Location Based Network Node Requirements

Status

Initial test packages were already released and seeds were initialized, now documenting, trying to collect feedback, test and make the release more open to the public. .

Working at: https://github.com/Fermat-ORG/iop-location-based-network

Introduction

Node operators set a location for these nodes that becomes part of their network identity. This location can not be anywhere in the globe. Their IP must resolve to that location and its latency measurements must be consistent with it, other wise, the network will reject them. Once accepted into the network, a new node becomes part of a network of nodes which topology is based on node's geographical location.

These nodes have 3 major interfaces, each one running on different ports:

1. **IoP Servers Interface:** Through this interface IoP servers installed on the same hardware are allowed to consume node services needed to participate in this network as IoP service providers.
2. **LBN Nodes Interface:** Through this interface nodes talk to their peers.
3. **Clients Interface:** IoP Clients consume node's services through this interface.

Node to Servers Use Cases

These are services that Nodes gives to Servers running on the same hardware.

- **USE CASE: Sharing Server Interfaces:** Server need to tell nodes at which interfaces clients can contact them: The node builds up a list of its server interfaces that clients can query, see Node to Client Use Cases.
    - **Publish My Client Interface (Server Type, Ip, Port):** Provides the Ip and port of the default client Interface. This is the informations clients needs from a node in order to contact a server running on top of it.
        - [Unrelated Nodes Release]
    - **Remove My Client Interface (Server Type):** If a server closes in a controlled way or loses network connectivity, it should ask to remove its Ip and port from the list of client interfaces. When a server changes its Port or Ip, it should remove itself from the list and register again with the changed details..
    - **Get Neighbour Nodes Ordered by Distance (KeepAlive?):** Provides a list of node profiles that this node has as neighbors. Useful for sharing information with neighbouring nodes, see e.g. shared profiles of the Profile Server. If the keepalive flag is set, the connection is kept alive and changes of the neighbourhood (i.e. node was added, updated or deleted) are continuously reported to the requesting local service.

        - [Unrelated Nodes Release]

Node to Node Use Cases

The Location Based Network needs to act like a coherent entity. This means that besides these core services nodes must also provide services to other nodes.

The separation of relationships between nodes: being unrelated, colleagues, neighbors determines the accessibility nodes have between them. These relationships are always mutual for both nodes. The access level of a remote node to the functionality of a node depends on the relationship the nodes have between each other. Abuse is controlled method by method against access logs kept for each of them.

We can classify the relationship between nodes like this:

- **USE CASE: Unrelated Nodes:** Nodes join the network unrelated to other nodes. Later they can become colleagues or neighbors. Unrelated peers by default don't trust each other. Any of them can be a malicious node of any kind. That means that interactions with unrelated nodes are taken with special care. There are certain services between unrelated peers:
    - **Get Node Count ():** Get number of nodes (no matter if colleagues and neighbours) known by this node. Useful for other nodes to determine a targeted map size while initializing the world map.
        - [Unrelated Nodes Release]
    - **List Random Nodes (Count, Include Neighbors?):** Provides a list of random nodes known by the requested node. Useful to collect node contacts while a new node is initializing its own maps.
        - [Unrelated Nodes Release]
    - **List Closest Nodes Ordered By Distance(Count, Radius, GPS Location, Include Neighbours?):** Provides a list of the closest nodes known by the requested node to the location provided. Useful while exploring the world to find nodes near a location. Nodes are returned by range from the targeted GPS location in ascending order.
        - [Unrelated Nodes Release]
- **USE CASE: Colleague Nodes:** Nodes must know nodes around the globe to help clients reach faraway lands. As a network, all nodes equally benefit if they can avoid non-profitable requests from clients, such as routing requests. For this reason, nodes exchange node profile info with each other and become Colleagues. Services associated with this type of relationship are:
    - **Accept me as a colleague (Node Profile):** One node request another to exchange their profile info and become colleagues. If the request is accepted, then the requested node returns its latest profile information (to make sure no outdated information is stored at the initiator side) and both node profiles end at both node tables until the information expires. The response should also contain the detected external IP address of the initiator node node to help it autodetect its connection information to be advertised to the network.
        - [Colleague Nodes Release]

    - **Renew Colleague Relationship (Node Profile):** This method is executed by the initiator of the Colleague relationship before the expiration date. It can also be used to update the profile info (e.g. if an IP change has been detected or server was restarted and its PORT has changed). By executing successfully this method the initiator also renews the expiration date of the local information of its Colleague.
        - [Colleague Nodes Release]

- **USE CASE: Neighboring Nodes:** Nodes must know their neighborhood very well. Being neighbors is a type of relationship between nodes. Services related to neighboring nodes are:
    - **Accept me as Neighbor (Node Profile):** This method is used by nodes entering the neighborhood. When they find other nodes that qualify to be their neighbors they execute on them this method to get the relationship established. The previous relationship of the nodes may be either unrelated yet or colleagues already, i.e. there is no need for nodes to be colleagues first. The response should also contain the latest node profile information of the requested node and the detected external IP address of the initiator node.
        - [Neighboring Nodes Release]
    - **Renew Neighbor Relationship ():** This method is executed by the initiator of the Neighbour relationship before the expiration date. It can also be used to update the contact info if IP or PORT has been changed. By executing successfully this method the initiator also renews the expiration date of the local information of its Neighbor.

Node to Clients Use Cases

The network itself must provide certain core services in order for end users to find each other in the network and later communicate. These core services are defined out of these known use cases:

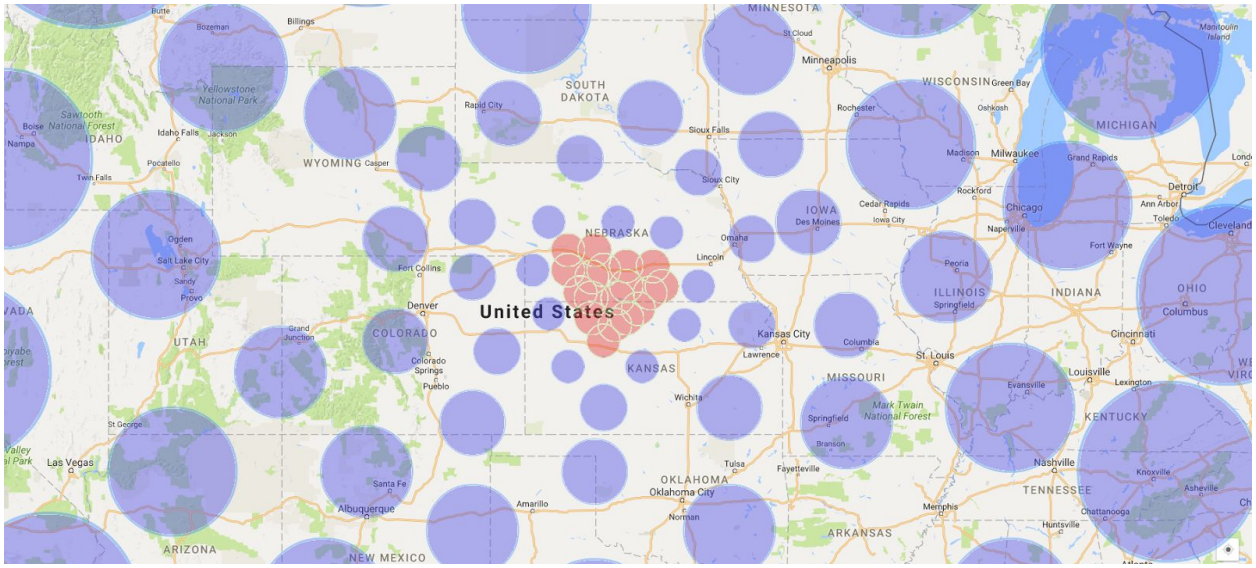
- **USE CASE: Finding a Server Public Interface:** Clients can ask anytime for information regarding the servers running on top of a network node. For those situations we have the following services:
    - **List Available Servers ():** This method returns the list of servers that previously sent their information to the network node including the IP and Port. This information was previously received when the server reported that to the node.
        - [Routing by Location Release]
- **USE CASE: Finding Known People for the First Time:** Here we assume that when people want to find someone they know, they also more or less know where they are and they can provide that information to be used by the client software to help them find them. In order for the network to scale to possibly millions of nodes, most of the work is pushed to clients, leaving the work done by nodes to the minimum possible. This means that when end users finally find each other, they must remember the server profile of the people found, because finding them again is an expensive operation for them (time wise) and for nodes (resources). This means that the use case for finding known people has a low frequency. The procedure to find known people for the first time starts at the profile server of the client performing the search and ends at the targeted people's profile server. The services to clients associated with this use case are:

- **List Closest Nodes Ordered by Distance(GPS Location, Count, Radius, Include Neighbours?):** This function returns a list of the nodes closest to a latitude / longitude pair. Using this service at different nodes, clients can navigate to a target region where the profile server of the person to be found should be. Once clients enter that person's profile server's neighborhood, all nodes will know about the target person and which of the nodes in the neighborhood is his profile server. If the searched person is not found at the expected location, the search should be continued outside the neighbourhood of the closest node, this is how setting the flag to false may be useful, There is a similar method to be consumed by nodes. The main differences with this one are the security measures implemented to avoid abuses [to be described later].
    - [Routing by Location Release]
- **Get Neighbour Nodes Ordered by Distance():** If a person was not found on a node, the search has to continue outside the neighbourhood. If closest nodes outside the neighbourhood (discovered through the previous operation) also do not know the profile, the client has to look even farther away. Doing so the client needs to know the area that has been already discovered, otherwise it might return to the same area once again. There is a similar method to be consumed by local services. The main differences with this one are the keepalive feature to receive updates and security measures implemented to avoid abuses [to be described later].

Local Network Map Algorithm

The network is location aware and every node knows only a part of the network. This local network map is unique to each LOC NODE and partly overlaps with information kept at other nodes. Each node knows better the part of the network that is closer to it and as the distance increases it knows less. Nodes implement an algorithm that helps them pick which nodes they need to know about and which nodes they don't. As the burden to announce themselves to the network is put on newcomers, already running nodes just need to wait to be contacted.

The Local Network Map consists of two parts with different goals and properties. The Neighbourhood Map aims to maintain a comprehensive list of the closest nodes. The World Map aims to provide a rough coverage of the rest of the world outside the neighbourhood. To prevent too much data and limit node density, the world map picks only a single node from each area. Covered areas are defined with circular "bubbles" drawn around single nodes that must not overlap. The World Map uses a changing bubble size to have a denser local and sparser remote node density. So farther the area, the bigger bubbles are, i.e. less nodes are included. Area bubbles are defined autonomously and specific to the LNM of each node. There is no node count limit for the World Map, the increasing size of bubbles serves as a limit for node counts. For the Neighbourhood map the bubbles may overlap, so we need an maximal count of neighbouring nodes.



Node distribution example. Distance required between nodes plotted as node bubble size for colleagues nodes. Neighboring nodes drawn in red.

To add a new node to our World Map, the new node has to agree and also include our node to its World Map. If this succeeds, they are considered to be colleague nodes. Consequently, the relation of being colleagues is mutual.

The *Local Network Map* algorithm uses a predefined formula to decide if a node should be added to the World Map. The formula described below is used against both the node that is to be included and the closest node on the LNM to this one.

Being:

A = Current node.

N = Newcomer node.

C = Closest node at the LNM to the newcomer.

Dan = Distance from A to N

Dac = Distance from A to C

Dnc = Distance from N to C

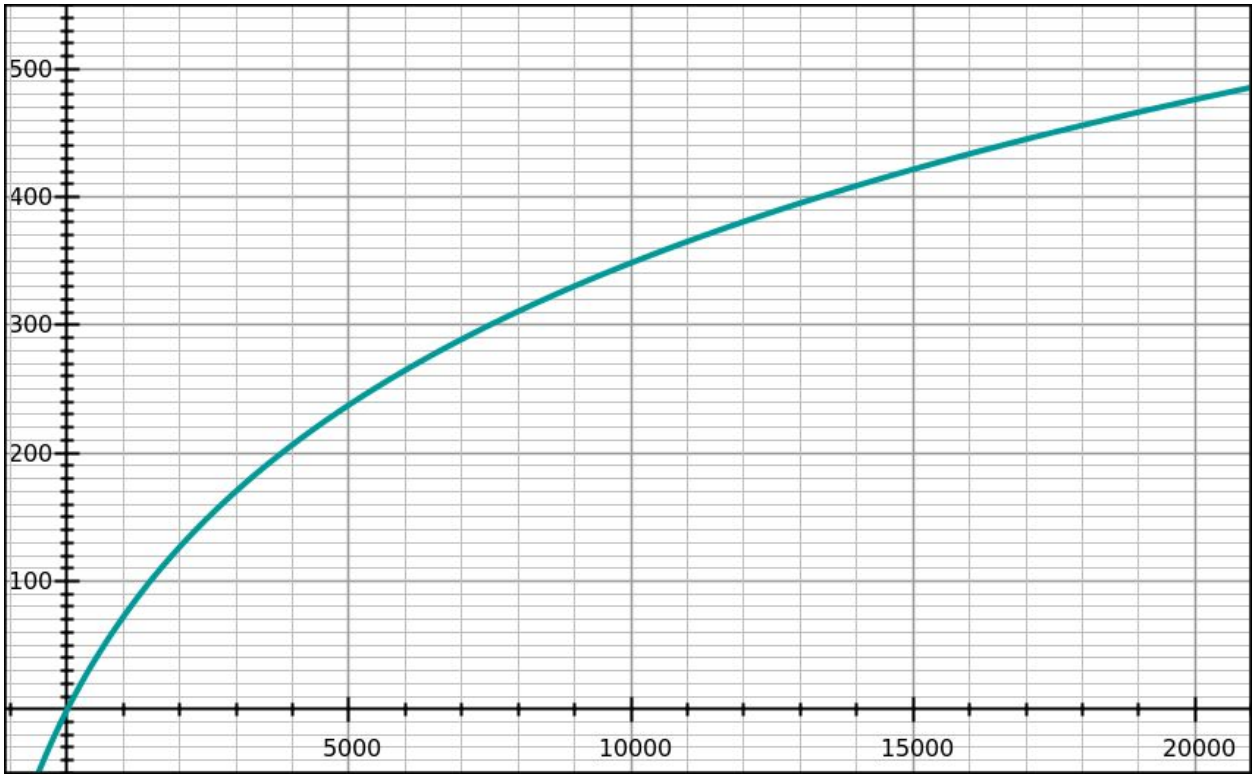
Nmax = Maximum number of nodes that will be added to the Neighbourhood Map. (estimated magnitude: 100, current exact value is 50)

d(x) = Node bubble radius at an x distance from A = log(x+2500) * 501 - 1700

Bf = Bubble radius required for nodes at (20,000 km) the farthest point on earth = (est. 480 Km)

Nbubble = Bubble radius of Dan

Cbubble = Bubble radius of Dac



In this graph the x axis is Dan [km], the y axis is the corresponding bubble size of a node at that distance.

We require that the bubble around each node must not overlap with any other bubble of colleagues nodes. To verify this requirement it's enough to check intersection with the bubble of the closest node. Thus our node inclusion criteria just verifies that the bubble of node C does not overlap with the bubble of node N. If the following formula applies then the new node is accepted.

d(Dan) + d(Dac) < Dnc

By executing this algorithm, each node would end up having a piece of a full network map, with a high density of nodes surrounding the subject profile server and the concentration decreasing smoothly as the distance from the subject increases. In this way the profile server can effectively route clients to far distances without the need to have the full network map.

For example, if:

Dan = 5000 km

Dac = 4500 km

Dnc = 350 Km

Nbubblesize = log(5000+2500) * 500 - 1700 = 237.53 km

Cbubblesize = log(4500+2500) * 500 - 1700 = 222.54 km

Nbubblesize + Cbubblesize < Dnc ⇒ 406.07 Km < 350 km ⇒ This node won't be included in the local network map because its bubble is overlapping other nodes bubble already on the local map according to the distance the new node is to the subject profile server.

Several processes are needed to keep a healthy Full Network Map distributed across different Local Network Maps:

1. **Local Network Map Initialization:** New nodes execute the Local Network Map algorithm while initializing. They use information of other nodes to scan the network and find possible candidates with whom exchange their Node Profile. They first validate using the same algorithm that a candidate is good for their local network map and after that they request the Node Profile exchange. The initialization process consist of the following steps:

    a. **World Discovery Phase:** To avoid every new node to follow the same path and go against the same set of nodes many random selections are done during this process.

        i. The first step is to randomly select a seed node from the ones available. Since seed nodes are old nodes we can assume they have their LNM without major holes and that we can ask them how many nodes they know in total. That would be our target amount of nodes.

        ii. Second, ask the selected node to provide a list of 100 randomly selected colleagues nodes from its own LNM. Though we are under the World Discovery Phase, we do not exclude neighbours because the algorithm would not work while the network is small.

        iii. Run the Local Network Map algorithm over the list of nodes and for the ones that qualify, try to exchange Node Profiles with them, i.e. try to make them colleagues.

        iv. See if the 75 % of the target was reached and added to the LNM already. If not, randomly pick one of the nodes in the current list and ask him a new list of a 100 colleagues and go back to ii) .

        v. Once the 75% of the target is added we are done. The rest will be filled either by nodes requesting to be added or by the maintenance process described below.

    b. **Neighborhood Discovery Phase:** In this phase the goal is to add to the LNM the subject node's neighborhood. The steps to do so are:

        i. Pick the closest node to the subject node on the LNM and get routed to your own location. This means acting as a client that needs to get the closest node to one location asking several nodes to find it. Continue with that closest node you have found after being routed.

        ii. Ask that node a list of the 10 closest nodes to the subject node. With this list create a first list of neighbors candidates. If the list is empty that means you are the first in that

neighborhood and no further action is required, just wait for neighbors to appear someday. If not, continue in iii).

iii. From the neighbors candidates list choose the closest node to the subject node location.

iv. Ask this node for a list of the closest nodes to the subject node. Avoid asking a node you already asked this list before. If you already asked this node before, then do it with the next closest to the subject node you haven't ask for its list yet.

v. Append the list of nodes received to the list of neighbors candidates.

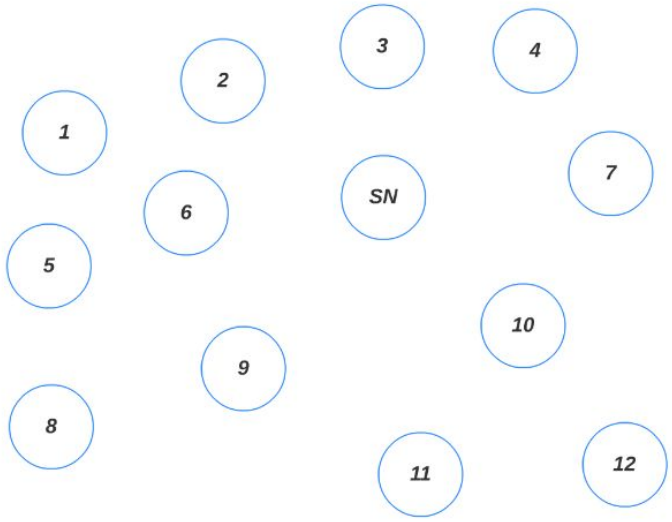
vi. If in iv) some nodes were added to the list, then goto iii) until no nodes are added anymore. We will end up with a list of all candidates nodes to be our neighbors.

vii. It can happen eventually that the number of neighbors identified exceed the Nmax value. For that reason, the list of neighbors now must be ordered by distance to the subject node (increasing). Next we need to loop from the beginning of the list exchanging profiles with these neighbors until we either exhausted the whole list or successfully exchanged profiles with Nmax nodes. Neighbour nodes must be flagged at the LNM as neighbors. It can happen that in a crowded area a new node appears and its natural neighbors already reached the Nmax threshold. In this situation they will still accept the newcomer even exceeding momentarily the threshold. They will return to the right range later, rejecting renewals from farther neighbors.

An example for Neighborhood Discovery Phase follows: (See the graph below).

1. Let's say the closest node at the subject node LNM is #1 and all the other nodes are unknown.
2. From #1 we get routed according to the Location Based Routing Procedure listed below, and in this example we end up the routing at node #2.
3. We ask #2 the list of closest nodes at our location and it returns is {3, 6} being that our first list of neighbors candidates.
4. The closest node to the subject location here is #3.
5. We ask #3 for its list of nodes closest to the subject node and it returns {6, 9, 10}
6. Now our candidates list is {3, 6, 9, 10}
7. As we added some nodes then we go to ii)
8. The closest node to the subject location continues to be #3.
9. We skip #3 and go and ask #6 for its list of nodes closest to the subject node and it returns {2, 3, 7, 10}
10. Now our candidates list is {2, 3, 6, 7, 9, 10}
11. As we added some nodes then we go to ii)
12. The closest node to the subject location here is still #3.

13. We skip #3 and #6 and ask #2 for its list of nodes closest to the subject node and it returns {3, 4, 9, 10}
14. Now our candidates list is {2, 3, 4, 6, 7, 9, 10}
15. As we added some nodes then we go to ii)
16. The closest node to the subject location is still #3.
17. We skip #3, #6 and #2 and ask #10 for its list of nodes closest to the subject node and it returns {3, 4, 6, 9}
18. Now our candidates list is {2, 3, 4, 6, 7, 9, 10}
19. No new nodes were added. It is time to move forward.
20. The process of exchanging profiles starts and we end up completing our neighbouring map.



Neighbors Discovery Phase example. SN = Subject Node

2. **Node Relationship Renewal:** Each record stored locally was generated either by the subject node requesting to be added on others node's network map or foreign nodes requesting the subject node to do so. For keeping information up-to-date, each node considers the records added by foreign nodes requests valid for a system wide constant period of time (est. 24 hours, but can be adjusted later). After that the record is considered expired and subject to be removed. Each node then is responsible for sending a renewal request to the nodes which previously accepted their request in order to extend the expiration time further into the future. It is reasonable to think that old stable nodes that arrived early enough will probably have less work than new or unstable nodes and that consequence is aligned with the resilience objectives of the network. The process to renew colleagues relationships should be run daily and follow these steps:

    i. Create a list from the LNM of all colleagues nodes which relationship will expire the following day.

    ii. Iterate through the list connecting to each node and executing the Renew Node Profile method.

3. **Local Network Map Maintenance:** Finally, nodes need to have a network map maintenance process that proactively try to fill the holes in their own map. This prevents incomplete maps when for any reason part of the map gets empty or keeps empty during the lifecycle of the network. This process is executed periodically (somewhere between the minutes and hours magnitude, currently set to 5mins) and follow these steps:

    a. Repeat the following steps a few (currently 5) times to discover potential colleagues

        i. Randomly choose a GPS location.

        ii. Pick from the LNM the closest node to that target location.

        iii. Ask this node for a list of nodes on its LNM around this target location.

        iv. From this list pick the closest node to the target location and try to create a relationship with the following rules: .

        v. If we don't have enough neighbours or this node is closer than our farthest neighbour, try to make it a neighbour.

        vi. Otherwise verify with the LNM algorithm if this node qualifies to be a colleague.If so then try to make it a colleague.

Location Based Routing Procedure

Both clients and nodes need to be routed to a certain location. Sometime this location is a city where a friend lives, other times it is the location of a node. In any situation the whole purpose of having a LNM is to allow location based routing.

The procedure always start asking a node for a list of nodes closest to a certain location. The expected amount of records returned and the maximum distance to the location provided can be specified as parameter to the method call and they very much depends on the app being routed.

Usually from the resultset received the app being routed will select the closest node to the target location and will repeat the query. It might provide the same parameters or not, again that depends on the app and the use cases that is supporting.

Some apps, after receiving the first result set with no nodes closer to the target location that the ones received before will stop there and the routing procedure is considered finished. Some other apps, will query the second closest node or even the third one just to be sure none of them have a closer node on their LNM. There is no guarantee that you will find the absolute closest to the target location.

Node operation "List Closest Nodes" always returns nodes in ascending order by distance. The client can always be sure that there is no other closer node known by the queried node instance than the first one returned.

Routing Model Properties (Routing Algorithm Test)

We would like to describe here a problem that we would like the routing model to successfully solve. Any model we suggest should be tested against this example to see if there are any problems.

Alice lives in London and she wants to find her friend Bob Doe and she only knows that he lives in Middlesbrough. The distance between them is about 340 km. Alice has her profile server in London. She doesn't know where Bob's profile server is.

The fact is (but Alice doesn't know that when Bob entered the network, he made a local search for nodes in his area and found out that the nearest profile server to his location is in Sheffield (130 km) and one more node is in Manchester (135 km), then one is in Glasgow (240 km), and one in Edinburgh (200 km). Being the cheapest offer and having a good connectivity to Manchester, he chose the node in Manchester to be his profile server.

Alice does her search for "Bob Doe in Middlesbrough", which starts at her profile server. She expects her profile server or her client device to know that Middlesbrough is a mid size city (population of 174,700), so to her query it can add something like 30 km radius to cover whole city and some area around, just to be sure Bob will be found.

Her profile server should be able to find one or more nodes that "cover" the target area. By "covering" we mean that there should be set of nodes that are somehow responsible for that area. There should not be a populated area in the world that should not be covered. Having find those nodes, Alice's profile server should be able to ask them for Bob's profile server and that information should be the part of the result of her search operation.

Alice expects to see on her screen profiles of all users whose account name is Bob Doe and who live in Middlesbrough area. She doesn't want to see profiles of people outside this area - e.g. from Manchester.

If a routing model is able to solve this problem efficiently, it can be a good model. The problem should not be solved, just by tweaking some parameters to fit this example. If it can't solve this problem efficiently, it is probably not what we are looking for as this is a very real situation and a basic search query that we want to support.

As variations on this test, we can add a node to the Middlesbrough, but either Bob will still prefer to select his profile server to be the node in Manchester, or the node in Middlesbrough will join the network after Bob established profile server agreement with Manchester's node.

Example #1: David from London searches John Doe that lives in Middlesbrough.

1. David's device request its profile server in London a list the closest nodes to Middlesbrough. 5 nodes are returned, none in Middlesbrough.

2. The closest one is 50 km from Middlesbrough, so it's density is of nodes at that range is quite high. This one return 3 nodes that are in Middlesbrough itself and some other scattered around.

3. David's device chooses the node closest to the city center and ask it yet for a final list of nodes. This one should know better since Middlesbrough is where its neighboring nodes should live. This one return 10 more nodes scattered around Middlesbrough.

4. From now David's phone starts querying these nodes, starting for the one closest to the target location. It asks for a list of John Does he knows there. Together with the list of 3 profiles that match the query, the list

of neighbour nodes covered by these results come back to David's device so the app can know the reach of its query.

5. David's app catches this list, that not only include a name and a thumbnail picture, but also the pub key of each profile, and the node profile of their profile server.

6. David's device avoid asking the second node because it was reported to be covered by the first node because they are neighbours and share profile information. So it goes to the third node which is not a neighbour of node #1 and ask the same question receiving a list of 5 profiles that match the query and another list of covered neighbour nodes.

7. The app running in David's phone believes this is enough to show a screen with results, so it stops there and displays the list on David's phone.

8. David see the list, scroll down forcing the app to search for more. In this situation the app continue asking node #4, #5 and showing the information on the screen.

9. At some point David recognizes the picture of his friend and clicks on it.

10. The app then goes to its node cache and gets the node profile of the profile server of the profile selected.

11. It connects to that node and request the profile data of that profile. The node returns a bunch of information it stores for that person, including more pictures, texts, etc. It also returns a flag that means that John Doe is online.

12. The app opens a detailed profile screen that shows that content and presents David the option to connect with his friend. Once David presses the connect button, the app requests application service call so as to connect with the same app on John's phone. The node receives that request and arranges the call. Next steps: out of scope.