Supplemental Material of

Merged Path: Data Dissemination for Large Scale Multiple Mobile Sinks WSN

Ammar Hawbani, Xingfu Wang, Omar Busaileh, Ammar Qarariyah, Liang Zhao, Mohammed A. A. Al-qaness, and S. H. Alsamhi

DESCRIPTION: This document is a supporting content for the manuscript (Merged Path: Data Dissemination for Large Scale Multiple Mobile Sinks WSN), given the limited number of pages allowed by the publisher. We provided more details about the algorithms and the simulation results. The source code for this protocol is available online here.

Index Terms: Wireless sensor networks, multiple mobile sinks, data routing, merged path.

1. Clustering and Bifurcating

1.1 Clustering Algorithm:

The goal of this algorithm is to divide the sinks into groups and to find the branches where the packet to should be traveled. Thus, this algorithm returns a list of branches. The **Branch** object contains **Cluster (**line #157**)**, **Start** and **End** points of the cluster. Cluster contains a list of sinks of the branch. Start point is the start point of the branch while End point is the end of the branch. Before explaining how to find the branches, we will first explain how the clustering algorithm runs. The high-tier node sends back the locations of the sinks at the network to the source node that has data to send. Thus, the clustering algorithm is performed at current source node (the current bifurcation point). Thus the inputs for the clustering algorithm are **bk_sinks** (the locations sinks in the field which responded by the high-tier node), the **b_k** (current bifurcation point), and the **clustingThreshould**. The source code for the **Clustering** algorithm is on the link . The object (`Angle` line#24) contains various of functions. The main function [`GetAngle` line #35] is to return the cosine similarity between two sinks(namely, `sink1 and sink2`) considering the current (`bifurcation`) point. The clustering algorithm starts at line #250. It runs in three steps. First, (`FindAngles` line #468) to find the cosine similarities of all nodes by calling the (`GetAngle` line #35) . All similarities are stored in an object named (`AngleSimlirityEdge` line #96). The second step is to remove the similarity edges which are greater than the threshold value (`FindMinusScors` line #553). The third step is to find the clusters (`ComputeClusters1` line #292). This function runs as explained in the main manuscript.

1.2 Bifurcation

The bifurcation point is exactly where the merged path should be branched to several sub-branches, and explicitly where the clustering process is computed. The source code of bifurcating is on the link. It contains the (`Branch` line #15) object the contains a **Cluster (**line #157). The main function in in the (`Bifurcation object`) is the **(**`FindBranches` line 32**)** that returns the branches at the current bifurcation point. This algorithms woks as follows. The source node divided the sinks into clusters as explained in the previous section (`ComputeClusters1` line #292). The main process here is to find the start and end of each branch. See the details code on the link.

2. Diagonal Virtual Line Construction

The construction process for the overlay structure is explained in the link (`DiagonalVirtualLine`). As explained in the manuscript, DVL defines two reference points for the diagonal for example (`Point(0, 0); and Point(NetworkSquareSideLength,NetworkSquareSideLength);`). After identifying these two points, the protocol starts by broadcast (`DVLConstructionMessage`). It has the following functions: `GeneratePacket`, `SendPacket`, `RecivePacket`, `SelectNextHop`.

3. Merged Path

The source code for data dissemination is online via the link here. It includes the implementation of different messages in the protocol as follows.

3.1 Report Sink Location

Whenever the sink changes its location, it implements (`ReportSinkPositionMessage`). The sink first selects its agent node which should be the nearest to the sink. The sink creates an object (`SinksAgentsRow`) to store the agent location and also compute the distance to the diagonal line. `ReportSinkPositionMessage` object contains multiple functions include the `ReportSinkPositionMessage`, `HandelInQueuePacket`, `GeneragtePacket`, `SendPacket`, `RecivePacket`, `SelectNextHop`. `ReportSinkPositionMessage` starts by checking the agent node, high-tier or low-tier node. If the agent is a high-tier node, then it just shares the sink location with all high-tier nodes (`ShareSinkPositionIntheHighTier`). If the agent is low tier node, then it starts the (`SendPacket`). In the (`SendPacket`) the nodes always employs (`SelectNextHop`). If a

next hop node is found then it goes to (RecivePacket). Otherwise, it goes to (SaveToQueue). In the (RecivePacket), if the receiver node is a high-tier node, the packet reaches its destination. The receiver high-tier node then employs (ShareSinkPositionIntheHighTier). The next hop function for (ReportSinkPositionMessage) is implemented on (SelectNextHop) which actually explained in the manuscript.

3.2 Obtain Sink Location

The source node that has a data employs (ObtainSinkFreshPositionMessage) object. It almost considered the two issued as in (ReportSinkPositionMessage), the high tier or the low tier node. If the source node is a high tier node, then it knows the location of the sink. It just go directly to (MergedPathsMessages). Otherwise, it generates an obtain packet by (GeneragtePacket) and sends it by (SendPacket). Note that before sending the packet, the sender node first selects the next hop node by (SelectNextHop) The packet is considered as received when it arrives to any high-tier node. once arrived, the high-tier node employs the response object (ResonseSinkPositionMessage).

3.3 Response Packet

The high-tier node which received the obtain (request) packet from a given source node employs the response mechanism which is implemented by the object (ResonseSinkPositionMessage). It first starts by generating a response packet (GeneragtePacket) that contains the new location of the sink. The next-hop node is implemented by (SelectNextHop) while both of sending and receiving the packet are implemented by (SendPacket) and (RecivePacket), respectively. The response packet is considered as delivered when it arrived at the source node that sent the request packet. After receiving the response packet, the source node starts preparing (PreparDataTransmission) for data disseminations (MergedPathsMessages).

3.4 Data Packet

After receiving a response packet, the source node starts by calling the (GetBranches) which returns a list of (List<Branch>). Then, the source employs the data routing mechsims implemented by (MergedPathsMessages) which takes three parameters as input, (Sensor currentBifSensor), (List<Branch> Branches) and (Packet packet). The source node generates a packet for each (Branch). The details of the branch should be saved at the packet. Data packet is generated, sent and received the employing the three functions (GeneragtePacket), (SendPacket), and (RecivePacket), respectively. The next hop is selected by (SelectNextHop). The data packets are disseminated towards the end point of the branch. When the packet arrived to the end of the branch, the sender node employs (HandOffToTheSinkOrRecovry) which hands the packet to the agent of the sink or recovery the if the sink has changed its location. The recovery mechanism is implemented by (RecoveryMessage) .

4. Simulation and Results Collection

4.1 Parameters Setting

From the main window, select the "Experiment" menu as arrowed in Fig1. Set the parameters as in Fig.2. Then click on "Start". Experiments of this work are concluded based on the network topologies shown in Fig.3.

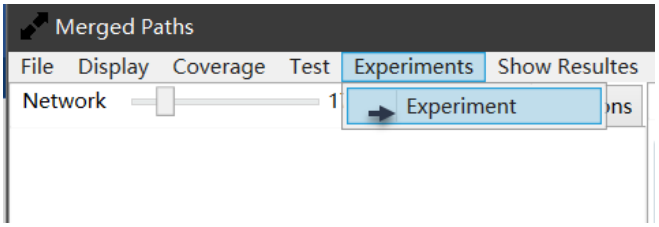

Fig.1: Main window

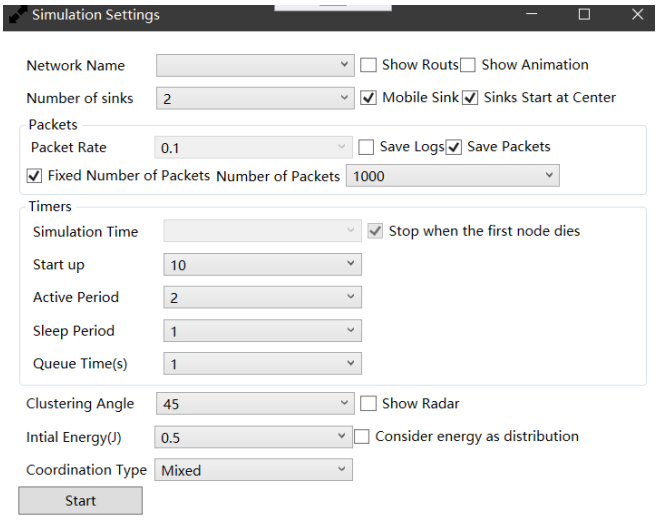

Fig.2: Parameters list

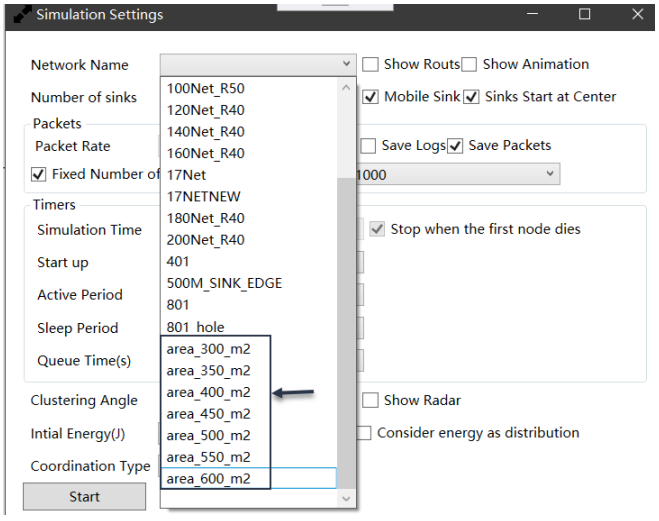

Fig.3: Network topologies

After clicking "**start**" button, the simulator starts running as shown in Fig.4.

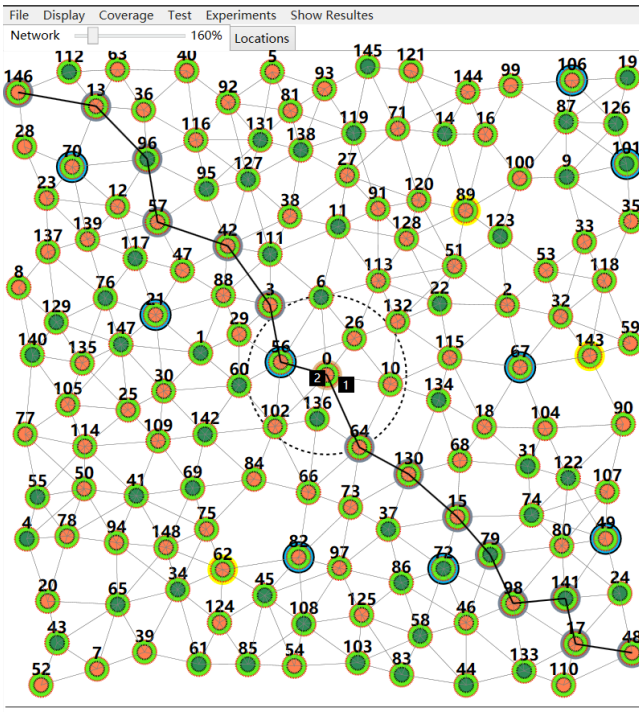

Fig.4: Simulation

The simulation results can be obtained by clicking on the menu "Show Results", as in Fig.5.

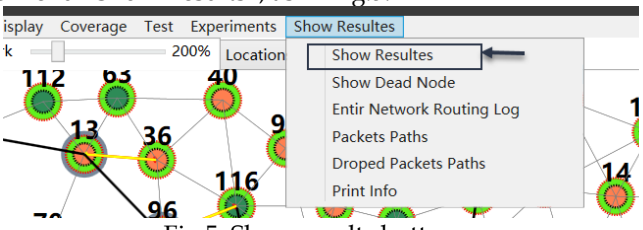

Fig.5: Show results button.

The results are listed in a data-grid as shown in Fig.6.

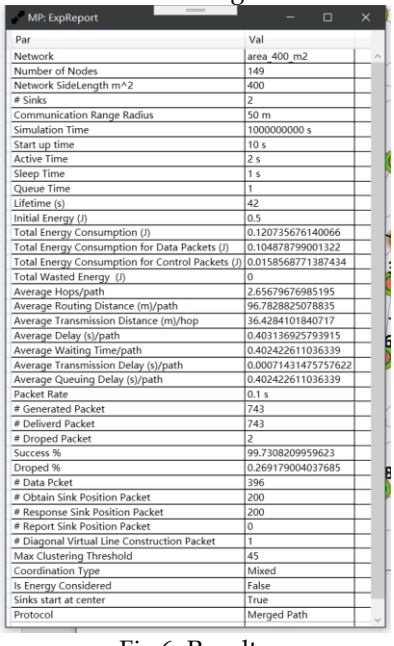

Fig.6: Results.

5. RING ROUTING AND

The source code for implementing the Ring Routing is available on the link, supporting multiple mobile sinks. The implementation of Ring Routing messages and packet forwarding mechanisms are in the link, which includes RingConstructionMessage , ShareSinkPosition-IntheHighTier, ResonseSinkPositionMessage, ReportSink-PositionMessage, ObtainSinkFreshPositionMessage, DataPacketMessages, and RecoveryMessage. The user is able to customize the setting via the "**simulation settings**" window from the menu bar "Experiments", as in Fig.7. The simulator will start running as in Fig.8.

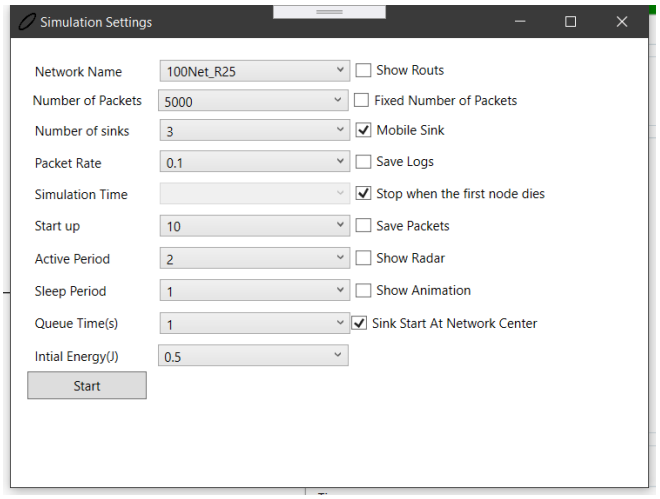

Fig.7: Ring Routing settings

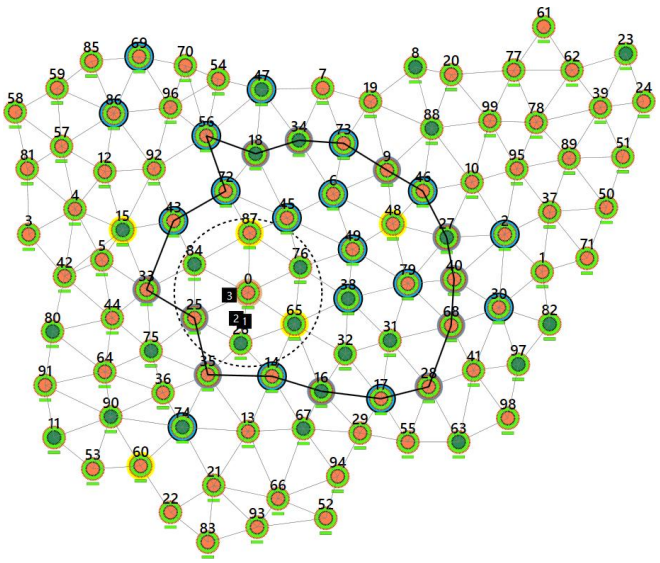

Fig.8: Ring Routing simulation

The source code for LBDD protocol is available on the link. The packets forwarding mechanism is implemented on the link. The user is able to customize the setting via the "**simulation settings**" window from the menu bar "Experiments", as in Fig.9.

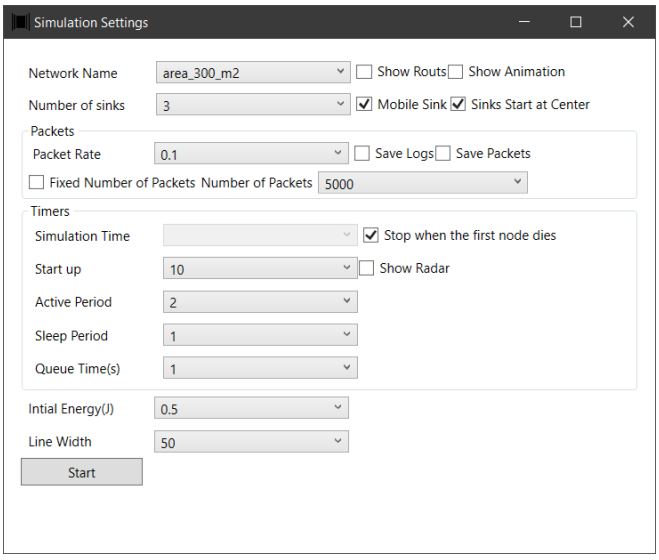

Fig.9: LBDD settings

The simulator will start running as in Fig.10.

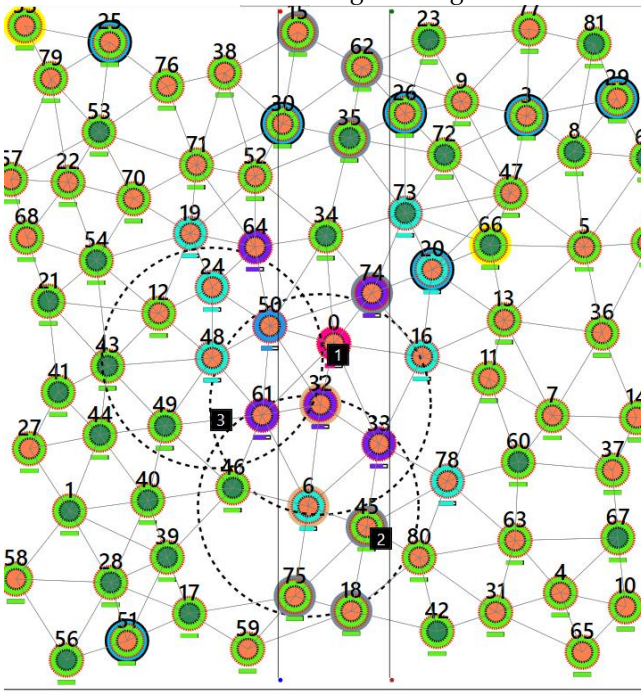

Fig.10: LBDD simulation

If the reader has any problem about the simulator, please contact our team via anmande@ustc.edu.cn