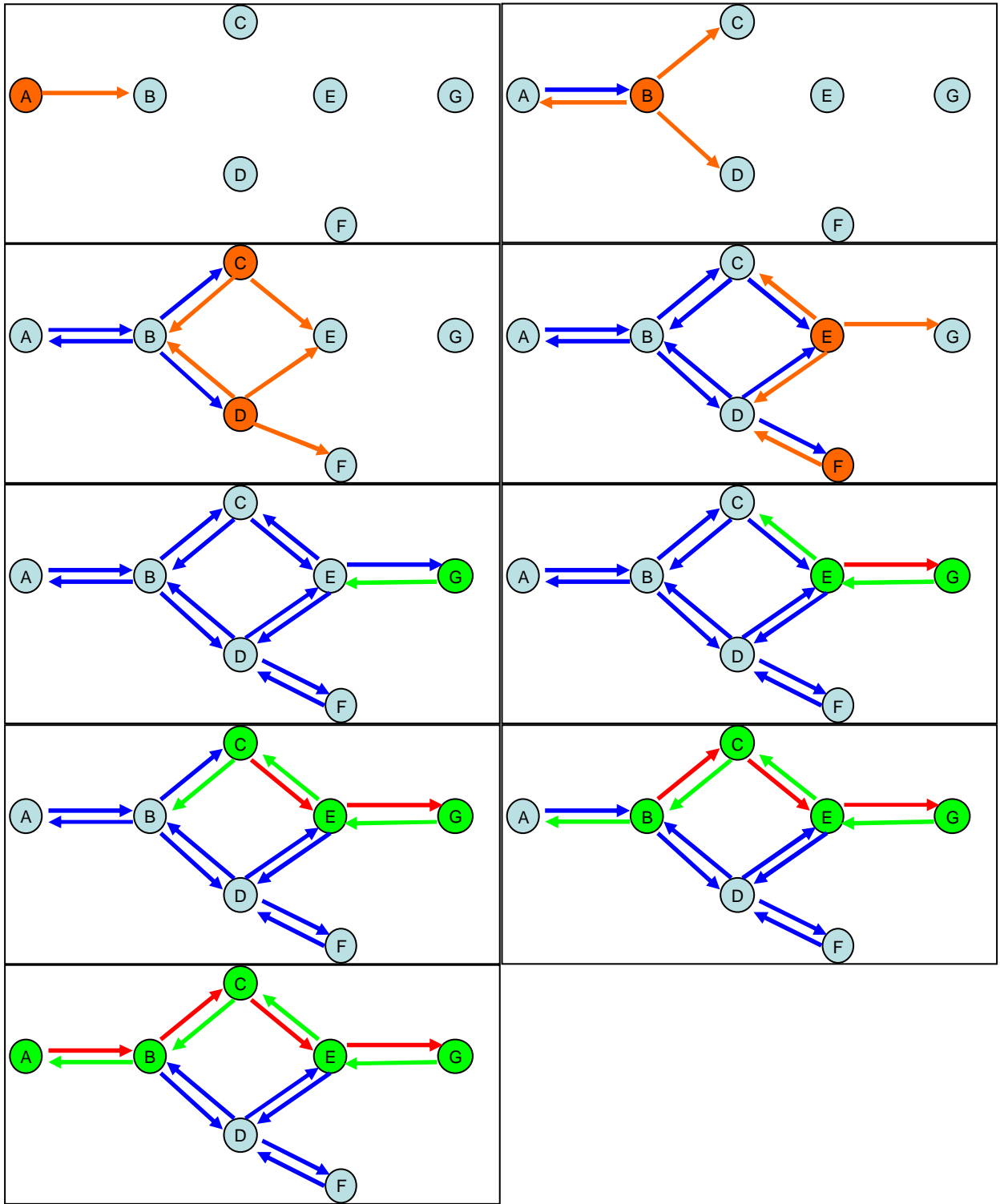


Figure 1: Route Determination by Flooding

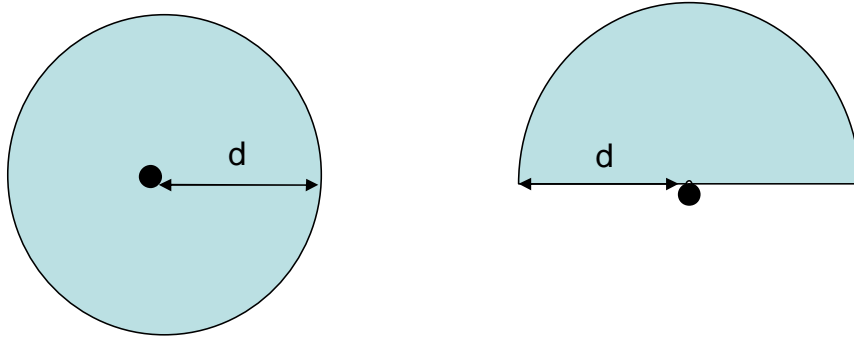


Figure 2: Estimation of  $B$ 's Current Location - *Expected Zone*

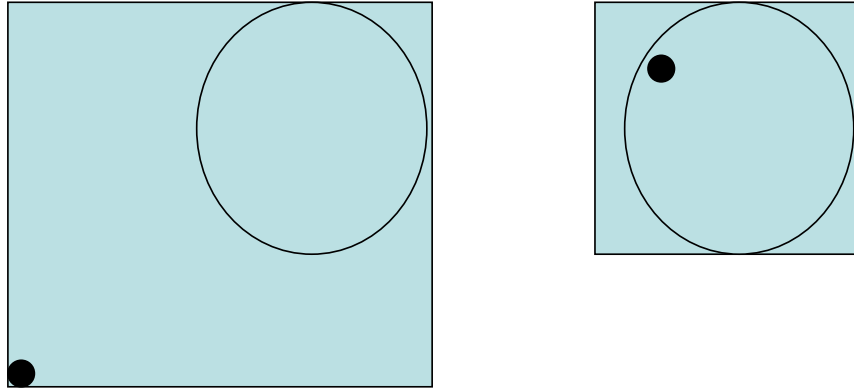


Figure 3: Scope of Flooding - *Request Zone*

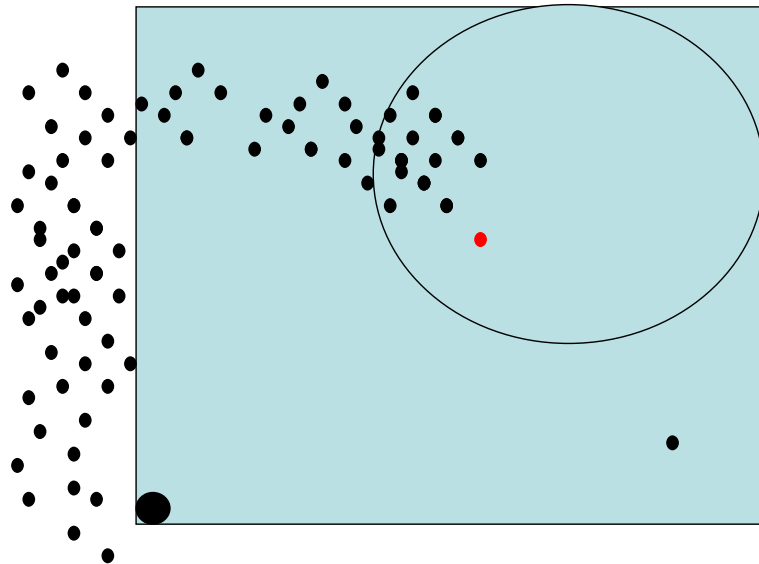


Figure 4: The path does not belong to the request zone

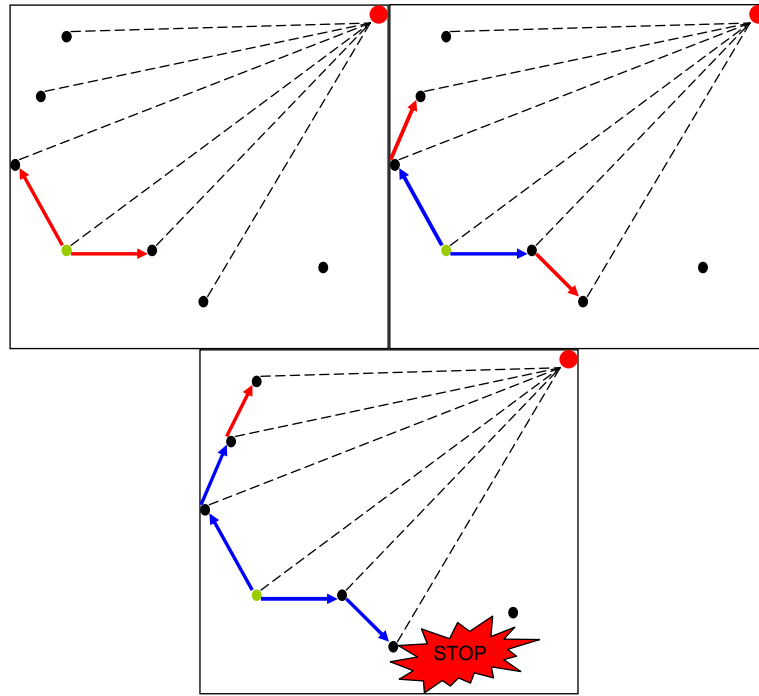


Figure 5: LAR - 2

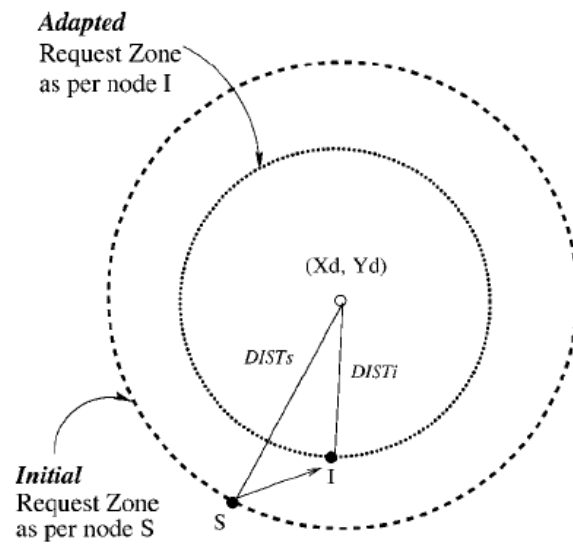


Figure 6: The request zone gets smaller with every round

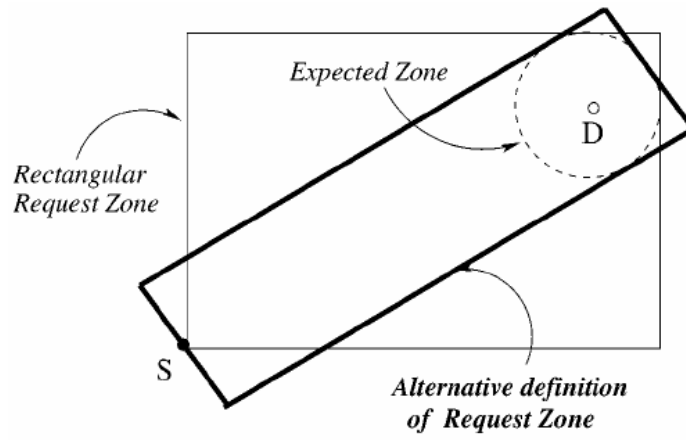


Figure 7: Optimization 1

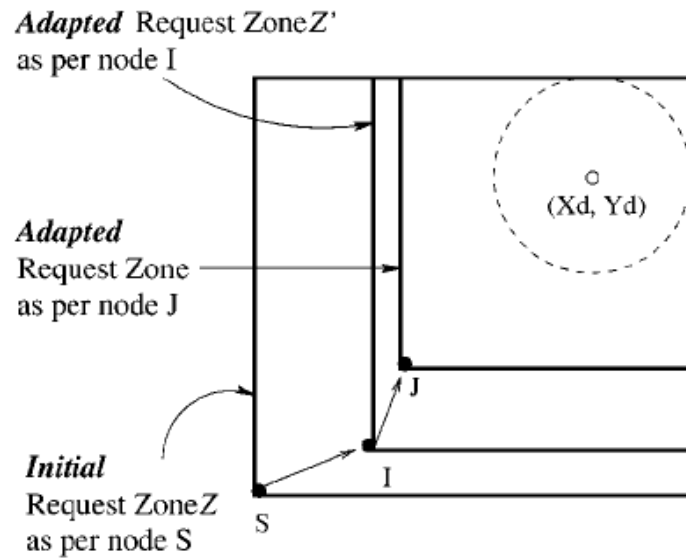


Figure 8: Optimization 2

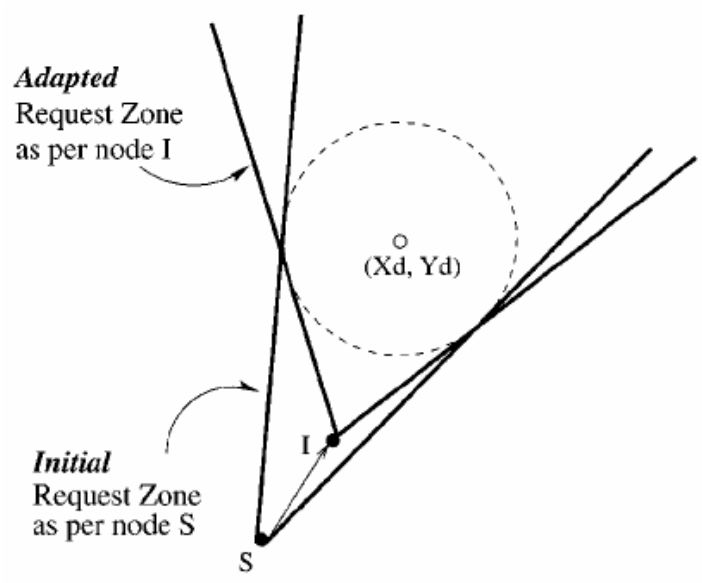


Figure 9: Optimization 3

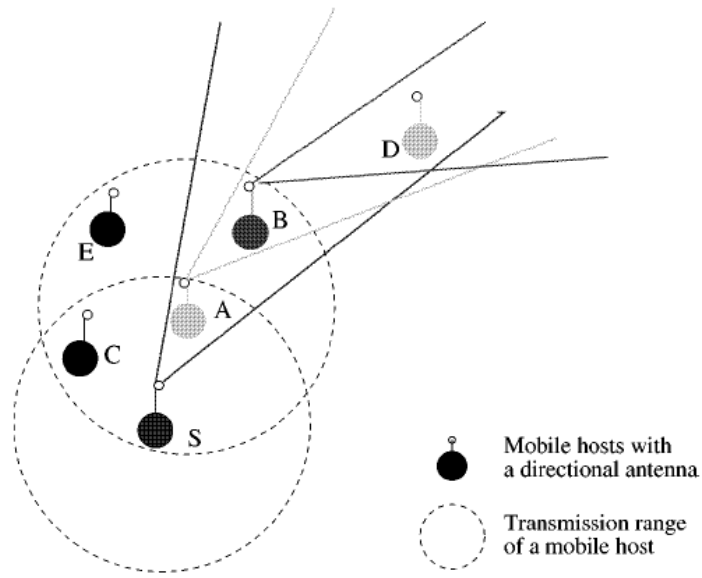


Figure 10: Optimization 4