Extending Network Lifetime in Wireless Sensor Networks using Power-Aware Geographic Routing

by

Amalia Tsanaka

A thesis submitted to the
University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Electronic, Electrical and Computer Engineering
College of Engineering and Physical Sciences
The University of Birmingham
May 2012

Abstract

Wireless Sensor Networks (WSNs) use geographic routing to deliver measurement data to one or more sink nodes. In employing geographic routing, where data packets are forwarded along approximate geodesic (shortest) paths, WSNs can distribute the forwarding task unevenly between nodes. Nodes closest to the data sinks, or the geographic centre of the network coverage area, as well as nodes at the periphery of void areas, carry significantly more traffic. Consequently, their batteries deplete at a faster rate than the remaining nodes, causing void areas to form and grow, thus decreasing the overall network lifetime below its potential value. This thesis proposes a novel power-aware routing protocol inspired by the propagation of light-rays in graded-index media, aiming to balance the load over the network and to extend the overall network lifetime with minimal, local co-ordination overheads only. The idea has been implemented in a custom-built network simulator, where two versions of the algorithm were constructed: Curvy routing, which uses a fixed refractive index distribution and Energy Aware routing, which takes into account energy level changes to update the refractive index. Comparisons have been made between those, a baseline protocol and existing solutions in the literature, where simulation results have shown that the proposed protocols perform better in sparse networks, providing a realistic and plausible solution which can be applied successfully in real-life scenarios.

Acknowledgements

Well, this is it. I never thought this moment would come. Writing the final parts of this thesis, made me replay frame by frame all the steps to today. Happy days, sad days, excitement, frustration, confidence, challenge, agony, impatience, hope... It was a great journey and it had it all. One of these unique and unforgettable experiences in life which mark you as a person and reshape your idiosyncrasy. It requires strength, persistence, courage and without certain people it would never be completed.

Words are not enough to express my gratitude to my supervisor, Dr Costas Constantinou, for his guidance, support and advice during all this time. The knowledge and experience he has passed on to me throughout this research degree are simply invaluable.

A great part of this project belongs to my "academic siblings" in the Communications Engineering Group for the help, encouragement and motivation to keep working and make it through. Big thank you to David Arjona, Paul Kiddie, Hani Mortazavi, Yuri Nechayev, Keita Rose and Debra Topham.

Acknowledgements would be incomplete without mentioning my family; my brother, Stelios, and my parents, Βασίλειος and Ελένη. I am truly grateful for their psychological and financial support all these years, although they did not fully understand what I did, why I did it or whether it was actually worth it.

And since friends are the family you choose, I take pride in having loyal friends for life who always stand by me, encourage me, support me, help me, motivate me, are patient with me, believe in me, make me stronger and have shown love and care altruistically. Special thanks to Christiana Koundourou, Vaios Mprouziotis, Alexandros Plakidas, Katarina Rigova, Dimitris Sampanis, Gaby Silva, Aimilia Sisamperi, Nansy Stathaki, Sissy Stefanidou, Amalia Syrri and Georgia Tsiantoula.

Finally, I feel the need to thank my partner, Joseph Bottomley, who is always there for me and understands me like nobody else has done or will ever do. I am lucky to have someone like you, because you are a diamond. You saw and accepted me at my worst and you definitely deserve my best!

... And because (unfortunately) money always matters, I have to thank EPSRC for paying my tuition fees.

Contents

List of Figures

List of Tables

Abbreviations

2D	Two Dimensions
3D	Three Dimensions
DCE	Data Combining Entity
ID	Identification
MAC	Medium Access Control
QoS	Quality of Service
SVD	Singular Value Decomposition
WSN(s)	Wireless Sensor Network(s)

CHAPTER 1

Introduction

1.1 Structure and Applications of WSNs

Wireless Sensor Networks (WSNs) consist of a number of wireless devices, called sensor nodes, which produce a measurable response signal to a change in an environmental condition. These include temperature, light, sound, pressure, humidity, motion and vibration; these signals are transmitted wirelessly to one or more base stations (sink nodes), which make decisions based on those measurements. In a WSN, sensor nodes are usually scattered in a known area and programmed in such a way to forward gathered data to the designated sink node(s).

WSNs have a lot of applications and cover many different aspects of human needs. They can be classified (amongst others) into environmental, health and military applications (Perkins, 2001).

- Environmental applications include weather forecasting, fire/flood detection, habitat monitoring, forest surveillance, air pollution, traffic control, even industrial sensing to prevent machine failure.
- Health applications of WSNs are of vital importance, due to the purpose they are built to serve. There exist sensors that monitor blood pressure, body temperature, cardiac activity and in combination with tracking abilities, they can provide constant remote supervision of patients.
- Military applications are aimed for security maintenance by monitoring enemy forces, detecting potential nuclear attacks (sensing toxic elements), targeting, secure communication, surveillance.

1.2 Geographic Routing

Geographic routing is broadly used for communication between sensor nodes, due to the nature of WSNs; since nodes are placed in a known area, information about position is available and easily retrievable. Geographic routing algorithms can provide a scalable solution for WSNs, reducing data overhead and rapidly adapting to changes in the network topology without requiring additional memory. Location information is exploited to achieve a resource-saving infrastructure, since only the neighbour and the destination nodes' location are sufficient to carry out packet-forwarding tasks (Stojmenovic, 2002).

Greedy (shortest-path) routing algorithms rely entirely on location information and base the next-hop decision on the Euclidean distance calculated from a neighbour to the destination, choosing the one closest to the sink. For dense inhomogeneous WSNs, clustering (Perkins, 2001) has been utilised to manage routing decisions, where the network is divided into clusters by location and data is passed initially to each clusterhead and onto the destination afterwards.

More sophisticated methods of routing have been deployed recently in an attempt to exploit the advantages of WSNs and extend their performance. Calculating the cost each transmission takes and comparing it with adjacent nodes in the vicinity, more efficient routing can be achieved by choosing the less costly path. These techniques have been incorporated to existing (Greedy) routing algorithms, leading to the so-called power- or energy-aware routing algorithms (Stojmenovic, 2002).

1.3 Research Challenges

Although WSNs have many applications and a lot of potential, their practical implementation is held back by the fact that, due to the size of the sensor nodes, they have very limited memory, processing and battery lifetime capabilities (Buratti,

Conti, Dardari & Verdone, 2009). It is, thus, essential to investigate methods for energy conservation during path discovery.

Simple stateless routing is commonly employed, in the form of geographic or position-based routing, to deliver sensor measurement data to one or more sink nodes. In employing geographic routing, WSNs distribute the forwarding task unevenly between nodes, where data packets are forwarded along approximate geodesic (shortest) paths. Therefore, nodes closest to the data sinks or the geographic centre of the network coverage area, and nodes at the periphery of void areas carry significantly more traffic. Consequently, their batteries deplete at a faster rate than the remaining nodes, causing void areas to form and grow, thus forcing the network to stay functional for shorter period of time. By having an even distribution of the paths over the network, it is believed that partition can be prevented or at least postponed and as a result, most of the nodes exhaust their batteries at about the same time, achieving maximum network lifetime. Therefore, extending the network lifetime by definition (here) implies delaying the time it takes for the network to partition.

This study presents a proposal for an energy-depletion sensitive routing algorithm, which is a modification of existing geographic routing techniques. The objective of this investigation is to explore the limits of energy-aware routing algorithms in terms of network characteristics, to experiment with changes in the topology and to identify the circumstances under which the proposed algorithm achieves maximum performance.

1.4 Contributions

The outcome of this study involves, most importantly, a novel solution to the energy management research question. A theoretical model was constructed and expressed mathematically to reflect the algorithm's behaviour, which is inspired by nature and

based on the theory of geometrical optics. Programming code was, then, created in order to represent the theory adopted and to aid in simulation implementation.

Two different versions of the proposed algorithm for geographic routing have been constructed to support the presented solution, while comparisons with existing routing techniques and a baseline protocol are performed to verify the algorithm's performance. Curvy routing uses a bulk network distribution to bend the paths away from the geographical centre of the network, while Energy Aware routing uses periodic energy level updates for task forwarding, avoiding not only central, but also low-energy areas.

Finally, the development of a custom-built network simulator completes the proposal and comprises a software that can be utilised for protocol evaluation and to perform a large range of simulations and which can be configured accordingly, because of its functional structure and class organisation.

1.5 Outline of Thesis

The rest of the thesis is organised as follows:

- Critical review is undertaken of existing energy-aware solutions proposed in the literature in Chapter 2, outlining their objectives. The assumptions made by these proposals are investigated and the methods used in order to achieve those objectives are discussed. Alternative solutions are also mentioned and reasons why they are not practically efficient are discussed. The existing literature is categorised by method, starting from global view to local view ones and narrowing down to energy-aware proposals.
- In Chapter 3, the proposed solution is presented and the methodology used is classified into the two different versions of the algorithm. The theory of geometrical optics geodesics in graded refractive index media is summarised, followed by the cost function estimation and further calculations essential for next-hop selection.

- The custom-built network simulator is outlined in Chapter 4, describing all its components, functions and features. Protocol evaluation is performed through a baseline scenario construction and the metrics used for results comparison are explained. Limitations and difficulties met during implementation are also discussed.
- Chapter 5 shows simulation results of the first version of the proposed algorithm, called Curvy routing, against the baseline Greedy protocol. Several scenarios are included to represent its advantage in comparison to Greedy routing and another existing routing technique called Curveball routing (Popa et al., 2007).
- Performance analysis continues in Chapter 6, where the more advanced version of the algorithm, Energy Aware routing, is compared versus both Curvy routing and Greedy routing. Experimentation with many network aspects (network density, topology, size, sink placement, transmission range) shows the supremacy of a more sophisticated algorithm and reflects its ability to adapt during rapid topological changes.
- Finally, in Chapter 7 conclusions are drawn based on the results outcome and the comparison analysis, followed by contributions of this study. Observations regarding network attributes' alterations are discussed and future work, to enhance the existing algorithm, is considered.

CHAPTER 2

Literature Review

There exist a number of proposals that address the energy efficient operation of WSNs, discussed in Section 1.3, but these only offer a partial solution to the above stated problem. The most relevant amongst these proposals were selected and classified using criteria based on the amount and type of information available for routing strategies.

Before discussing in detail the literature, an introduction to the concepts of geographic/greedy routing and cost function is necessary. Stojmenovic (2002) presents an overview of position-based routing and the advantages of locally exchanging information. It is claimed that using location information to forward data across the network, provides a scalable solution for WSNs and it has been confirmed experimentally by Stojmenovic (2002) that protocols not using geographic location in the routing decision are not scalable. Simulation results in Stojmenovic (2002) have also shown that when simple geographic forwarding is deployed, more packets are delivered while fewer network resources are consumed. Additionally, when forwarding decisions rely on routing tables, the size of these tables is logarithmic versus linear in the presence of location information. Greedy routing is used to provide simple and effective next-hop selection, as it only needs knowledge of the one-hop neighbours' position and the position of the destination. It calculates the distance from all the neighbours to the destination and forwards the data to the one which makes maximum progress (closest to the destination or most further away from the source). The metric used in simulations is usually hop count. Figure 2.1 depicts the method used to select the next hop. S being the source and D the destination, S chooses M over G amongst its neighbours, because M is closer to D. The selected path is SMUVD.

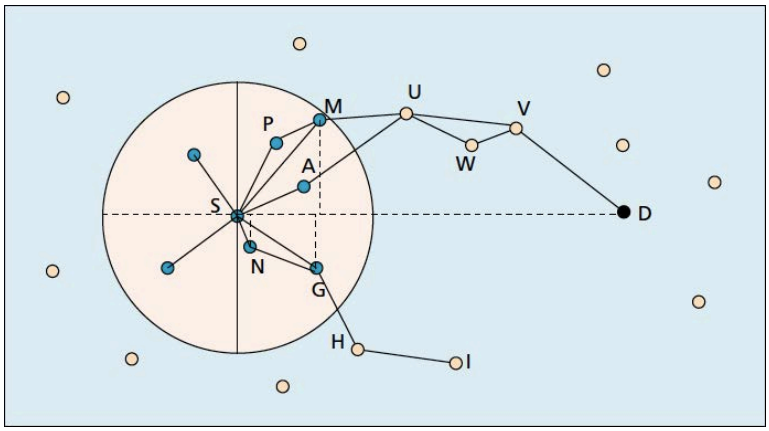


Figure 2.1: Greedy routing forwarding path (Stojmenovic, 2002)

A cost function is a formula or a series of combined formulae which satisfy the proposed solution and agree with the theory it is based on. Basically, a mathematical expression of a solid theorem, modified and adjusted to the assumptions of the method to be used, including the parameter(s) chosen to experiment with and which affect the performance of the proposal (i.e. the cost). Once this function is determined, the goal is clearly to identify the tradeoff linking the factors which minimise it. More practically, after determining mathematically the relation between those parameters, they are altered empirically in the simulation in order to achieve the desired optimal outcome.

2.1 Taxonomy of Methods

The taxonomy of the methods used in the approaches mentioned earlier is depicted in Figure 2.2, where the first level categories contain the global view proposals (discussed in Section 2.2) and the local view ones (cf. Section 2.3). These terms refer to whether the knowledge used initially (or needed to make a forwarding decision) is limited between small areas of the topology, or whether information about the entire network is required by the routing algorithm. The second level

categories account for further use of available information towards routing choices and management of performance parameters. Global view approaches involve path computation and proactive route discovery (Section 2.2.1), as well as necessary cognition to perform clustering (Section 2.2.2). Local view approaches are divided into those where data is only shared between one-hop neighbours (Section 2.3.2) and those where neighbourhoods are expanded to two or more hops (Section 2.3.1). Focusing on the former, further categories exist to represent the type of information that is used. There are approaches which base their routing decisions on traffic demand and others, including the one proposed here, which perform task forwarding by employing residual node energy data. As the basis of the proposed protocol exploits an analogy with ray-optical propagation in geometrical optics and employs concepts from the calculus of variations, the review of the relevant literature that addresses similar problems, based on ideas from physics, is also highlighted (Section 2.3).

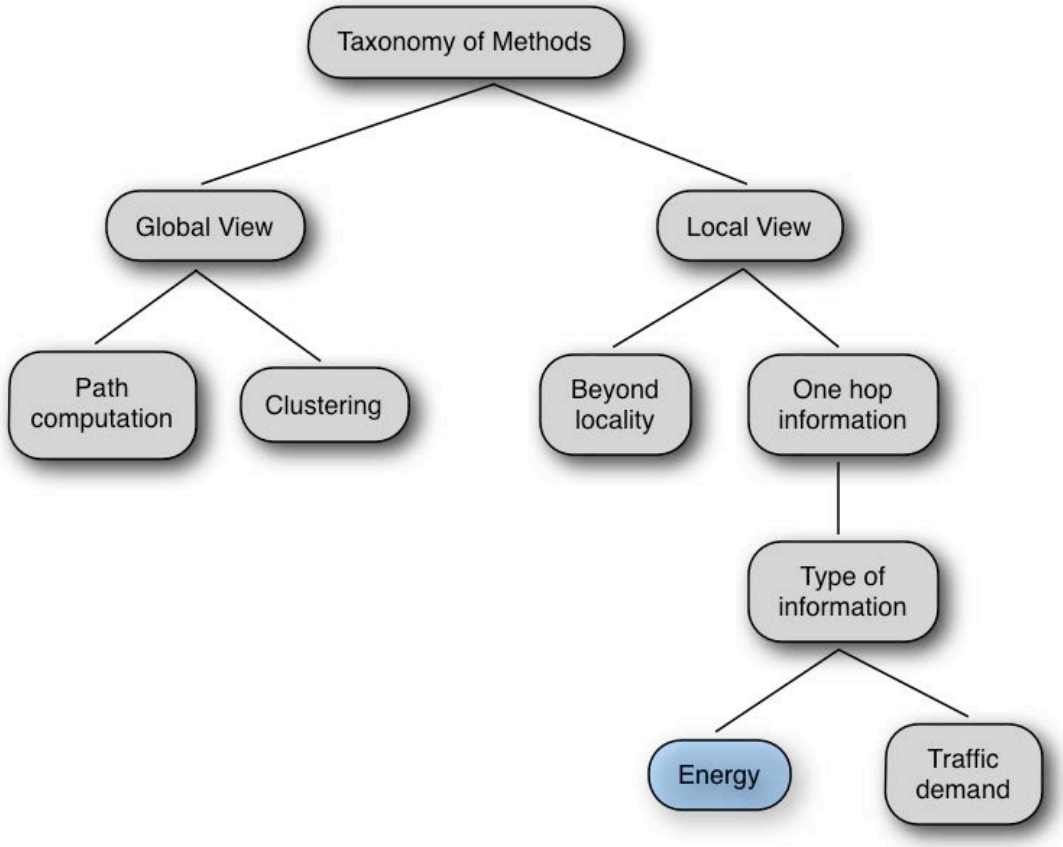


Figure 2.2: Taxonomy of methods in literature

The literature presented below has the form of initially stating the objectives of the proposal, discussing the assumptions made under which the problem is approached, explaining the concepts it is based on and finally presenting the method of evaluation chosen, as well as commenting on the usefulness of the findings and whether they have managed to fulfil the primary expectations.

2.2 Global View

Methods needing global information in order to perform routing, whilst providing a solution to the uneven energy consumption issue, are discussed below. A few distinctive approaches have been selected from the literature; multiple path calculation, proactive and table-driven protocols. Cases of clustering have also been included, since global information is initially used to decide on ways to divide the topology and select cluster heads. Since the global view approaches are not directly relevant to the proposed solution in this thesis, they will only be mentioned briefly as a reference to their energy management technique, in order to provide an integrated overview of the literature.

2.2.1 Path Computation

The performance of a customised Energy Aware algorithm is presented by Vidhyapriva & Vanathi (2007) to provide a reliable transmission environment with low energy consumption. The algorithm proposed has been influenced by the fact that it is compared to on-demand routing protocols, thus *a priori* known information is required to estimate signal strength and to discover multiple routes. The best possible route to destination is identified by utilising the energy availability and the

received signal strength of the nodes. During the initial broadcast, the intermediate nodes indicate whether they have sufficient energy levels to participate in the reply process. Power conservation is achieved by forcing nodes not being selected back to sleep mode, since during sleeping their receiver/transmitter is switched off.

An approach closer to reality is taken by Le, Johns & Pister (2009) to address the limited energy issue in WSNs. A method to keep the nodes alive until the next scheduled maintenance is proposed, utilising an adaptive energy-slope control. This combines three different power sources in WSNs (battery-powered, energy scavengers and AC powered) to make sure all nodes are alive for a required period of time in a home/building automation. As soon as the charge drops below that threshold, the energy metric will increase the cost of the route and these nodes will enter the sleeping mode until their charge becomes above the threshold line over time, and they become eligible for being used for routing again. The nodes having energy scavengers are charged to reach the current threshold, in order to keep (this way) the network connected for longer. Routing tables are used and updated periodically to keep a balance between energy level changes, network connectivity and excessive overhead.

The same technique described by Le, Johns & Pister (2009) is used by Rabaey & Shah (2002) in a different network setup with PicoNodes with the aim to find the minimum energy path to optimise energy usage at a node. An office environment is monitored by 76 nodes capable of sensing, controlling and actuating. The algorithm used is characterised by a reactive behaviour in order to cope with mobility and destination-initiated requests. Pre-calculated routes are used for routing and they are stored in the forwarding table, which is constructed using the cost calculations. The probability assigned to each node, after the construction of the forwarding table, is inversely proportional to the cost. It is reported that the overhead is significantly low, although transmission delays are not disclosed due to the routing method being used; calculation of the cost of multiple possible paths and randomly choosing the less costly one is known to be time and memory intensive.

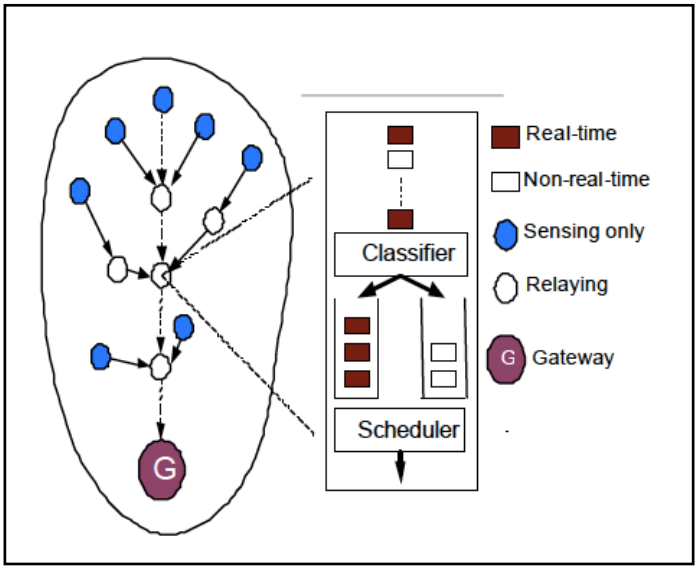


Figure 2.3: Queueing model (Akkaya & Younis, 2005)

Quality of Service (QoS) with energy concerns is the main focus of Akkaya & Younis (2005). The aim is to find the optimal path in terms of energy consumption and error rate, while taking into account end-to-end delay requirements. Additionally, emphasis is given to maximise the throughput for non real-time traffic and deal with the overhead created by maintaining both the node states at each sensor and the multiple paths to the sink. The protocol employs a queuing model and calculates end-to-end delay and link costs using the distance between nodes, current residual energy of each node and error rate on the link between the nodes. Routing is performed by utilising a cost function based algorithm for each link to find a set of candidate routes, which are checked against end-to-end constraints and the one yielding to maximum throughput is chosen. The queueing model (Figure 2.3) designed considers the bandwidth ratio to determine the order of the packets to be transmitted.

2.2.2 Clustering

Cluster based routing is a form of hierarchical routing used in many types of wireless networks. Global knowledge of the network geography is necessary in order to decide the manner in which the area is divided. Depending on the scope of the routing protocol, the network is split into sub-networks and one of the nodes in each sub-network becomes the cluster head, where all the data is passed onto. Only the cluster heads of each sub-network can communicate with each other and exchange information; the rest of the nodes communicate between one another within the same sub-network (or cluster) and, of course, with their cluster head. Because of all these properties, clustering has the advantage of topology control and management of data flow. Figure 2.4 shows an example of clustering hierarchy, where normal nodes are represented in grey and intra-level cluster heads in black. Inter-level cluster heads are square.

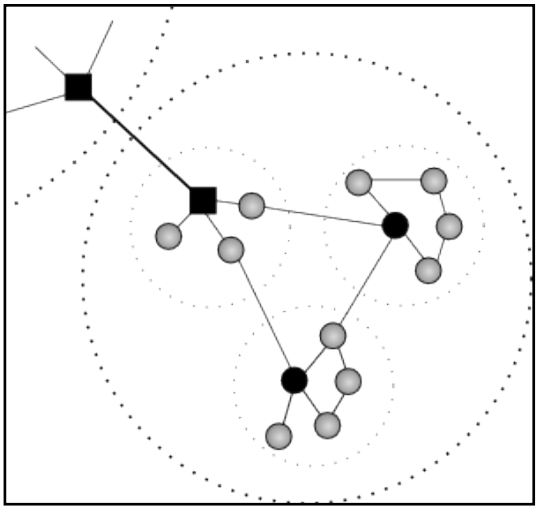


Figure 2.4: Clustering hierarchy (Langkemper, 2006)

Cases of clustering such as the one presented by Yang, Li, Sawhney & Wang (2009) are popular in trying to solve the power management problem. The algorithm proposed is of a hybrid nature and is based on the geographic layout of the network, which is laid out into two layers. The base station and the cluster

heads belong to the top layer, while the rest of the sensor nodes belong to the lower layer, as shown in Figure 2.5. The target region is divided into sub-regions and the nodes are grouped together with a number of sensors in each of them. The groups created, then, are randomly distributed to the sub-regions. The energy model used typically involves the distance between two nodes and the energy expenditure per transmission unit. The metric used for routing considers as the link cost both the maximum and minimum residual energy and the minimum and maximum communication energy consumption. The path with the lowest energy consumption is chosen and recalculated after every broadcast.

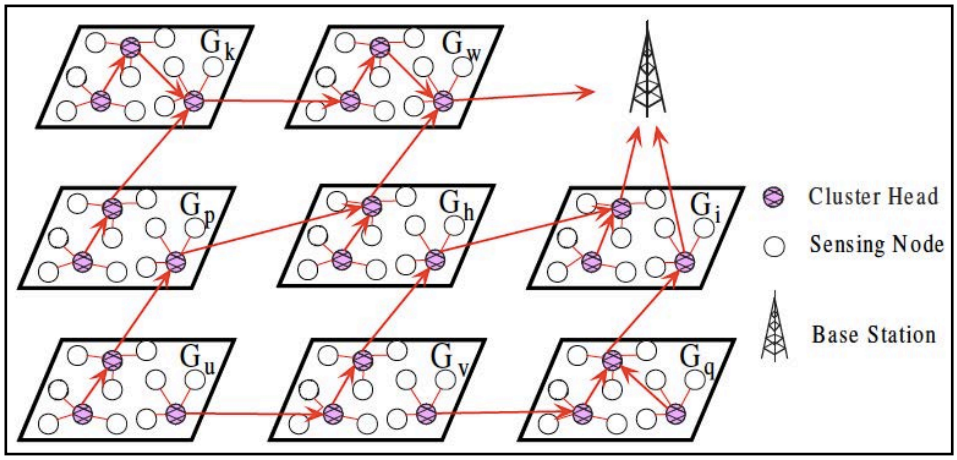


Figure 2.5: Two-layer sensor network (Yang, Li, Sawhney & Wang, 2009)

Optimum parameters such as one-hop distance and clustering angles are investigated by Min, Wei-ren, Chang-jiang & Ying (2010) to reduce energy consumption of the way clusters are formed and updated between inter- and intra-cluster layers. The frequency of updating cluster heads and the energy consumption of establishing new ones can be decreased by using the proposed mechanism. Clustering is done by separating the uniform density area into wedge shapes (V) of a clustering angle and then dividing each one of them into rings starting from the base station, where the distance between the rings is defined as inter-cluster one-hop communication, as shown in Figure 2.6. The clustering angle is tuned empirically to define the size of each cluster in order to control the inter-cluster

communication, the data flow and, hence, the energy consumption. The base station is informed of the cluster heads' location during setup and decides which one acts as the first, and then via advertisement messages each node decides which cluster it belongs to. The current cluster head receives information from the rest of the members and the lower layer cluster head with the same clustering angle (Figure 2.6), while it sends the fused data to the upper layer cluster head. Energy conservation is achieved by working with different time slots and switching off the radio of each node until its allocated transmission time.

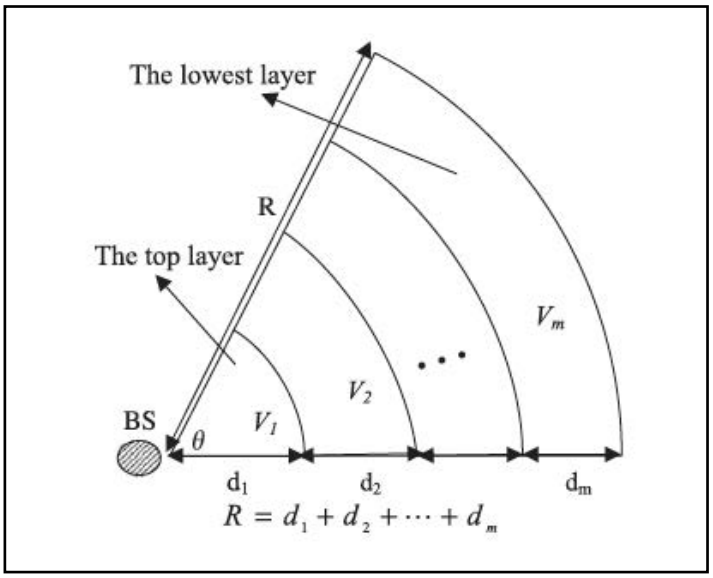


Figure 2.6: Layers of clustering (Min, Wei-ren, Chang-jiang & Ying, 2010)

Focus is given on the optimal number of clusters to maximise overall lifetime in heterogeneous networks including some more powerful overlay sensors (Duarte-Melo & Liu, 2002). The energy consumption is examined based on the clustering mechanism with sensing field parameters. The goals are to either maximise the lifetime of such a network for a given amount of energy, or to retrieve the same data using the least amount of energy. The lifetime is measured in rounds, where data collection is performed periodically. The area is assumed to be a square and the density is controlled by adjusting the sides of the area. Uniform distribution is also assumed and the overlay sensors are randomly placed across the field, where they

become cluster heads in turns. The rest of the nodes select which cluster head to send data to, based on signal strength from the broadcast; the strongest the signal the closest to the cluster head. Overlay sensors that decide not to be cluster heads are set to sleep mode to reserve energy, which is helpful in identifying the average number of clusters in the network. The energy model adopted contains amounts of energy spent in transmission, reception and sensing, and is set to be equal for all three (50 nJ). Collecting all this information, the maximum lifetime in simulation rounds for a given initial amount of energy per sensor is calculated and compared between the scenarios.

Although the above techniques present some advantages and produce favourable results, the tradeoff of utilising such methods is not to be overlooked. Sharing information globally adds to the overall overhead and causes serious transmission delays, as well as meets difficulties in coping with rapid and frequent changes in the topology. Dense networks can benefit from global knowledge dissemination, since it is extremely accurate and convenient to use for management, but local view techniques below will show why detailed information is not vitally necessary for efficient routing.

2.3 Local View

Methods limited to local information and relying on topologically restricted data for routing decisions are outlined below. The first group of methods mentioned utilise both local and extended topological information in an attempt to achieve a more balanced solution, where forwarding decisions are made strictly locally. Approaches taking into account network traffic are, then, highlighted and separated from those using residual energy levels in order to extend the network lifetime.

2.3.1 Beyond Locality

A scheme on "optical" routing is proposed by Catanuto, Morabito & Toumpis, (2006) based on dynamic programming applicable to WSNs with very large numbers of nodes only. The scalability issue in such networks is highlighted, where routes become longer and hence harder to discover and maintain, due to the increased number of hops per transmission and the additional overhead present. Because of the large size of the network, a microscopic view does not provide enough information for accurate routing and, additionally, a macroscopic (bird's eye) view is also necessary. The position of the nodes is not important either, since there are so many of them. Instead, node density is measured in nodes per square meter and the cost function is calculated globally, rather than by individual nodes. The whole network is considered as a heterogeneous medium whose refractive index is analogous to the cost function, the concept of which was introduced earlier. It consists of the incremental cost c of transmitting the packet over a distance e at a given location r ($c = \delta c(r)/e$). This cost function can be modified according to network metrics to be optimised using equations borrowed from the theory of light ray propagation, e.g. bandwidth utilisation, energy minimisation in transporting a packet, etc. The differential equation for optimisation is solved locally for each node and the packet is transmitted forward without determining the exact route. If two nodes propagate route request packets (to their neighbours), the rays (routes) created intersect with each other. Intermediate nodes placed at these intersections, realise that the initial nodes want to communicate and they inform each end-node of one another's location. Optimal routes can be chosen by either requesting, creating and using additional rays or by using the existing ones and choosing between multiple segments at every intersection. Additional rays are also launched during routing to cover the areas of the network avoided due to high refractive index values. Dynamic programming (Cormen, Leiserson, Rivest & Stein, 2001) is used to overcome the cumulative cost changing over time and depending on the present

state. Based on Fermat's principle (Marchard, 1978), a local minimum can be achieved by any ray between a source and a destination node. With the aid of dynamic programming, the geometrical optics (eikonal) equation (Paris & Hurd, 1969) can be minimised. Simulation results are not available and the only comparison/analogy provided is between equations of dynamic programming, shortest path and optimal routing in terms of current and candidate state/node/position, minimum cumulative cost from a node to the destination and additional communication cost of a transmission between two intermediate nodes.

Further work (Catanuto, Toumpis & Morabito, 2007) takes the cost function to a more detailed level, adjusting it to Simple and Distributing routing in massively dense networks with a setup of 5000 nodes in a 100m x 200m area. Questions arising from the macroscopic view with regard to finding the minimum cost route for two known locations (Simple routing) and when the end-point of the route is unknown (Distributing routing) are attempted to be addressed. In the microscopic view, bandwidth and energy limited networks are examined and forwarding rules are made based on the analogy with geometrical optics. A Poisson distribution (Diestel, 2005) is adopted for node deployment and the per-hop transmission cost is set to the hop distance squared. The forwarding rule selects the closest node to the destination with the minimum transmission cost c. Balance between small transmission cost and sufficient progress toward the destination is attempted to be achieved. The cost function is, then, the ratio one over the square root of the Poisson spatial distribution density. For energy limited networks, the cost function C is a sum of the energy consumed by the transmitter power amplifier (depending on the distance d) and the energy consumed in the electronics of the transmitter and receiver c ($C = d^b + c$), while the exponent b involved of the first term depends on the environment and varies between the values of 2 and 6. Simulation results are not shown in the paper due to lack of space and because the intention was to provide proof of concept and to elucidate the theory.

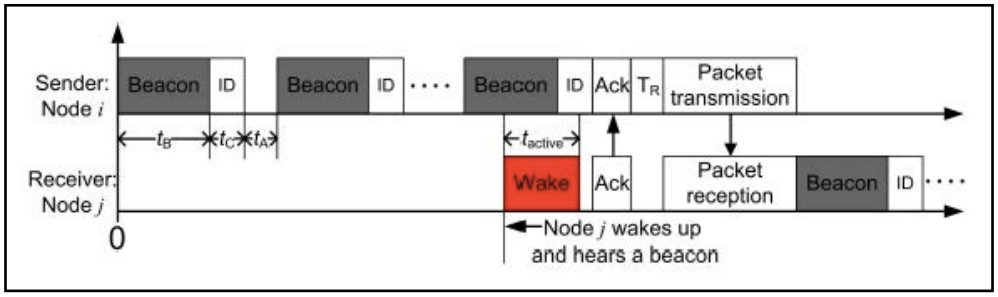


Figure 2.7: System model (Kim, Lin, Shroff & Sinha, 2010)

A numerical approach is used by Kim, Lin, Shroff & Sinha (2010) with the objective, primarily to minimise the delay in sensor networks and secondarily to maximise the lifetime, by solving the joint control problem of sleep-wake scheduling and anycast packet forwarding. The wake-up rate is adopted from a Poisson process (Diestel, 2005) with which each node wakes up and the awake probability of a node is a function of the wake-up rate, the beacon signal and the ID signal (Figure 2.7). The performance metrics used are end-to-end delay (time when an event occurs until the first packet of this event is received by the sink) and network lifetime (defined as the energy available to a node over the units of energy it consumes each time it wakes up times the wake-up rate). Minimisation of the delay is achieved by choosing the way anycast policy is performed. A local delay relationship is derived as the sum of the one-hop delay and the expected delay from the next-hop node to the sink. The algorithm, then, uses the local delay relationship to derive an optimal forwarding node set. Global information is derived to develop a second algorithm for computing the optimal anycast policy, where each node uses the delay values of neighbouring nodes from the previous iteration to update the current forwarding set and the current priority assignment. The problem of the network lifetime maximisation is solved with the aid of the global optimisation algorithm. Initially, the sink sets the optimal lifetime value to half of its possible maximum value, the algorithm is run and it finds the optimal anycast policy and the optimal delay value and returns the maximum number of possible iterations.

Extended topological information was considered to be adopted in the proposed solution and the tradeoff of using it was examined all over again, but simulations showed that, in sparse networks, an extended view beyond a node's immediate locality does not make significant difference during decision making. The overhead and the delay added, therefore, are not worth sacrificing and one-hop information is sufficient to perform geographic routing, while information on the rest of the topology can be approximated.

2.3.2 One-hop Information

Coming to the last branch of Figure 2.2, forwarding decisions are exclusively based on information available within the boundaries of the neighbourhood. The proposals summarised below examine the way traffic information can be used in favour of power conservation and load balancing in the network, followed by proposals where the main focus is shifted to the utilisation of energy depletion information to prolong the network lifetime.

2.3.2.1 Traffic Demand

Schurgers & Srivastava (2001) argue that optimal routing in large-scale sensor networks is infeasible due to the limited energy supply in each node and propose a gradient-based solution which aims to minimise the energy consumption of transmissions and exploit the multi-hop aspect of network communications. The methods introduced are aggregation of packet streams in a robust way and shaping the traffic flow for uniform resource utilisation. Routing is performed for energy efficiency instead of energy optimisation, where information from the past and the present is required to forward traffic. As the main purpose is to maximise the network's functionality, the metric used is lifetime, defined as the time until the first

node's energy is exhausted. Basic routing techniques are based on the fact that traffic should be spread evenly over the topology and all nodes should carry the same degree of importance. Flooding the type of information a user is interested in, gradients are established in each node and this way, the minimum number of hops to the user (called a node's height) can be recorded. Nodes in the same vicinity having multiple traffic streams, can create a Data Combining Entity (DCE) to take care of data compaction, instead of selecting a cluster head to combine each cluster's data. In the second technique, a node can increase its height in order to avoid getting selected in case its energy reserve has dropped below a threshold (energy-based scheme). Diverting new streams away from the nodes that are part of another path will not affect the original stream since those nodes have not updated their height, but will cause the creation of other streams toward the destination (stream-based scheme). Simulation results compare the two schemes (energy-based and stream-based) with the standard version initially introduced. There is a slight improvement in terms of energy consumption which favours the stream-based scheme, although it does not always increase the functionality.

Kirchhoff's Voltage Law is used by Toumpis & Gitzenis (2009) to balance the network load by monitoring traffic of nearby links and modifying it along cycles. The goal of the proposal is to find the traffic flow that minimises the cost function introduced below. The cost optimisation of a link between two nodes is discussed and motivated by the nature of the wireless channel: the communication is more costly in terms of delay, energy and bandwidth due to interference from the neighbouring links. The total communication cost function is introduced and is subject to the distance from the neighbours and the individual link costs between the node and each neighbour ($cost_T = cost_{ij} - cost_{ji}$). The network is parallelised as an electric circuit where each node is a current source that brings current into the network equal to the node divergence, and each link between two nodes is an electric element that exhibits a voltage drop when a current intensity flows through it. Figure 2.8 depicts the optimal flow in a square grid wireless network with 40 sources and 2 sinks (dashed boxes). The load balancing algorithm consists of two

phases: A set of cycles is constructed first and the nodes that belong to each cycle in the span calculate its cost collectively. If the cost is positive, the nodes insert a small circulation in the opposite direction of the set of cycles and if the cost is negative the direction is the same. Differential costs of neighbouring cycles are affected by inserting circulations and thus the second phase is applied repeatedly until the cost is increased. The cycle cost is calculated as the flow moves continuously along the cycles and partial derivatives for each link are added to it.

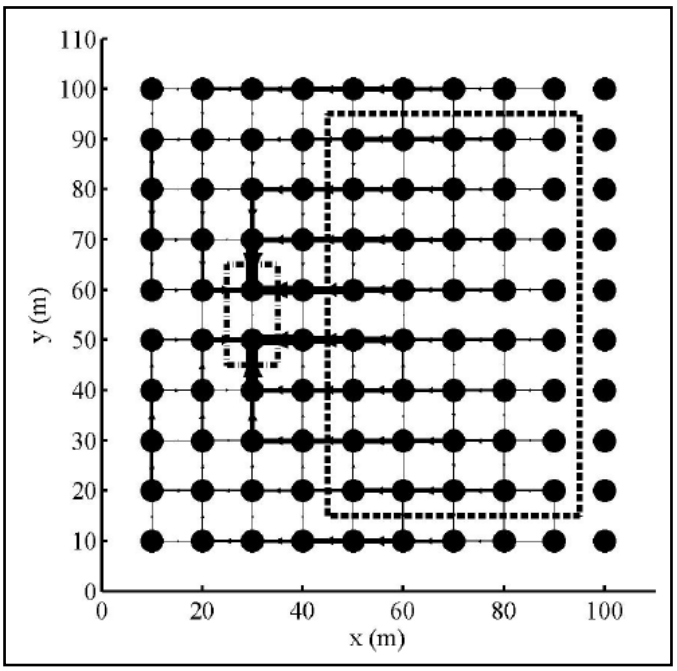


Figure 2.8: Traffic flow (Toumpis & Gitzenis, 2009)

The problem Jacquet (2004) is trying to address is the shape of the shortest path when the traffic density is not uniform. A theoretical approach is presented showing that a variable traffic density impacts the curvature of paths in a dense wireless ad hoc network the same way a variable optical density bends light paths. A Poisson process (Diestel, 2005) is used to set up general rules that paths must follow in the presence of traffic flow density and construct, this way, a global propagation model. Massively dense networks are assumed with the equation of the optimal path to be given by a function of the traffic density at a point, the number and length of hops

passing in the vicinity. Looking for the path with minimum number of hops, is equivalent to looking for the light ray in a medium with non-uniform optical index. Given the fact that the distribution of the paths impacts the traffic density, a flow density factor is introduced to represent the flow of the packets at a one metre segment centred at the node and is measured in bits/metre. Simulation results have not been generated and, generally, this is a purely theoretical proof of how the traffic flow is affected and there is no numerical evaluation of the routing performance to support the conclusions drawn.

Traffic demand is tremendously relevant to the routing algorithm proposed in this study and traffic information has been considered to be incorporated into the data forwarding task. The idea has not been rejected and reasons why it is not part of this proposal are explained in Chapter 7.3, where it has been left as a future implementation.

2.3.2.2 Energy

Although the solutions described previously presented some interesting simulation results, they lack sophistication over the nature of the network. Analysing the local energy information at given time slots and using it towards updating and optimising the cost function, more efficient routing can be achieved in terms of network lifetime and load balancing. The first proposal analysed below offers an integrated solution, simulating and presenting results in three different environments (Popa, et al., 2007). This algorithm has been used as a benchmark for comparisons to the work of this thesis, as it has been reproduced and simulation results are presented in Section 5.2 (referred as Curveball routing). The second proposal is based on the same routing technique, but uses a different way of mapping the network.

The approach of Popa, et al. (2007) is based on geometrical optics and aims to address the problem of balancing the traffic load in wireless networks. Geographical routing (Stojmenovic, 2002) is assumed to be used to cope with scalability and overhead, and it is stated that the maximum load can be decreased if the packets followed curved paths. The crowded centre effect is explained with the load probability of central nodes being higher than the rest (as well as their energy consumption) and is calculated with respect to a first assumption, namely that the shape of the network being a disc. Neighbours within a communication range are determined, as the location of the nodes and their neighbours is assumed known. Greedy routing (Cormen, Leiserson, Rivest & Stein, 2001) is applied where the closest to the destination neighbour is selected and enhancements such as perimeter routing are left aside. The network is viewed as a continuous space instead of individuals points due to the assumed high number of uniformly deployed nodes. Simulations of random source-destination pairs are performed to monitor the behaviour of this model and more specifically to calculate the load probability density of nodes. This is shown to be a function proportional to the square difference of the radius of the network and the distance from the node to the centre of the disc. The cost function is approximated by an algorithm and the cost of the link joining two nodes is the product of its length and the average cost of its end points. Using linear programming, the load of each node is estimated and equalised by multiple iterations, until all nodes have a load of common value. Polynomial interpolation is, then, used to determine the cost function. Exact trajectories cannot be calculated during routing, but the absolute direction of the path can be known using differential equations from the Calculus of Variations and the neighbour which deviates the least from this direction, is chosen. The above routing method is said to be causing excessive overhead, as large pre-computed tables and numerical integrals are required and its only purpose was to show that the minimum maximum load can be approximated numerically to match the discrete set of node costs. The alternative solution proposed is called Curveball routing and involves mapping a disc-shaped network on a sphere, as illustrated in Figure 2.9, and applies Greedy

routing using the virtual mapped coordinates obtained. A projection point is selected on the sphere (the pole, for better results) and the virtual position of the nodes is computed and included into the routing algorithm. Greedy routing is performed in the new spherical coordinates and the forwarding task yields the most geometric progress toward the destination. In theory, when 3D routing fails, the method falls back to 2D routing, but as shown in the presented results this was found never to be the case due to the large number of nodes used. A discussion of mapping limitations and other heuristics is also involved with a description of unsatisfactory attempts to modify density (by using a factor to transform the node coordinates and compensate for the load imbalance) and to provide ring routes (the same way congestion is avoided using ring roads in large cities). Due to the selected way of mapping, the network is not uniform anymore and the sphere is not fully covered. Three different simulation environments are used: A high level environment to observe the behaviour of the algorithm and to experiment with large networks for many simulation scenarios, a low level environment to evaluate the algorithm and a physical testbed implementation to validate the simulation results and to show practical deployability. The metrics used are the average load and the maximum load, which are the average number and the maximum number of packets handled by a node in a given annulus, respectively. The number of nodes used is initially 15000 and later 1000, denoting the very high density of the network with the intention to observe the behaviour of the algorithm. Extensive simulation results are presented and compared with 2D Greedy routing, showing the efficiency of the method in terms of maximum and average load. Varying the radius of the sphere seems to be affecting the performance as well as the number of neighbours; best performance is achieved when the radius R of the sphere is R/1.2 and the mean node neighbourhood size is 20. Coming to the second simulation environment, the setup is 400 nodes with 8 neighbours each and the outcome is throughput measurement versus transmission rate and the number of flows. Curveball routing performs a little better (15% maximum) than the simple Greedy routing. The third environment consists of 90 MicaZ nodes on a rectangular testbed with 2.5 aspect

ratio. Only the load was measured in this scenario, where it is lower for Curveball routing compared to the node peaks of Greedy routing. The load does not appear to be dramatically decreased, instead it seems a little more spread across the topology. Although this paper balances the load in the network using local information, energy levels are not taken into account, as the network lifetime or overhead and delay are not studied. One interesting and useful observation includes the impact of path length, which is significantly increased, and this was experienced in this thesis's simulations as well (Section 7.2).

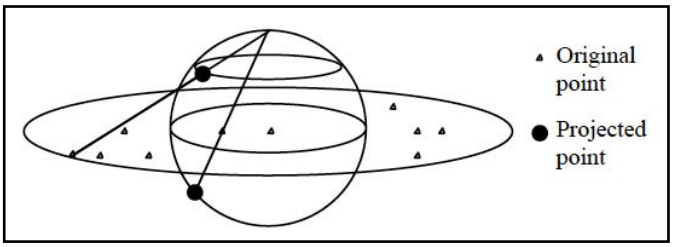


Figure 2.9: Sphere mapping (Popa, et al., 2007)

A similar way of mapping a network is investigated by Xiaokang, et al. (2011) with the goal to address the problem of scalable and load balanced routing in WSNs, based on Curveball routing (Popa, et al., 2007). Instead of straight-forward spherical embedding, optimisation as a polyhedron is performed to achieve a more uniformly distributed network, using connected graphs. Greedy routing is still used on the new transformation with 3D distances, called polyhedron routing, where the polyhedron surface is used as a metric and is modified proportionally to account for Gaussian path curvatures calculated using a heat diffusion approach, producing a reduced graph (Figure 2.10). In order to achieve uniform energy distribution, efforts have been put to make the circles at the vertices have similar size, since the circle centred at each node is proportional to the residual battery level. Results are provided in terms of average load, success rate and load distribution, compared to pure Greedy and Curveball routing. Mathematical equations optimising the metrics and leading to the numerical comparisons of average load are neither introduced,

nor discussed. This method is theoretically proven to perform a little better in terms of delivery guarantee and load distribution, due to more available options of the large number of nodes (average node density is 14.88). The average load percentage is slightly higher than both Greedy and Curveball routing, implying that polyhedron routing uses longer paths. Discussion on route adjustment to battery and density levels is, finally, included showing that polyhedron routing is optimised with respect to battery levels and thus, routes tend to pass through regions of higher battery levels. The comparison to Greedy and Curveball routing is somewhat tenuous, as these two schemes do not consider energy levels during routing.

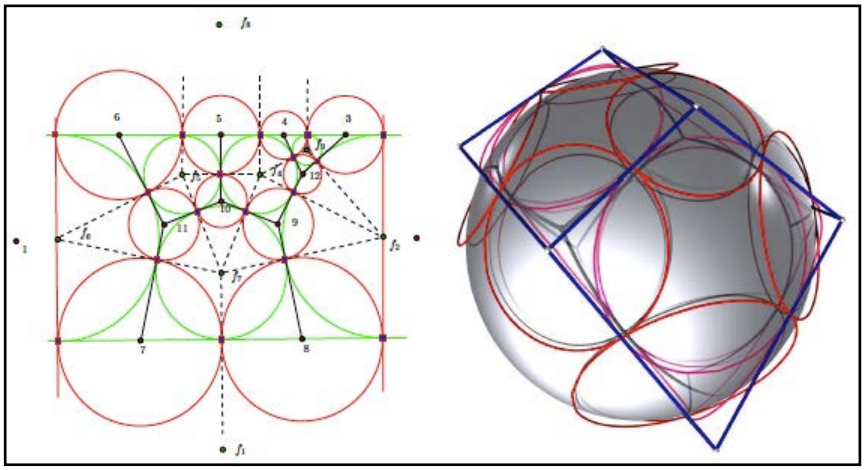


Figure 2.10: Spherical embedding representation in 2D and 3D (Xiaokang, et al., 2011)

Recent publications have proposed various solutions to the uneven energy depletion problem combined with large void areas in WSNs. As outlined above, assumptions that do not always correspond to reality are often made such as deploying massively dense networks, which are convenient theoretically but cannot be applied to real-life scenarios. Having discussed the limitations of WSNs in domestic, commercial, health and military uses in Chapter 1, along with current research areas in general, the question is narrowed down to cases where restricted amounts of energy are available to networks with densities giving a maximum of 10-15 neighbours per node on average. Due to their low density and cost, these networks are implementable in a practical way, where proven solutions can further

be tested on actual hardware. The aim of this investigation is to create a routing protocol that is energy-aware and suitable to be used in such practical network deployments, regardless of its disadvantageous performance in highly dense WSNs.

CHAPTER 3

Curvy Routing and Energy Aware Routing: A Novel Solution

3.1 Introduction and Objectives

This thesis presents a proposal for an energy depletion and data traffic volume sensitive routing algorithm. The concept is to adaptively distribute the data forwarding task as evenly as possible, taking into account the shape of the network coverage area, the local data traffic intensity and the local energy state of nodes. The objective of this study is to accomplish the global optimisation of the WSN lifetime through limited local co-ordination only. Following the existing literature towards a satisfactory solution proves to be inefficient, since - taking local information availability as a point of view - void areas are created in smaller-scale networks as time progresses. Applicable (cost-efficiently), manageable (in terms of design and programming complexity) and realistic sized wireless sensor networks are investigated, examining the conditions under which the algorithmic parameters are optimised and the network performance is maximised.

In most proposals, the phenomenon where key nodes die faster is common, since the paths are more concentrated and alternative routes are not available, leading to early network partition. Spreading the paths more evenly across the network, partitioning can be delayed by additionally using the energy of the nodes placed on the periphery of the topology. The tradeoff of this method is that the paths used are longer on average and therefore, some specific scenarios do not provide improved results due to the increased energy consumption resulting from the higher number of hops per transmission. The conclusion drawn from a large number of detailed

simulations indicates that, as a consequence, there does not exist a clear solution that yields an advantage in all WSN scenarios unconditionally. Part of this proposal's investigation is also to identify under which circumstances the adaptive routing protocol achieves significantly better performance against shortest-path (Greedy) routing.

3.1.1 Geometrical Optics and Analogy with WSNs

The research question to be addressed is establishing a tradeoff between the length of the paths and the overall energy consumption. Solving this problem in the absence of a continuum of alternative paths involves a high level of difficulty, since unrealistic, massively dense networks are required and such assumptions are not part of this investigation's objectives. Solving the problem discretely will face a considerable degree of complexity in calculating the terms representing the standard deviation of the energy and will increase the overhead and delays, which is again outside this investigation's objectives, as the proposed algorithm is aimed to be flexible and highly adaptive. To meet all the objectives set and to satisfy the assumptions made, the solution chosen and proposed here is a discrete approximation to the continuous solution; the terms of the continuous function (introduced as the cost function in Section 3.1.2) only need to be calculated approximately in order to establish the aforementioned tradeoff.

Geometrical optics can provide a solution to the challenge of spreading the geodesics evenly over the entire network and thus increase its lifetime. The idea proposed refers initially to a bulk distribution across the network accounting for the overall network geometry. It is known (Marchard, 1978) that the rays of light tend to bend towards optically denser areas, namely the circumvent areas with high refractive index. If the network is mapped as an optical medium and high data traffic areas along with regions of nodes with low-energy battery state are treated as "optically dense" areas, then those areas with high traffic and/or nodes with

batteries running low can be easily avoided by biasing the selected routes to "bend" around them. According to Fermat's principle (Marchard, 1978), a ray joining any two points in a medium is such that the light-path integral taken along this ray from the first point to the second, is stationary relative to its value for any nearby curve joining the two points. This implies that, energy-wise, the cost of a single transmission between two end points remains the same regardless of whether the link is a curve or a straight line.

In order to represent the refractive index distribution, the theory of spherical gradients (Marchard, 1978) has been adopted. This distribution is based on Maxwell's fish eye lens theory (Born & Wolf, 1970), where every point on the region can be sharply imaged, and involves an index function having spherical symmetry about a point chosen as a reference each time. In a perfectly homogeneous medium the rays of light have the form of straight lines. In the chosen system the medium is heterogeneous, the rays are curved and they lie in a plane through the origin.

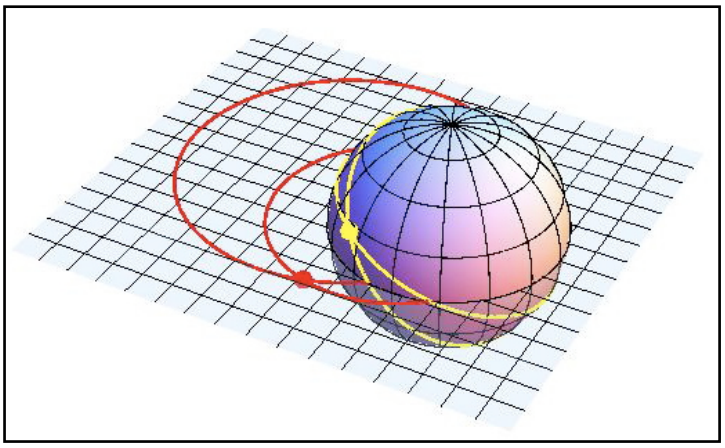
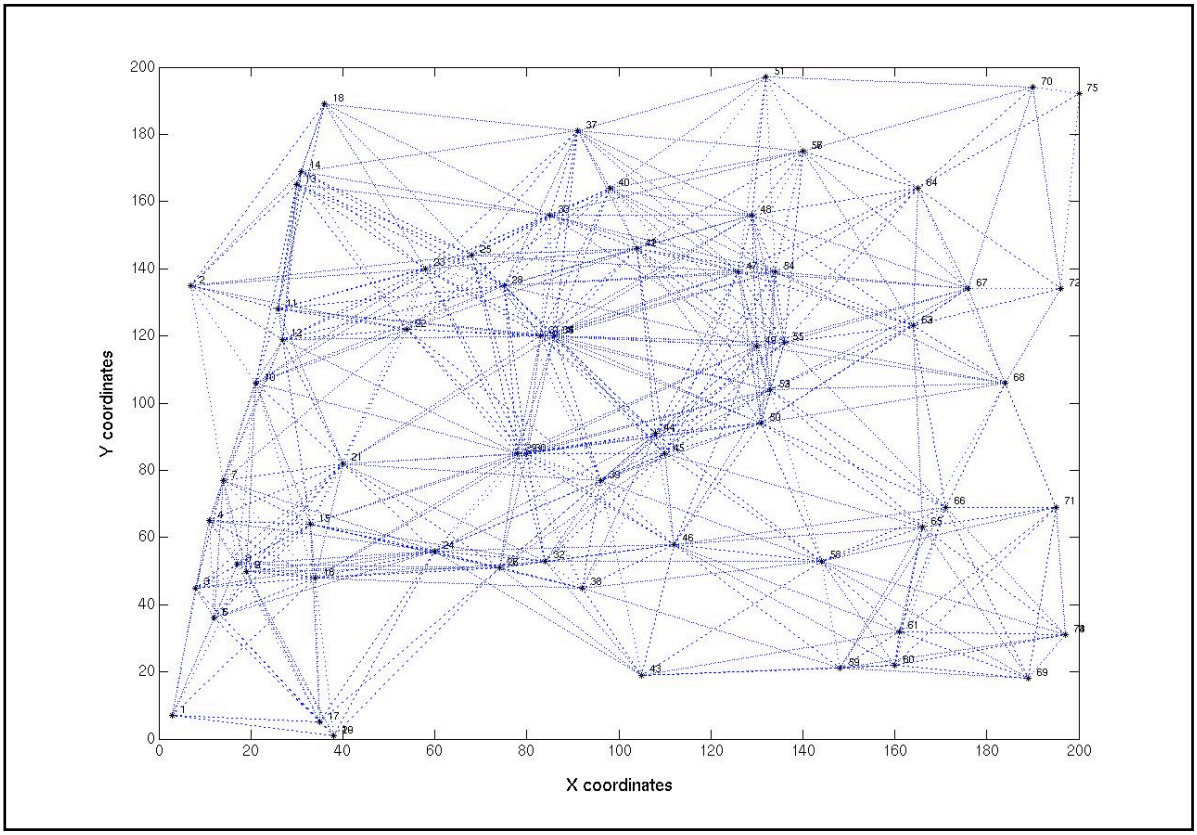


Figure 3.1: Representation of Maxwell's fish eye lens light rays propagation (Leonhardt, 2009)

Figure 3.1 shows the representation of Maxwell's fish eye lens, as rays propagate in a heterogeneous medium. The virtual space of Maxwell's fish eye lens is the surface of the sphere and in the case of the topologies in this investigation the virtual

space is the upper hemisphere. The rays on the sphere are represented in yellow and the ones on the 2D plane are in red. A WSN is treated in the same way, where routing is performed on the hemisphere after converting the plane (network) to a sphere. The approximate, convex network outline is transformed to a circle representing the sphere in 2D, where the spherical gradient equations can be utilised. The centre of that circle has been chosen as the origin of the refractive index distribution. The refractive index is selected to be a function of the radial coordinates of the points in the considered area from a suitably selected origin at the centre of the geographical extent of the WSN. As the bulk topology transformation is only approximate, a spherically symmetric refractive index distribution is chosen for which ray paths have well-understood analytical solutions that can be computed very fast.

The procedure of the circle transformation (rotation angle, stretch factor and transformed coordinates) can be found in Appendix A.2.2 as Matlab code (transformation.m). Figure 3.2 shows the original network topology and its spherical representation after the transformation is applied. The mathematics for this transformation are explained in Chapter 4.



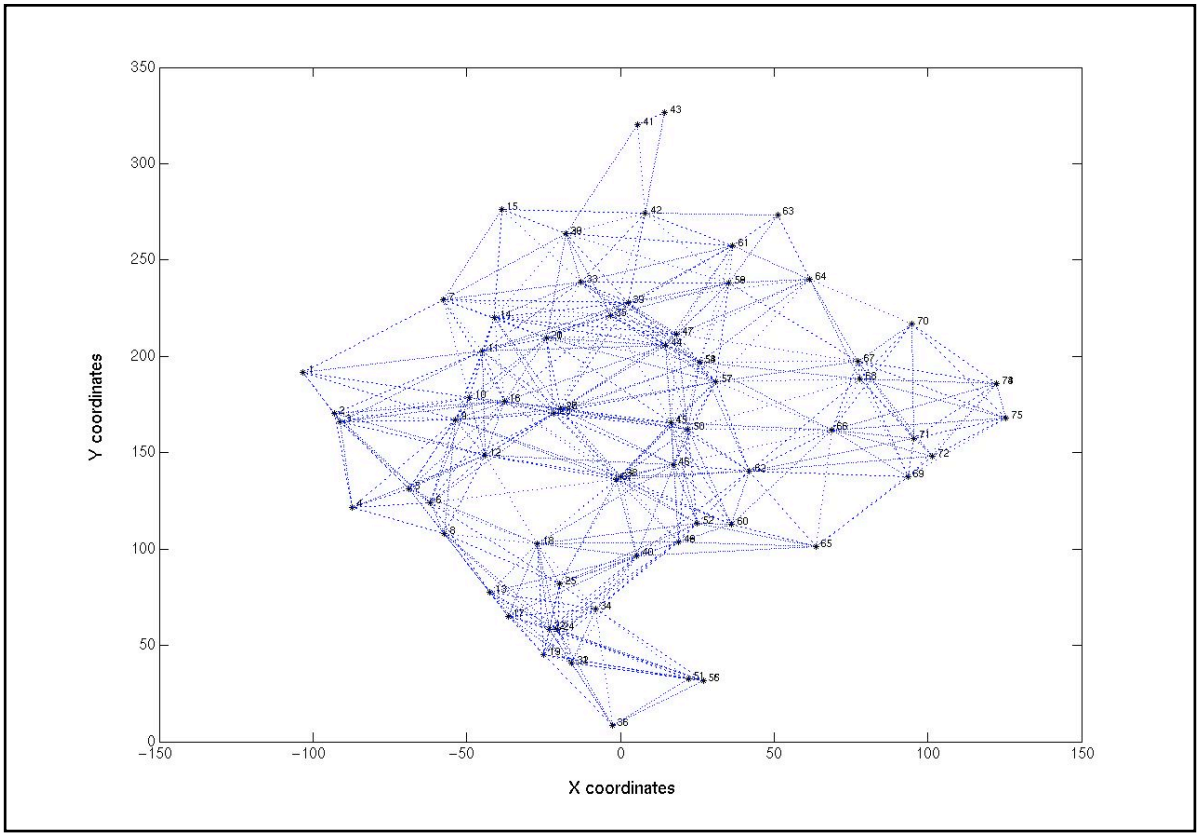


Figure 3.2: Initial WSN topology (top) and its spherical representation (bottom)

3.1.2 Cost Function and Routing Algorithm

The light ray spreading property of Maxwell's lens is based on a spherical gradient described by,

$$n = \frac{n_o}{1 + (\frac{r}{a})^2}, \tag{3.1}$$

where r represents the distance from a fixed point (the centre of the lens), while n_0 and a are constants. It will be shown later on in the equations that the absolute value of n_0 and of the refractive index in general do not dramatically affect the decision making; what actually matters is the gradient of the refractive index. The angle and the curvature of the ray (and therefore of the path) are directly dependent on the gradient of the refractive index. Since local forwarding decisions are made, accurate detailed calculations all the way to the final destination need not to be done. The only information needed is a reasonably accurate direction angle for the local tangent to the end-to-end path in order to be able to locally forward packets towards the destination, which results in minimised overheads.

Routing is then done through local, next-hop neighbour selection along modified geodesic, curved paths for the chosen Maxwell's fish eye refractive index distribution. The neighbouring node further along the curved geodesic path is selected using only geodesic path tangent and curvature, locally at each forwarding node. The next-hop neighbour selection is illustrated with reference to Figure 3.3. The current forwarding node is denoted as a "source" node, s. The tangent unit vector, $\hat{\mathbf{t}}$, of the geodesic path, e, at s indicates the sense in which the data packet is to be forwarded. The local centre of curvature, C, of the geodesic path at s is determined and the "advance angle", ψ, for each eligible neighbour, N, is computed.

The neighbour whose "advance angle" ψ is greatest is then chosen to be the next-hop neighbour. The remaining symbols in Figure 3.3 are listed in Table 3.1.

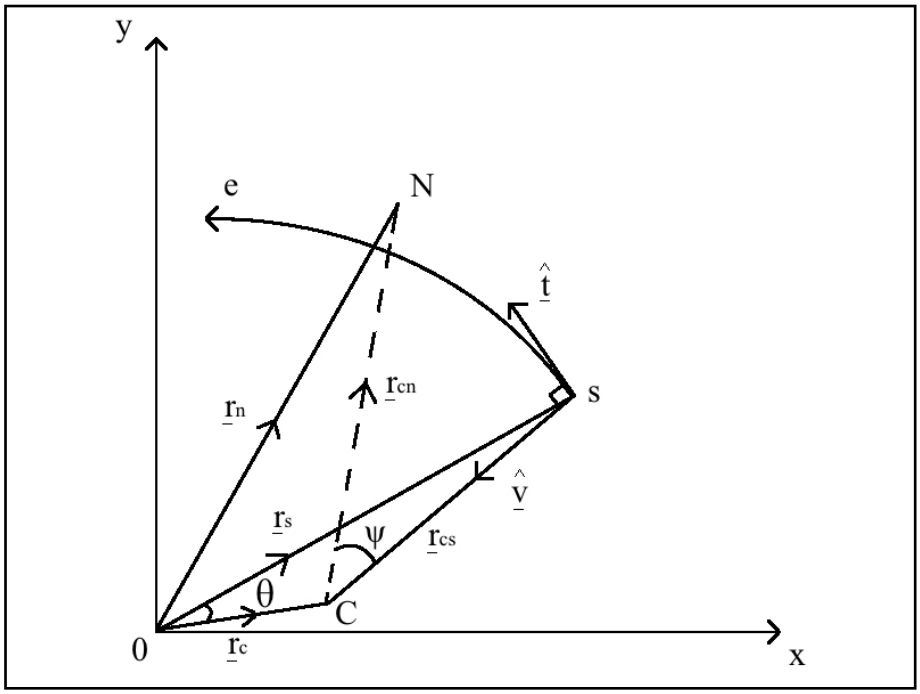


Figure 3.3: Illustration of routing vectors

0	origin of coordinates
C	centre of curvature
s	source node
N	neighbour node
e	geodesic path
$\hat{\mathbf{t}}$	tangent unit vector
$\hat{\mathbf{v}}$	unit principal normal at a point of the ray
$\mathbf{r}_s$	vector from origin to source node
$\mathbf{r}_c$	vector from origin to centre of curvature
$\mathbf{r}_n$	vector from origin to neighbour node
$\mathbf{r}_{cn}$	vector from centre of curvature to neighbour node
$\mathbf{r}_{cs}$	vector from centre of curvature to source node (radius)
ψ	angle between $\mathbf{r}_{cn}$ and $\mathbf{r}_{cs}$
θ	angle between $\mathbf{r}_s$ and $\mathbf{r}_c$

Table 3.1: Index of symbols used in Figure 3.3

After describing the way a local decision is made towards the next hop, the steps the algorithm follows leading to that decision need to be introduced. Once the refractive index distribution has been determined, the polar coordinates (code included in Appendix A.2.8) need to be computed, since routing is performed on those new coordinates after the circle transformation. Local information is exchanged via "hello" message broadcasts so that each node is aware of all its one-hop neighbours' location and battery level. This way, a local neighbour table can be constructed and updated periodically. When a node's battery is totally depleted, this node is not part of its neighbours' table anymore. Every time a node needs to make a forwarding decision, it checks its neighbour table in order to find out which of its initial neighbours are still alive. Afterwards, the ψ advance angle of the all the alive neighbours (shown in Figure 3.3) is calculated. In order to control the accuracy locally towards the direction, a neighbour eligibility criterion has been set. This criterion states that a neighbour is eligible to be chosen if its distance from the centre of the curvature C minus the radius of the circle $\mathbf{r}_{cs}$ is less or equal to twice the mean distance from the source node s to all its neighbours. Figure 3.4 depicts the eligibility criterion, where the eligibility threshold band is drawn in blue. It is clear that neighbours N, N_1, N_2, N_3 and N_4 are within the band and therefore eligible for getting selected, while neighbours N_5, N_6, N_7 and N_8 are not considered. Finally, based on the absolute value of the advance angle ψ, the eligible neighbour which makes maximum progress towards the destination is chosen over the rest. Table 3.2 summarises the above algorithm, whose functionality is outlined in Section 4.3 and its complete code can be found in the Appendix A.

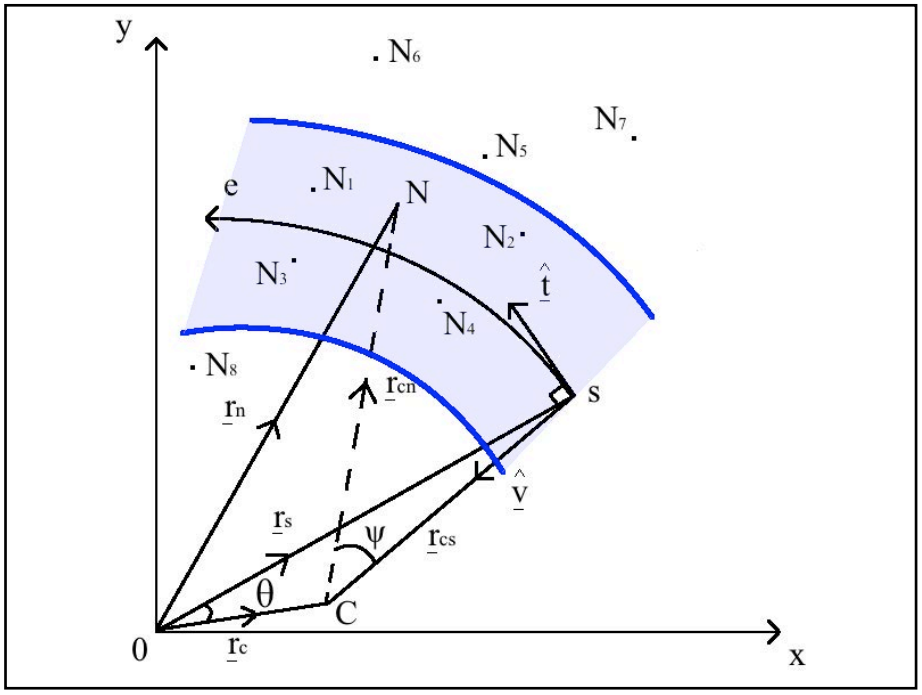


Figure 3.4: Neighbour eligibility criterion

The algorithmic steps leading to next-hop selection
1: Compute polar coordinates for each node 2: Find alive neighbours 3: Calculate the ψ advance angle of the neighbours found 4: Set a threshold for eligibility based on mean distance from neighbours 5: Choose as next hop the neighbour with the largest absolute value of advance angle ψ (maximum progress towards destination)

Table 3.2: Steps of the algorithm

The algorithm introduced uses a refractive index distribution comprising two components, as shown in equation (3.2). The first term refers to the initial bulk distribution across the network establishing the overall geometry. Depending on the shape of the network, there are different ways of implementing the distribution, but the most convenient (corresponding to highest performance) is on a disc. It is assumed that all nodes have pre-programmed information for the approximate overall shape of the network coverage area. The second term accounts for the mean energy stored in the immediate locality of a node and is responsible for the local

deflection of the path forwarding direction away from the bulk geodesics in such a way as to avoid/mitigate for significant battery-stored energy differences between nodes. Information about each node's battery is stored into "hello" packets and is periodically disseminated to their one-hop neighbours only. Regular updates are performed during the simulation, as equation (3.2) is an outcome of reported energy data in a certain position and at a certain time.

$$n(\mathbf{r}, t) = n_{bulk}(\mathbf{r}) + n_{battery}(\mathbf{r}, t). \tag{3.2}$$

Moreover, traffic demand issues were considered in terms of further modifying the refractive index, through the introduction of a third term in the equation. Due to implementation and time limitations, discussed in Chapter 7, this is left as part of future work.

3.2 Curvy Routing

The steps of the algorithm and the calculations involved are common for the first analytical term and the second term of equation (3.2), which employs a numerical approach. The difference between the two comes at step 3 in the algorithm (Table 3.2), which refers to the advance angle ψ (Figure 3.3) calculations. Additional variables are employed to include local energy information and contribute to the next hop selection. Background theory on properties of rays is utilised and presented in parallel with the actual usage in WSNs accompanied, together with the assumptions holding to justify the functionality of the adopted idea.

As outlined in Chapter 2 and stated in the objectives of this investigation, the most important assumption to be considered during computation is immediate locality. It is crucial for the accuracy of the algorithm to extract and use as much local information as possible. The determination of the ray equation (and hence the path)

is essential for the local computation of the vectors connecting the origin of the plane and the points (nodes) laying in it.

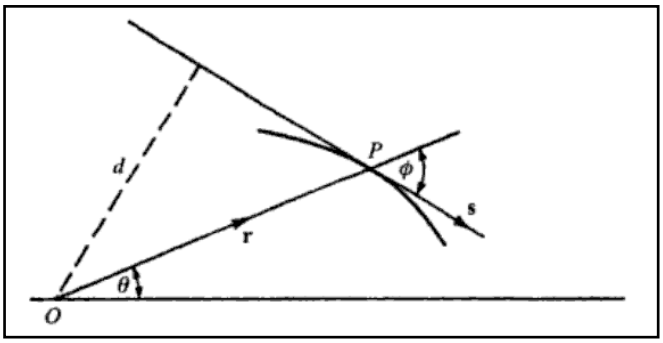


Figure 3.5: Rays in a medium with spherical symmetry (Born & Wolf, 1970)

Figure 3.5 represents the rays in a spherically symmetric medium, where a formula in dynamics holds expressing the conservation of angular momentum of a particle moving under the action of a central force (Born & Wolf, 1970), and where *nd* is constant. Given the polar coordinates (r, θ) of a plane curve, the angle φ is given by

$$\sin\varphi = \frac{r(\theta)}{\sqrt{r^2(\theta) + (\frac{\mathrm{d}r}{\mathrm{d}\theta})^2}}, \tag{3.3}$$

and since $nr\sin\varphi = nd$ = constant,

$$\frac{\mathrm{d}r}{\mathrm{d}\theta} = \frac{r}{c}\sqrt{n^2r^2 - c^2}\,. \tag{3.4}$$

The polar equation $r(\theta)$ of any ray in a medium with spherical symmetry can be, therefore, represented as,

$$\theta = \theta_o + c\int_{r_o}^{r} \frac{\mathrm{d}r}{r\sqrt{n^2r^2 - c^2}}, \tag{3.5}$$

where r_0 and θ_0 are values at any point on the ray and c is a constant to be determined later. Substituting the cost function introduced in equation (3.1) into equation (3.5), the polar equation of rays takes the form of

$$\sin(\theta - \alpha) = \frac{c}{\sqrt{a^2 n^2 - 4c^2}} \frac{r^2 - a^2}{ra}, \tag{3.6}$$

and the constant c satisfies

$$c = \frac{rna^2 \sin(\theta - \alpha)}{\sqrt{(r-a)^2 + 4}}. \tag{3.7}$$

The geodesic path between the source and destination nodes is characterised by an invariant, e, given by (Born & Wolf, 1970)

$$e = \frac{r^2 - a^2}{r \sin(\theta - \alpha)}, \tag{3.8}$$

and this represents the one-parameter family of rays passing through a fixed point. α here is a constant of integration, but geometrically it represents the geodesic angle shown in Figure 3.6.

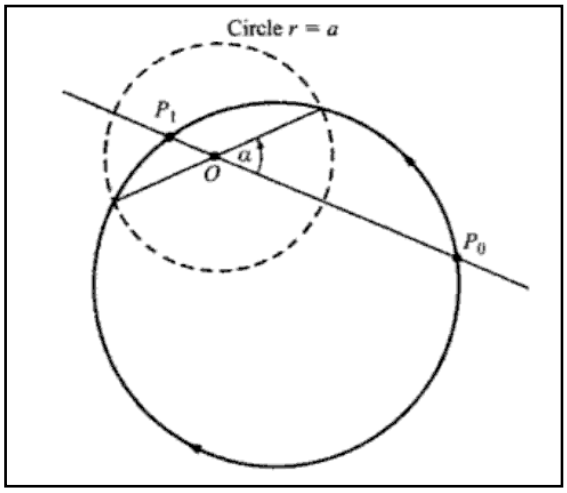


Figure 3.6: Rays in Maxwell's fish eye (Born & Wolf, 1970)

A source/destination pair (r, θ) and (r', θ') of equation (3.9) is written as

$$\frac{r^2 - a^2}{r \sin(\theta - \alpha)} = \frac{r'^2 - a^2}{r' \sin(\theta' - \alpha)}, \tag{3.9}$$

and it can be seen that no matter what the value of α is, equation (3.9) is satisfied by $r = r'$ and $\theta = \theta'$. Equation (3.9) above can be used to calculate the geodesic angle α as

$$\alpha = \tan^{-1} \frac{\sin\theta' r^2 r'^2 - \sin\theta' a^2 r'^2 - \sin\theta r^2 r'^2 + \sin\theta a^2 r^2}{\cos\theta' r^2 r'^2 - \cos\theta' a^2 r'^2 - \cos\theta r^2 r'^2 + \cos\theta a^2 r^2}, \tag{3.10}$$

by using numerically the polar coordinates of a source node ($r\sin\theta$) to a destination node ($r'\sin\theta'$). Having determined α, equation (3.8) can now be solved for $r(\theta)$ giving

$$r(\theta) = \frac{-e\sin(\theta - \alpha) \pm \sqrt{(e\sin(\theta - \alpha))^2 + 4a^2}}{2}. \tag{3.11}$$

The ray equation $r(\theta)$ is of second order and hence it always has two values; depending on the relation between the geodetic angle and the azimuthal angle, usually there is one positive and one negative value. In reality this represents the two arcs of the circle starting from point P_0 and finishing at P_1, shown using directional arrows in Figure 3.6. The algorithm is set to always choose the shorter arc in order to avoid unnecessary angle conversions and restrict (as much as possible) the length of the paths. In the rare case where the points are antipodal to each other and the arcs are equal, the choice of the arc is random.

As seen in Figure 3.3 and the algorithm of Table 3.1, in order to decide the next-hop neighbour, the angle ψ needs to be calculated for each neighbour and to be compared with each other in order to pick the neighbour whose position corresponds to the maximum progress towards the destination. The sine of this angle is given by the cross product of the two vectors $\mathbf{r}_{cn}$ and $\mathbf{r}_{cs}$.

$$\psi = \sin^{-1} \frac{\mathbf{r}_{cs} \times \mathbf{r}_{cn}}{|\mathbf{r}_{cs}||\mathbf{r}_{cn}|}. \tag{3.12}$$

Looking at Figure 3.3, $\mathbf{r}_{cn}$ is the vector connecting the neighbour node N with the centre of the curvature C and $\mathbf{r}_{cs}$ is the vector from the source node s to the centre of the curvature C. It follows that

$$\mathbf{r}_{cn} = \mathbf{r}_n + \mathbf{r}_c\,, \tag{3.13}$$

where $\mathbf{r}_n$ is given by the neighbour polar coordinates, and

$$\mathbf{r}_c = \mathbf{r}_s + \mathbf{r}_{cs}\,, \tag{3.14}$$

$$\mathbf{r}_s = \hat{\mathbf{r}}_i r(\theta), \tag{3.15}$$

where $\hat{\mathbf{r}}_i$ is the unit radial vector of the current position and *r(θ)* is the polar equation of the ray calculated previously.

Given the fact that in a heterogeneous medium all the rays are plane curves, situated in a plane through the origin (Born & Wolf, 1970), it can be seen from Figure 3.5 that

$$\mathbf{r} \times n\mathbf{t} = c\,, \tag{3.16}$$

where *c* is the constant appearing in equation (3.4) and determined in equation (3.7). The tangent $\hat{\mathbf{t}}$ (symbolised by **s** in the figure) can be derived from equation (3.16) along with the principal normal, unit vector $\hat{\mathbf{V}}$, which is perpendicular to it (Figure 3.3). The second part of equation (3.14), $\mathbf{r}_{cs}$, is given by multiplying the actual radius of curvature (ρ) with the direction of $\hat{\mathbf{V}}$.

$$\mathbf{r}_{cs} = \rho\hat{\mathbf{V}}\,. \tag{3.17}$$

The radius ρ is reversely proportional to the magnitude of the curvature vector of a ray, **K**, (Born & Wolf, 1970):

$$\rho = \frac{1}{|\mathbf{K}|}. \tag{3.18}$$

Based on the vector form of the differential equations of the light rays by definition (Born & Wolf, 1970), it shows that the gradient of the refractive index *n* lies in the osculating plane of the ray and, therefore, the curvature vector is also related to the gradient of the refractive index by

$$|\mathbf{K}| = \hat{\mathbf{v}} \nabla \ln n\,. \tag{3.19}$$

Combining equations (3.18) and (3.19) shows that proceeding along the principal normal, the refractive index increases, proving that the ray bends towards the region of high refractive index. By being able to control the refractive index (equation 3.2), the curvature of the ray can be controlled as well. Therefore, the extent to which the paths are bent can be handled in such a manner to achieve the optimum tradeoff discussed previously: a balance between the length of the paths and the energy depletion.

Finally, in a sparse network, neighbours are unlikely to lie on the geodesic path *e* to the destination (Figure 3.3) and therefore an obvious and ideal next-hop choice based only on the advance angle ψ is not possible. Nearby neighbours need to be considered for selection and although they are one hop away, their distance from the geodesic path may vary significantly. Hence, their location can interfere with the curvature and sequentially with the final path length in an undesirable way. For that reason, a radial band centred on *e* is defined, whose width is determined by the mean distance $\bar{d}$ between nodes in order to identify which neighbours are eligible to undertake the role of the next-hop node (Figure 3.4), given by,

$$||\mathbf{r}_{cn}| - \rho| \leqslant 2\bar{d}\,. \tag{3.20}$$

3.3 Energy Aware Routing

The second part of the proposed refractive index in equation (3.2) is presented and analysed below. The aim of a more adaptive version of this algorithm is to dynamically further equalise energy depletion across a network and thus keep it alive for as long as possible, so that the majority of the nodes suffer from battery exhaustion as late as possible.

Local energy levels in the neighbourhood are taken into account by the second term of the proposed algorithm (equation (3.2)). All previous symbols have remained the same. Primed symbols represent the parameters which have changed due to the local energy periodic updates and the actual change to those parameters is represented by the symbol δ in front. The way the next-hop decision is made is not affected and the calculation of the advance angle ψ proceeds as before.

The new refractive index (n') consists of the previous Curvy routing value and an additional new local energy term n_e. The cost function (equation (3.1)) now becomes

$$n' = n + n_e. \tag{3.21}$$

Subsequently, the parameters directly dependent on the new refractive index need to be re-calculated. Given the fact that only the second term is different in equation (3.21), only the incremental change (denoted by δ) of the parameters involved needs to be defined and then added to the existing Curvy routing value. As discussed in Section 3.2 and showed in equations (3.18) and (3.19), calculation of the curvature ray $\mathbf{K}$ and therefore the gradient of the refractive index are essential for establishing the desired curve. Starting with the differential equation (3.22) of light rays (Born & Wolf, 1970):

$$\frac{\mathrm{d}}{\mathrm{d}s}(n'\frac{\mathrm{d}\mathbf{r}}{\mathrm{d}s}) = \nabla n', \tag{3.22}$$

it follows that

$$\frac{\mathrm{d}\mathbf{r}}{\mathrm{d}s} = \mathbf{t}' = \mathbf{t} + \delta\mathbf{t}, \tag{3.23}$$

$$\frac{\mathrm{d}\mathbf{t}}{\mathrm{d}s} = \mathbf{K}' = \mathbf{K} + \delta\mathbf{K}, \tag{3.24}$$

and

$$\frac{\mathrm{d}n}{\mathrm{d}s} = (\mathbf{t} + \delta\mathbf{t})\nabla n, \tag{3.25}$$

$$\frac{\mathrm{d}n_e}{\mathrm{d}s} = (\mathbf{t} + \delta\mathbf{t})\nabla n_e. \tag{3.26}$$

Substituting equation (3.21) to equation (3.22) gives,

$$\frac{\mathrm{d}}{\mathrm{d}s}[(n + n_e)\frac{\mathrm{d}\mathbf{r}}{\mathrm{d}s}] = \nabla n + \nabla n_e. \tag{3.27}$$

Following the notation stated previously about the change of the parameters, it is clear that the values needed to be re-calculated are n_e, ∇n_e and $\delta\mathbf{K}$, while $\delta\mathbf{t}$ can be computed after having determined $\delta\mathbf{K}$.

3.3.1 Determining the local energy term of the refractive index n_e

As seen by the equations presented above, the mean energy level available to the neighbourhood is directly dependent on the gradient of the refractive index (∇n_e), which sequentially affects the change to the curvature vector of the ray ($\delta\mathbf{K}$). Practically, this means that the slightest change to the residual energy affects the curvature of each path, adding some correction to its direction each time information about energy levels is updated.

It follows that, this method is highly sensitive to variations of the local energy levels. The local energy term of the refractive index (equation (3.21)) has a relation with the energy level ε_k at a given time and location. An additional parameter, ξ, is introduced to express this relation and reflect the sensitivity to energy changes.

$$n_e = \xi \, \varepsilon_k \,. \tag{3.28}$$

The parameter ξ represents the same relation between the refractive index in Curvy routing n and the initial energy level per node ε_{max} (equation (3.29)), but it was not utilised before since the refractive index never changed once the distribution was established and further computations were not needed. The initial value of ξ was therefore set as shown in equation (3.29) and it was tuned afterwards according to the algorithm's performance.

$$\xi = \frac{n}{\varepsilon_{max}} \,. \tag{3.29}$$

3.3.2 Calculating the gradient of the refractive index's local energy term ∇n_e

Using the first order Taylor series expansion of a function with two variables:

$$\varepsilon_i(x_i, y_i) = \varepsilon_o(x_o, y_o) + \frac{\partial \varepsilon}{\partial x}|_{x=x_o}(x_i - x_o) + \frac{\partial \varepsilon}{\partial y}|_{y=y_o}(y_i - y_o)\,, \tag{3.30}$$

where ε_i (i = 1, 2, ... , N) is the energy level of a node at location (x_o , y_o) and N the number of its neighbours.

Singular Value Decomposition (SVD) proved useful in the case of more equations than unknowns present (over-specified) (Press, 1986), where matrices of local energy and node coordinates were constructed to yield the gradient of the n_e. The format of

these matrices is shown in Figure 3.7, where **A** and **b** are known and **x** is the unknown.

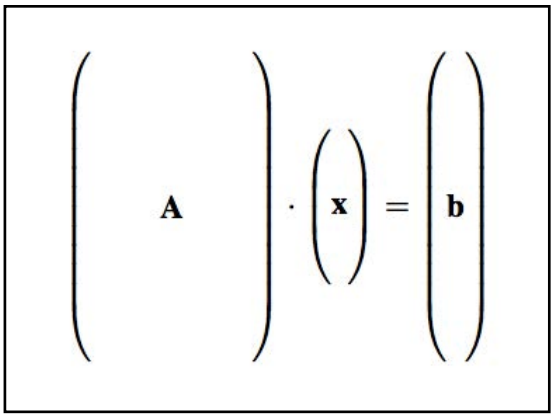


Figure 3.7: Singular Value Decomposition matrices (Press, 1986)

The the gradient of n_e can be expressed locally as

$$\nabla n_e|_{r_o} = \hat{\mathbf{x}}\frac{\partial \varepsilon}{\partial x}|_{x_o} + \hat{\mathbf{y}}\frac{\partial \varepsilon}{\partial y}|_{y_o}, \tag{3.31}$$

where (x_o , y_o) are the current node position coordinates.
Solving equation (3.30), it takes the form of,

$$\varepsilon_i(x_i, y_i) - \varepsilon_o(x_o, y_o) = \frac{\partial \varepsilon}{\partial x}|_{x=x_o}(x_i - x_o) + \frac{\partial \varepsilon}{\partial y}|_{y=y_o}(y_i - y_o) \tag{3.32}$$

and the matrices to be used for SVD are easily extracted:

$$\mathbf{b} = \begin{pmatrix} \varepsilon_1 - \varepsilon_o \\ \vdots \\ \varepsilon_n - \varepsilon_o \end{pmatrix}$$

$$\mathbf{A} = \begin{pmatrix} x_1 - x_o & y_1 - y_o \\ \vdots & \vdots \\ x_n - x_o & y_n - y_o \end{pmatrix}$$

$$\mathbf{x} = \begin{pmatrix} \frac{\partial \varepsilon}{\partial x} \\ \\ \frac{\partial \varepsilon}{\partial y} \end{pmatrix}$$

Matrix **b** contains the local energy, matrix **A** contains the node coordinates and matrix **x** contains the two components of ∇n_e in equation (3.31).

3.3.3 Computing the change of the curvature vector δK

Expanding equation (3.27), leads to

$$\frac{\mathrm{d}}{\mathrm{d}s}[(n+n_e)\frac{\mathrm{d}\mathbf{r}}{\mathrm{d}s}] = \nabla n + \nabla n_e \tag{3.33}$$

and substituting equation (3.23) gives,

$$n\frac{\mathrm{d}}{\mathrm{d}s}(\mathbf{t}+\delta\mathbf{t}) + (\mathbf{t}+\delta\mathbf{t})\frac{\mathrm{d}n}{\mathrm{d}s} + n_e\frac{\mathrm{d}}{\mathrm{d}s}(\mathbf{t}+\delta\mathbf{t}) + (\mathbf{t}+\delta\mathbf{t})\frac{\mathrm{d}n_e}{\mathrm{d}s} = \nabla n + \nabla n_e. \tag{3.34}$$

Using equation (3.24), transforms equation (3.34) into,

$$n\mathbf{K} + n\ \delta\mathbf{K} + \frac{\mathrm{d}n}{\mathrm{d}s}\mathbf{t} + \frac{\mathrm{d}n}{\mathrm{d}s}\delta\mathbf{t} + n_e\mathbf{K} + n_e\ \delta\mathbf{K} + \mathbf{t}\frac{\mathrm{d}n_e}{\mathrm{d}s} + \delta\mathbf{t}\frac{\mathrm{d}n_e}{\mathrm{d}s} = \nabla n + \nabla n_e, \tag{3.35}$$

which (working backwards equations (3.22), (3.23) and (3.24)) eliminates into,

$$n\ \delta\mathbf{K} + \frac{\mathrm{d}n}{\mathrm{d}s}\delta\mathbf{t} + n_e\mathbf{K} + n_e\ \delta\mathbf{K} + \mathbf{t}\frac{\mathrm{d}n_e}{\mathrm{d}s} + \delta\mathbf{t}\frac{\mathrm{d}n_e}{\mathrm{d}s} = \nabla n_e. \tag{3.36}$$

If (3.25) and (3.26) are substituted into equation (3.36), yields,

$$n\ \delta\mathbf{K} + [(\mathbf{t}+\delta\mathbf{t})\nabla n] + n_e\mathbf{K} + n_e\ \delta\mathbf{K} + \mathbf{t}[(\mathbf{t}+\delta\mathbf{t})\nabla n_e] + \delta\mathbf{t}[(\mathbf{t}+\delta\mathbf{t})\nabla n_e] = \nabla n_e, \tag{3.37}$$

and solving for δ**K** then gives,

$$\delta\mathbf{K} = \frac{\nabla n_e - \nabla n\ \delta\mathbf{t}(\mathbf{t}+\delta\mathbf{t}) - \nabla n_e\ \mathbf{t}(\mathbf{t}+\delta\mathbf{t}) - \nabla n_e\ \delta\mathbf{t}(\mathbf{t}+\delta\mathbf{t}) - n_e\mathbf{K}}{n+n_e}. \quad (3.38)$$

The new values retrieved are used the same way as previously to calculate the advance angle ψ. In Chapter 6, the paths between random source-destination pairs are compared and show the difference between the two routing techniques. The paths chosen by the Energy Aware routing algorithm are much more accurate in terms of load balancing and energy management across the network.

3.4 Protocol Evaluation

Since the evaluation method for the proposed protocol was decided to be carried out via simulations, a network simulator is necessary. A custom-built network simulator is presented and outlined in Chapter 4. All the properties included in the simulator along with the available features are described to meet the requirements of simulating a large variety of different network scenarios.

CHAPTER 4

Network Simulator

4.1 Implementation Limitations and Network Simulator Selection

From the early stages of the inception of the Curvy routing method to the actual implementation of it, many difficulties have been met and thus, inevitably, have shaped the final solution.

At a theoretical level, the concept of making the paths bend over void areas was proposed and ways of realising it have been sought: Existing published proposals were examined (Popa, et al., 2007; Catanuto, Morabito & Toumpis, 2006) and the attempt of reproducing them proved restrictive in terms of implementation complications and network density assumptions. Alternative mathematical models have, then, been targeted expressing physical theories of optical lenses (Marchard, 1978) and gravitation (Mollerach & Roulet, 2002), with the objective to provide an integrated and acceptable foundation for the realisation of the idea and its parallelisation to WSNs. Maxwell's fisheye lens and spherical gradient theory (Born & Wolf, 1970) were eventually selected, as discussed in Chapter 3.

Coming to building the simulation scenarios, the choice of a network simulator was necessary and research on existing solutions was carried out. At the time, OPNET (OPNET, 2007) seemed to be promising in terms of maturity, being powerful, graphical representation, with extensive tools and module availability, sophisticated results analysis and traffic monitoring. However, due to its complicated nature, it proved to lack programming flexibility and thus, unable to be modified according to the desired simulation scenarios. The decision was made to

move to Matlab (Matlab, 2009) and build a customised simulator precisely suited to the research project requirements (validated in Section 4.2.1). The scenarios were constructed successfully to represent the different stages of the algorithm, but the nature of the analysis had to vary to a more numerical low-level approach, shifting the focus away from traffic/packet quality analysis. Because of the research challenges in WSNs and their characteristics discussed in Chapter 1, this shift did not affect the way the network is observed. In fact, focusing entirely on the survivability of WSNs, tackles the energy management issue more accurately and more in detail. A packet-level discrete event simulator (like OPNET) is unnecessarily complex for the study of WSNs, because the analysis it provides does not reflect the problem in reality since it is designed to cover a large range of different network types with different characteristics and different problems to be addressed.

Finally, hardware equipment limitations on power and memory issues have also shaped the implementation of the method; large scale scenarios were not feasible, while unrealistic energy depletion models were adopted. These assumptions along with the level of abstraction and real-life WSN deployment are outlined later on in this chapter.

4.2 Protocol Evaluation

Since the entire simulation environment was built in Matlab, it provides a great level of programming and implementation freedom. Efforts have been made to simplify the simulator and focus exclusively on the routing technique and the geographical information available within the neighbourhood. Since emphasis is given to a numerical approach, high-level simulation components have been omitted in order to offer flexibility, time conservation, overhead minimisation and programming efficiency. Transmission schemes controlled at the MAC (Medium Access Control) layer are not investigated, while traffic monitoring details and

Quality of Service (QoS) are outside of the project's focus. The results obtained are not affected by the presence of such mechanisms and therefore the reason for those to exist is unnecessary. Responsibilities handled at the Network layer level are enough to take control of the routing technique and experiment with the survivability of WSNs associated with energy consumption.

The routing protocol is not built in a way that corresponds to reality, neither are the simulations set up with real-life scenarios. Assumptions for the convenience of the implementation have been made regarding the network area, the number of nodes placed, the size of neighbourhood, the communication model, the transmission range, the update frequency, the data rate, the definition of (simulation) time and the energy consumption model. Some of those settings have been chosen empirically because they do not make much difference in the deployment and some are chosen to serve certain purposes discussed below. Performing simulations under these assumptions, aims to provide information and results which can potentially be used to actualise these simulations on hardware and to examine how the proposed algorithm can be utilised in real-life deployments.

An overview of the simulations ran is shown in Table 4.1. The network area has been kept constant at 200m x 200m throughout the experiments and it was chosen empirically, in order to approach a real-life WSN deployment. The parameters which vary are the total number of nodes in the area, the placement of the sink node (in the centre, near the edge of the network or random source-destination pairs), the neighbourhood size in average number of neighbours per node and the network density in nodes per square metre. The transmission range was initially set to be 62 m and, by varying the neighbourhood size each time, the rest of the parameters can be derived. The first simulation setup had 30 neighbours per node and a central sink node. The neighbourhood size kept being reduced (25, 23 and then 20), comparing results between the algorithms, until experimentation with the position of the sink node took place. The combination of these two parameters (neighbourhood size = 20 and sink node near the edge of the network) produced some satisfactory results and revealed the functionality of the algorithm in sparse networks, where its limits

were decided to be tested. The boundary of network connectivity was touched (neighbourhood size = 7), still producing advantageous results. Experimenting with the transmission range afterwards, altered the total number of nodes in the area (initially 100 and then 140) and, sequentially, the average number of hops to the destination, allowing space for the curved paths to propagate. The simulations not included in the results chapters are marked with an asterisk and for the reason that they do not produce better or different results. Only for neighbourhood sizes smaller than 20 the results show significant difference, due to the sparse topologies. The rest of the scenarios are part of Chapters 5 and 6, demonstrating the performance of the algorithms against Greedy routing.

Network area = 200m x 200m			
Total number of nodes	**Sink placement**	**Neighbourhood size**	**Network Density**
Transmission range = 62 m			
* 100	central	30	2.5×10^{-03}
75	central / s-d pairs	25	1.875×10^{-03}
* 60	central	23	1.5×10^{-03}
50	central / s-d pairs / edge	20	1.25×10^{-03}
40	edge	15	1×10^{-03}
38	edge	10	0.95×10^{-03}
35	edge	7	0.875×10^{-03}
Transmission range = 42 m			
100	edge	7	2.5×10^{-03}
Transmission range = 31 m			
140	edge	7	3.5×10^{-03}

Table 4.1: Overview of simulations

Simulation time issues were investigated next, where a propagation model was created. As discussed previously, high-level mechanisms are not needed and an accurate, realistic model will only add to the complexity without serving any particular purpose. A simple transmission model was adopted, where each activity takes one time slot referred as "simulation second". For simplicity, an end-to-end transmission takes one simulation second and updates are initiated after the completion of each packet delivery. The update frequency was chosen for the convenience of constructing an energy level table of all one-hop neighbours. Since only immediate locality is considered and events are not run in parallel, simulation time synchronisation was never needed. Overheads are minimised, this way, since the energy level and neighbourhood updates are carried out just before the discovery of a new path. The initial amount of energy available at each node can vary depending on the purpose of the simulation, but is usually set to be ten times the total number of nodes in the network. Battery depletion is set to 75 mJ, 5 mJ and 1 mJ for one transmission, one reception and idle state per time slot respectively. It can be argued that large amounts of energy are consumed per transmission which do not correspond to reality. This is because the object of the simulation is to investigate the algorithm's behaviour under rapid topological changes due to node battery depletion. Since large-sized simulations are not permitted, in order to achieve fast network partition (defined later on in Section 4.2.2) and force the algorithm to the extremes, the above energy depletion settings are adopted. The code for Greedy, Curvy and Energy Aware routing algorithm can be found in the Appendix A and is described more technically in Section 4.3.1 below.

4.2.1 Assessment of Protocol Implementation

The basic Greedy routing algorithm (Cormen, Leiserson, Rivest & Stein, 2009) has been implemented and validated in several scenarios to verify the correctness of the

simulation environment and to provide a benchmark/comparison baseline protocol. The topology is modelled using the unit disc graph (Clark, Colbourn & Johnson, 1990), which is a graph formed from a collection of points in the Euclidean plane in which two points are connected if their distance is below a fixed threshold. Exploiting this property, neighbourhood is defined and recorded for each node, should the distance between communicating nodes is smaller or equal to the notional transmission range. Dijkstra's algorithm (Dijkstra, 1959) is used to verify (among others discussed below) that the network considered in each simulation is connected. A sink node has been used as the destination at locations both near the centre and the edge of the geographical area covered by the WSN. Every node sends a packet towards the sink in random order and each transmission is assumed to take one normalised time interval. Local information at one-hop neighbours is available to each node including exact location, energy level and timestamp, obtained through periodic hello beacon broadcasts. The next-hop node for Greedy routing is chosen by calculating the distance from each neighbour to the sink and selecting the nearest neighbour to the sink, provided it still has enough energy. In cases when Greedy routing fails, face routing (Bose, Morin, Stojmenovic & Urrutia, 1999) is used to circumvent the void areas ahead and ensure successful packet delivery.

Curvy routing is performed as described in Chapter 3, applying the refractive index distribution and spreading the paths more evenly over the network compared to Greedy routing. The curvature of the paths is calculated along with the angle between the nodes and their neighbours, choosing the one that makes maximum progress to the sink. When Curvy routing fails, routing falls back to Greedy for guaranteed delivery.

Employing the second term in equation (3.2) in the algorithm presented in Chapter 3, routing is performed considering the local energy levels of the one-hop neighbours. There are two ways of gathering information; taking into account the residual energy of each neighbour individually or taking into account the average residual energy of the current node's neighbourhood. The new refractive index is

calculated each time and updated before each transmission, to make sure the next hop selection is accurate. The rest of the process towards the optimum neighbour is identical to the Curvy routing.

All three versions of the protocol presented above (Greedy, Curvy and Energy Aware routing), were validated manually by carrying out the exact same computations and comparing the outcome with the one produced by the Matlab-based code. The way neighbourhood is established was checked by calculating the Euclidean distance from a node to the rest of the network using their coordinates. A node is a neighbour to another node if the distance between them is smaller or equal to the transmission range. The neighbour table was reproduced by hand and compared with the one in Matlab. After keeping track of the random path order, the next-hop selection was validated; in Greedy routing by using the Euclidean distance as before and measuring the maximum progress to the destination, while in Curvy and Energy Aware routing by computing the advance angle which determines that progress. The manually created paths were compared with the ones produced by the simulation, since a record is kept for that reason. All mathematical equations of the parameters presented in Chapter 3 were solved individually for each scenario and for each version of the algorithm to ensure accuracy between theory and simulation. Finally, the energy consumption model was validated thoroughly, by reproducing all energy tables and checking the amount of energy set to be depleted at each transmission/reception/idle state and also by calculating the total energy consumed per path.

As mentioned previously, Dijkstra's algorithm (Dijkstra, 1959) has been used for validating the benchmark (Greedy routing) algorithm. This validation method is widely known and used in the literature, because of its simplicity and effectiveness. The way it works is, it calculates the cost of transmission between each node to the rest of the nodes in the network (after neighbourhoods have been established by the unit disc graph (Clark, Colbourn & Johnson, 1990)). This cost is the distance of the hops between the start and end nodes, and once it has been calculated, the shortest path is recorded and back-traced for double-checking. The code of Dijkstra's

algorithm is available in Appendix A. In addition to the above, the algorithm proposed by Popa et al. (2007), called Curveball routing, was reproduced in the custom-built simulator using the author's original source code. The simulations of those scenarios produce the same results presented in Popa's et al.(2007) publication and can verify the validation of the algorithms proposed here. Moreover, it has been used as a justified comparison algorithm in Chapter 5. Table 4.2 below shows the validation process of the operational and functional tests executed at all levels of the construction of the algorithms.

Level	Validation Method		Outcome
	Matlab	Manual	
Neighbourhood	Unit disc graph / neighbourhood tables	Calculation of distance between nodes and comparison with transmission range	Test passed
Connectivity and path discovery (Greedy routing)	Dijkstra's algorithm / back-tracing	Calculation of shortest path and comparison with trace files	Test passed utilising Face routing, when Greedy failed
Energy consumption	Energy model: Transmission = 75mJ Reception = 5mJ Idle = 1mJ	Reproduction of energy tables and comparison with Matlab / calculation of total energy consumed per path	Test passed
Simulation time	Transmission model: one simulation second per end-to-end transmission / frequency of updates before each transmission	Synchronisation checked with updated energy and neighbourhood tables	Test passed
Next hop selection (Curvy and Energy Aware routing)	Code creation to reflect the theoretical model	Manual solving of mathematics and comparison of the "advance" angle values with Matlab	Test passed dropping back to Greedy routing, when curved paths failed
Curveball routing reproduction	Author's original source code	Accuracy of paths comparison with published results	Test passed

Table 4.2: Algorithm's validation tests

4.2.2 Evaluation Metrics

In order to verify that the proposed routing algorithm exhibits the desirable behaviour intended, a number of different simulation scenarios have been run varying the shape of the network, the topology, the density, the location of the sink node, or deploying random source-destination paths. Initially, the end time was set to the time when network partition occurs. A universal definition of network partition is hard to be given and it depends largely on the topology. It has mentioned before that when Greedy routing fails (meaning when there are no neighbours ahead to forward the packet to), face routing is used to find the next hop. As a rule, face routing is terminated when it eventually finds a neighbour ahead (and switches back to Greedy routing) or when it reaches the destination. Face routing guarantees delivery (Bose, Morin, Stojmenovic & Urrutia, 1999) and when it fails, it means that the network has been partitioned and there is no connection between the remaining nodes and the sink or the sink's neighbours. Other simulation scenarios were run until the sink has received packets from all the nodes in the network, where comparable results have been derived and discussed in Chapters 5 and 6. In general, for the same simulation duration, network partition is experienced in Greedy routing, while Curvy and Energy Aware routing complete the simulation as expected.

The metrics used for the performance evaluation and comparisons are:

- **Average energy**: The average residual energy (mJ) of all the nodes in the network as a function of simulation time. The rate the average energy reduces during each simulation run is studied.
- **Standard deviation of node energy**: Based on the residual energy levels, the standard deviation represents the spread of unequal energy consumption in the WSN. Once again, of particular interest is the rate the standard deviation increases with respect to time during each simulation run.

- **Time to network partition**: The simulation time when network partition occurs is recorded in normalised time units (= 1 simulation "second").
- **Time until the battery of the first node is depleted**: The time (in simulation "seconds") when the first node in the network runs out of battery is recorded.

To complement the above metrics, figures of energy levels (mJ) are shown against simulation time at every node location (x-y coordinates), implicitly illustrating that low energy levels correspond to high traffic. They are used to exhibit how data traffic reduces the residual node energies across the WSN, which aim to show that, using curved paths, the load is more balanced compared to Greedy routing.

Moreover, screenshots at random simulation time slots reflecting how the refractive index adopts to local energy level alterations are presented in tandem (energy and refractive index distribution) and discussed in terms of rapid and accurate response to those alterations.

4.3 Functionality of Algorithm

Following the algorithmic steps mentioned briefly in Table 3.2, a flowchart has been created (Figure 4.1) to represent in detail the way the routing algorithm is designed to function. This flowchart is universal for Greedy routing, Curvy routing and Energy Aware routing. The algorithm functions in exactly the same manner, apart from the step where the next hop selection is calculated. The detailed code description is following in Section 4.3.1.

Starting from the top of the flowchart, the network setup is first determined in order to create a topology inside the area designated with the desired number of nodes spread randomly within the area dimensions (fixed at 200m x 200m). The node coordinates are, then, retrieved by scanning the area and recording the location of the points on it, with regard to the network dimensions (x = 0-200 m and y = 0-200 m). When Curvy routing and Energy Aware routing are simulated, the

circle transformation is executed and since it alters the coordinates, it precedes the determination of them.

A few more parameters are initialised afterwards, providing necessary information to carry on with the simulation: The allocation of the sink is important to define the destination of the paths and it is chosen according to the purpose of the simulation (centre/side). In order to approach reality, a random order of paths from each node to the sink is generated and used later on. The nodes' transmission range is declared at this stage as well, in order to establish neighbourhoods and define immediate locality based on the unit disc graph method (Clark, Colbourn & Johnson, 1990) described in Section 4.2.1. Finally, the initial energy level for each node is also set according to the size of the network, the number of the paths (in the case of random source-destination generation) and the objective of the simulation. As mentioned previously, the initial energy is usually set to be ten times the total number of nodes in the network to allow enough resources for task forwarding. For example, if there are 50 nodes in total, the initial energy for each node will be 500 mJ; for a 35-node network, 350 mJ will be available initially for each node, and so forth.

Using the initial parameters described above, hello broadcasts are broadcast containing the location, the energy level and timestamp of each node. This way, one-hop neighbourhoods are established and fed with information needed for the next-hop calculation. The first time a hello broadcast is flooded across the network, Dijkstra's algorithm (Dijkstra, 1959) is utilised to double check there exists connectivity between all node pairs. The function of Dijkstra's algorithm and the way it is used here are described below in Section 4.3.1. After this first time, the flooding routine remains unaltered.

The main part of the routing process is the selection of the neighbour to which tasks are forwarded. Following the necessary calculations analysed in Chapter 3, the next-hop node is chosen and added to the path from the source node to the destination. When a forwarding decision cannot be made, face routing is performed to select the next hop. This is where Greedy, Curvy and Energy Aware routing

techniques differ from each other and where additional calculations are included to calculate the next-hop neighbour selection using the appropriate theoretical approach. This stage is executed into a continuous loop until the destination is reached and the current path is complete.

The sequence of the nodes forming each path, is recorded and kept in trace files for validation of the routing technique, comparing it with manual computations (Section 4.2.1). An outer loop has been created containing the hello broadcast stage, the next hop selection and the recording of each path, in order to provide frequent updates on the topology and the residual energy level of each node's neighbours. After the completion of the simulation, data of each node's state is collected and used for further analysis of the algorithm's behaviour.

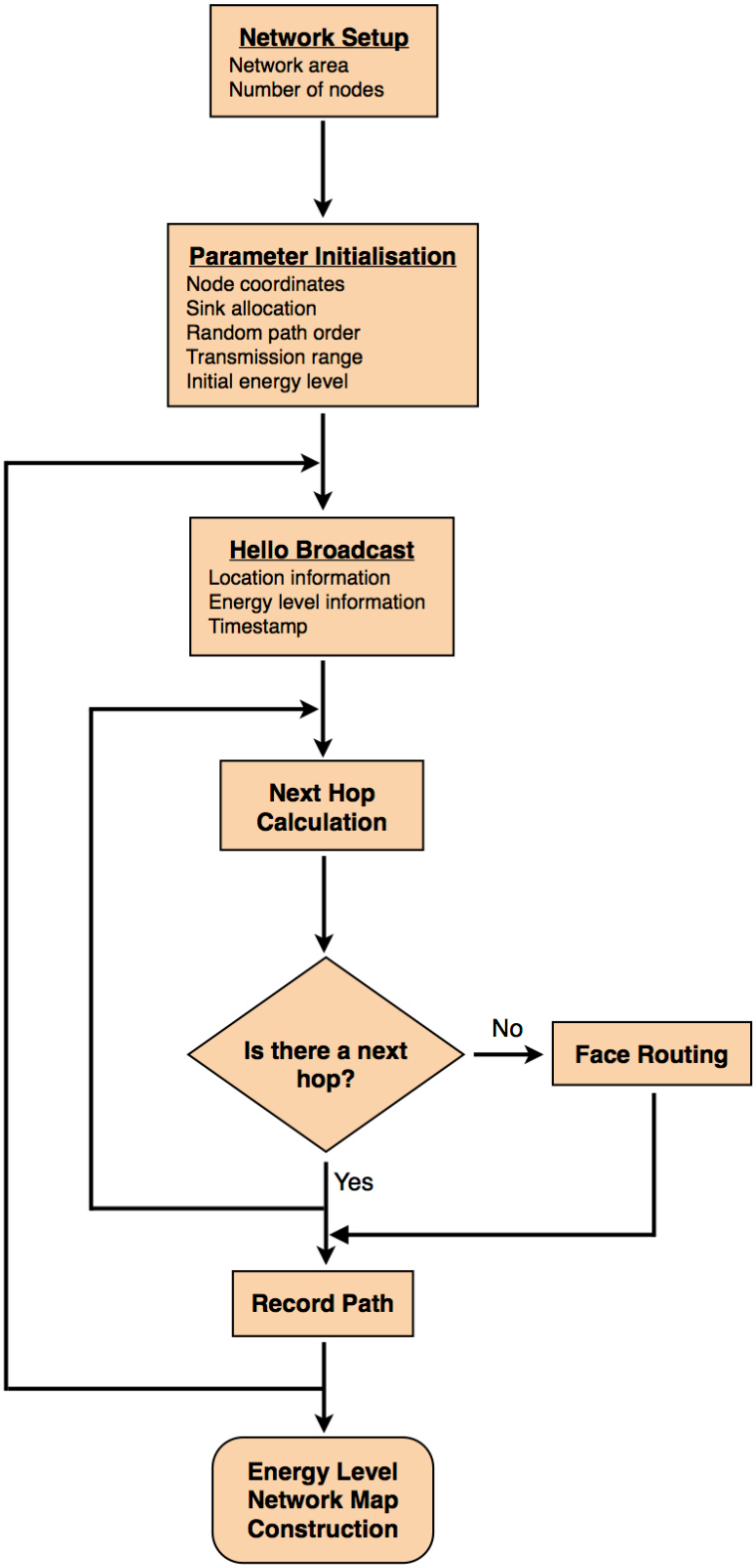


Figure 4.1: Flowchart of algorithm

4.3.1 Code Interpretation

Each function in Matlab is written into a different .m file and is called when in need. The functions are presented in actual order of being called and are notated in the text with double quotation, along with other names of parameters directly extracted from the source code. The entire code for a general simulation scenario is included in the Appendix A. For the full comprehension of this section, reference to the code is recommended.

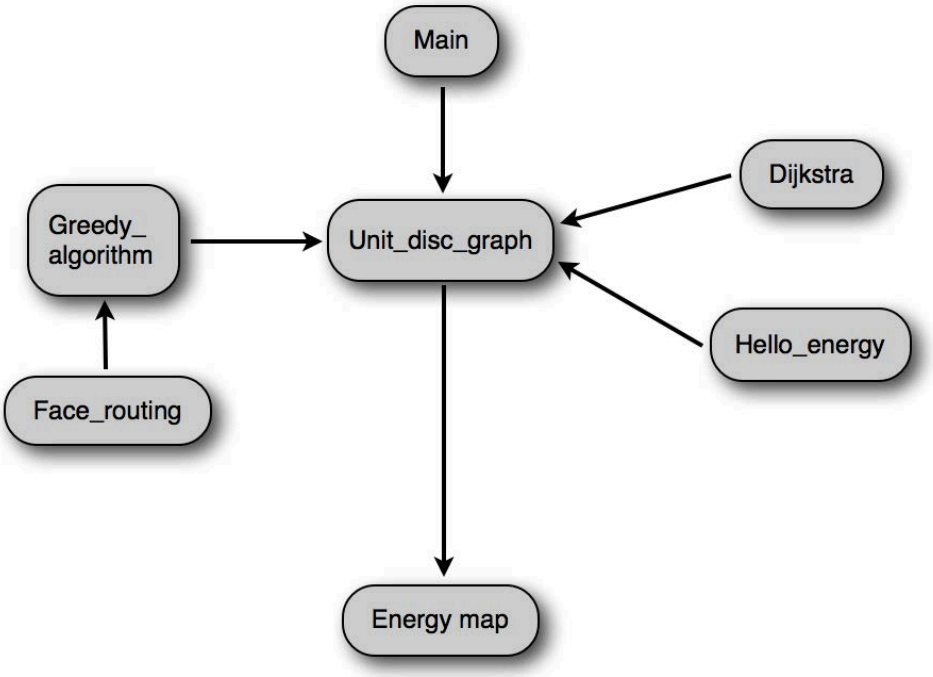


Figure 4.2: Data flow in Greedy routing

The **Greedy routing** scenario (Figure 4.2) includes the functions:

- **Main**: The simulation is initiated by calling the "Main" function. The initialisation is established declaring the desired number of nodes, the dimensions of the network area and the transmission range. Random x and y coordinates are generated for each node on the network and a node ID number is assigned to each of them. The coordinates of nodes are retrieved and the same topology is

used to simulate Curvy and Energy Aware routing. The "Unit_disc_graph" function is, then, called.

- **Unit_disc_graph**: This function is the core of the simulation construction, where the neighbourhoods are established for each node using actual distance calculated between one another. Coordinates falling within the designated transmission range are added to a node's "neighbour_matrix". All the remaining functions are called from within the "Unit_disc_graph". Dikstra's algorithm (Dijkstra, 1959) is invoked next to check whether connectivity has been attained in the network. A hello message is broadcast from each node to its neighbours and information about position, energy level and time is exchanged for each of them, constructing an "energy_matrix" which is updated regularly by the method of hello broadcasts. A random number generator is created to determine the order the paths are going to be discovered and saved for Curvy and Energy Aware scenarios. The "Greedy_algorithm" is called to construct the paths with the next-hop selection and whenever it fails (for energy depletion reasons), "Face_routing" is invoked to find a route around potential void areas. The paths constructed are placed in single large arrays and saved in the route record as trace files. Each time a path is completed, the energy and neighbour matrices are updated to keep up with the rapid changes in the topology due to node battery depletion. Finally, using the "energy_matrix" updated at the end of the simulation, a map is constructed representing the energy levels per simulation second at each node in the network.
- **Dijkstra**: This function is freely distributed and available for download online and can be used to establish network connectivity. It calculates the cost of the path between two nodes and provides a transition (or adjacent) matrix. After the first hello broadcast, "Dijkstra" discovers the shortest path between a node and all other nodes in the network. When needed, it can also be used to validate the result by back-tracing the shortest path found.
- **Hello_energy**: This is invoked whenever energy updates are needed; namely every time a new path is constructed, while the first time it is called, the initial amount of energy available for each node is set. It uses the "neighbour_matrix" as

an index and for each node (including the intermediate ones) used in a path, it decrements the corresponding amount of energy to the current send/receive operation. It is also responsible for decrementing energy corresponding to the idle state of every node in the network. Every time it is invoked, it produces a new "energy_matrix" which is advertised to a node's neighbours.

- **Greedy_algorithm**: The next-hop selection is decided after calling this function. A node is randomly chosen depending on the objective of the simulation (as discussed previously) and is again used for the rest of the scenarios for comparison purposes. The distance between the sink and each of the node's neighbours is calculated and the one that makes maximum progress to the destination is chosen. This procedure is repeated until the destination is reached, while every single selection is recorded in an array, as explained previously. This function also outputs the selected path to the screen during the simulation.
- **Face_routing**: In the case where there are no available neighbours ahead, "Face_routing" is invoked to calculate the next hop in a different way. Face routing (Bose, Morin, Stojmenovic & Urrutia, 1999) is a technique to circumvent void areas in the network broadly used when Greedy (and other types of) routing fail. It uses a Gabriel graph (Bose, Morin, Stojmenovic & Urrutia, 1999) of the remaining nodes and treats the network as a graph with the minimum number of links. It draws a line between the sink node and the current node (whose neighbours are not eligible for next-hop selection), and searches for intersections with other links. It always moves following the left hand-side or right hand-side rule (here the left one is picked) to avoid loops and choses the left of the two nodes whose link was intersected by the initial drawn line. If there is no intersection, it uses cosine angle calculations to find the neighbour which is furthest to the left and forwards the packet there. This is repeated until there exist eligible neighbours ahead, where Greedy routing is executed as before, or until the destination is reached. The face routing technique is represented in Figure 4.3, where a line is drawn from source s to the destination t which divides

the graph into four faces (F_1 - F_4). As seen, the right hand-side rule is followed and the path chosen is s-v_1-v_2-v_3-v_6-v_9-v_{10}-v_{11}-v_{13}-t.

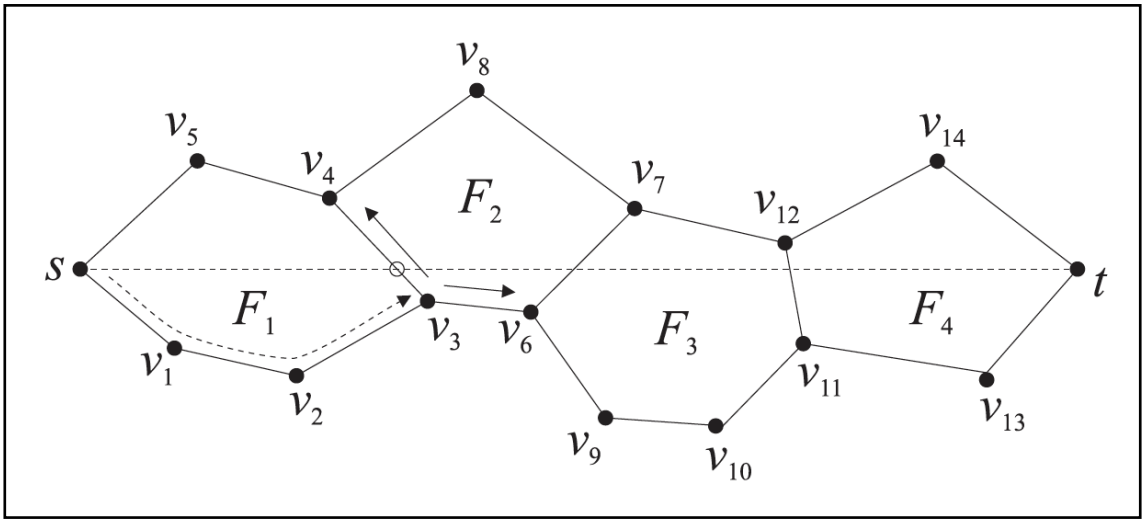


Figure 4.3: Face routing technique (Frey & Stojmenovic, 2006)

Curvy routing is different in many points across the construction of the simulation. The functions of the scenario are largely the same (depicted in Figure 4.4), whereas the additional ones are highlighted in italics:

- **Main**: This time, the "Main" function loads the topology and the sink node created and used previously in Greedy routing. The randomly generated order of transmission by the remaining nodes is loaded as well, along with the number of nodes, the network area and the transmission range initialisation values. As explained in Chapter 3, a transformation of the node coordinates is needed in order to compress and rotate it to fit a circle whose diameter is as large as the network area side (200m). The new transformed coordinates are obtained after calling this function (described below) and are, then, sorted in order from left to right so that the routing technique is not interrupted and is performed using ascending node numbers as before. A data structure ("name_index") is employed to map the sorted nodes to the old enumerated labels, including the sink node. The "Unit_disc_graph" function is invoked as previously.

- ***Transformation***: In order to obtain the transformed coordinates described in Section 3.1.1, a transformation function is needed. The network is firstly rotated using the first and last nodes' coordinates to fit into the borders of the network area set at the initialisation (200m x 200m), forming an ellipse. The rotation angle γ is calculated using cross and dot product calculation of the vectors created by the old coordinates (e_x) and the new coordinates (e'_x), whose sine (equation (4.1)) and cosine (equation (4.2)) form a "rotation_matrix" (equation (4.3)).

$$\cos\gamma = \frac{e_x \cdot e'_x}{||e_x||\ ||e'_x||} \tag{4.1}$$

$$\sin\gamma = \frac{e_x \times e'_x}{||e_x||\ ||e'_x||} \tag{4.2}$$

$$R = \begin{pmatrix} \cos\gamma & \sin\gamma \\ -\sin\gamma & \cos\gamma \end{pmatrix} \tag{4.3}$$

Intermediate coordinates (X′, Y′) are created by multiplying the "rotation_matrix" with the initial coordinates (X_o, Y_o) reduced by 100m (half the side of the area, to fit inside the circle's radius) as shown in equation (4.4).

$$\begin{pmatrix} X' \\ Y' \end{pmatrix} = R \times \begin{pmatrix} X_o - 100 \\ Y_o - 100 \end{pmatrix} \tag{4.4}$$

A stretch factor is produced by the maximum distance of the intermediate coordinates Y′ over the maximum distance of the coordinates X′ (equation (4.5)).

$$f = \frac{Y'_{max} - Y'_{min}}{X'_{max} - X'_{min}} \tag{4.5}$$

A matrix to apply the stretch factor on the right components of the "rotation_matrix" was created. Multiplying this "stretch_factor_matrix" with the

"intermediate_coordinates", yields to the new transformed coordinates (X″, Y″) in equation (4.6).

$$\begin{pmatrix} X'' \\ Y'' \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & f \end{pmatrix} \times \begin{pmatrix} X' \\ Y' \end{pmatrix} \tag{4.6}$$

- **Unit_disc_graph**: This function has the same structure as previously, only it uses the transformed coordinates to construct the neighbour_matrix. Dikstra's algorithm and "Hello_energy" functions are used the same way and in the same order as in Greedy routing. "Greedy_algorithm" is invoked afterwards for the next-hop selection; although the name of the function has been kept the same for programming convenience reasons, inside this routine, the new "Curvy" function is invoked. In principle, the whole Curvy method could have been contained within this substituted routine, but a new function has been created for programming management convenience and efficiency. The energy and neighbour matrices are updated regularly as before and the energy level map is created at the end in the same way as previously, before the simulation is terminated.
- **Dijkstra**: Dijkstra's algorithm is called as before, but using the sorted transformed coordinates for its connectivity check.
- **Hello_energy**: Being used the same way and for the same purpose as earlier, the "Hello_energy" function uses the (transformed) "neighbour_matrix" for its energy updates.
- **Greedy_algorithm**: The name and the structure of this function is the same for programming convenience, though the part where the next hop is calculated has been replaced with an external invocation of the "Curvy" function. This is the part of the Curvy routing method where the geodesics are redefined from straight lines to curves. Once the most eligible neighbour is selected by the latter, it is passed onto "Greedy_algorithm" and then recorded in the path array.
- ***Curvy***: "Curvy" function implements all the calculations described in Section 3.2. Polar coordinates of the existing nodes are used for these calculations and

therefore, a separate function is invoked. All the parameters needed are obtained analytically and determining the advance angles ψ_i, the neighbour, i, with the maximum progress (angle value) is chosen as the next hop.

- ***Polar***: The calculations in this function are straight forward; the angular coordinate θ and the radial coordinate r are needed to convert the Cartesian coordinates and yield to the "polar_coordinates" matrix used in the "Curvy" function.
- **Face_routing**: "Face_routing" was actually never needed, but still exists in the Curvy routing scenario as an extra feature. Cases where it is needed or could be used and ways of deploying it are discussed in Section 7.3.

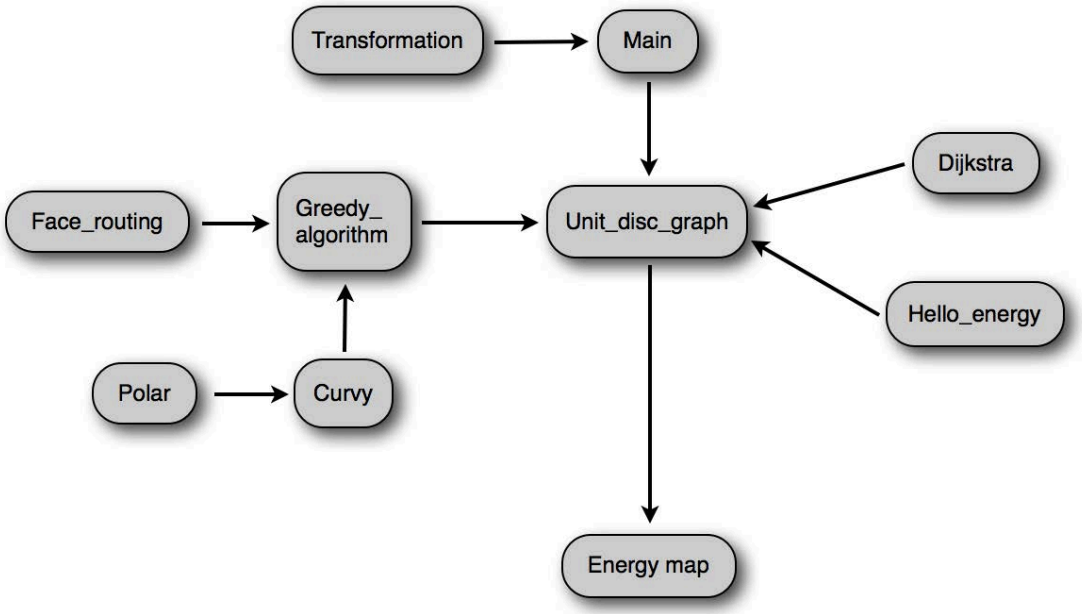


Figure 4.4: Data flow in Curvy and Energy Aware routing

The **Energy Aware routing** code is identical to the Curvy routing one (Figure 4.4). The only difference comes to the "Curvy" function, where additional parameters are used towards the next-hop calculation (as mentioned previously and described in Section 3.3). Because this version of the algorithm takes into account local energy changes, the "energy_matrix" is imported and used in the calculations. The rest of the simulation is carried on the same way, once the next-hop selection is

passed onto the "Greedy_algorithm" function and the "Unit_disc_graph" function produces the final energy level network map.

4.4 Results

The performance of the algorithm is observed in terms of the distribution of energy level depletion evolution on the topology, average node energy and standard deviation per simulation time. The behaviour of the refractive index (n' and n) is monitored with snapshots during the simulation along with instantaneous energy levels on the topology. The lifetime of the network is recorded with regard to period of time until the first node dies and comparisons of Greedy, Curvy and Energy Aware versions are outlined. Discussions also involve alterations observed changing the density, the local energy calculation, the position of the sink node and the transmission range. Performance comparisons are presented into two different Chapters (5 and 6); the initial version of the algorithm (Curvy routing) versus Greedy routing and the completed version (Energy Aware routing) versus both Curvy and Greedy routing.

CHAPTER 5

Performance Comparison: Curvy Routing vs Greedy Routing

In this chapter, the performance of Curvy routing is compared against the established Greedy routing technique (described in Chapter 4), through a series of different simulation scenarios. Simulation results vary depending on the shape of the network, the precise position of the nodes, the average size of the neighbourhood of each node and the position of the sink. Cases are presented where Greedy routing performs efficiently and cases where it fails completely, because of the creation of void areas in the network and sparser topologies. Section 5.2 compares both Curvy and Greedy routing with the Curveball routing technique by Popa, et al. (2007) outlined in Chapter 2.

5.1 Scenarios

An area of 200m x 200m has been chosen to define the dimensions of the network, while its density is controlled by the number of nodes populating it. Multiple scenarios for each density have been simulated and the ones chosen to be presented below reflect the observed performance in each category. In all graphs in this chapter depicting the topology, the sink node is marked in bold font. The rest of the topologies generated for simulation can be found in the Appendix B.

5.1.1 75-node scenario, central sink

Initially, the sink node was placed in the centre of a network with a density of 1.875×10^{-03} nodes per square metre (Figure 5.1). Given the network area and the transmission range at 62 m, as discussed in Chapter 4, this number corresponds to an average number of 25 neighbours; considered as a medium sized, well connected network with plenty route choices. The metrics introduced in Section 4.2.2 were plotted to show the performance comparison between the shortest-path Greedy routing and this proposal. Figure 5.2a depicts the rate at which average (node) energy is consumed in the network at each simulation second. Due to the high density of nodes, Greedy routing performs better than Curvy routing throughout the simulation. There are many next-hop node choices available in route discovery, so that the option of face routing (Bose, Morin, Stojmenovic & Urrutia, 1999) is hardly ever invoked. Similarly, in Figure 5.2b the standard deviation of the energy dissipation rate remains lower throughout the simulation compared to Curvy routing. Greedy routing meets no difficulties finding the shortest paths, while Curvy routing selects longer paths, thus causing an increase to the spread of node residual energies in the WSN. The sudden changes in Curvy routing graphs are caused when a relay node has been depleted and alternative next-hop options are sought. Nevertheless, comparing Figures 5.2c and d it can be observed that the load is actually shifted away from the centre and spread over the topology, increasing the number of the nodes being used as relays.

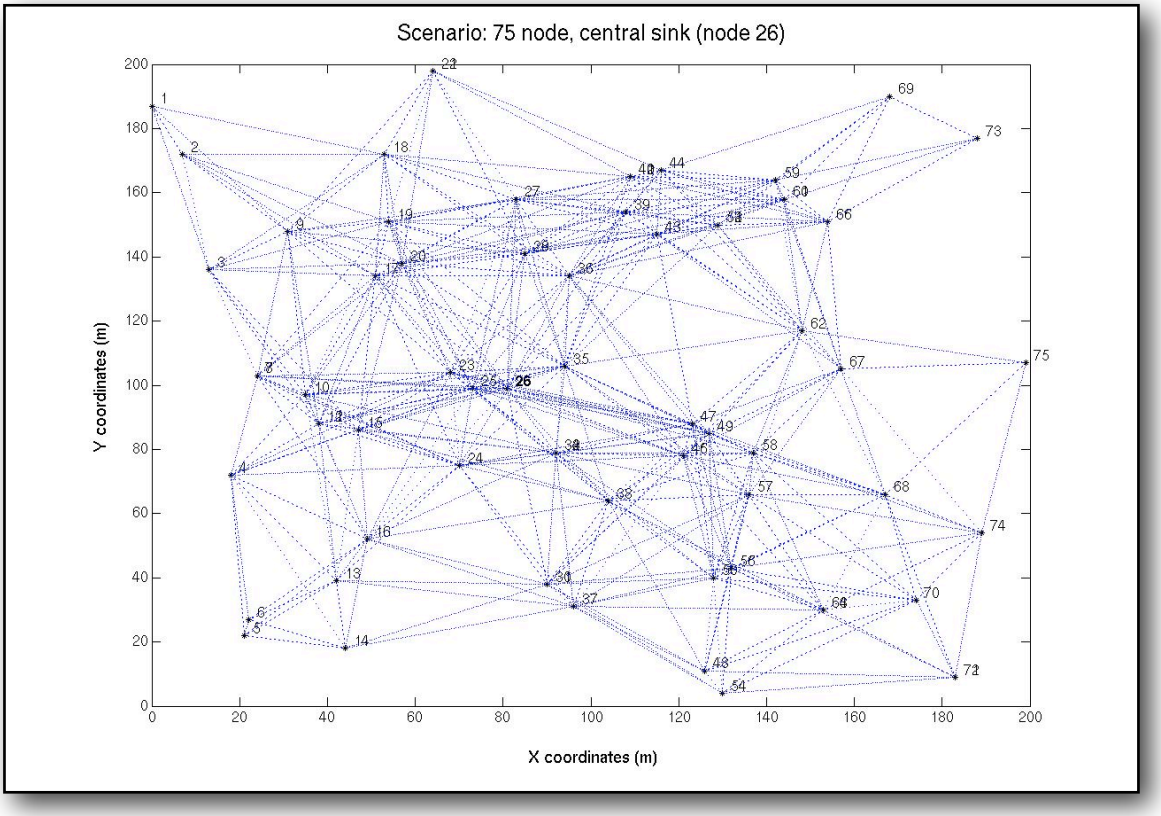


Figure 5.1: Topology with 75 nodes, central sink (node 26)

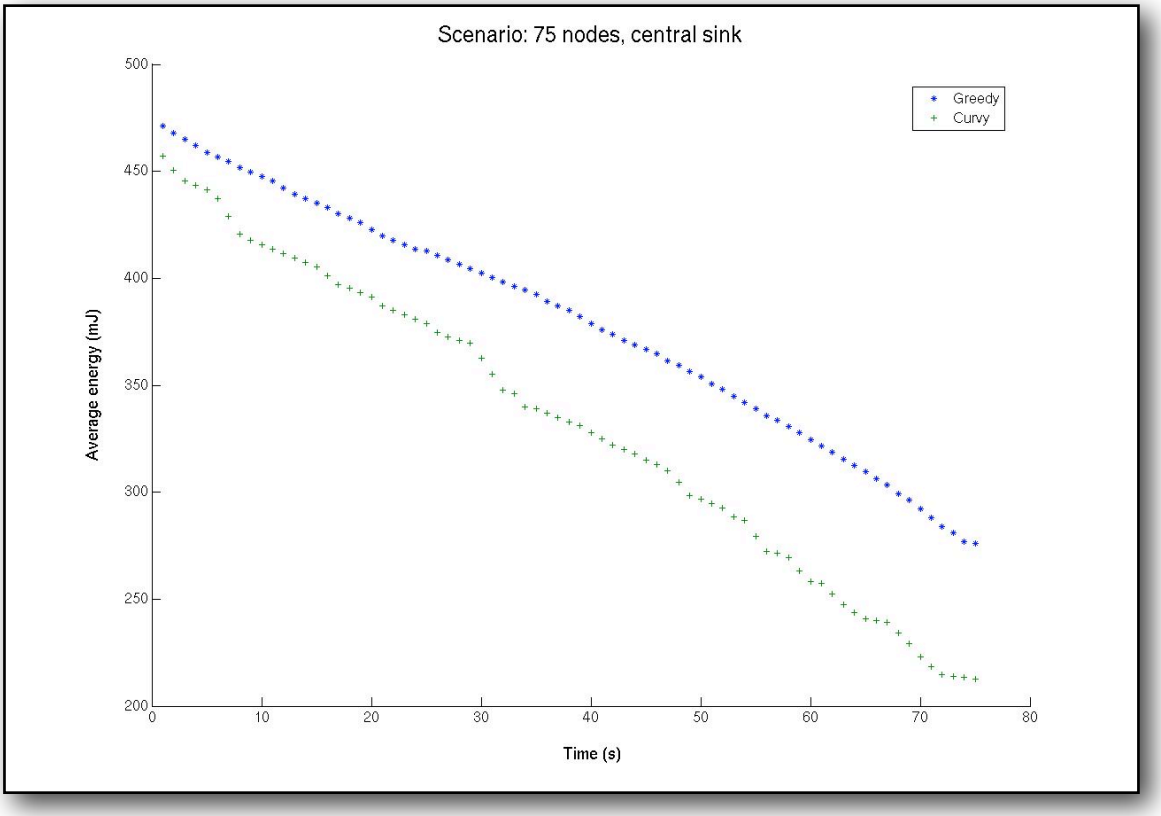


Figure 5.2a: Average energy vs Time - 75 nodes, central sink

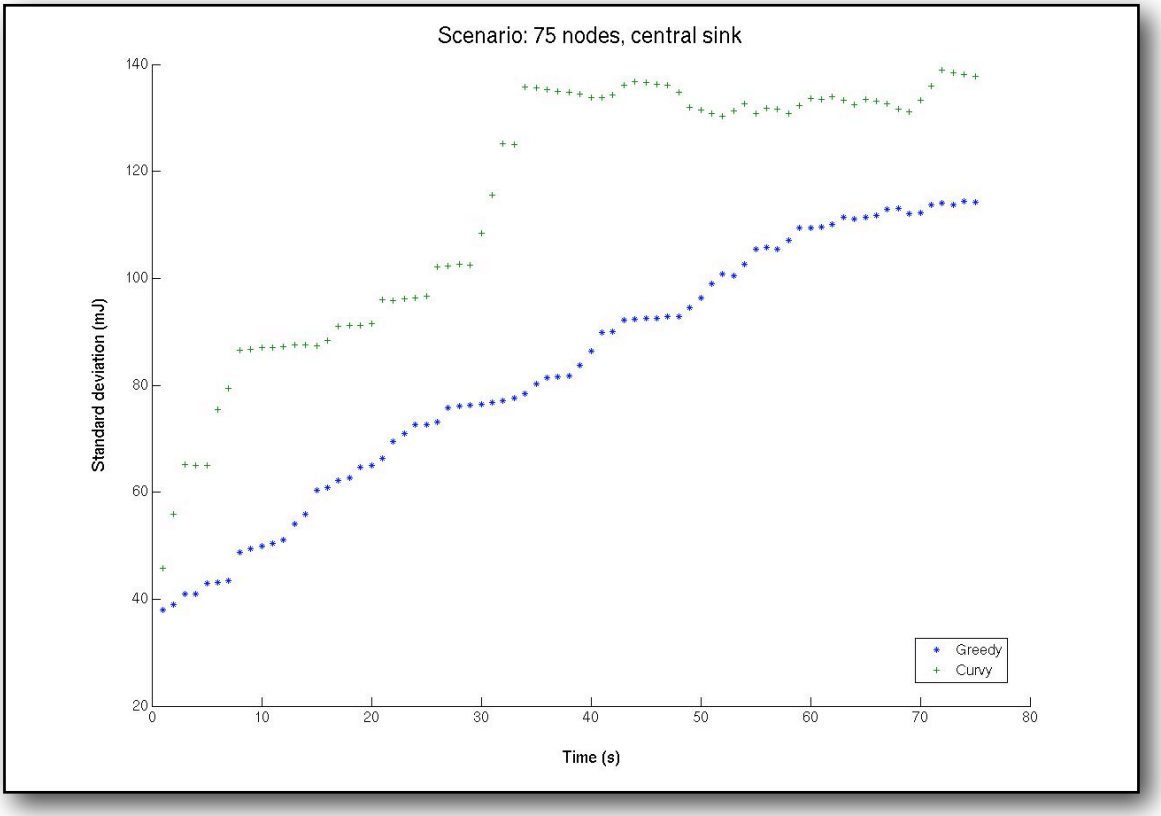


Figure 5.2b: Standard deviation of energy dissipation vs Time - 75 nodes, central sink

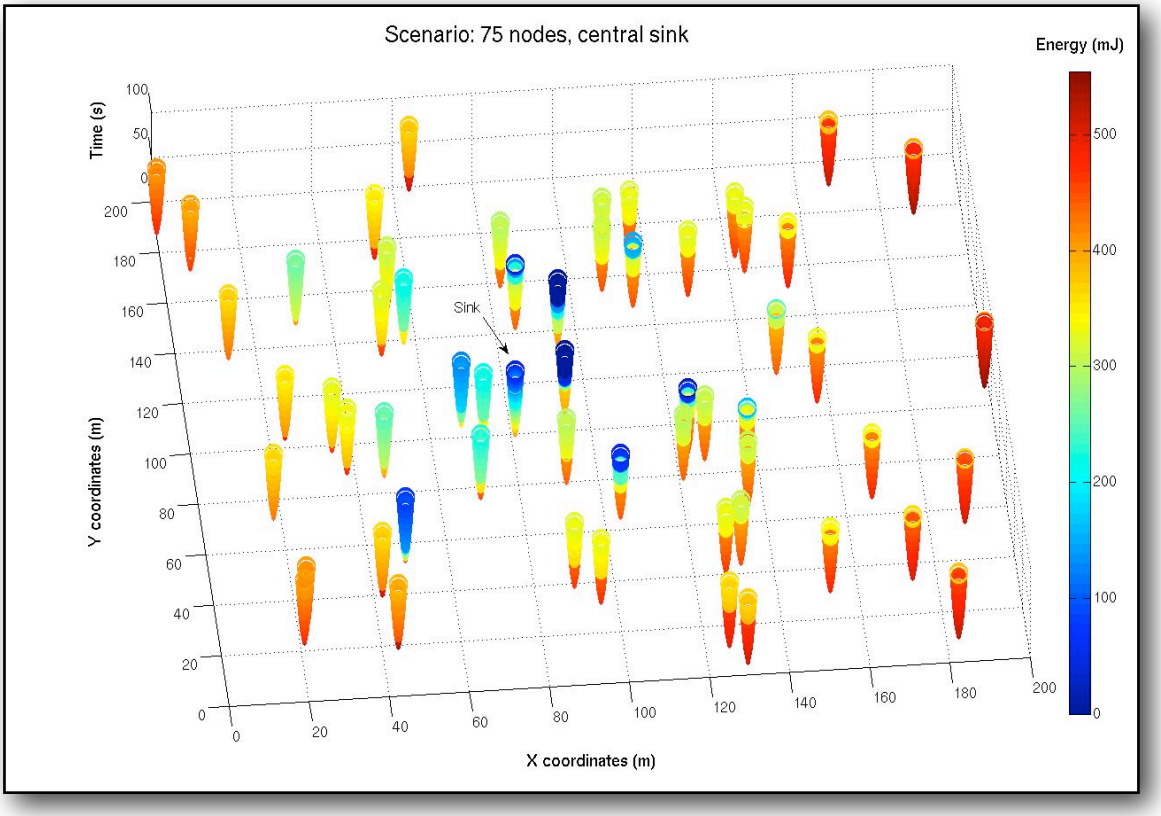


Figure 5.2c: Residual energy map, Greedy routing - 75 nodes, central sink

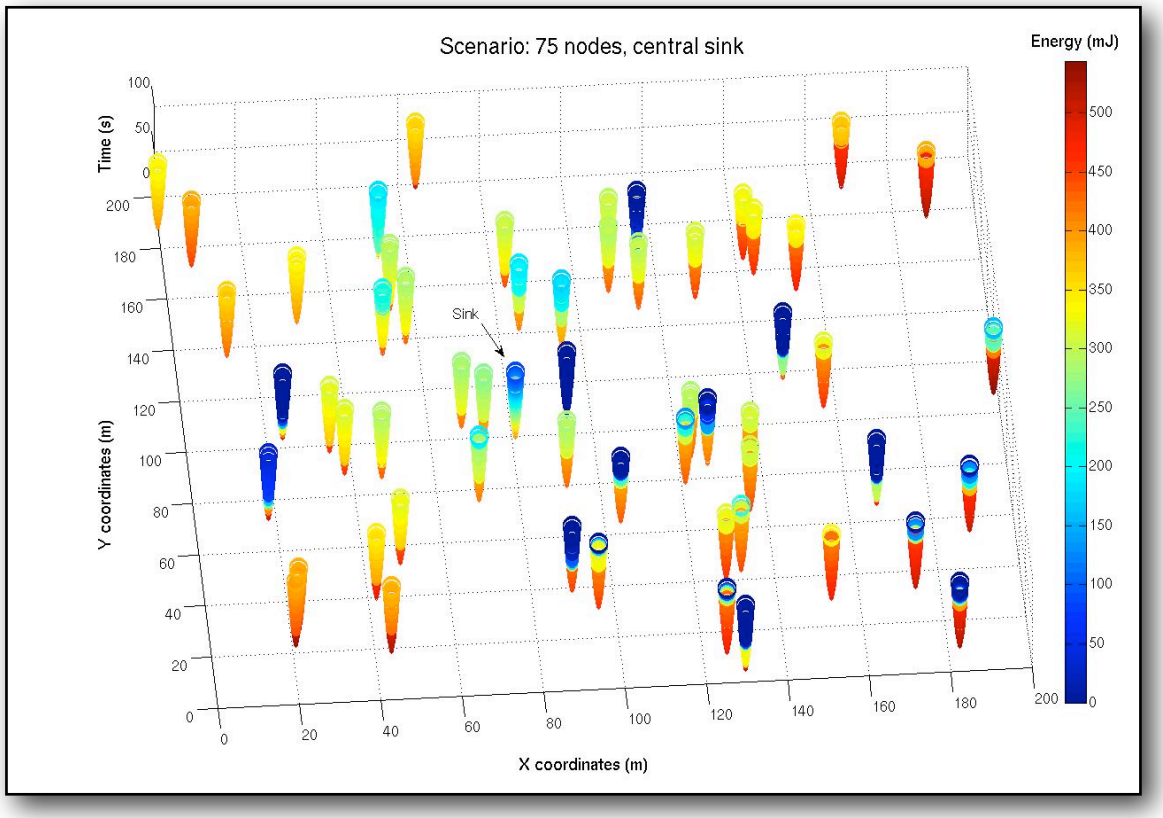


Figure 5.2d: Residual energy map, Curvy routing - 75 nodes, central sink

5.1.2 50-node scenario, central sink

The next simulation had a decreased density of 1.25×10^{-03} nodes per square metre, keeping the sink node in the centre as shown in Figure 5.3. This time the network is a little sparser than the previous scenario and each node has an average of 15-20 neighbours. The simulation was run again with the same parameters, except that the number of nodes is now 50. The performance results for Curvy routing are slightly improved, but Greedy routing still maintains a significant advantage as can be seen in Figures 5.4a and b. Both protocols aim to maintain balance in the network load and, ideally, to decrease the standard deviation. Figure 5.4b shows that both protocols can achieve that, since there are enough next-hop options to control the mean energy dissipation. The advantages of Curvy routing in maintaining a better geographical spread of paths are also maintained as before (Figures 5.4c and d).

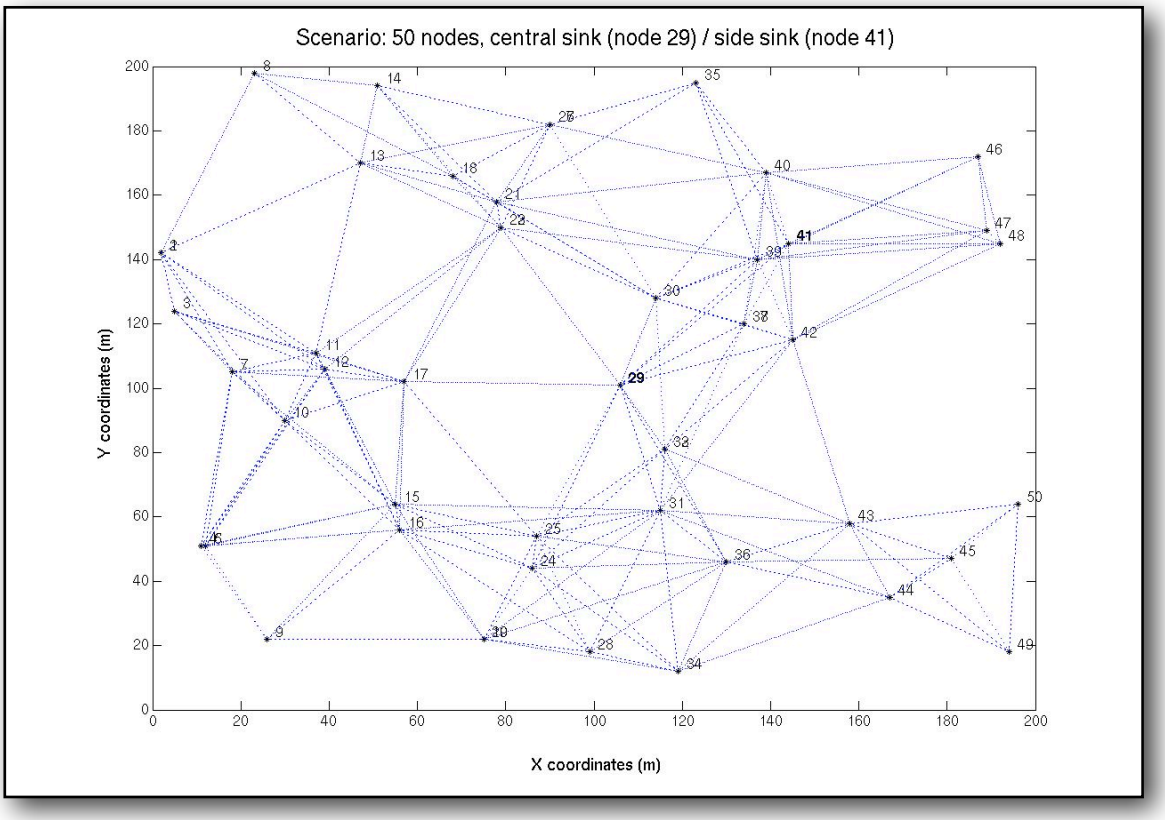


Figure 5.3: Topology with 50 nodes, central sink (node 29) / side sink (node 41)

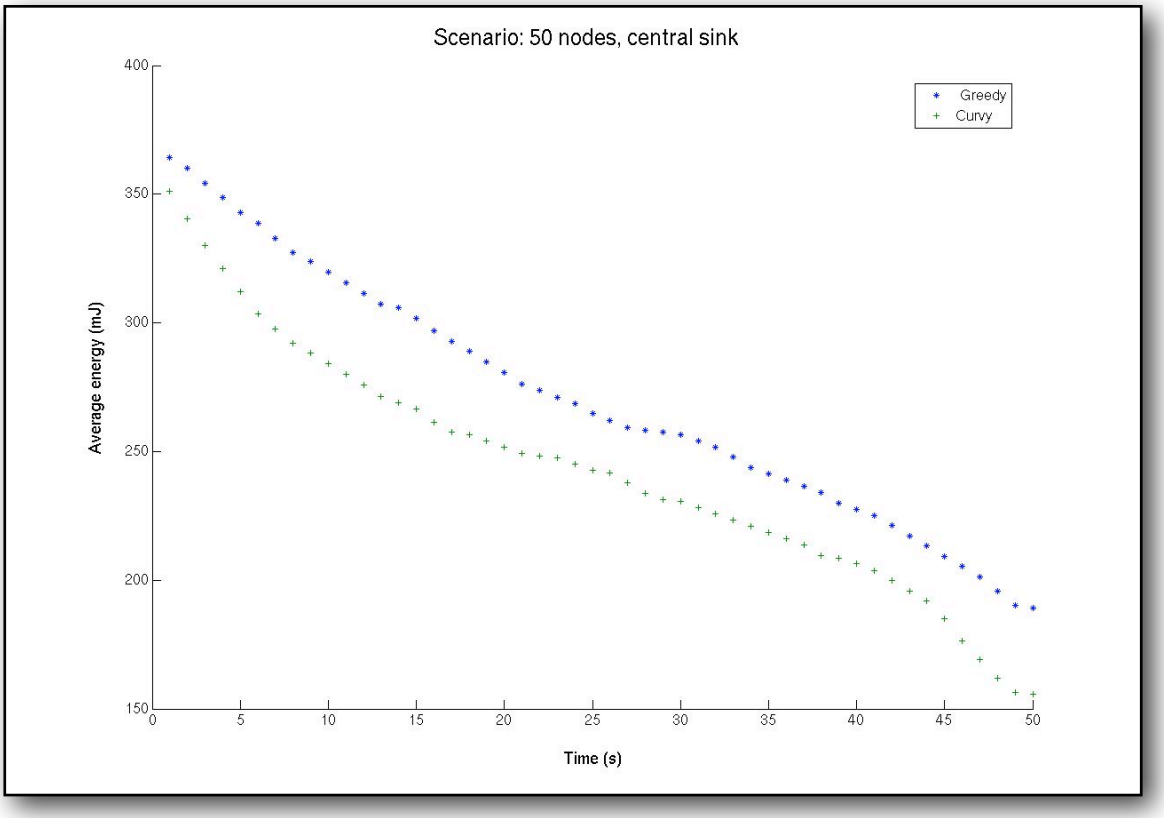


Figure 5.4a: Average energy vs Time - 50 nodes, central sink

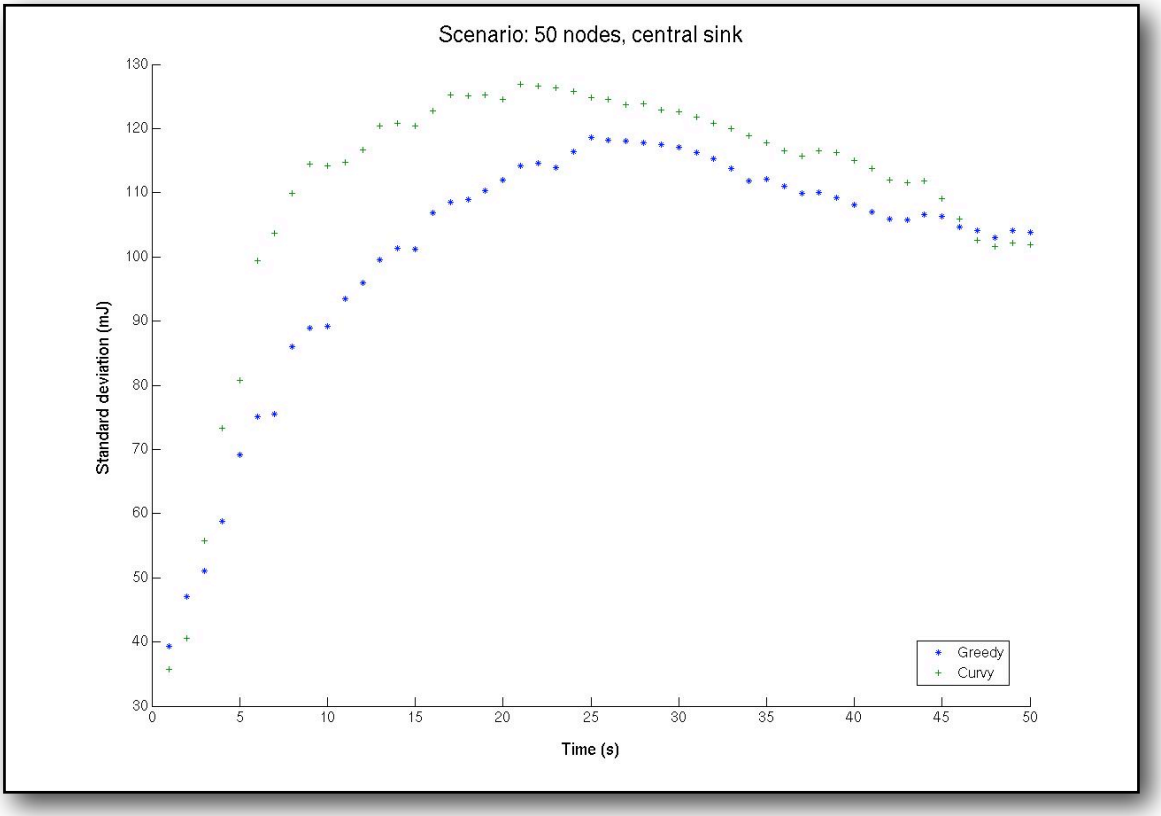


Figure 5.4b: Standard deviation of energy dissipation vs Time - 50 nodes, central sink

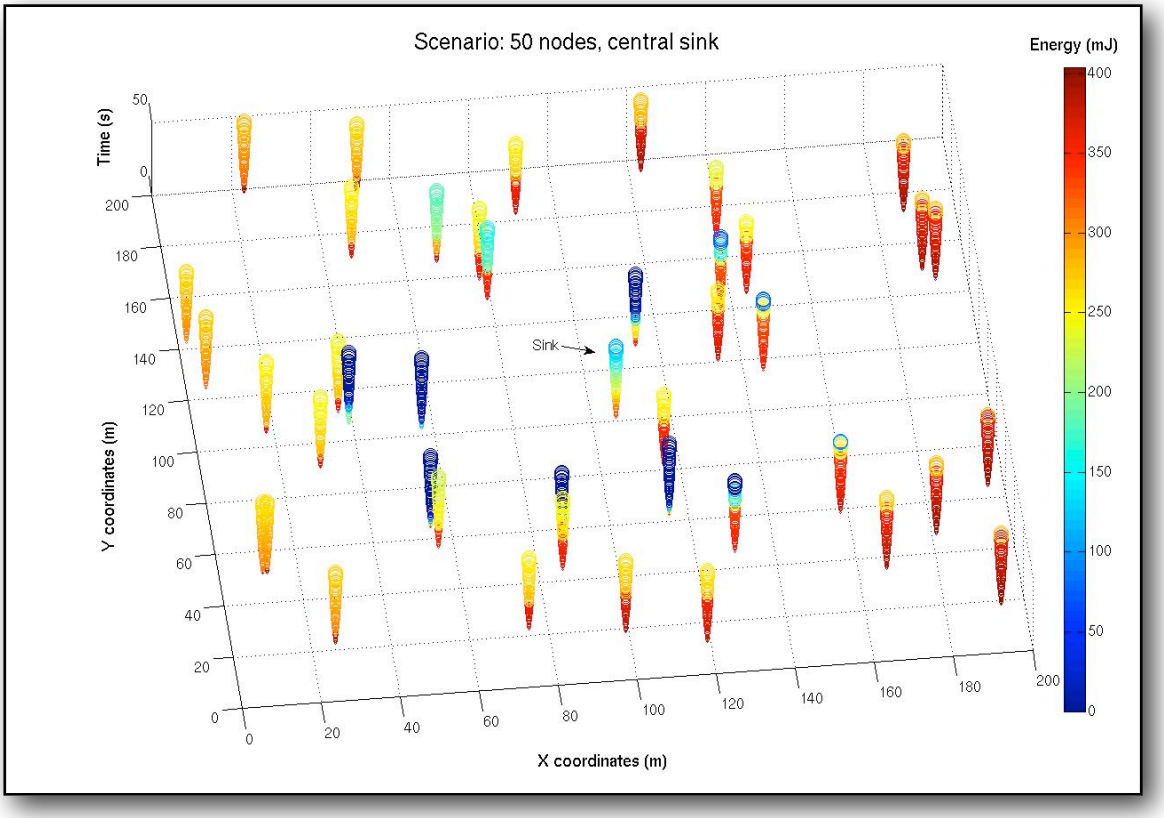


Figure 5.4c: Residual energy map, Greedy routing - 50 nodes, central sink

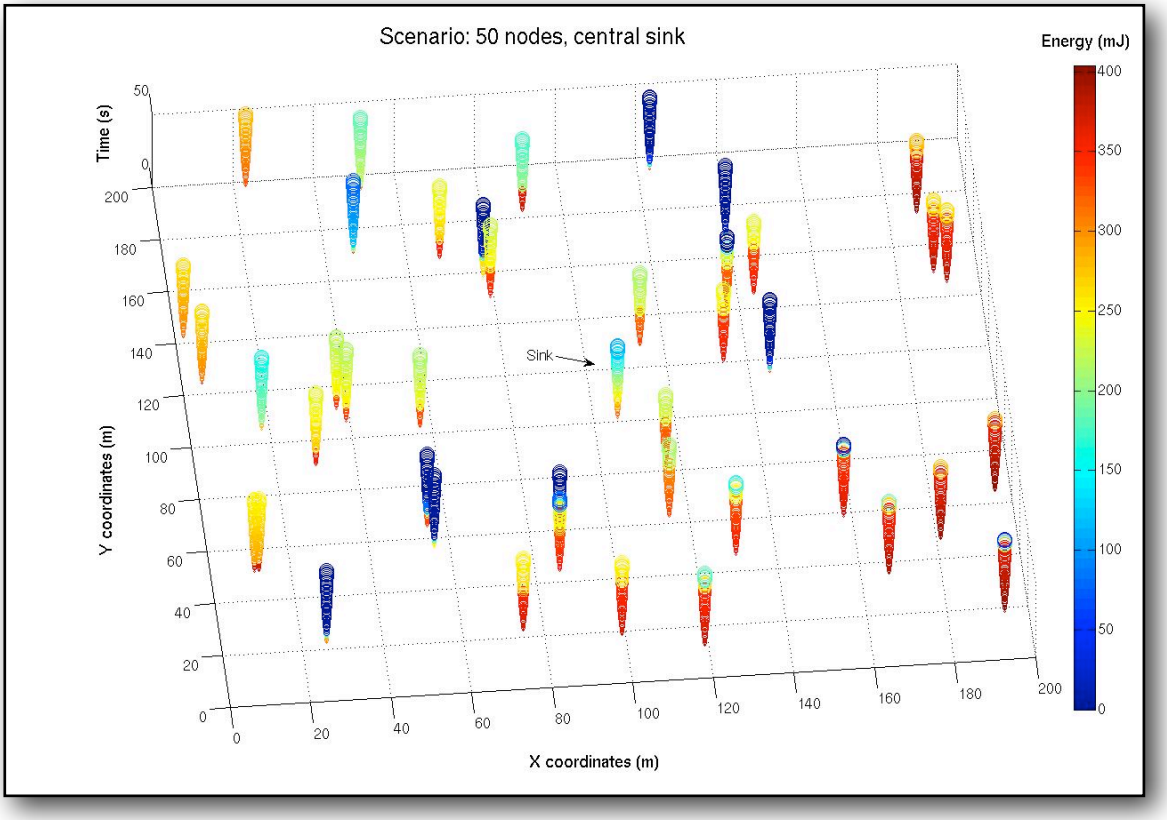


Figure 5.4d: Residual energy map, Curvy routing - 50 nodes, central sink

5.1.3 50-node scenario, sink near edge of network

Moving the sink to a different position and otherwise keeping the exact same topology (Figure 5.3), proved to be more favourable for Curvy routing. This is certainly more realistic, since in practical WSN deployments sink nodes are more likely to be placed at an accessible "end" location. A typical example of WSN is the PermaSense project (Talzi, Hasler, Gruben & Tschudin, 2007) deploying a sensor field in the Swiss Alps. Figure 5.5 shows the system framework used, placing the sink node at the very edge of the field.

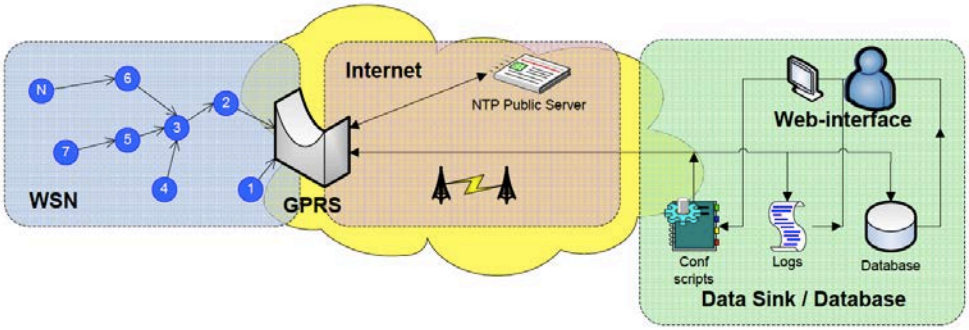


Figure 5.5: PermaSense system framework (Talzi, Hasler, Gruben & Tschudin, 2007)

As seen in Figure 5.6a, Greedy routing completely fails, suffering from a network partition after only 29 simulation seconds. The face routing option, available here as before, did not help much with the network survivability. Curvy routing keeps the network alive for almost double the simulation time, stabilising the average energy depletion rate. This is also evident in Figure 5.6b, where the standard deviation keeps increasing until network partition occurs for Greedy routing, while in Curvy routing it stabilises after 10 simulation seconds and even decreases towards the end of the simulation run. Comparing the network load (Figures 5.6c and d) shows that in Greedy routing, nodes at key central positions become depleted extremely fast, leaving no routing choices for the algorithm to exercise.

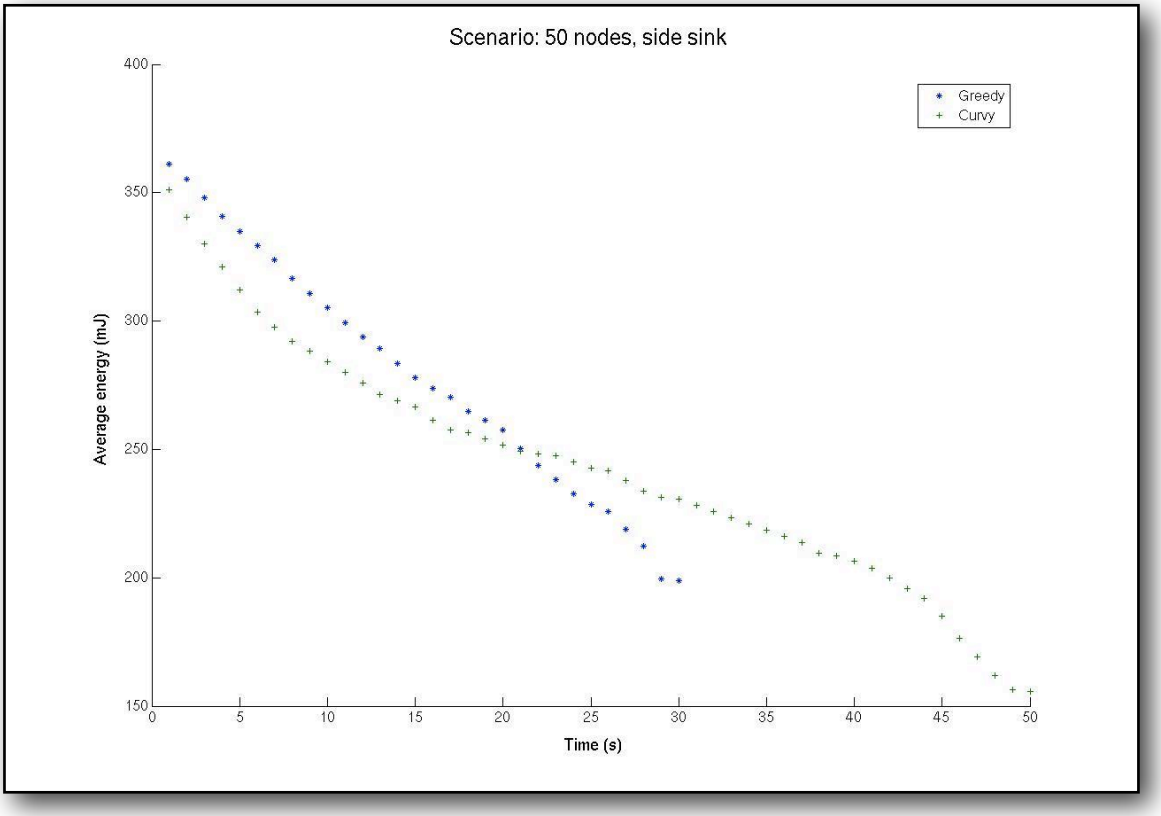


Figure 5.6a: Average energy vs Time - 50 nodes, side sink

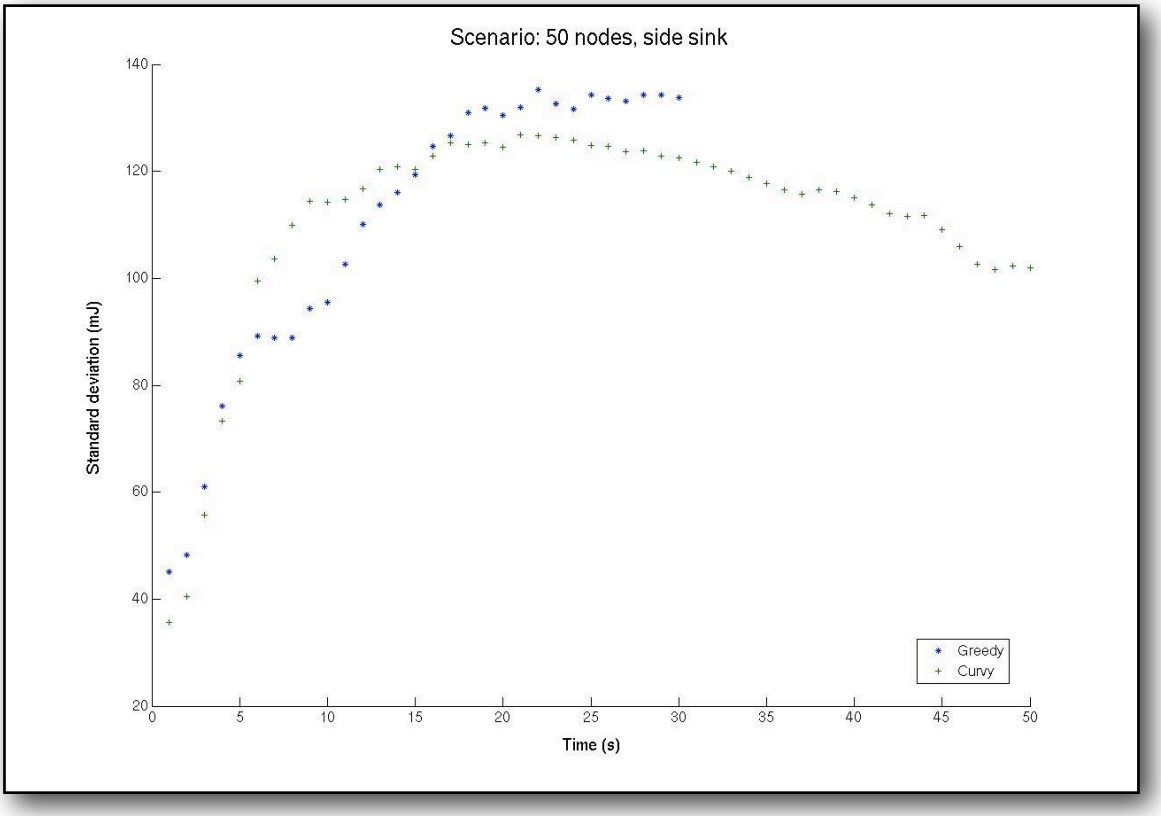


Figure 5.6b: Standard deviation of energy dissipation vs Time - 50 nodes, side sink

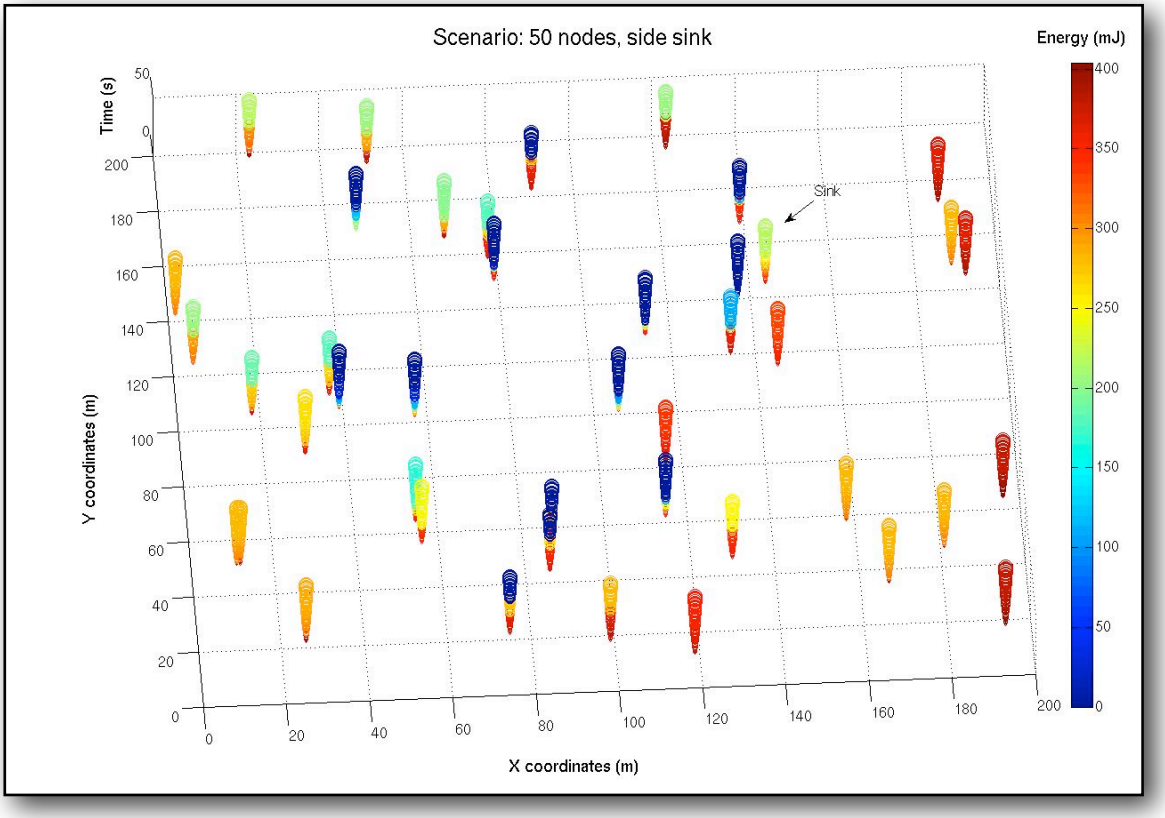


Figure 5.6c: Residual energy map, Greedy routing - 50 nodes, side sink

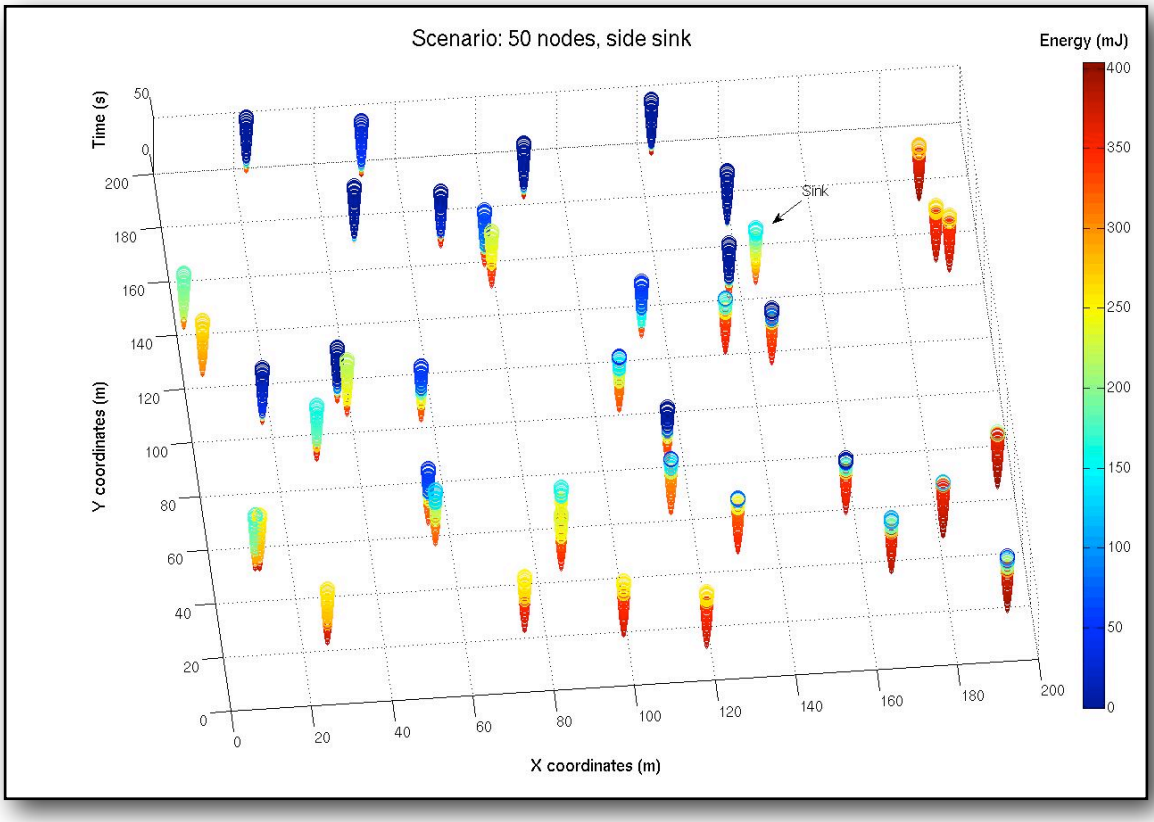


Figure 5.6d: Residual energy map, Curvy routing - 50 nodes, side sink

5.1.4 50-node scenario 2, sink near edge of network

Several deployment topologies were randomly generated with the same density ($1.25x10^{-03}$ nodes per square metre; average of 15-20 neighbours for each node) and the sink on the side of the topology (Figure 5.7), all confirming the performance advantage of the proposed algorithm. More precisely, the number of the topologies generated in Matlab (randomly) was around 30 and are included in Appendix B. Those representative topologies were simulated including ones with void areas, different network shape, uneven node distribution. The results produced demonstrate an even stronger performance for Curvy routing, which is reflected by the topologies chosen to be presented here. Observations on those categories mentioned above are discussed in Sections 5.3 and 7.2.

Figures 5.8a and b show the average energy and standard deviation of energy dissipation respectively, where network partition for Greedy routing is experienced as early as 17 simulation seconds. On the contrary, Curvy routing copes well circumventing the void areas and keeping the network alive for significantly longer, while spreading the load a lot more evenly (Figures 5.8c and d).

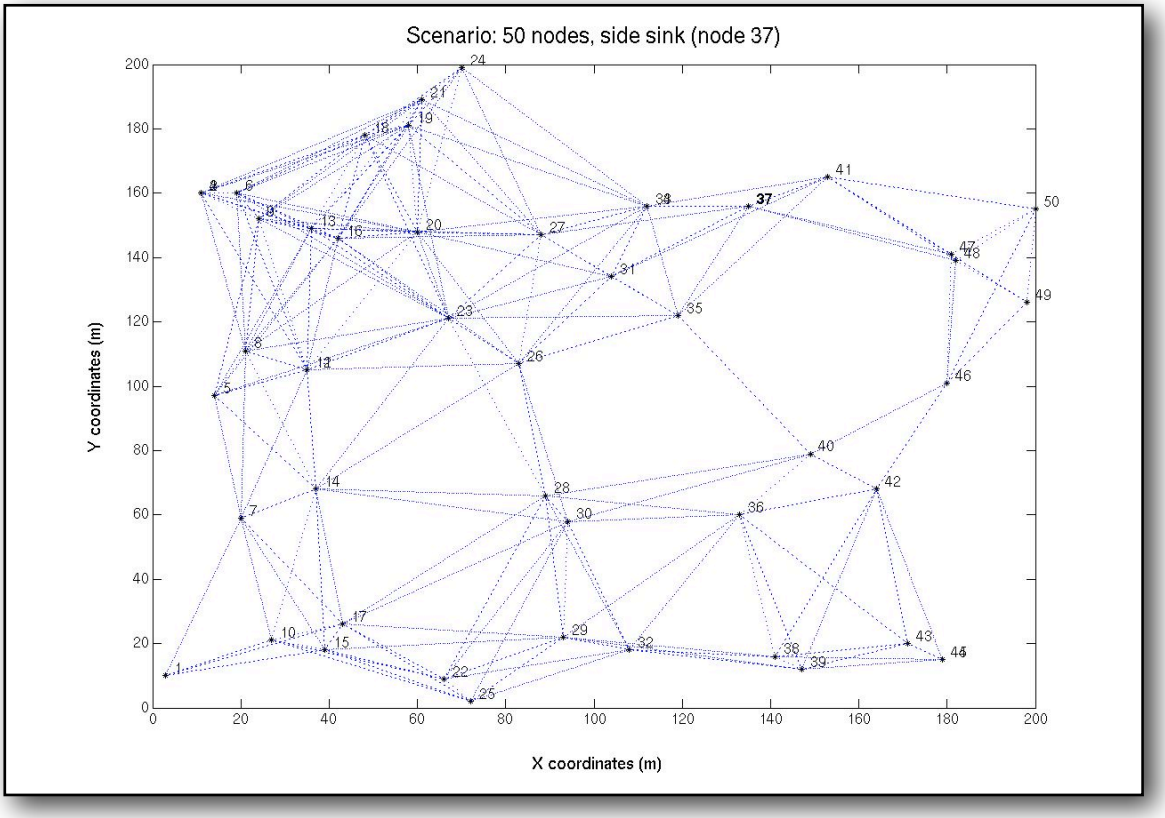


Figure 5.7: Topology with 50 nodes, side sink (node 37)

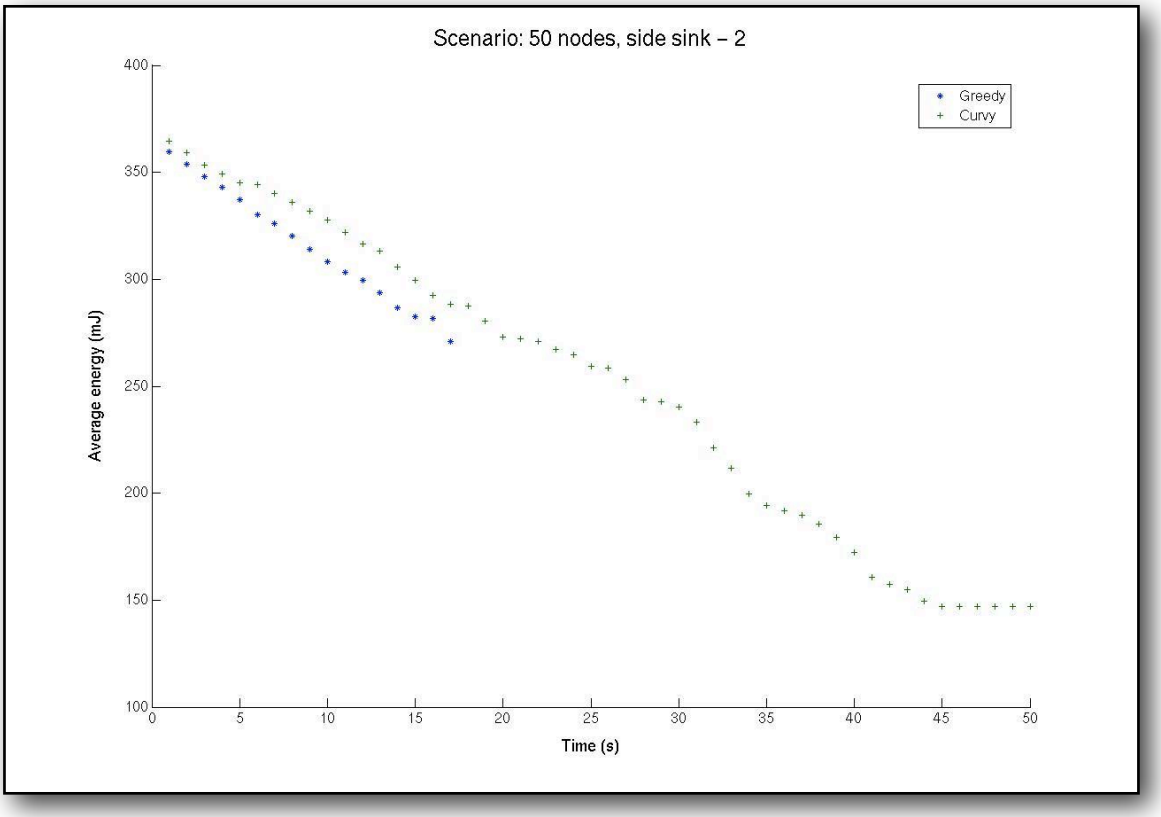


Figure 5.8a: Average energy vs Time - 50 nodes, side sink

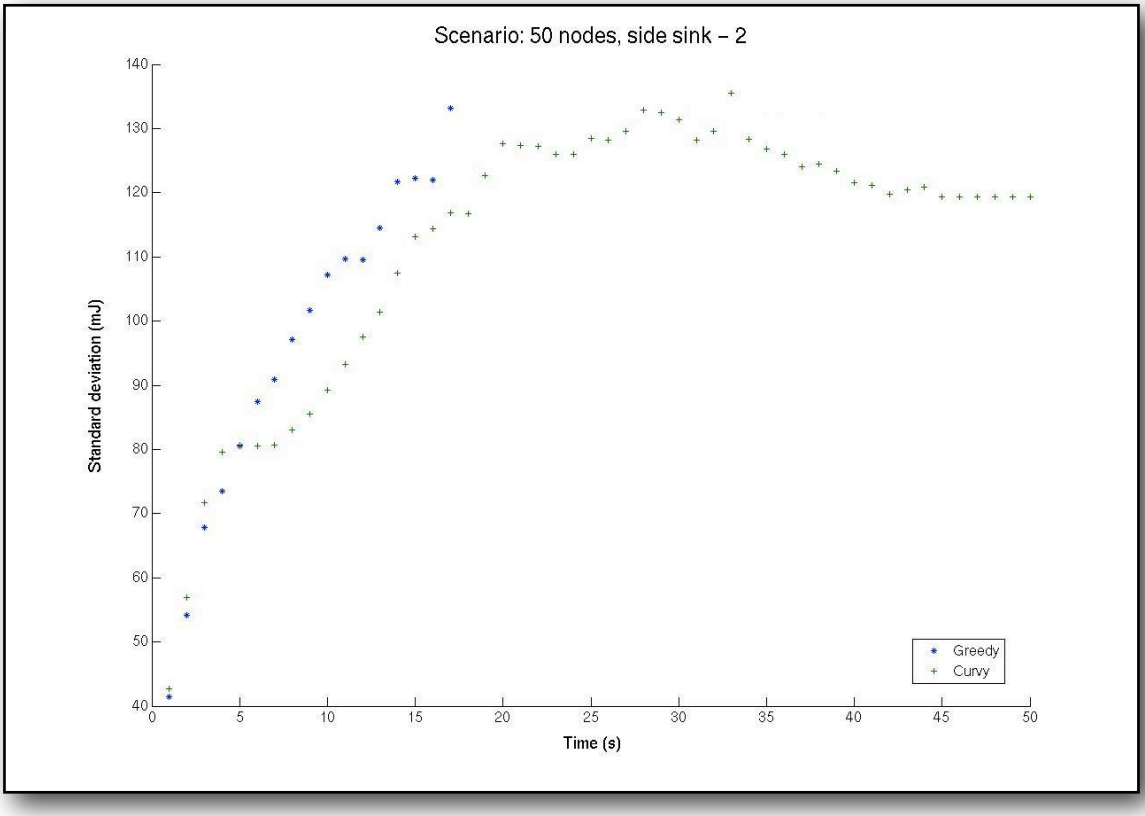


Figure 5.8b: Standard deviation of energy dissipation vs Time - 50 nodes, side sink

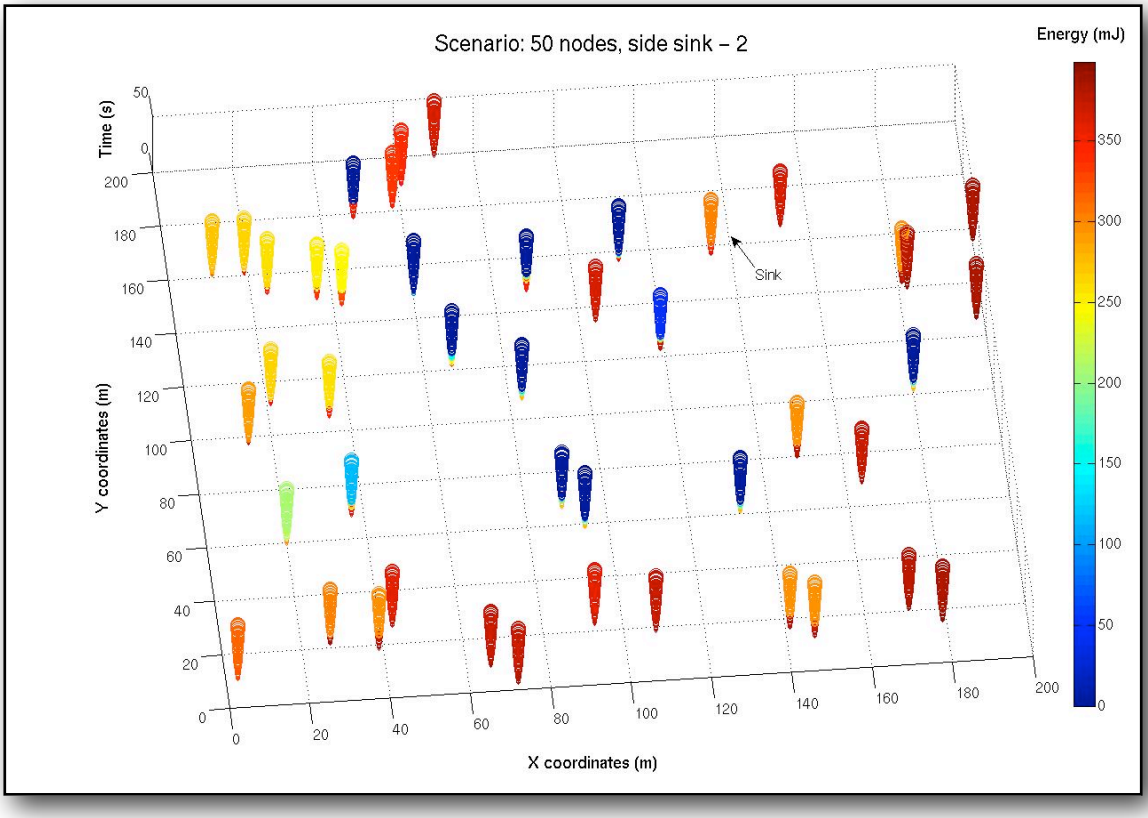


Figure 5.8c: Residual energy map, Greedy routing - 50 nodes, side sink

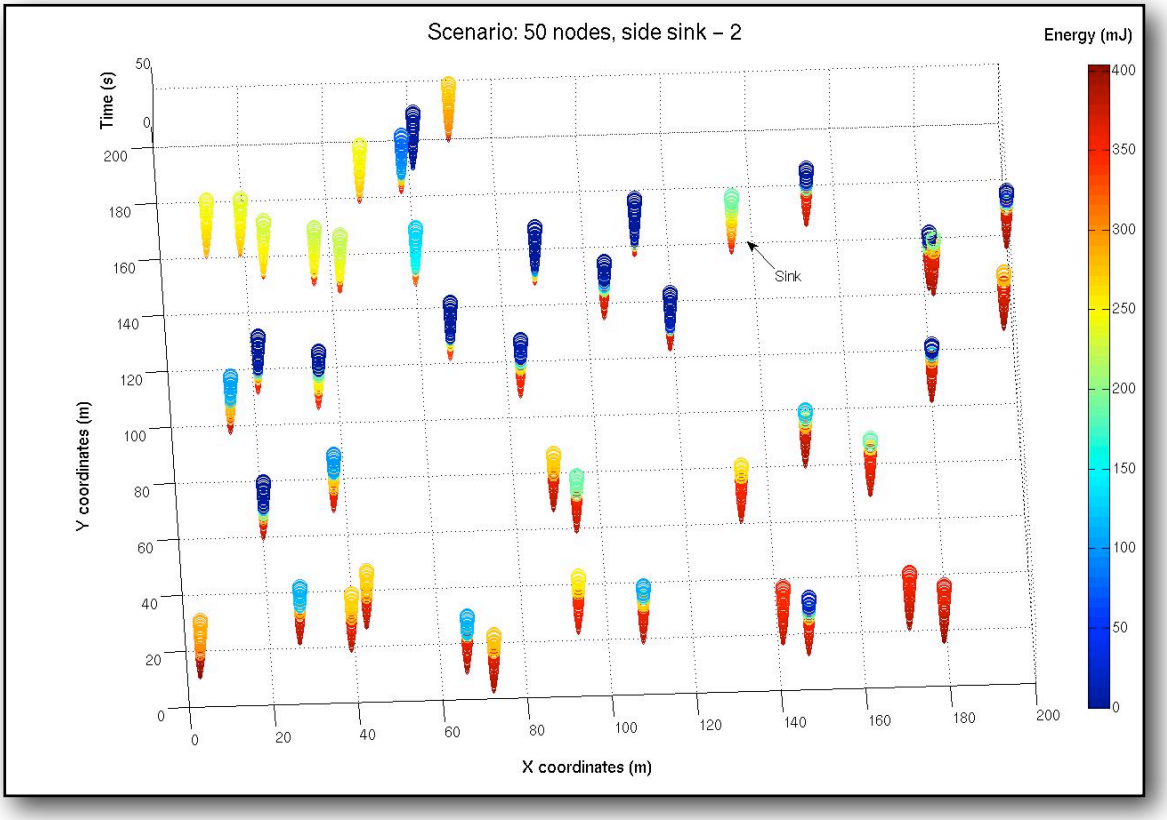


Figure 5.8d: Residual energy map, Curvy routing - 50 nodes, side sink

5.1.5 38-node scenario, sink near edge of network

Pushing the Curvy routing algorithm to the limits and forcing it to perform on even sparser networks of 0.95×10^{-3} nodes per square metre (corresponding to 10-15 average neighbours per node), proved to be even more advantageous. As before, a similar number of randomly generated topologies was created, but this time the categories were found to be fewer (about 5) due to the limited number of nodes in the area. In some networks, connectivity was hard to be established and for this reason, they were not included in the simulations. The rest of the categories were simulated as normal and observations on those are again discussed in sections 5.3 below and 7.2.

Greedy routing failed again after 20 simulation seconds, while Curvy routing kept the average energy depletion rate steady (Figure 5.10a) and the standard deviation

lower than any other scenario (Figure 5.10b). Due to the node position distribution and the associated topology, shown in Figure 5.9, the void area in the middle of the network plays a catalytic role in the spreading of the network load. In some cases, it is a matter of the random order the paths are initiated with and that explains the interruptions in both the average energy and its standard deviation. Once the difficulty in discovering a certain path towards the sink has been overcome, balance in the load can be restored by a "less-demanding" route. Greedy routing is finding it hard to overcome the existing void area, while Curvy routing meets no difficulty in energy-efficient route discovery (Figures 5.10c and d).

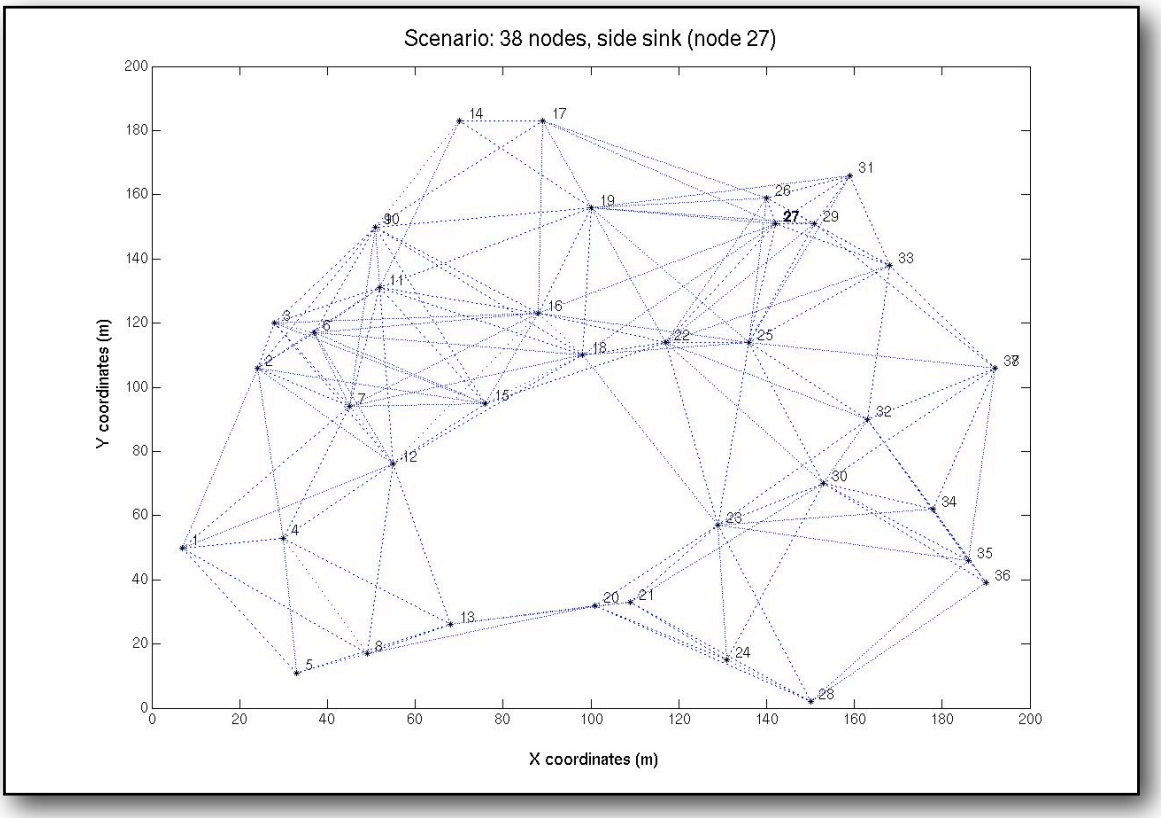


Figure 5.9: Topology with 38 nodes, side sink (node 27)

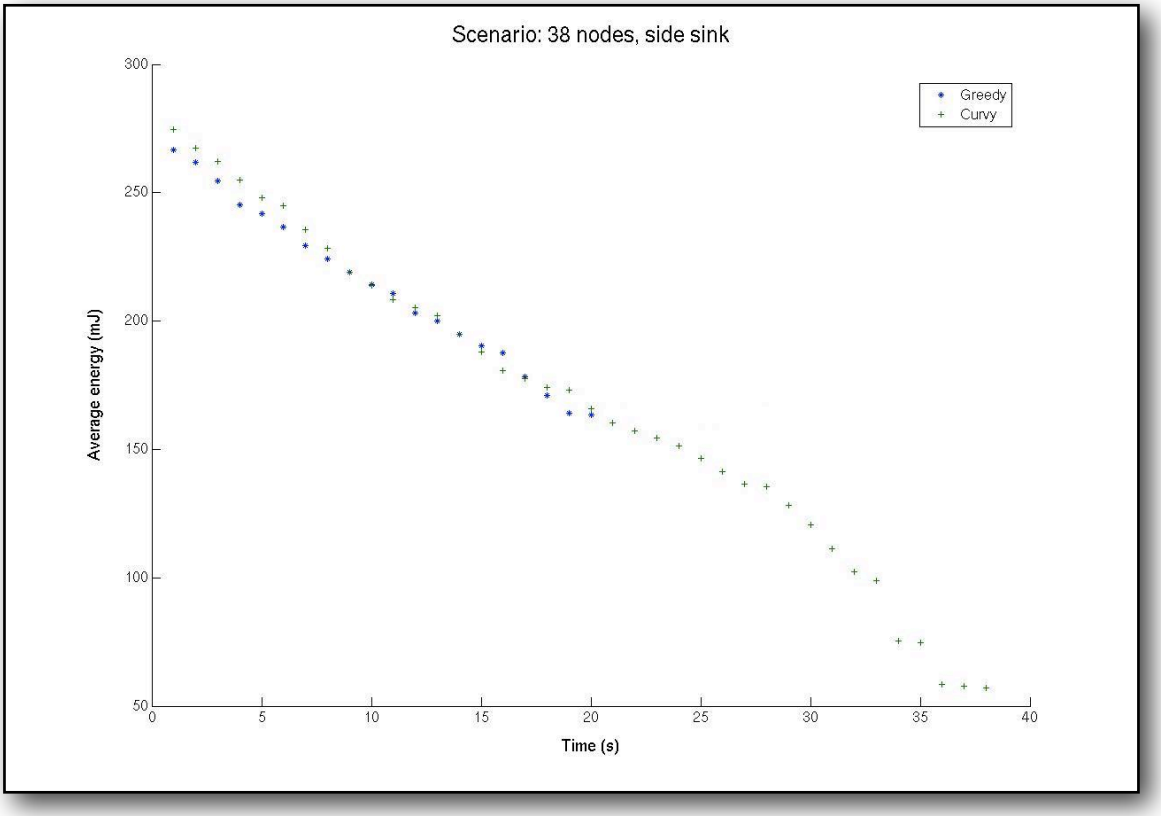


Figure 5.10a: Average energy vs Time - 38 nodes, side sink

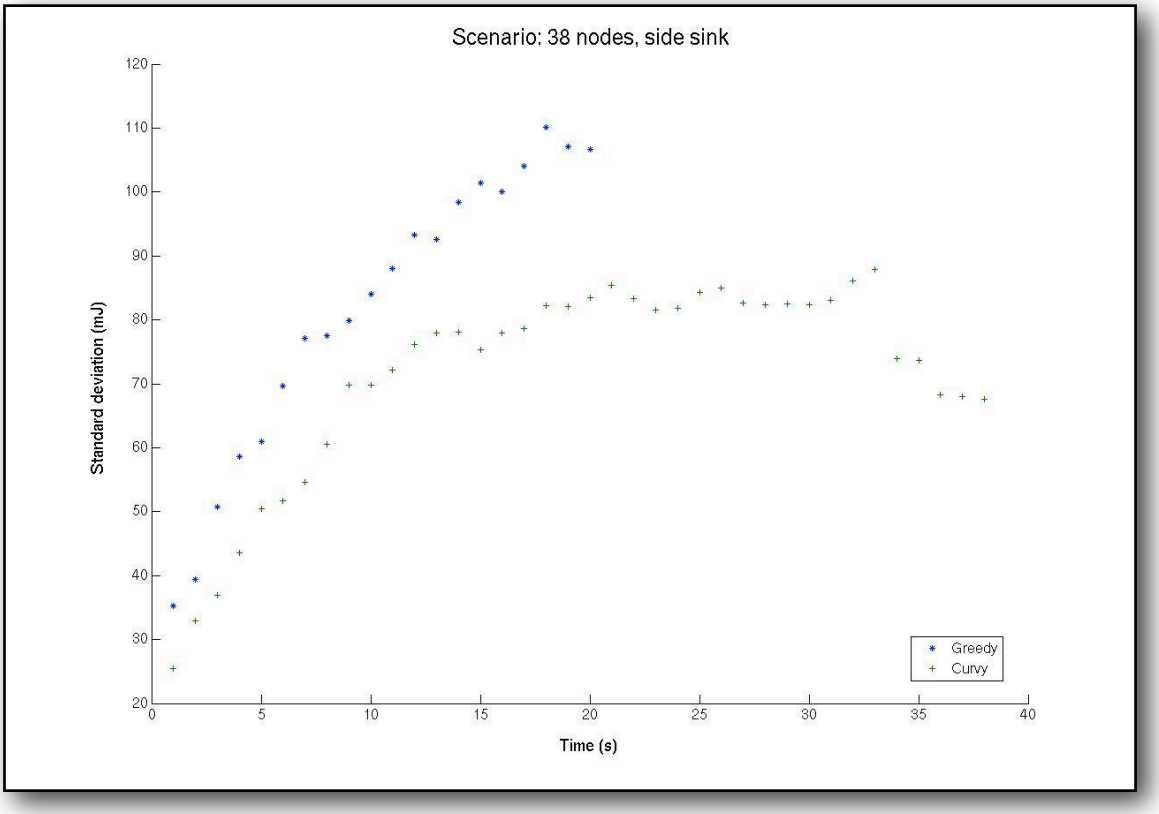


Figure 5.10b: Standard deviation of energy dissipation vs Time - 38 nodes, side sink

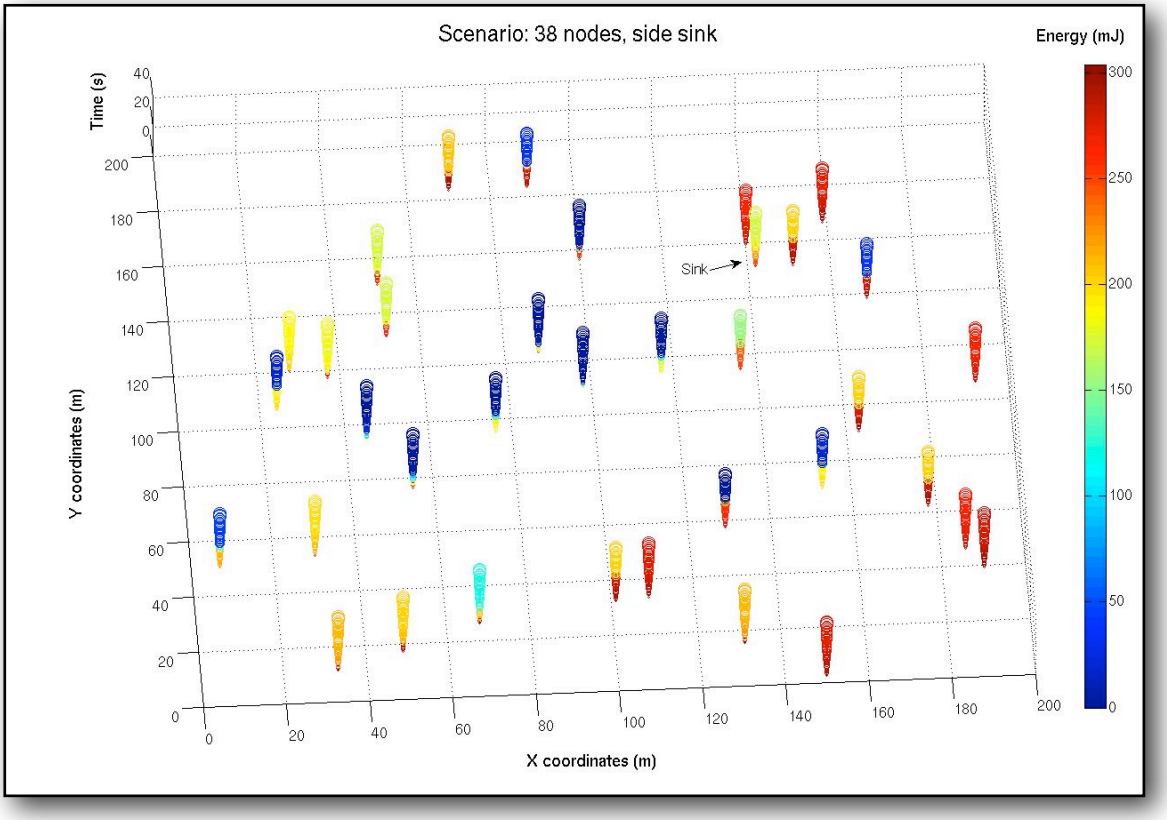


Figure 5.10c: Residual energy map, Greedy routing - 38 nodes, side sink

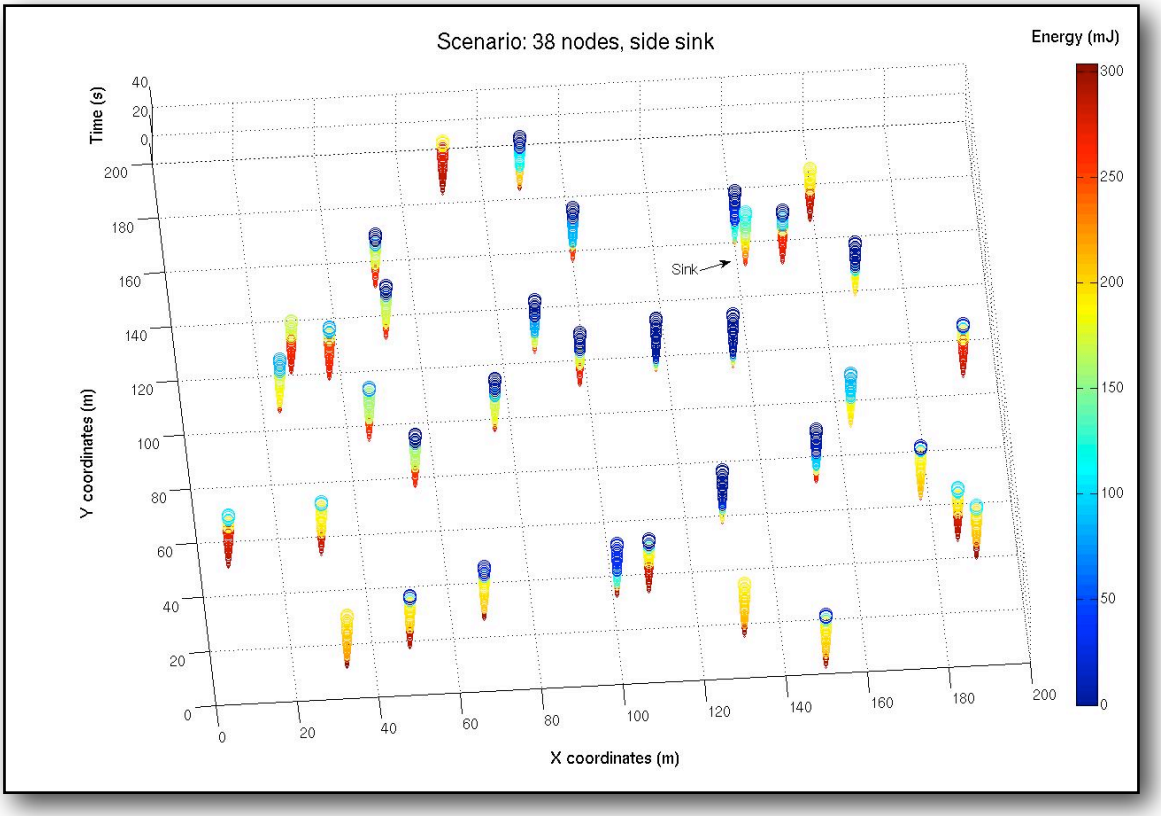


Figure 5.10d: Residual energy map, Curvy routing - 38 nodes, side sink

5.2 Comparison with Curveball routing

Curveball routing (Popa et al., 2007) was reproduced using Popa's source code for the sphere mapping of the nodes and it was incorporated into the network simulator (Chapter 4) in order to perform Greedy routing on the 3D coordinates, as described in Chapter 2. The rest of the parameters remain the same and scenarios with various densities and shapes were simulated as previously with the sink near the edge of the network, choosing a few examples to present below.

5.2.1 50-node scenario, sink near edge of network

Figure 5.11 shows one of the scenarios with 50 randomly distributed nodes in a 200m x 200m network area.

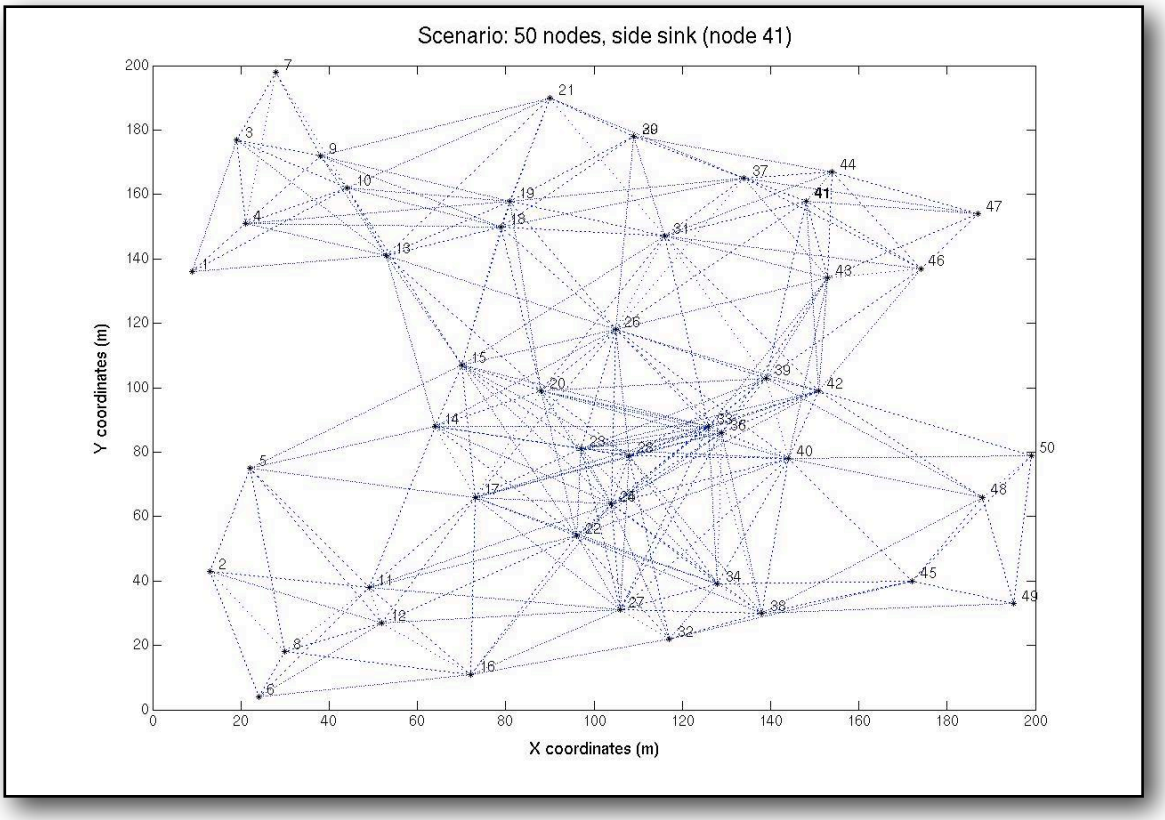


Figure 5.11: Topology with 50 nodes, side sink (node 41)

The average energy per node is shown in Figure 5.12a. Curveball routing faces network partition after 26 simulation seconds, Greedy routing fails after 35 simulation seconds, while Curvy routing finishes the simulation normally. It can also be seen that Curveball routing is incapable of keeping the average energy steady, which is believed to be caused by the position the nodes are mapped on the sphere. Figure 12b represents the standard deviation of the energy dissipation for Curveball, Greedy and Curvy routing. Figures 12c, d and e show the network load spread across the network for Greedy, Curvy and Curveball routing respectively. This scenario does not differ much for Greedy and Curvy routing and it shows as previously how Curvy routing balances the load more evenly. Curveball routing in Figure 12e handles the load better than simple Greedy routing, but the crowded centre effect is still not tackled efficiently and nodes at the centre can be seen to have depleted quite quickly and simultaneously, leaving the network without intermediate relays.

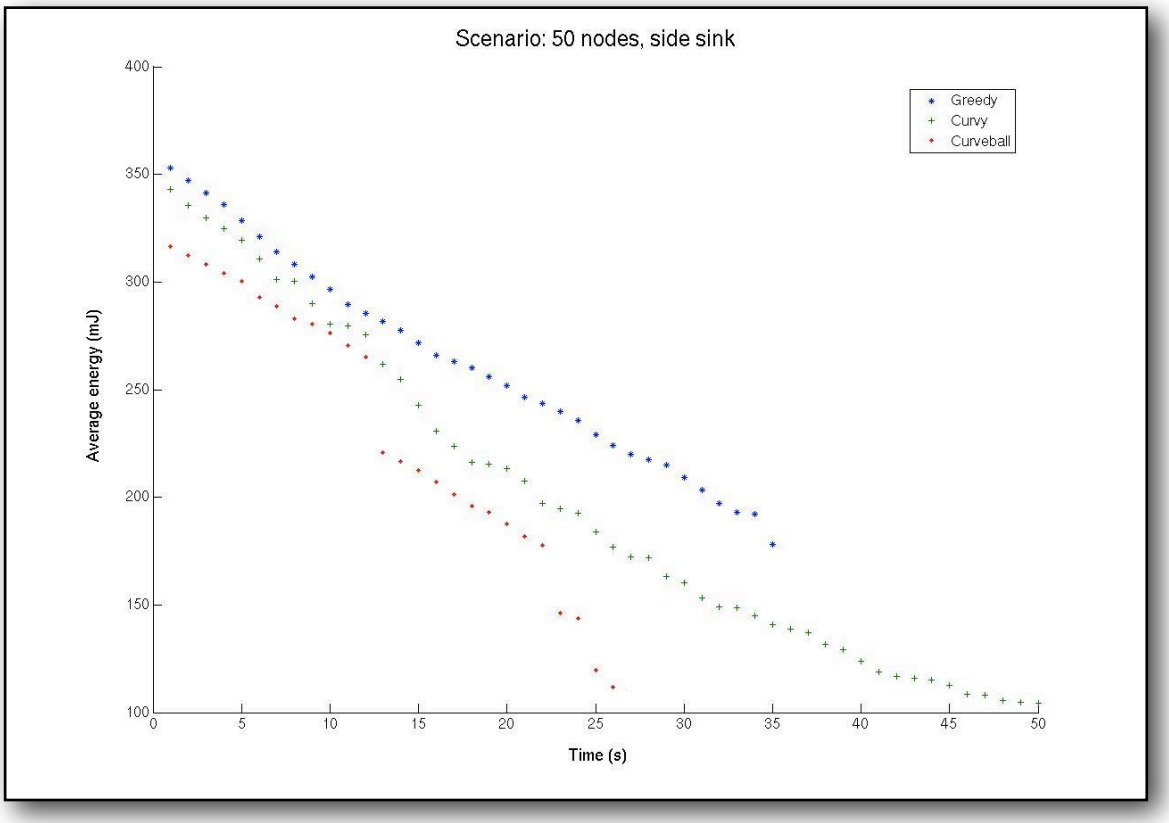


Figure 5.12a: Average energy vs Time - 50 nodes, side sink

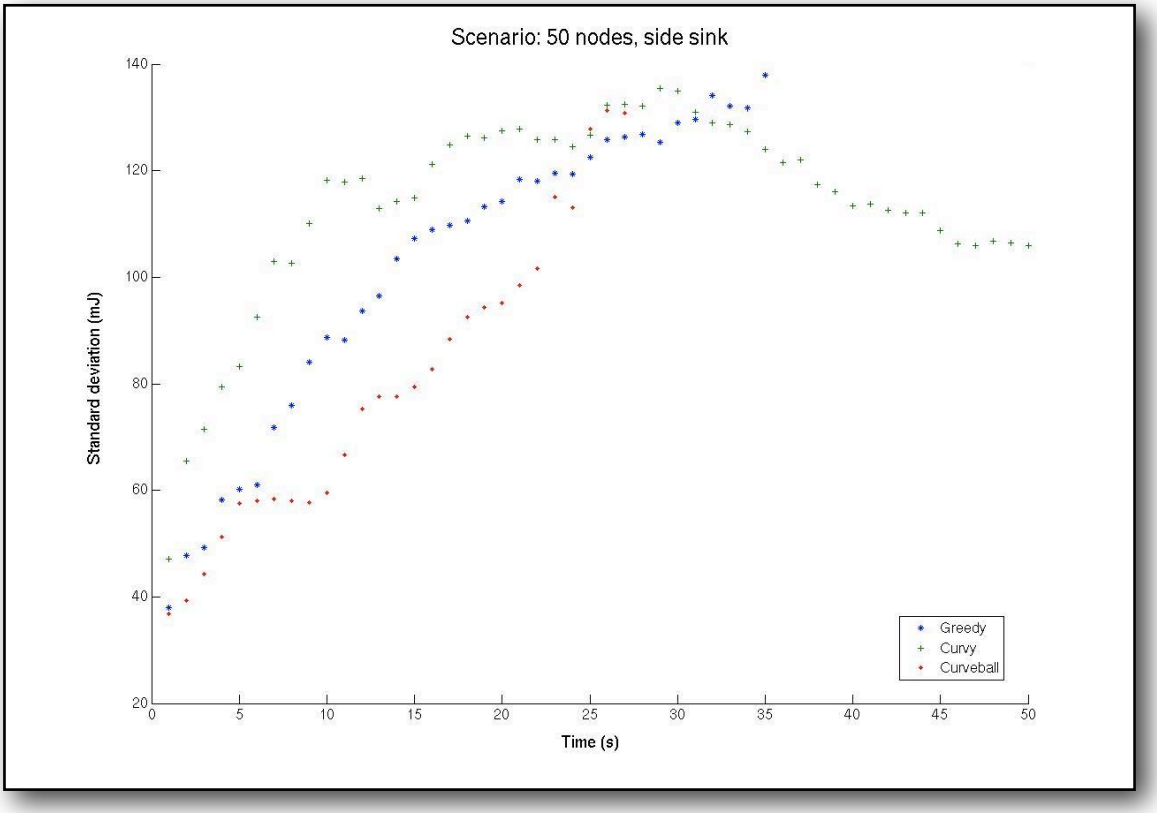


Figure 5.12b: Standard deviation of energy dissipation vs Time - 50 nodes, side sink

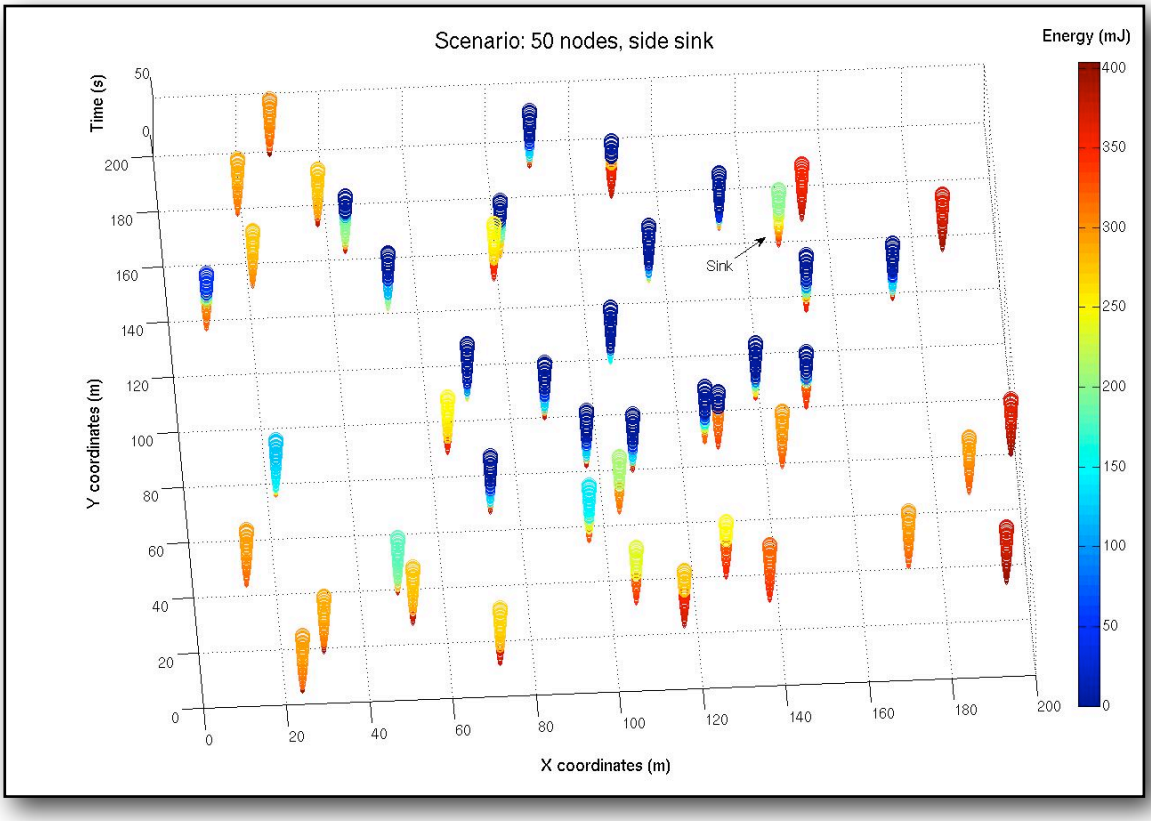


Figure 5.12c: Residual energy map, Greedy routing - 50 nodes, side sink

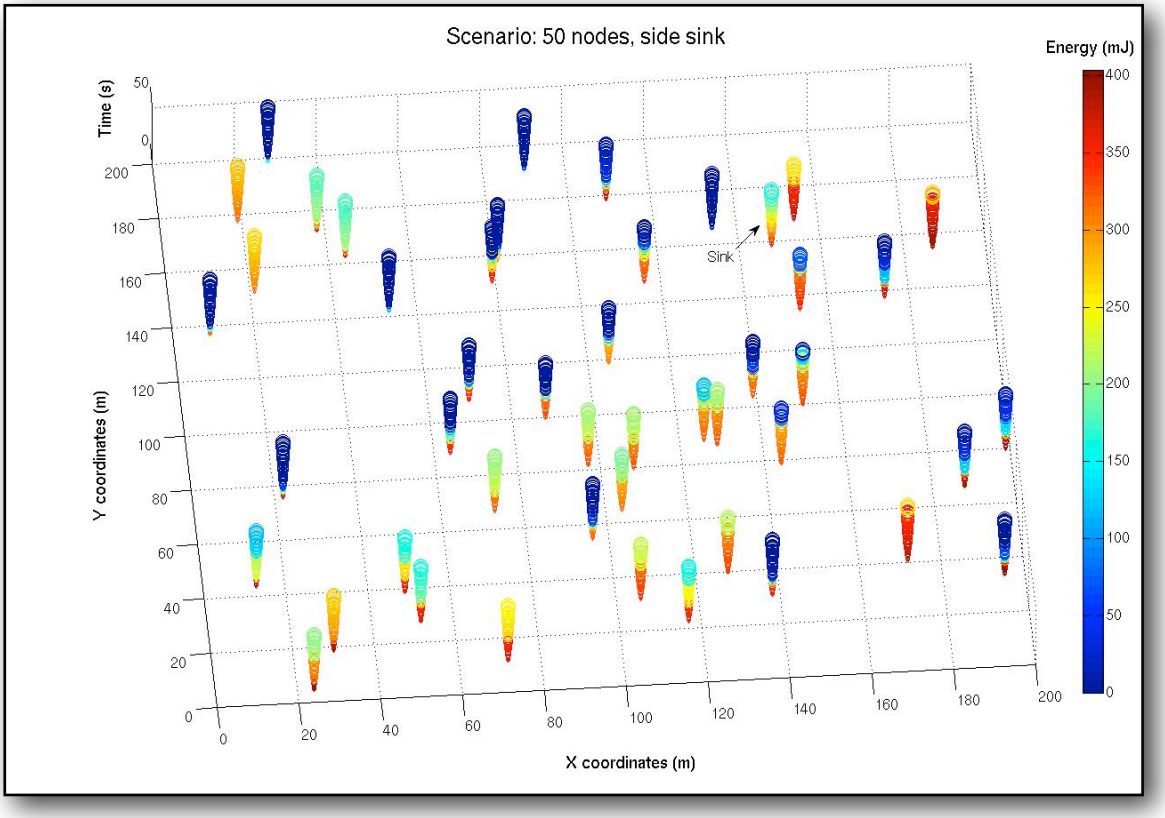


Figure 5.12d: Residual energy map, Curvy routing - 50 nodes, side sink

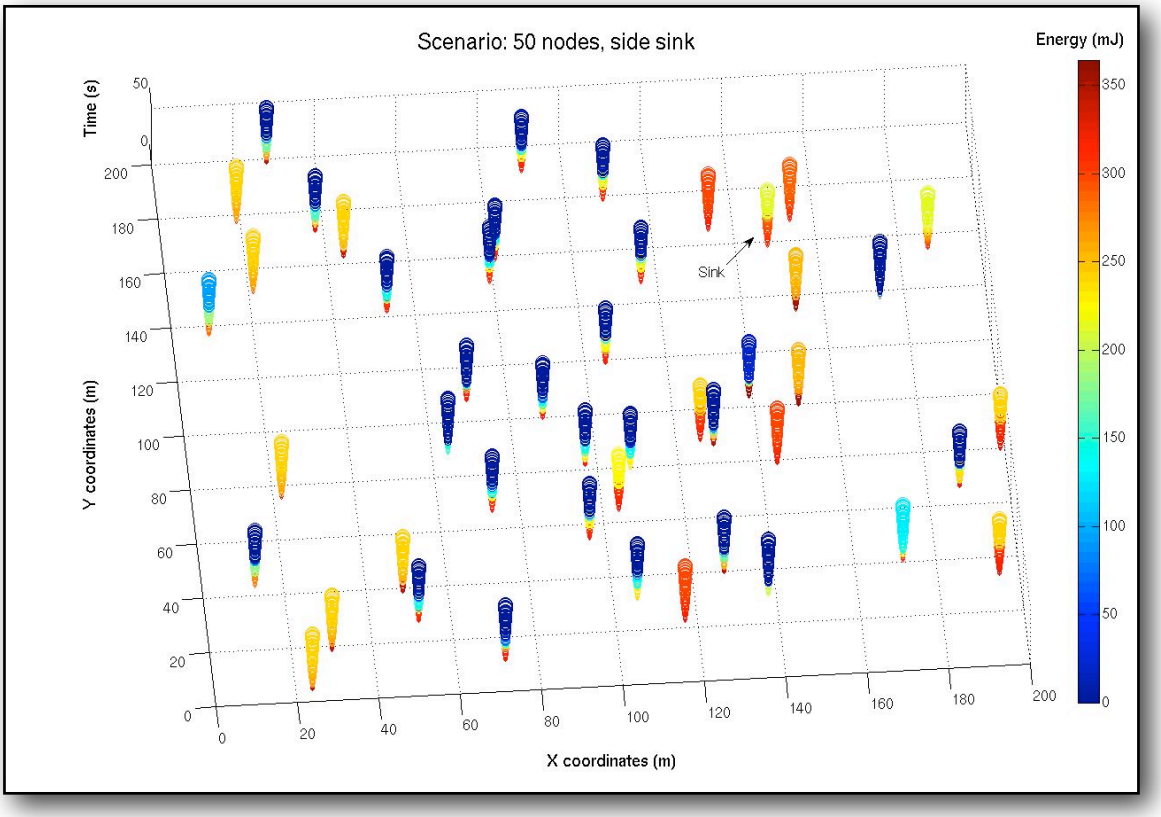


Figure 5.12e: Residual energy map, Curveball routing - 50 nodes, side sink

5.2.2 50-node scenario 2, sink near edge of network

A more extreme topology is presented here (Figure 5.13) with nodes concentrated at the left and right side of the network, having very few in the middle to connect them. This topology will highlight the difficulty met by Greedy routing to complete the simulation, the failure of Curveball routing to map such topologies into a sphere and the flexibility of Curvy routing to cope with limited forwarding choices.

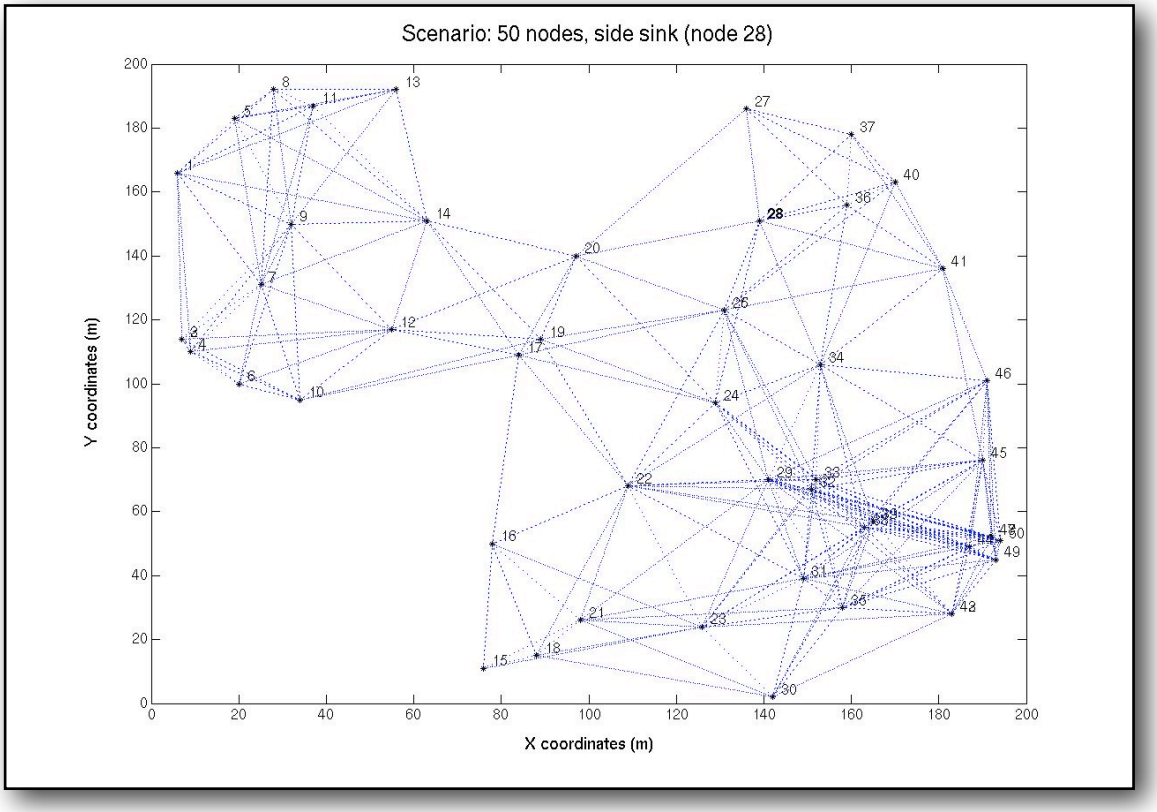


Figure 5.13: Topology with 50 nodes, side sink (node 28)

As expected, Greedy routing and Curveball routing face network partition in just 11 and 12 simulation seconds respectively. The average energy per node is represented in Figure 5.14a and shows how it has been formed throughout the simulation run. It is clear that Curveball routing starts with a lot lower residual average, because of the size and shape of the neighbourhood caused by the sphere

mapping, causing face routing to be invoked in order to establish the paths to the sink. The standard deviation of the residual energy in Figure 5.14b starts higher for Curveball and Curvy routing, while Greedy routing initially keeps it lower due to the construction of shorter paths. For Curvy routing, it increases gradually at the beginning, then it is stabilised and even decreases until it reaches the point where some of the middle nodes totally run out of battery. Looking at the network load map, network partition is clear for Greedy routing (Figure 5.14c) where all the intermediate nodes connecting the network are depleted early, leaving no room for communication. Figure 5.14d shows that Curvy routing copes well with the extreme topology protecting the intermediate nodes and keeping the network alive until the end of the simulation. The load in Curveball routing looks unevenly spread (Figure 5.14e) and the majority of the nodes run out of battery very fast. This is due to the employment of face routing applied on the periphery of the network until destination is reached. The paths are a lot longer than Curvy routing and the number of nodes is not enough to support them, which in result causes accelerated energy depletion and premature network partition.

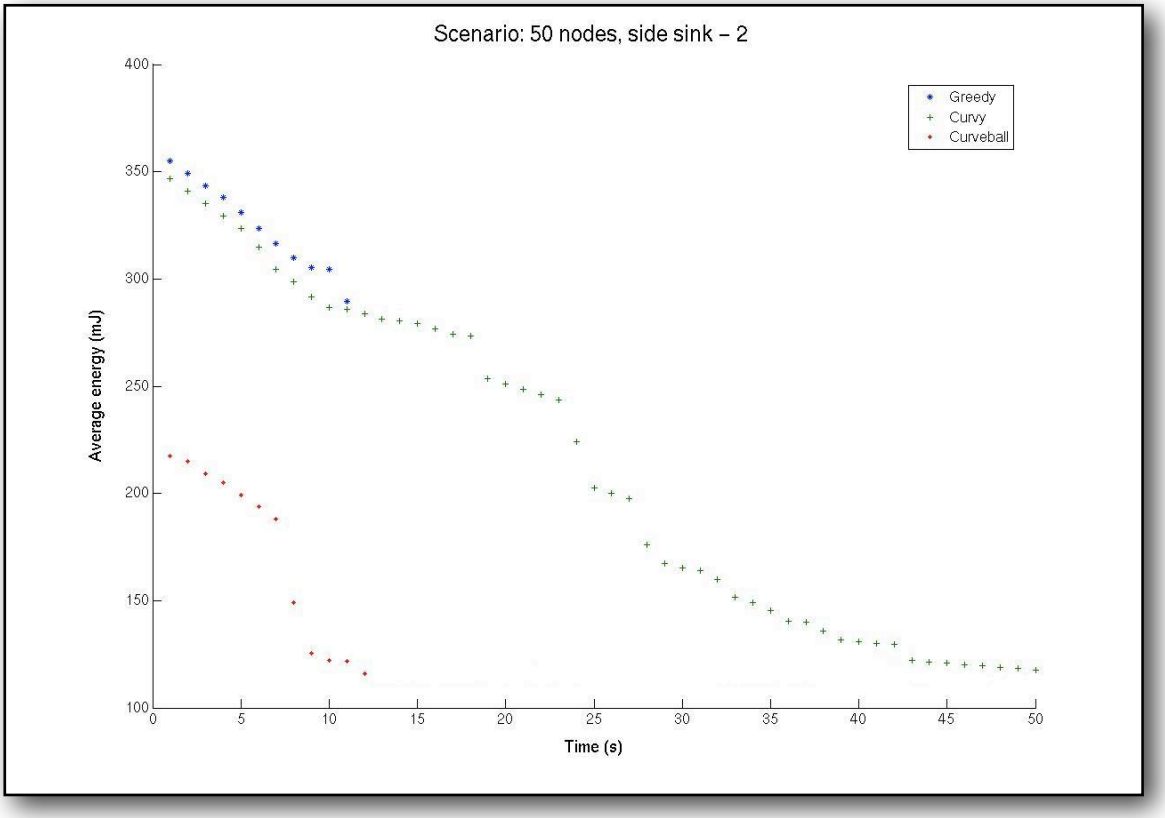


Figure 5.14a: Average energy vs Time - 50 nodes, side sink

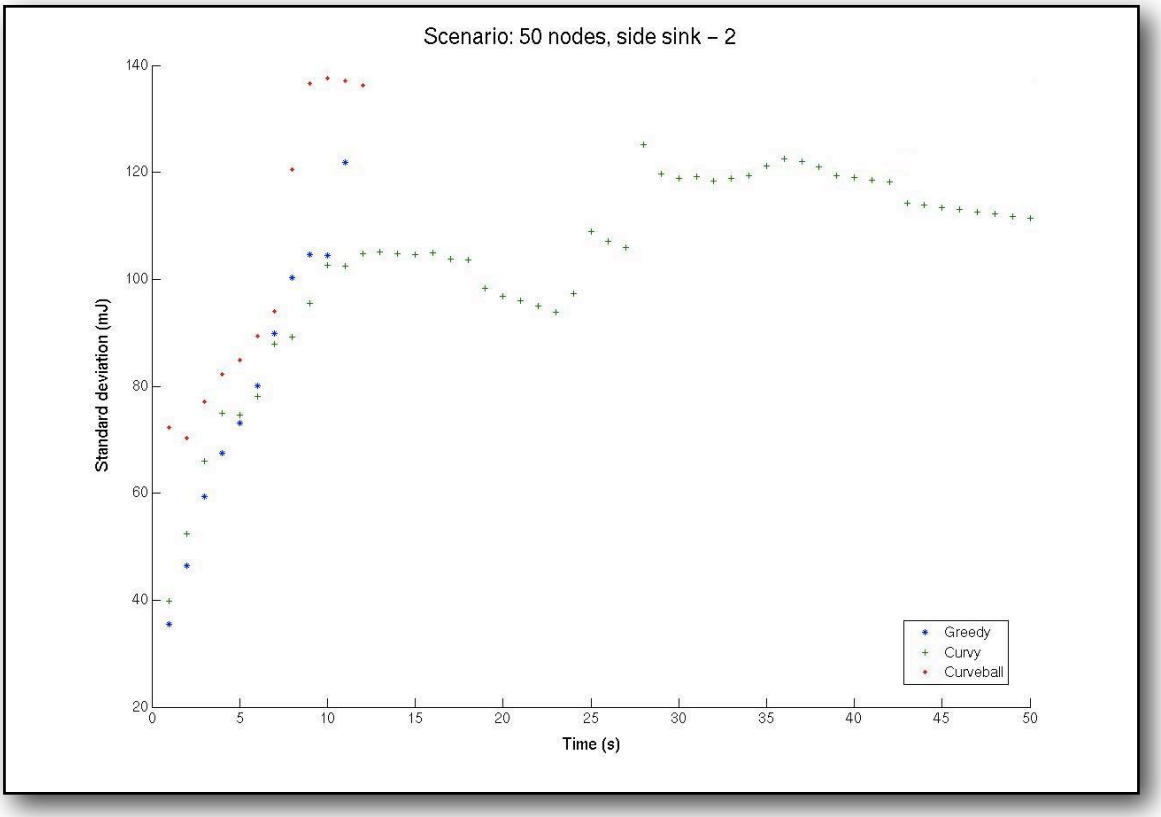


Figure 5.14b: Standard deviation of energy dissipation vs Time - 50 nodes, side sink

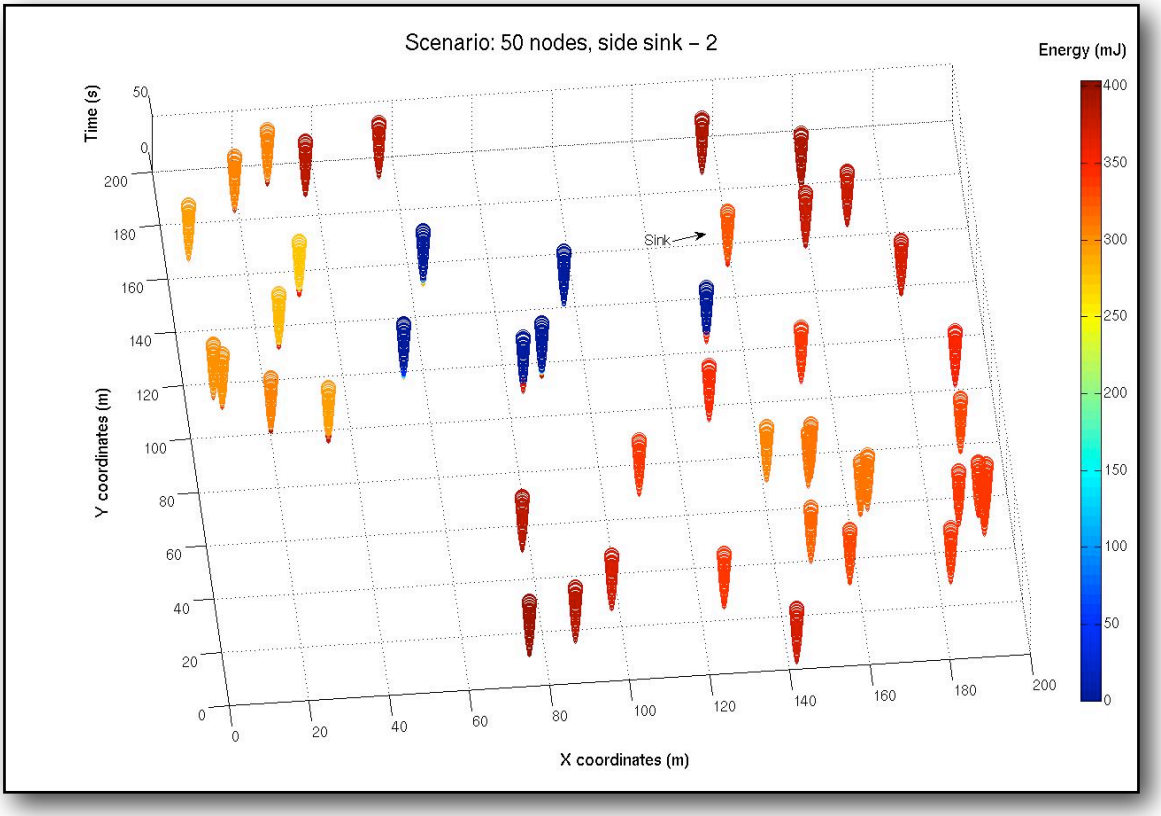


Figure 5.14c: Residual energy map, Greedy routing - 50 nodes, side sink

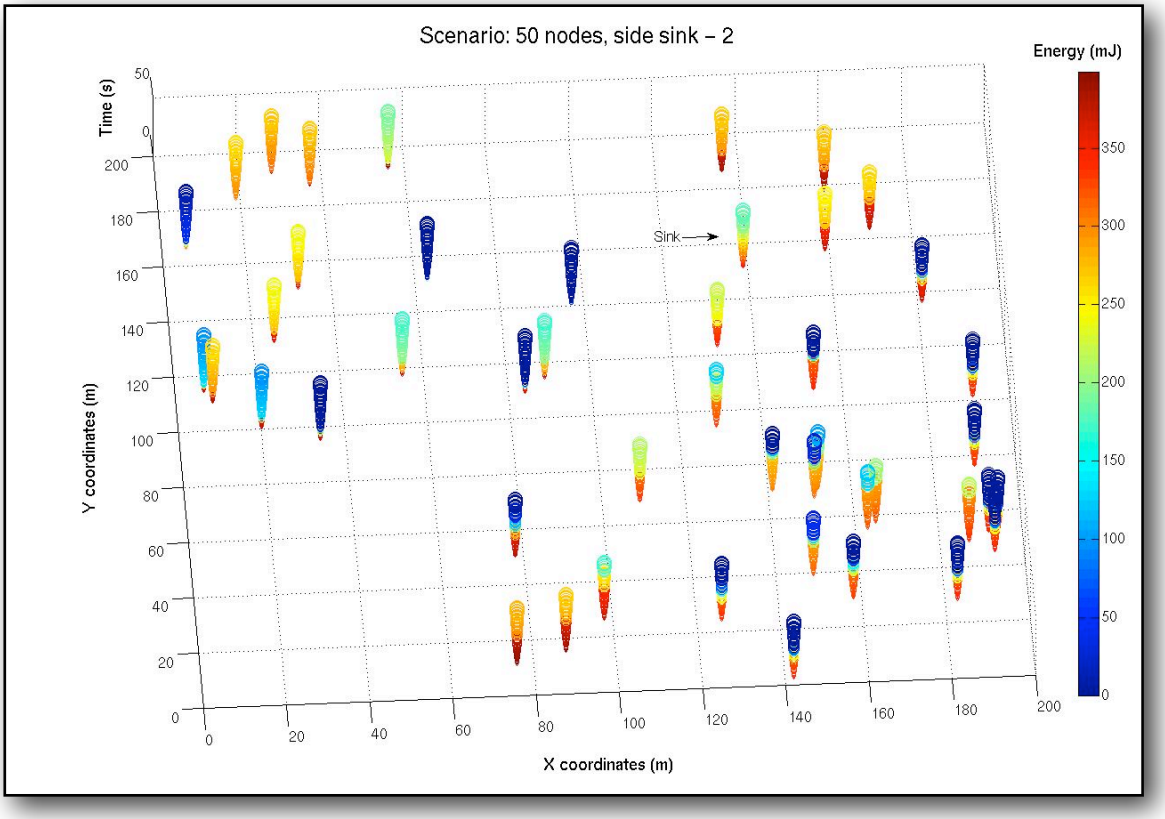


Figure 5.14d: Residual energy map, Curvy routing - 50 nodes, side sink

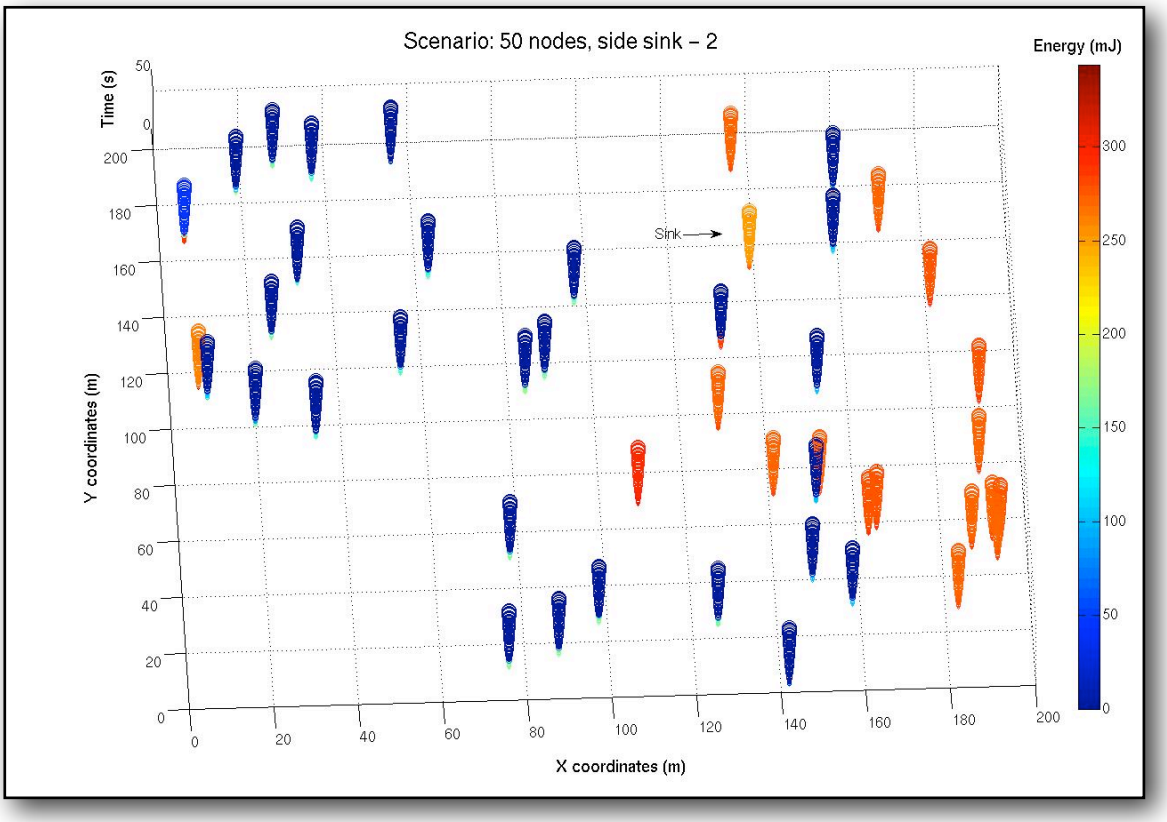


Figure 5.14e: Residual energy map, Curveball routing - 50 nodes, side sink

5.2.3 38-node scenario, sink near edge of network

Working on an even sparser scenario, as in section 5.1.5, another topology of 38 nodes (Figure 5.15 below) has been simulated. The distribution is more even this time and nodes cover most of the network region without creating any void areas in between.

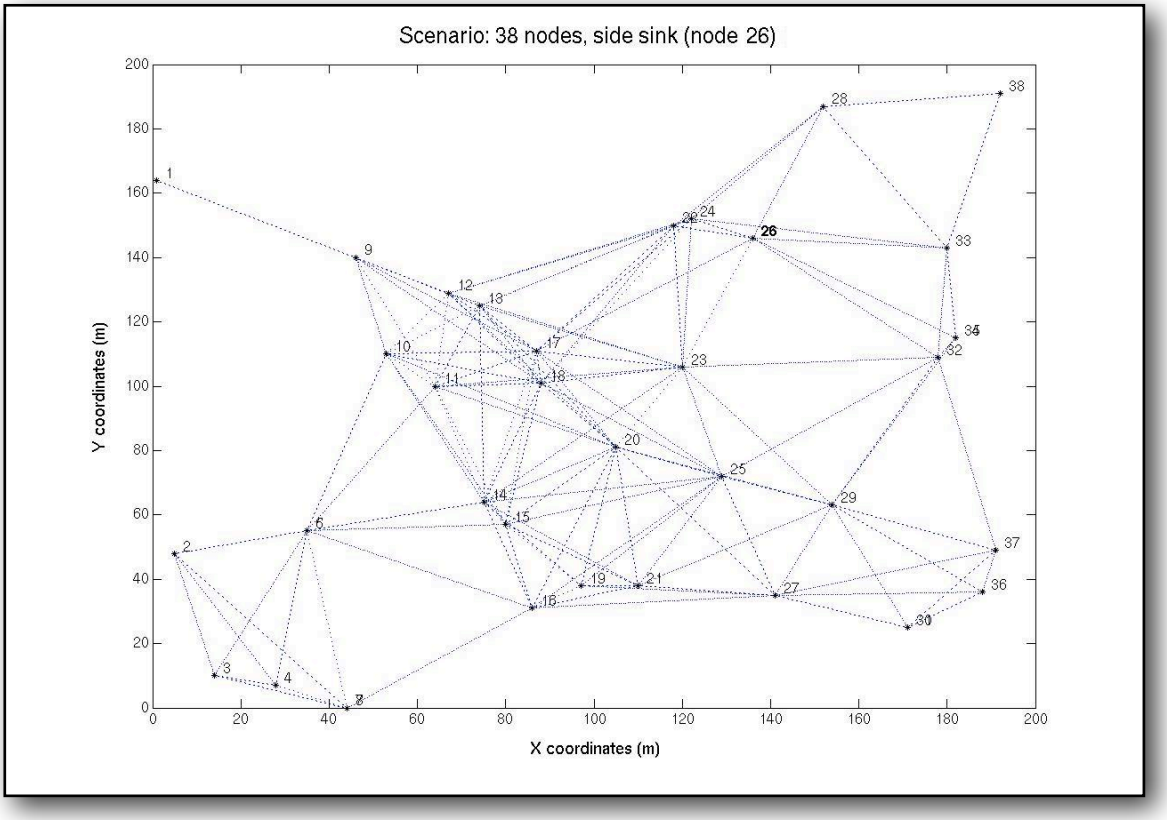


Figure 5.15: Topology with 38 nodes, side sink (node 26)

Figure 5.16a shows the node average energy and it can be seen that Greedy routing and Curveball routing face network partition at about the same simulation time. However, Curveball routing starts with a lower average than Greedy due to the hello broadcasts across the larger sphere-mapped neighbourhoods. Curvy routing performs well keeping the average higher than the rest with efforts to stabilise it halfway onwards. As usually, Greedy routing maintains a high standard deviation of the residual energy in Figure 5.16b, where Curveball and Curvy routing are almost equal up to Curveball's network partition point. Curvy routing manages to control the standard deviation and even to decrease it by the end of the simulation. Depicting the network load, shows the overloaded central area for Greedy routing in Figure 5.16c and the partition caused. Curvy routing achieves a balance between handling path length and addressing the crowded centre effect, as illustrated in Figure 5.16d. Looking at Figure 5.16e, the area around the sink is protected in a better way using the Curveball routing method in comparison with

Greedy routing, but it fails to complete the simulation run. Taking into consideration the previous scenarios demonstrated in sections 5.2.1 and 5.2.2, the difference in the number of nodes is not the main issue in this method: the shape of the network and the distribution of the nodes mostly define the mapping on the sphere, which have the biggest influence in determining the performance of the protocol.

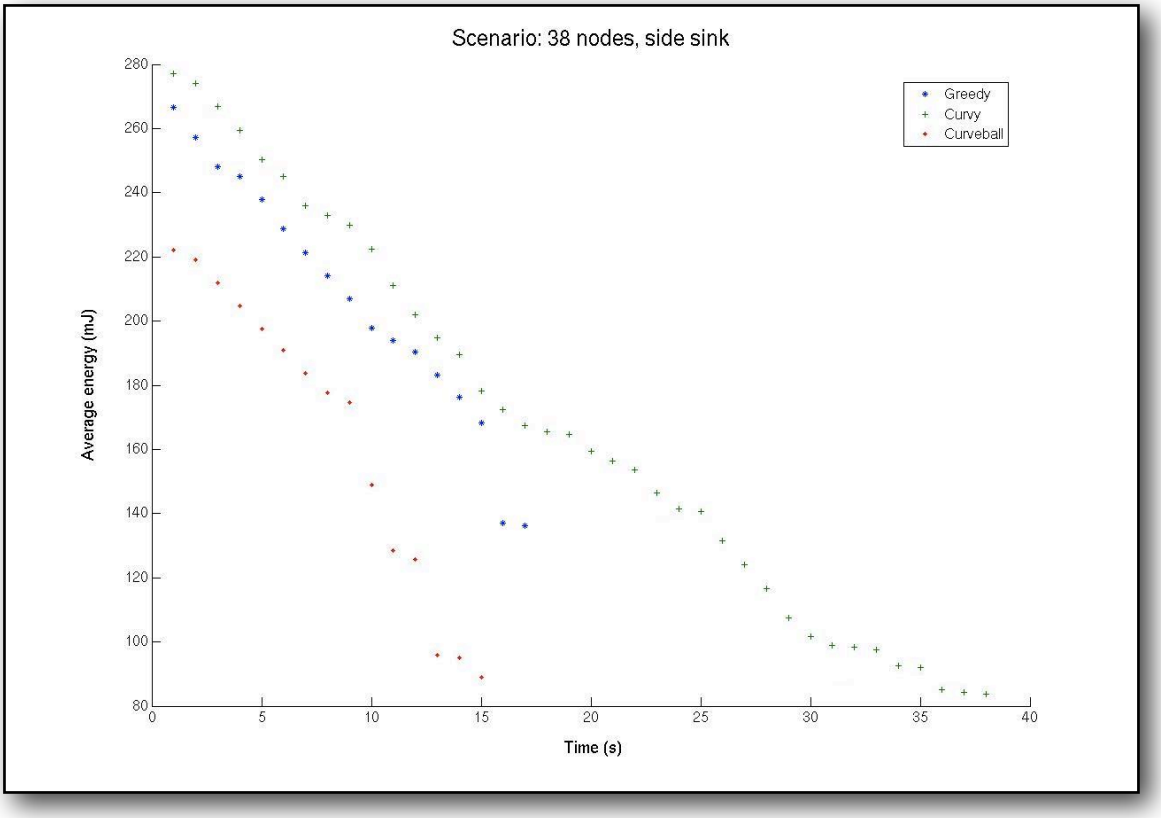


Figure 5.16a: Average energy vs Time - 38 nodes, side sink

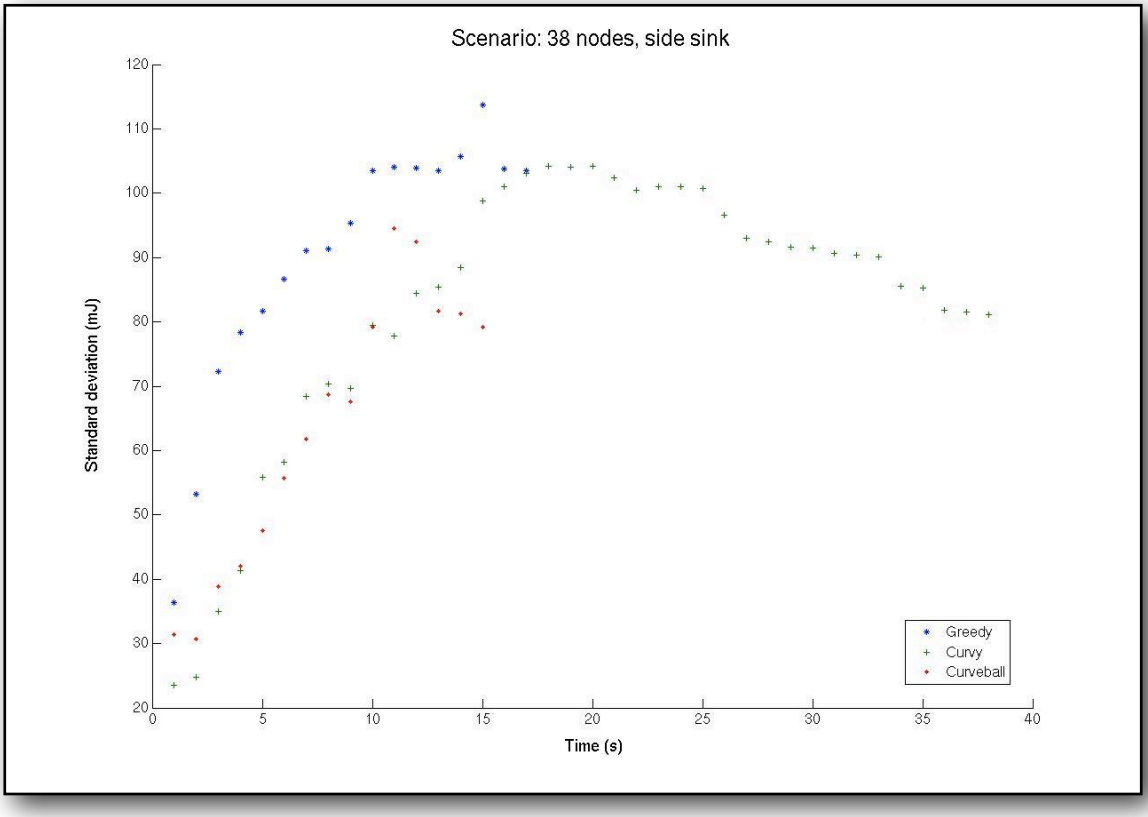


Figure 5.16b: Standard deviation of energy dissipation vs Time - 38 nodes, side sink

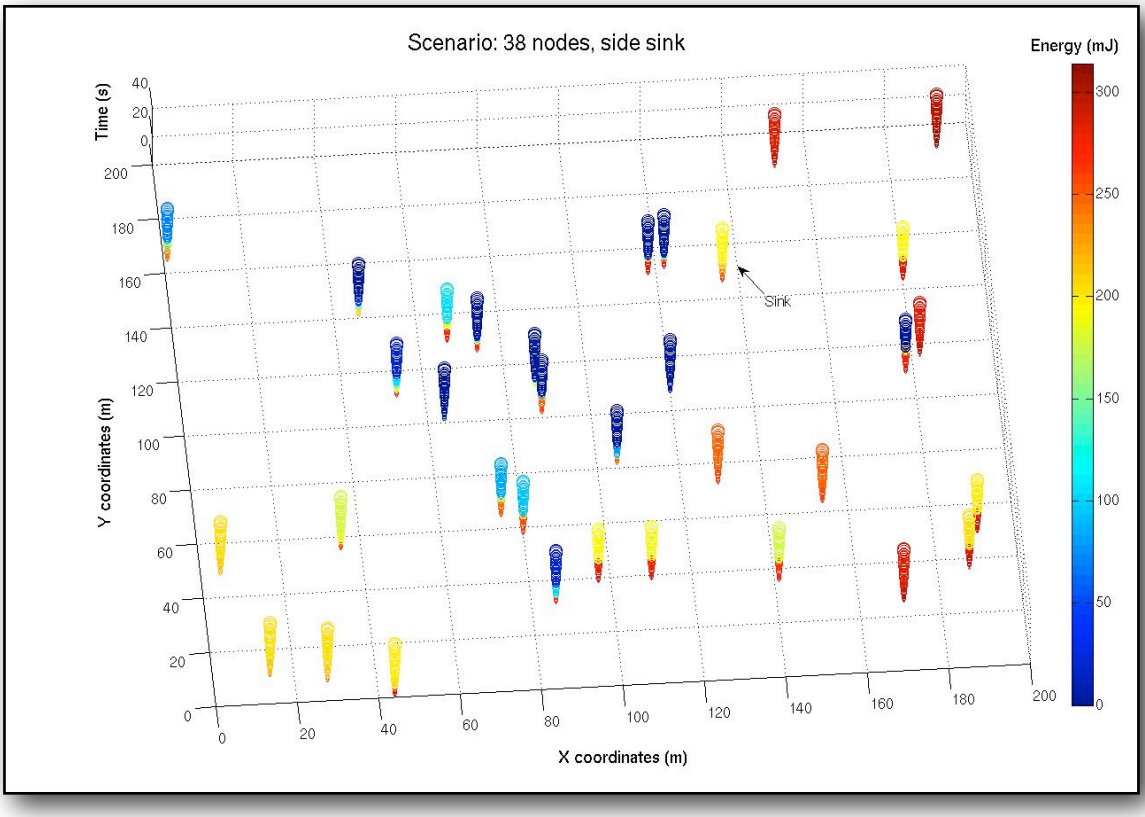


Figure 5.16c: Residual energy map, Greedy routing - 38 nodes, side sink

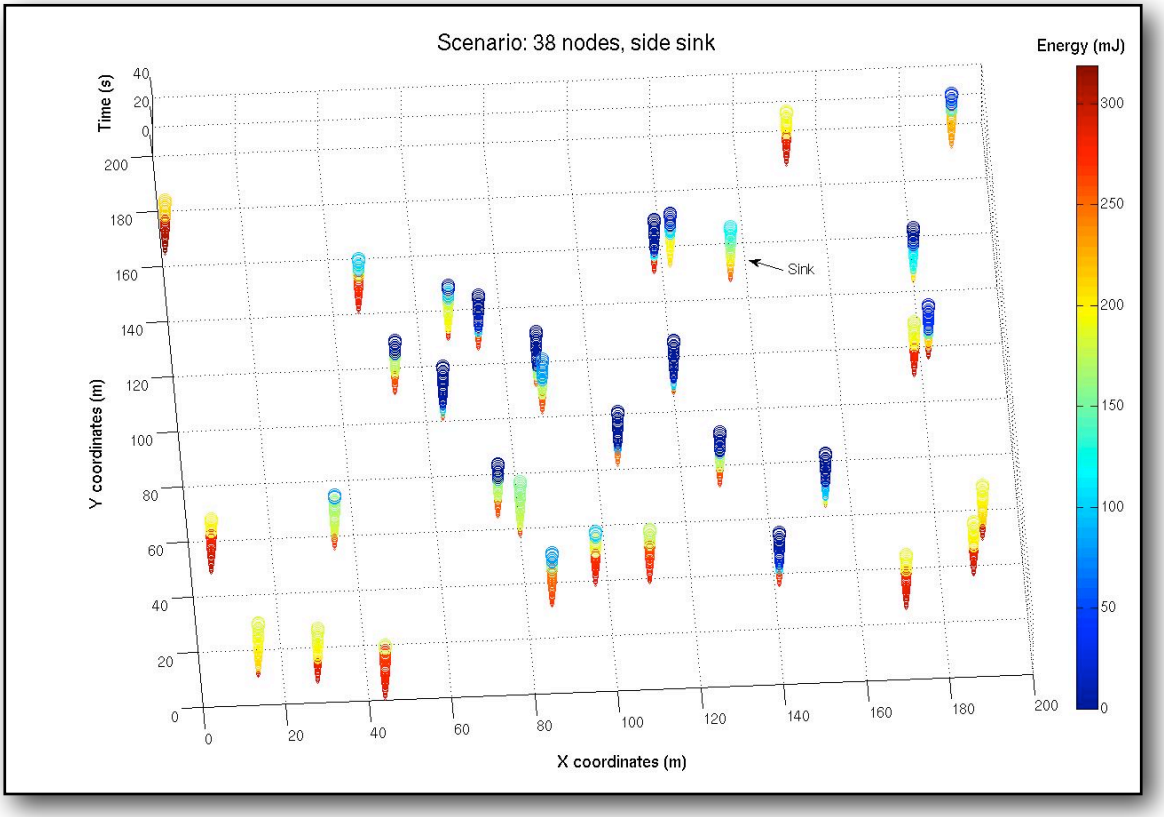


Figure 5.16d: Residual energy map, Curvy routing - 38 nodes, side sink

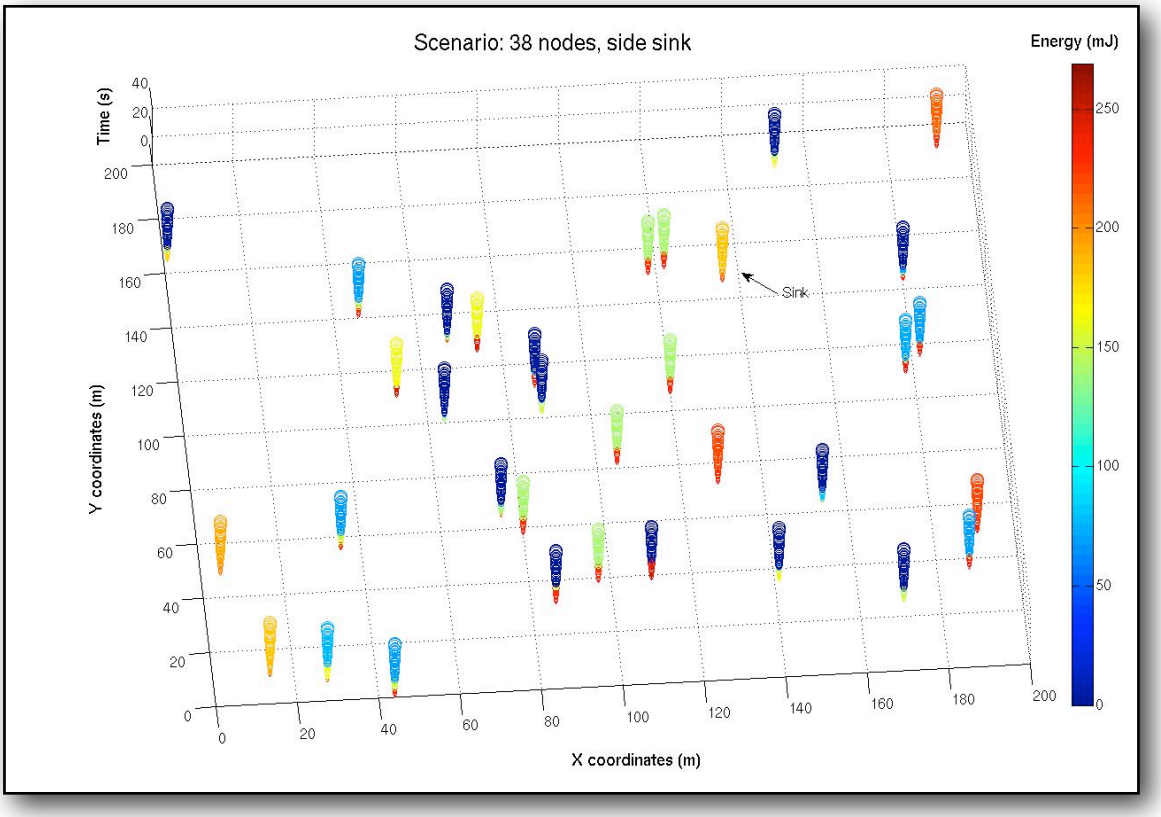


Figure 5.16e: Residual energy map, Curveball routing - 38 nodes, side sink

5.2.4 50-node scenario, random s-d pairs

As Popa et al. (2007) include random source-destination (s-d) pairs in their simulations, it is a fair comparison to create similar scenarios, although such configurations are not practical for information exchange as discussed previously in Section 5.1.3. For reasons of comparing the performance with the single-node scenarios, the topology chosen (Figure 5.17) is the same as in Section 5.2.1.

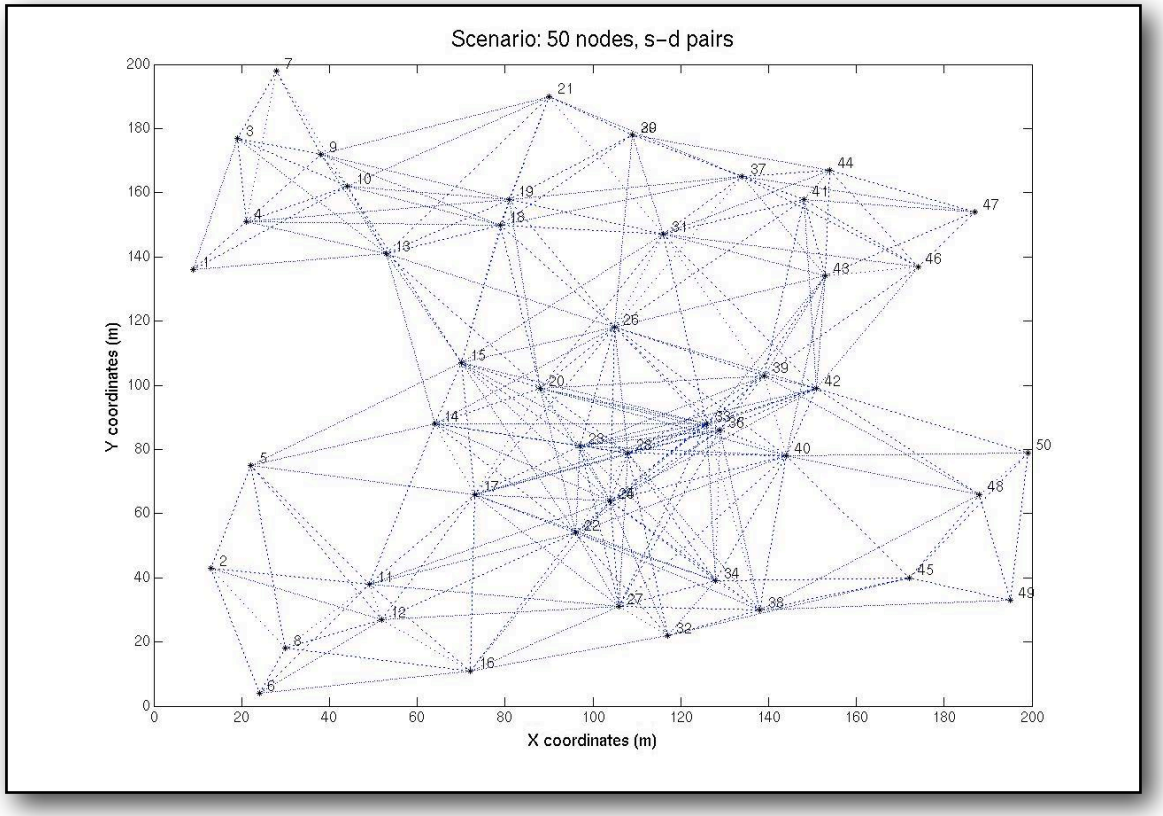


Figure 5.17: Topology with 50 nodes (s-d pairs)

The average energy graph (Figure 5.18a) shows that Greedy routing can not cope with random s-d pairs either and it fails after 29 seconds of simulation. Curvy routing on the other hand keeps the average steady and meets no difficulty in completing the simulation run. Curveball routing also copes well up to the 30th second and afterwards it struggles to control the energy depletion. This is due to the

design of Curveball routing (discussed in Chapter 2), which returns to Greedy routing once 3D routing fails. The standard deviation of the energy dissipation in Figure 5.18b verifies the performance of the protocols. Greedy routing starts with the lowest value, but cannot finish the simulation, while Curveball routing has a difficulty in handling the increase and seems to loose control between the 16th and the 29th second. Curvy routing maintains lower values than Curveball routing during the entire simulation run and shows its ability to adopt to different scenario configurations. Figures 5.18c, d and e depict the network map of residual energy for Greedy routing, Curvy routing and Curveball routing respectively. Greedy routing suffers from early energy depletion of nodes in central location, not allowing paths go through after a certain period of time (around 30 seconds). Curvy routing achieves a balance across the network, utilising nodes in the periphery of the network during path discovery. A few nodes run out of battery faster than the rest, but that totally depends on the random choice of s-d paths. Finally, Curveball routing performs well initially and does not experience network partition in this scenario. A load imbalance can be observed initially because of the sphere mapping and the use of the Greedy routing technique, but the objective of protecting the centre of the network is met.

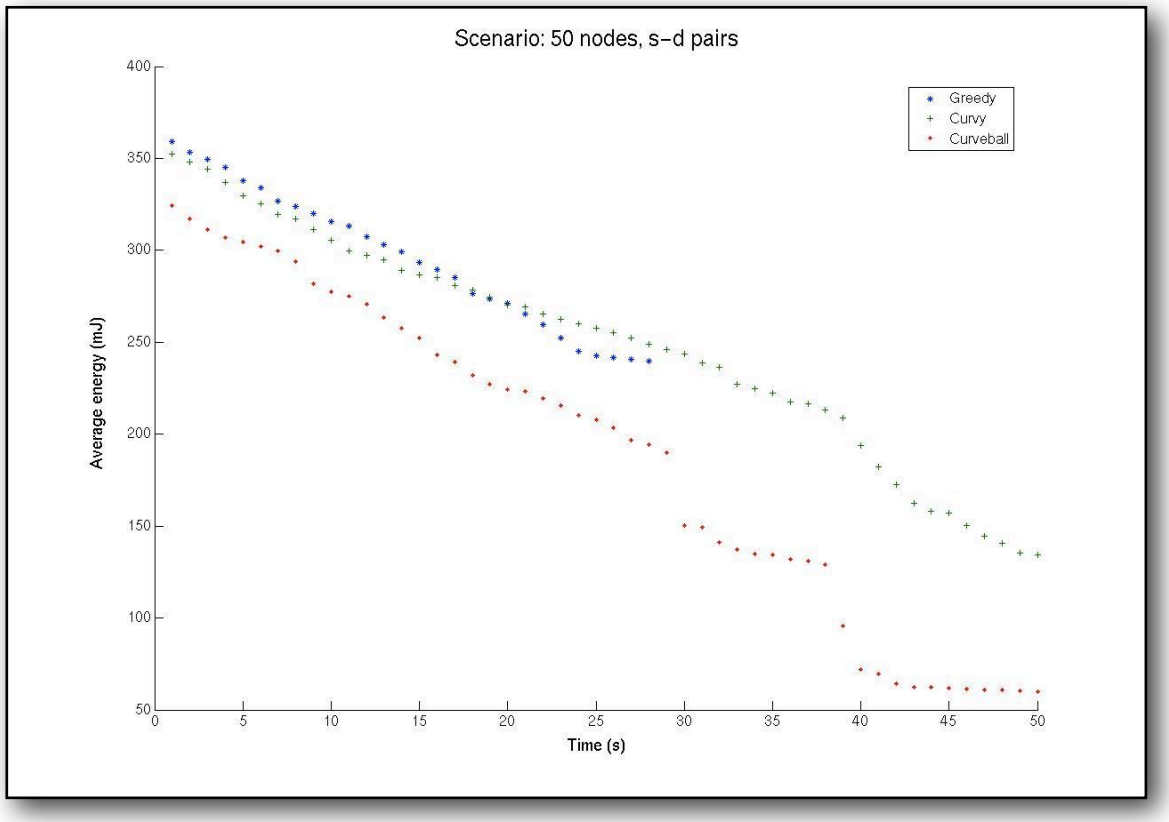


Figure 5.18a: Average energy vs Time - 50 nodes (s-d pairs)

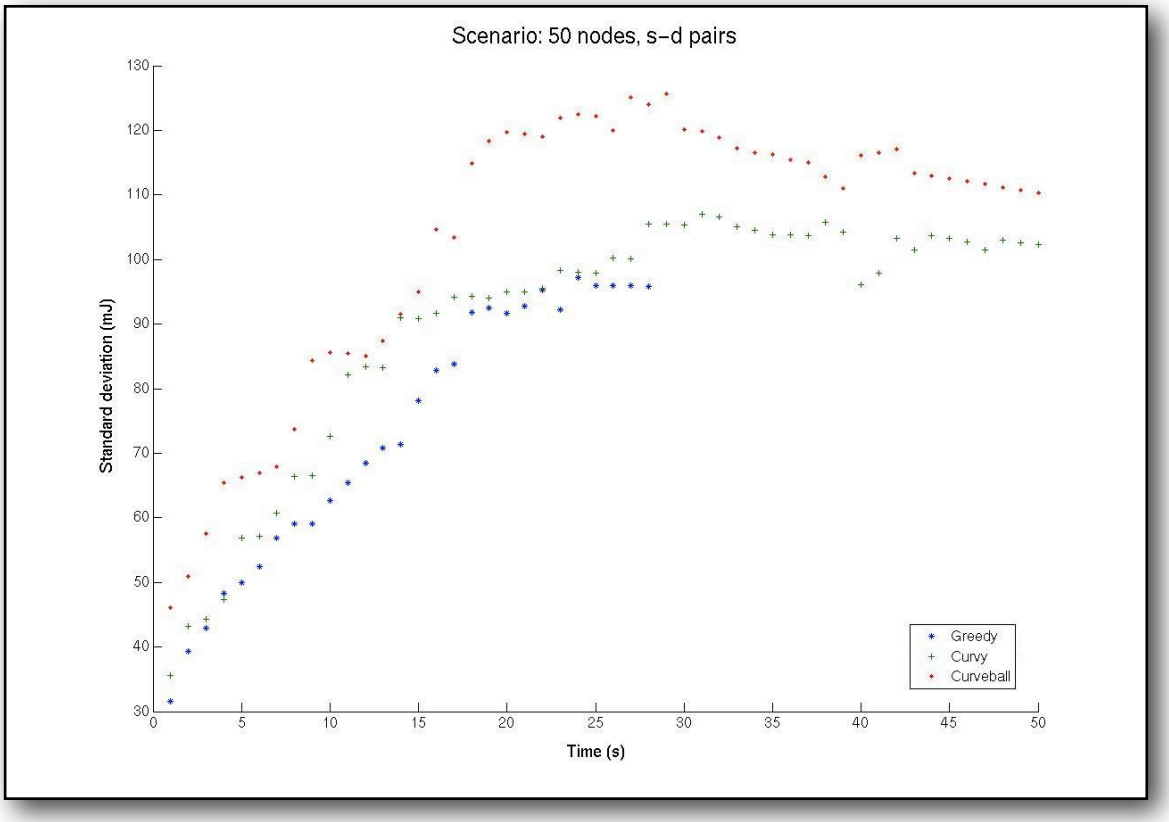


Figure 5.18b: Standard deviation vs Time - 50 nodes (s-d pairs)

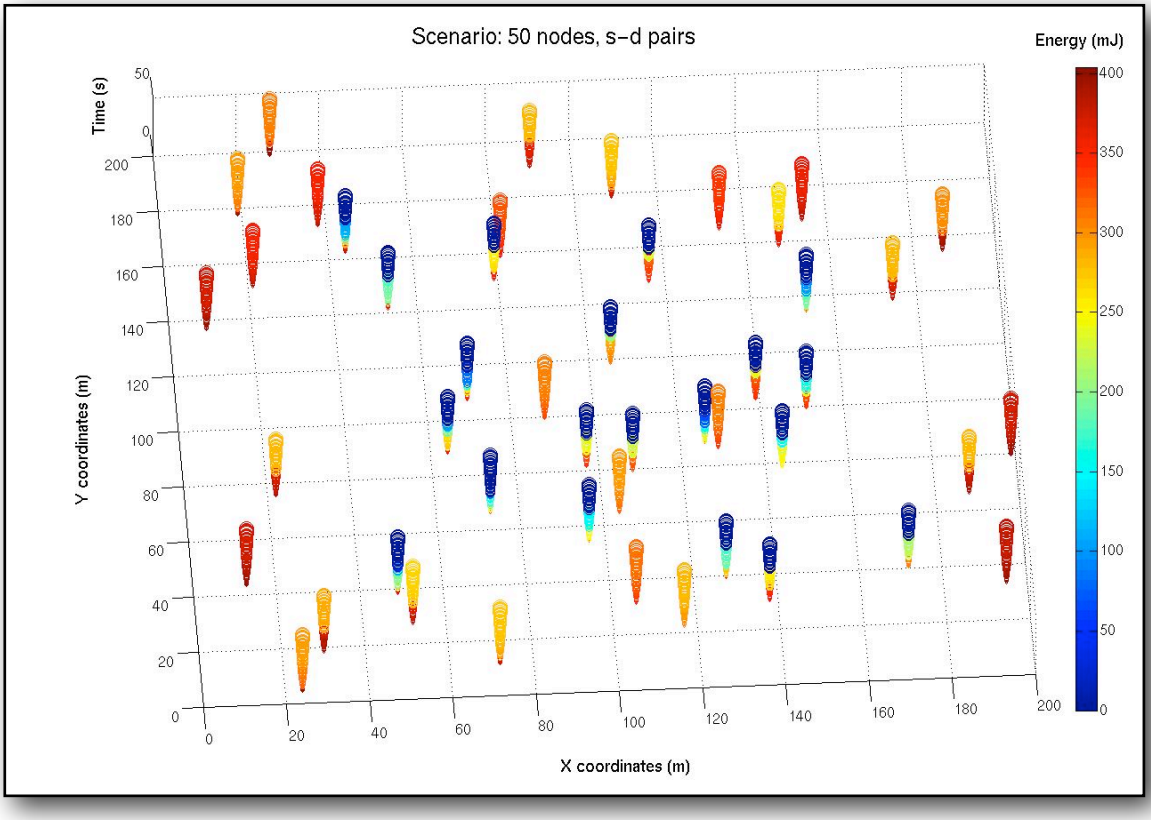


Figure 5.18c: Residual energy map, Greedy routing - 50 nodes (s-d pairs)

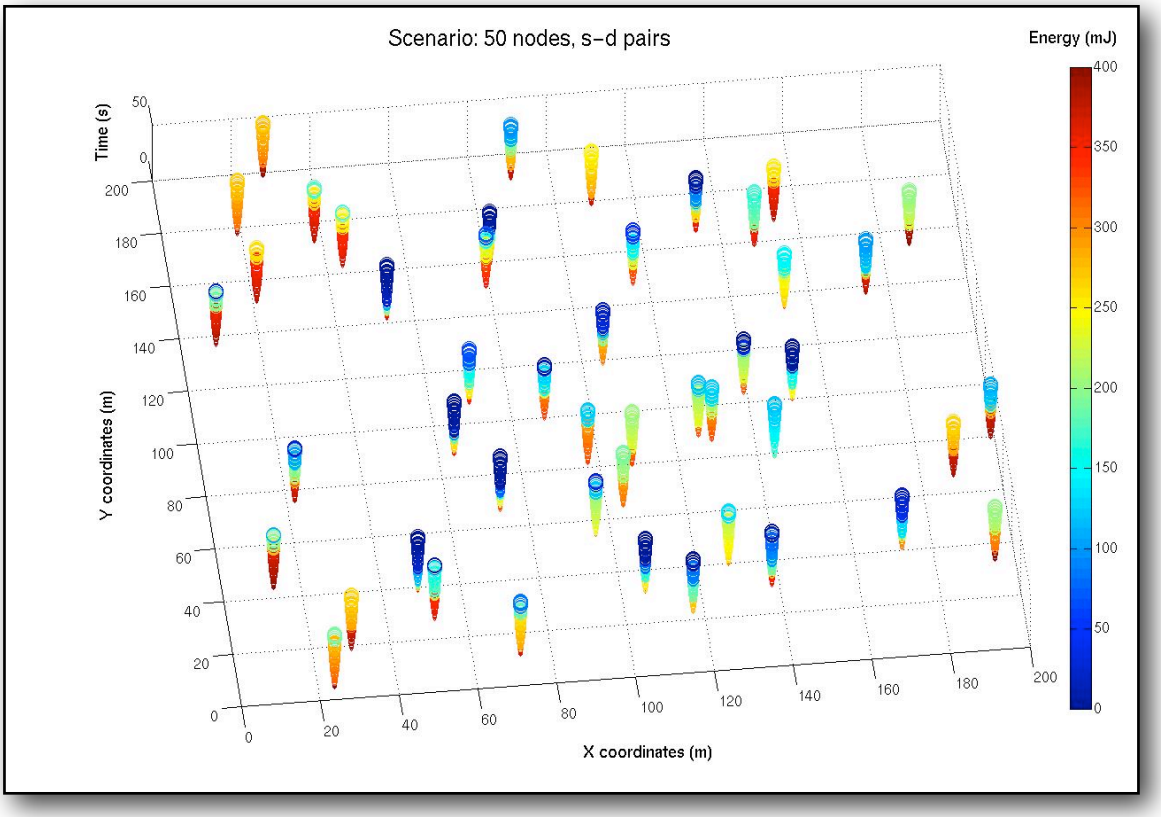


Figure 5.18d: Residual energy map, Curvy routing - 50 nodes (s-d pairs)

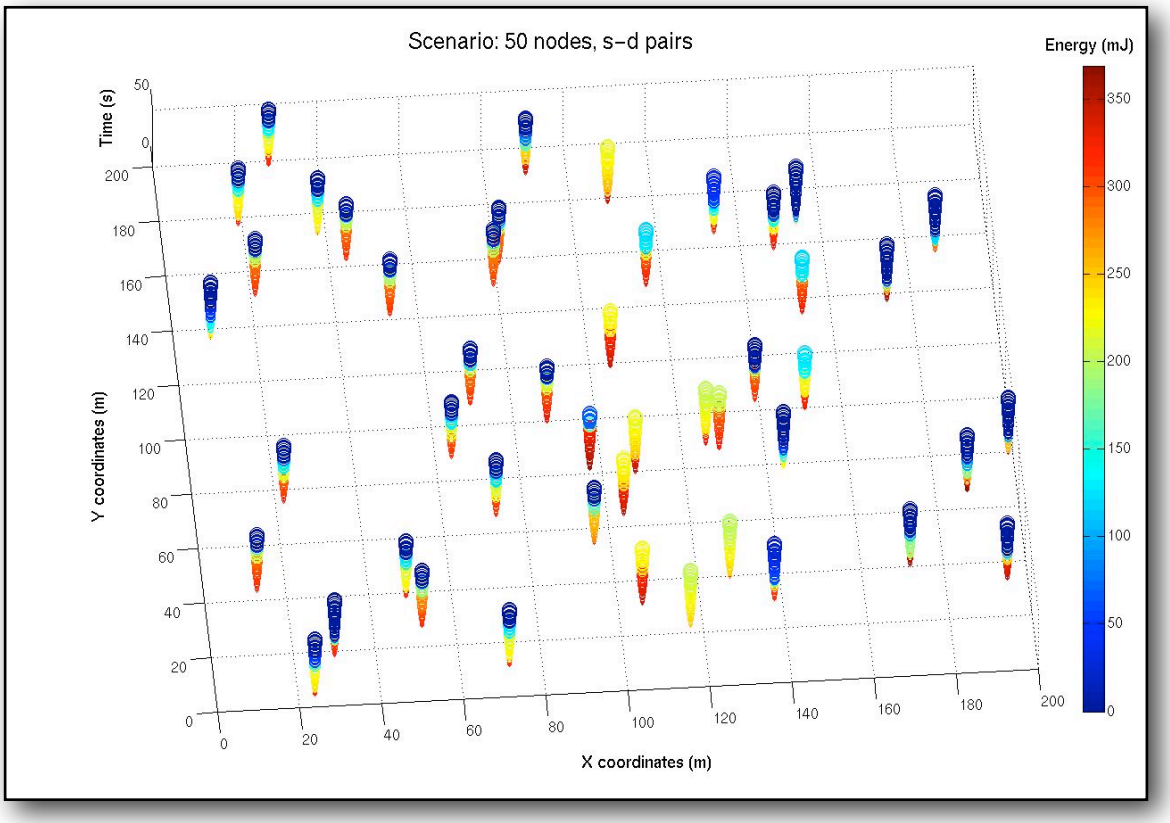


Figure 5.18e: Residual energy map, Curveball routing - 50 nodes (s-d pairs)

5.3 Summary

Useful conclusions for the practical implementation of the algorithm have been drawn after running a number of simulations, large enough to apply to a range of different network layouts. A notable observation is the fact that Curvy routing is extremely effective in sparse topologies with visible void areas and it significantly outperforms Greedy routing.

It has also been observed that the algorithm performance is strongly dependent on the shape of the network. Shapes approximating a disc result in more favourable metrics as opposed to rectangular ones, providing an easier construction of the curved paths. This is to be expected, as a radially symmetric refractive index is being employed. Rectangular shapes suffer from uneven path distribution with reference to the corners of the network. Moreover, it is easy to predict which shapes

are in favour by looking at the circle transformation used to convert the network plane to a sphere. If the initial topology appears similar to the transformed one, this means that the network shape is very close to approximating a circle.

Regarding the Curveball routing implementation, there has been observed a difficulty in executing the sphere mapping due to the low density network scenarios. As discussed in Chapter 2, assumptions of highly dense networks have been adopted towards implementing the proposed solutions. Although these prove that they work perfectly in such unfeasibly dense networks, when they are applied on more realistic scenarios, they fail to perform well. Popa et al. (2007) use 15000 nodes in their simulations and mapping that number of nodes on a sphere does not leave any part of its surface uncovered. Mapping 50 or fewer nodes, though, the surface cannot be always covered sufficiently no matter how small the size of the sphere is. Nodes are unevenly and randomly spread around the sphere and sometimes network connectivity suffers because they are not mapped on the same hemisphere or even the same side of the sphere. This mapping limitation has already been identified by Popa et al. (2007), but never caused much trouble in high densities and large network sizes.

The results derived by adjusting the density of the network were examined and it has been concluded that, in general, Curvy routing performs best when the number of nodes for the 200m x 200m area is between 35 - 40, namely 0.95×10^{-3} nodes per square metre, which corresponds to an average number of 7 - 10 neighbours per node. Finally, extensive experimentation was carried out with the position of the destination node (sink) and, as shown above, best performance is achieved when the sink is away from the geographical centre of the WSN.

CHAPTER 6

Performance Comparison: Energy Aware Routing vs Curvy Routing vs Greedy Routing

After presenting comparison results between Curvy routing and Greedy routing (Chapter 5), the second version of the proposed algorithm is compared here against these two. More variations in network size, density and transmission range are presented investigating even sparser topologies, as well as different network shapes to show the sensitivity of the algorithm performance with respect to path computation methods and next-hop selections. Timestamped screenshots are included to depict the levels of the residual energy across the topology and the way the refractive index is changed through simulation time for the Energy Aware protocol. Additionally, source - destination pairs have been simulated to display the difference of the calculated paths by the Greedy, Curvy and Energy Aware routing algorithms between pairs of random locations.

The results presented and discussed below show significant improvement in the performance of the Energy Aware version of the algorithm, especially in terms of spreading the traffic load evenly across the network and minimising the standard deviation of the average network energy. It has been observed that topologies with sufficient next-hop choices on the periphery of the network favour the Energy Aware routing protocol, even if the centre of the network is void or extremely sparse. The conclusions drawn in Section 6.2 show the conditions under which the Energy Aware routing protocol performs better, as well as limitations associated

with the simulation environment and alternative solutions are proposed to overcome these.

6.1 Scenarios

The dimensions of the network area have been kept exactly the same as previously at 200m x 200m and scenarios varying in number of nodes, density, shape, size and transmission range are selected to compare the performance and observe the behaviour of the Energy Aware, Curvy and Greedy routing techniques. The scenarios are representative examples from each category and have been selected amongst a wide sample of simulations with identical network parameters. In all graphs in this chapter depicting the topology, the sink node is marked in bold font. The rest of the topologies simulated for each scenario (with similar results) are included in Appendix B.

6.1.1 50-node scenario

As seen and discussed in the previous results chapter, the 50-node scenario does not provide the ideal density to highlight the performance of the algorithm, but for the sake of comparing the three protocols altogether and making meaningful the comparisons with Chapter 5 in terms of different network shape, a scenario with density of 1.25×10^{-03} nodes per square metre (Figure 6.1) and the position of the sink node to the side is presented.

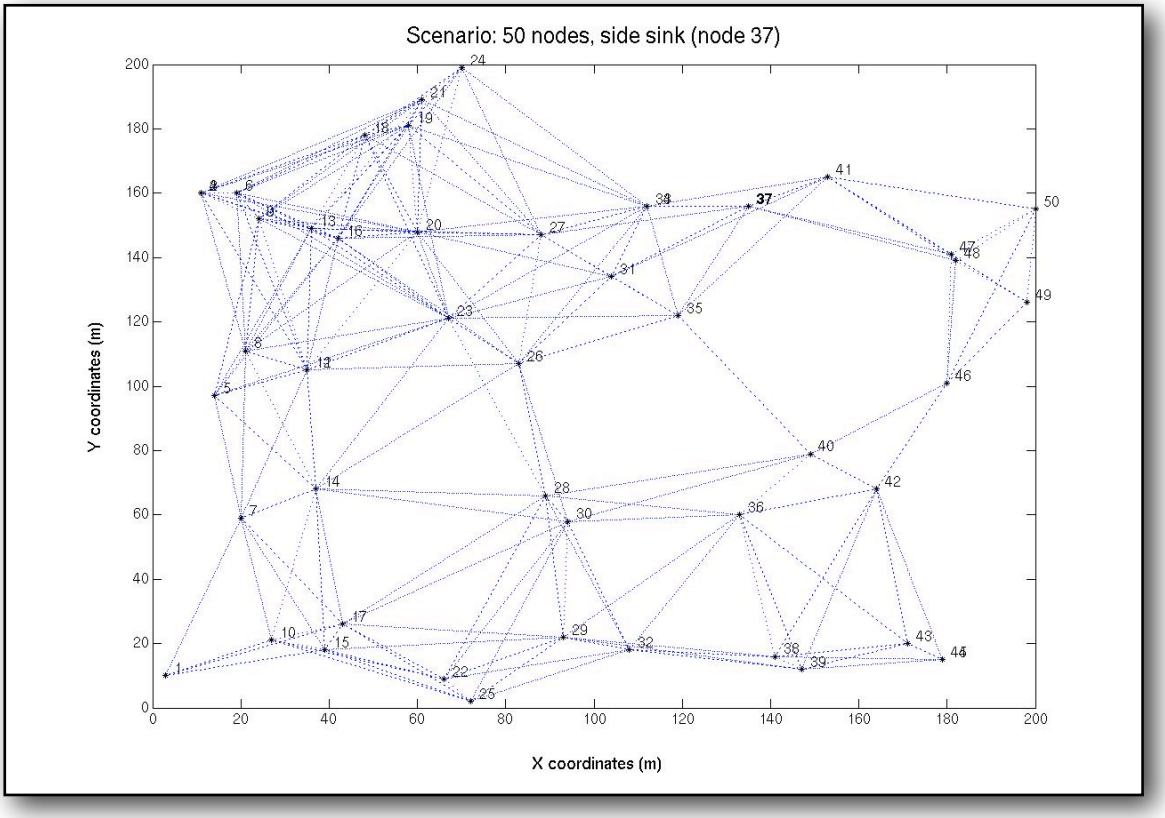


Figure 6.1: Topology with 50 nodes, side sink (node 37)

Figure 6.2a depicts the average node energy (mJ) as a function of simulation time. Greedy routing fails to complete this simulation run, as a network partition occurs after 35 simulation seconds. The Curvy routing, though, manages the energy dissipation a little more efficiently than Energy Aware routing, because the latter uses longer paths while avoiding local areas of low energy. The standard deviation of the node energy depletion shown in Figure 6.2b, on the other hand, reveals a clear advantage of using local energy information for the next-hop calculation from the 39th simulation second onwards. Sudden changes in the slope are experienced by Greedy routing, caused by depleted nodes at vital positions. When this happens, Greedy routing is forced to discover a new route, which is no longer the shortest.

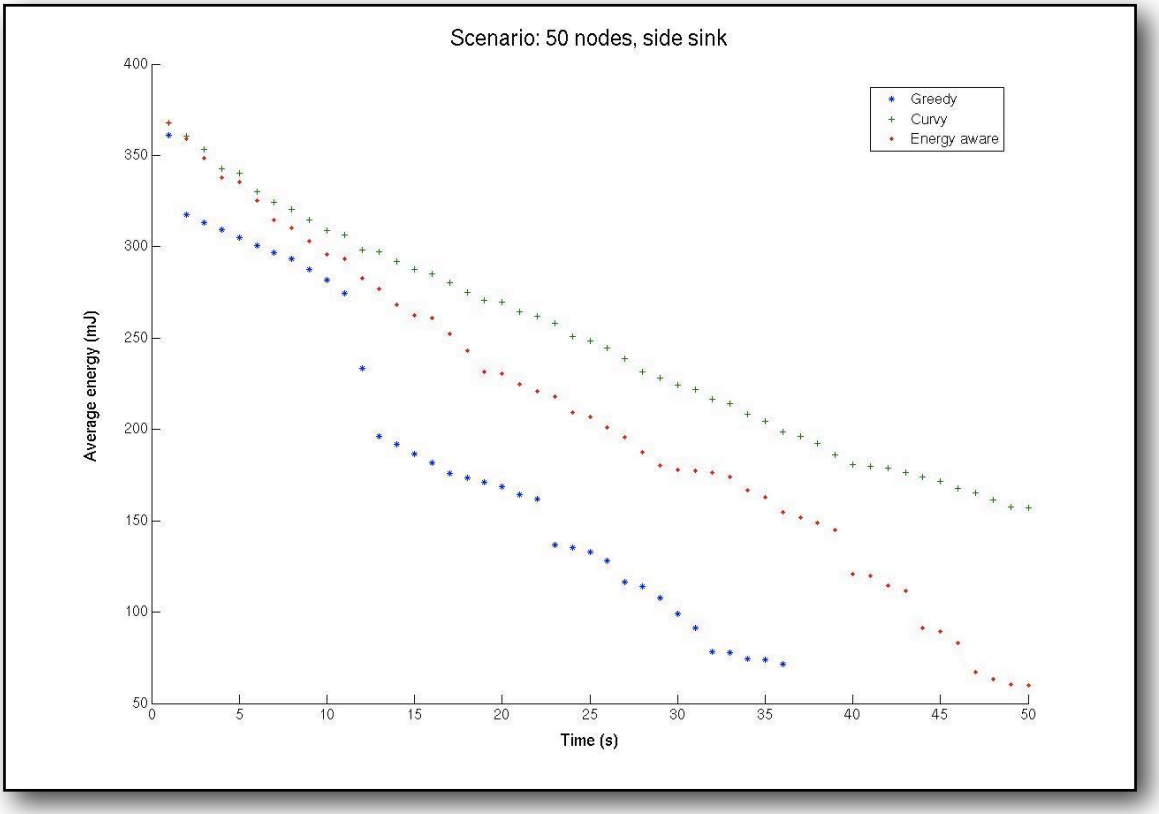


Figure 6.2a: Average energy vs Time - 50 nodes

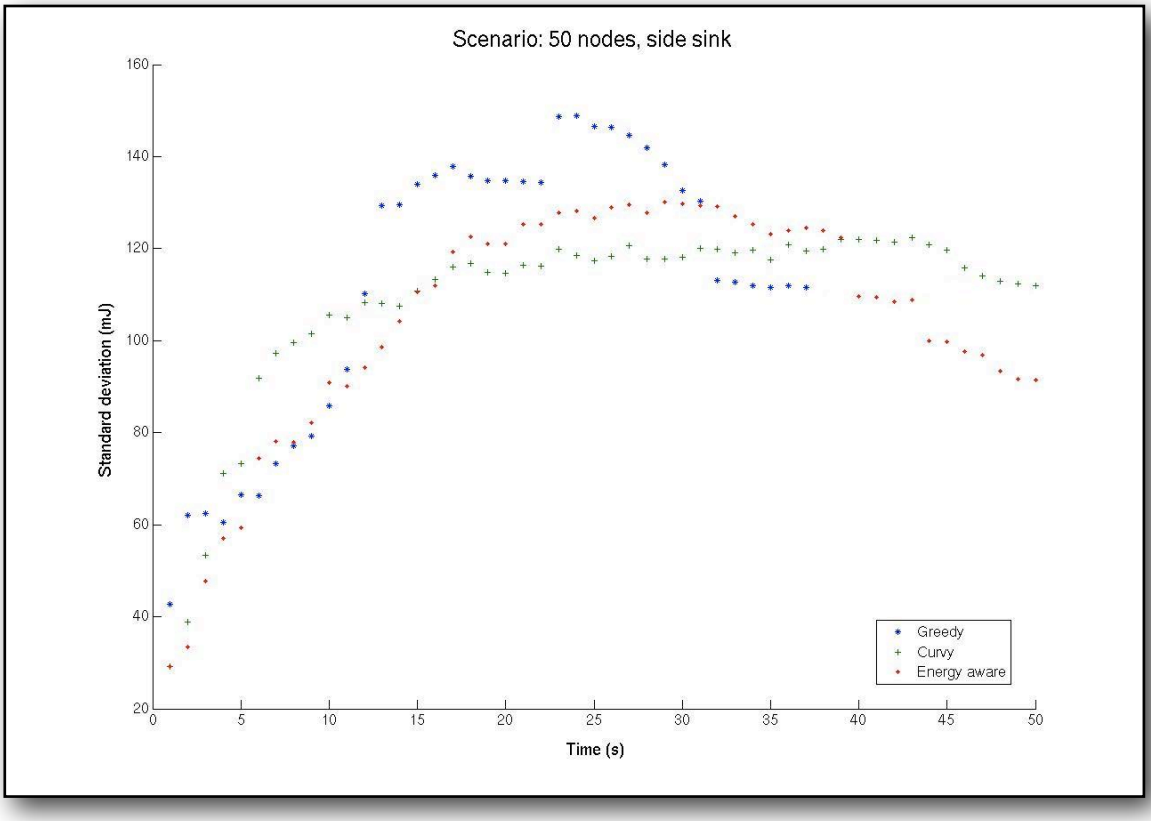


Figure 6.2b: Standard deviation of energy dissipation vs Time - 50 nodes

Figures 6.3a, b and c illustrate the load spread across the network. As previously, the x and y axis represent the node coordinates, the z axis represents the simulation time in seconds and the colour band is used to represent the time evolution of the residual energy levels at each node in mJ. Observing the three figures, the network partition for Greedy routing can be seen clearly. The load is spread for Curvy routing keeping the network alive, but not as evenly as it happens for the Energy Aware routing where more nodes end up with similar residual energy level. More specifically, in Figure 6.3b, more nodes than Greedy routing are used due to the deployment of curved paths. The phenomenon of traffic accumulated closer to the sink node (marked on the figure) has not been eliminated, though, and the battery of these nodes has depleted a lot faster, leaving the ones at the periphery still with high amounts of residual energy.

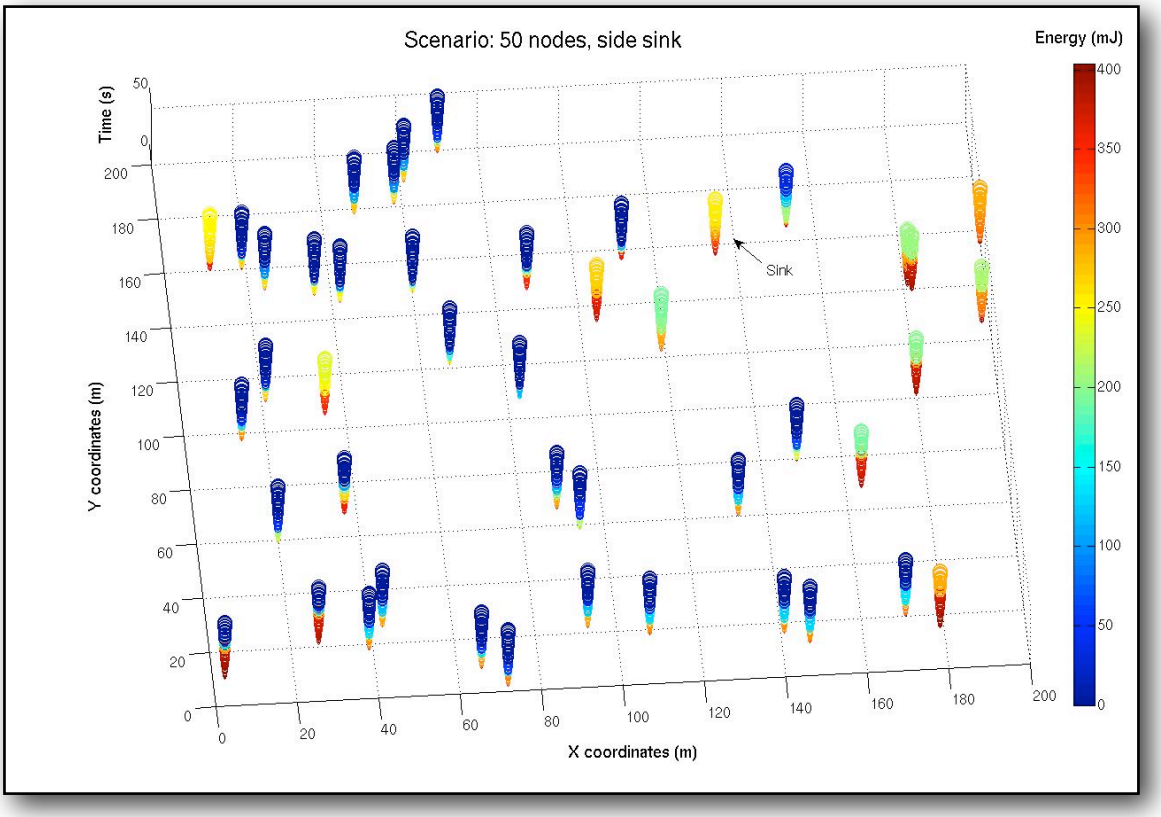


Figure 6.3a: Residual energy map, Greedy routing - 50 nodes

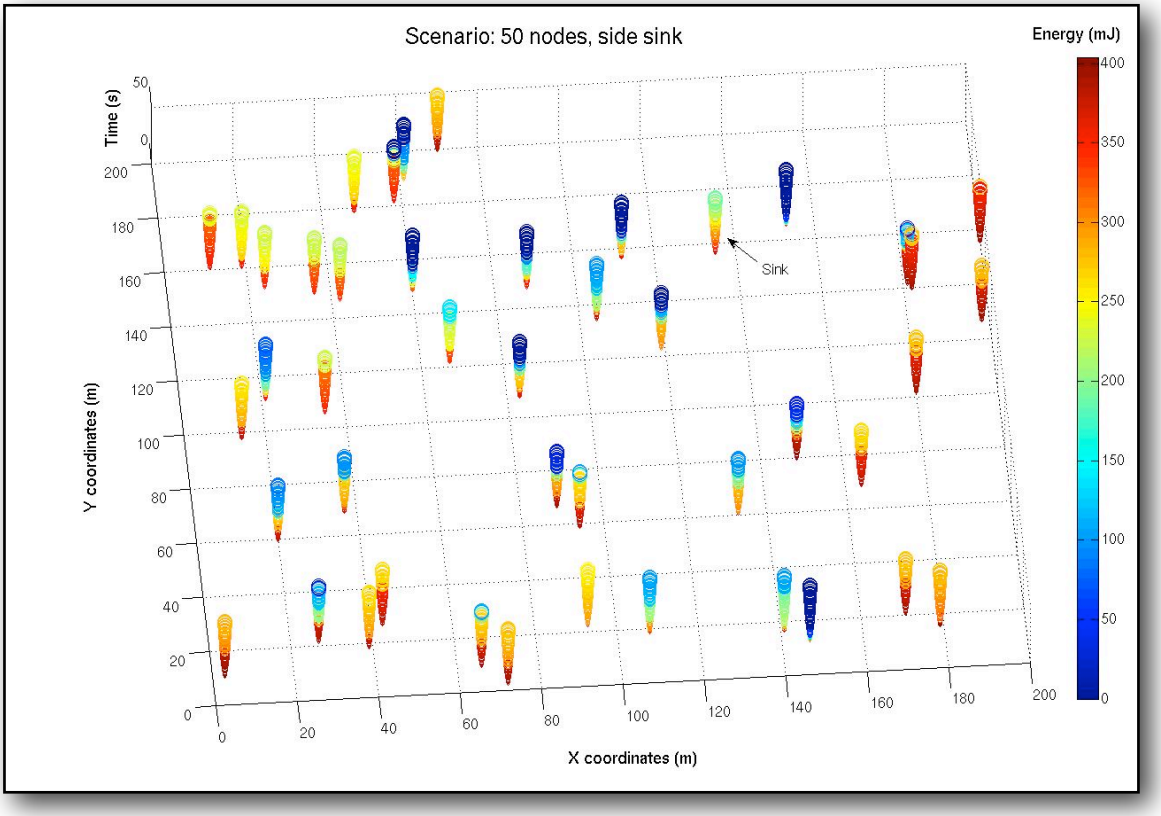


Figure 6.3b: Residual energy map, Curvy routing - 50 nodes

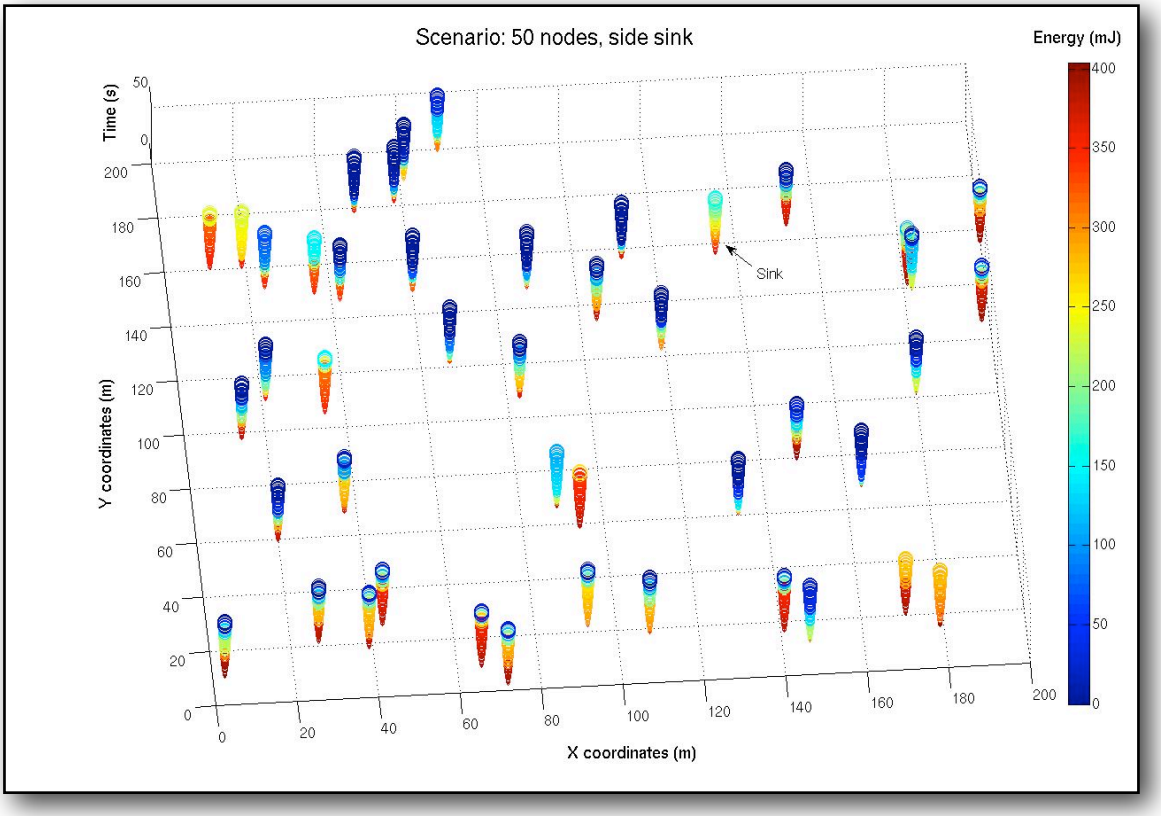
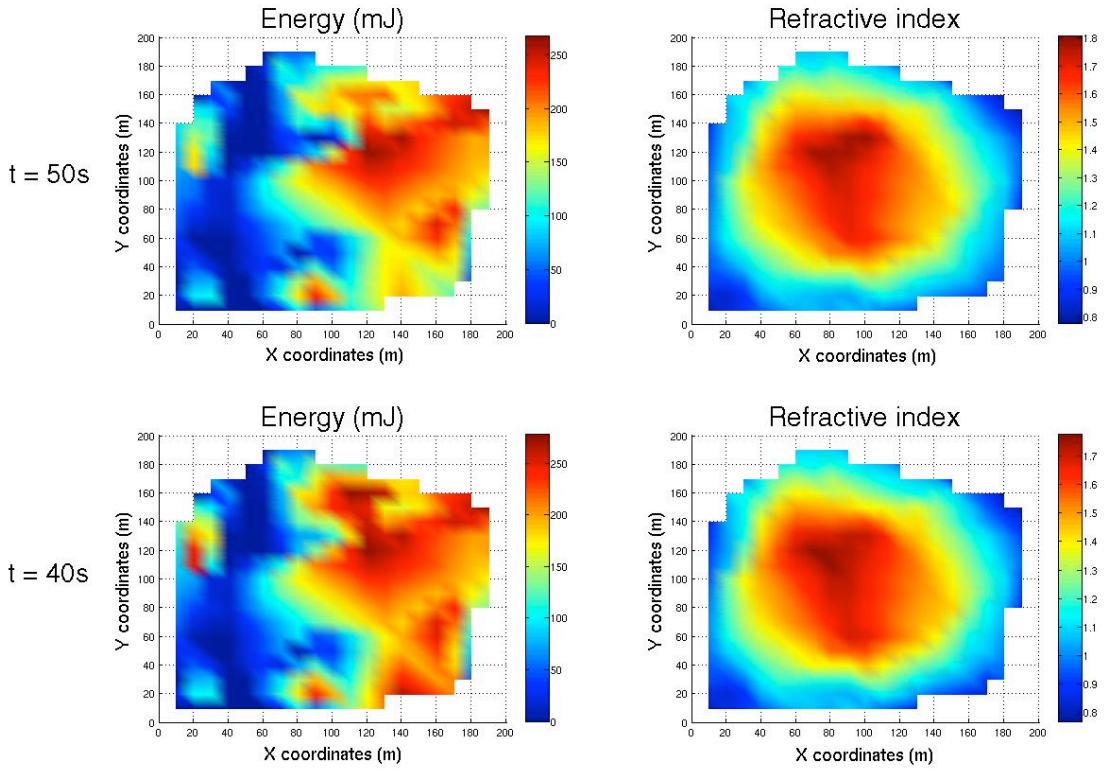


Figure 6.3c: Residual energy map, Energy Aware routing - 50 nodes

In order to explain the way the Energy Aware protocol works, snapshots of the residual energy distribution are presented at different simulation time slots, while the associated change of the refractive index is compared in parallel to show how it is adjusted to cope with the changes to the energy levels and avoid the low-energy areas. In order to construct this energy distribution, only local information has been gathered from one-hop neighbours at randomly selected simulation times. These energy levels are topologically averaged, meaning each neighbourhood provides an average residual energy level which can be used to construct the local energy deviation distribution. The screenshots are shown in pairs at 1, 5, 15, 30, 40 and 50 simulation seconds in Figure 6.4 below (continued over two pages). The full size figures are available in the Appendix C.1.



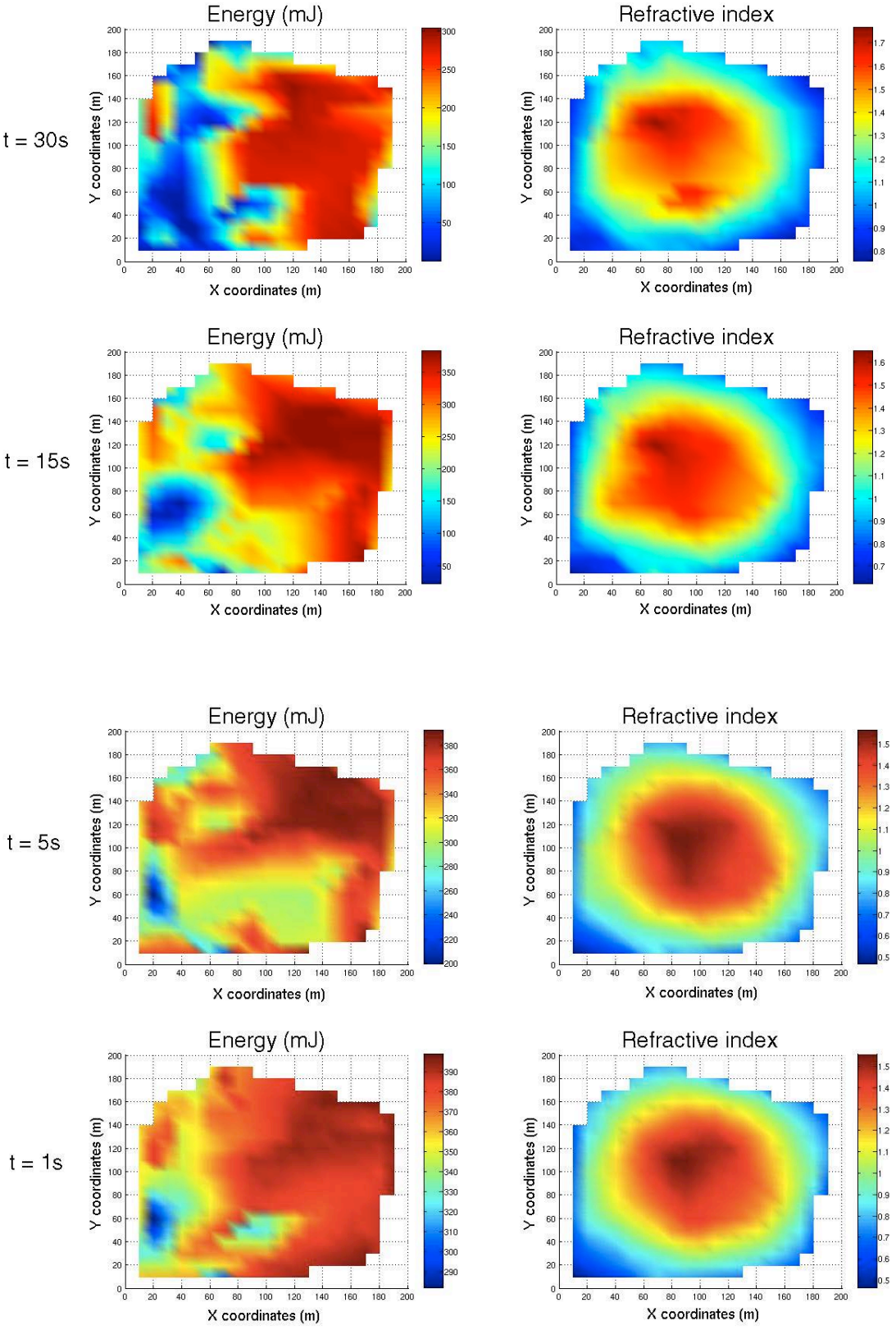


Figure 6.4: Snapshots of Energy and Refractive index distribution - 50 nodes

Starting from bottom to top of Figure 6.4, the energy distribution at the 1st simulation second is illustrated on the left side, while the refractive index one is on the right side. As time evolves and the energy distribution is altered (5 simulation seconds), the area at the bottom of the network experiences low energy levels and the refractive index is increasing towards the lower energy area in order to prevent routing paths passing through. The energy levels drop after 15 simulation seconds at the lower left part of the network area, where the refractive index becomes higher forcing routes to be more frequently used in the upper half of the network. The snapshot at 30 simulation seconds shows that residual energy levels become lower around the centre and the refractive index mirrors this well by developing two distinct local maxima in response. After 40 simulation seconds the energy of half the area is consumed and by looking at the refractive index distribution, the peak area is united again and is more intense towards the upper left side. The shots at the end of the simulation show that more energy is consumed at the top right of the network area as well and the refractive index ends up expanding to that part of the network as well.

The limitation of the assumption summarised in Chapter 3 on the spherically symmetric energy distribution appears in practice with topologies like the one in Figure 6.1 and can be seen in Figure 6.4 above. The energy distribution is skewed towards the right from the beginning, due to the uneven node concentration across the network. This uneven energy distribution is consequently caused after the first hello broadcast, since the size of each neighbourhood varies significantly and the cost of the information exchanged is not harmonious. Therefore, assuming spherical symmetry and constructing the initial refractive index according to it, limits its response time. Referring only to the initial distributions, this is the tradeoff paid for using local information. The refractive index distribution is "blindly" constructed assuming spherical symmetry and it does not always match the energy level one. It is easy to interpret that the more spherically symmetrical the initial energy

distribution, the more accurate the refractive index distribution and, therefore, the better the performance of the Energy Aware algorithm.

6.1.2 40-node scenario

Altering the network density and shape again, 40-node scenarios have been simulated keeping the sink node position towards one side of the network. The different topologies generated can be found in the Appendix B. Figure 6.5 depicts the one chosen to be presented here with density of $1x10^{-03}$ nodes per square metre.

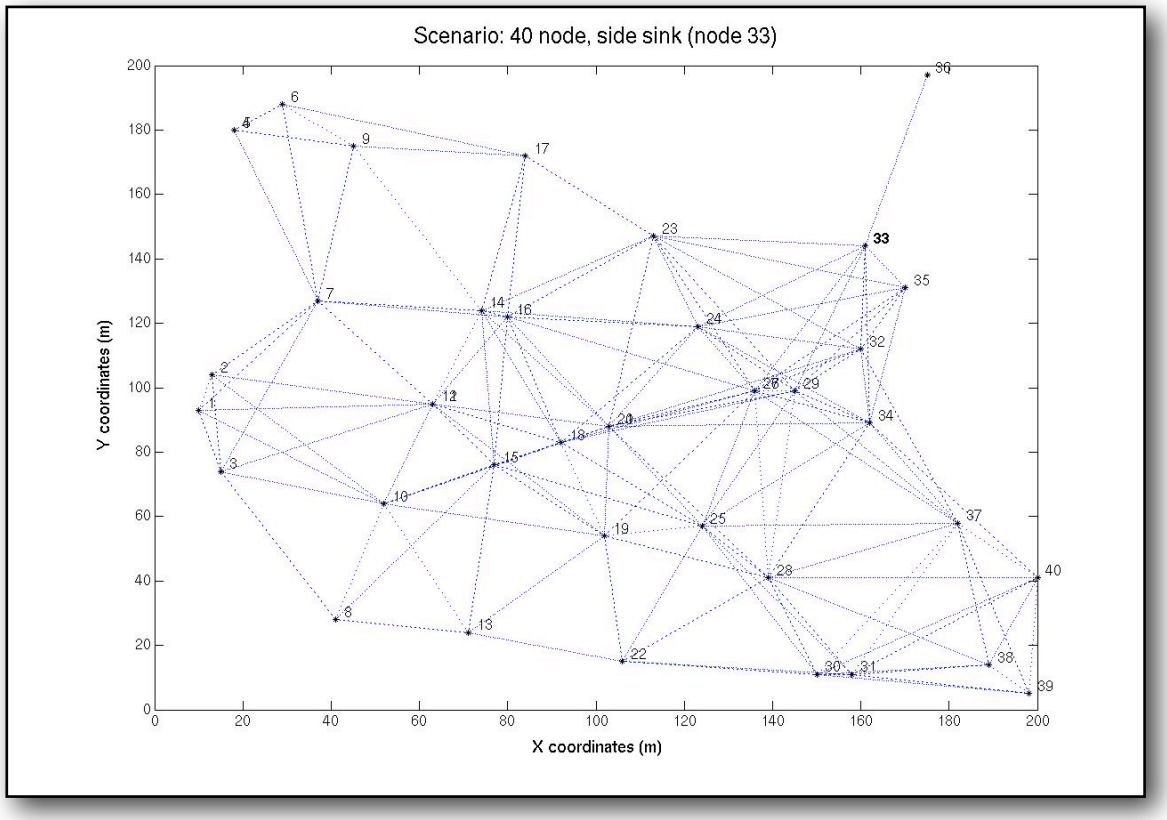


Figure 6.5: Topology with 40 nodes, side sink (node 33)

The average energy (mJ) per simulation time for Greedy, Curvy and Energy Aware routing is shown in Figure 6.6a. As previously, network partition occurs for Greedy routing after only 15 simulation seconds. Despite the fact that the slope of

the curve is slightly smaller (due to selected shorter paths) in Curvy routing, Energy Aware manages to keep the average energy at a higher level at the end of the simulation run. On the contrary, the standard deviation of the average energy (Figure 6.6b), shows a less steep initial slope for Energy Aware and a lower peak than Curvy routing. This indicates the ability of Energy Aware routing to manage the overall residue energy in a more efficient way across the network.

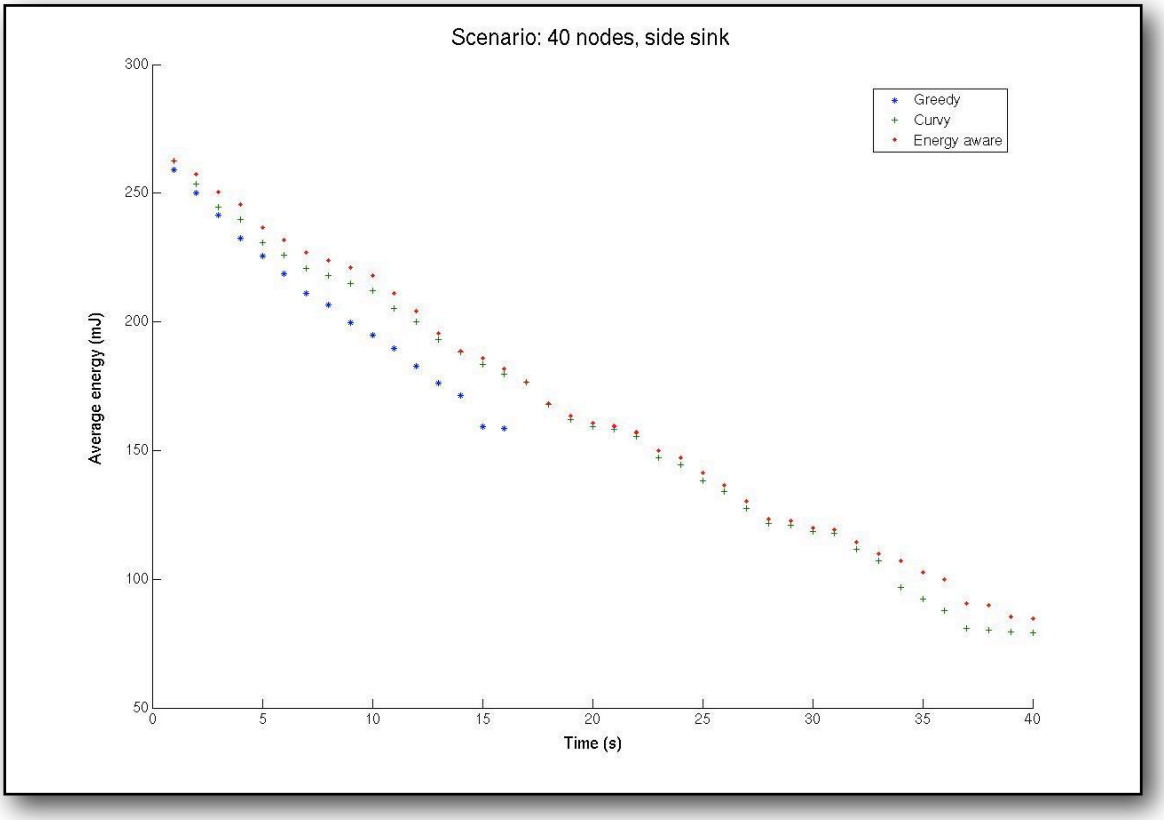


Figure 6.6a: Average energy vs Time - 40 nodes

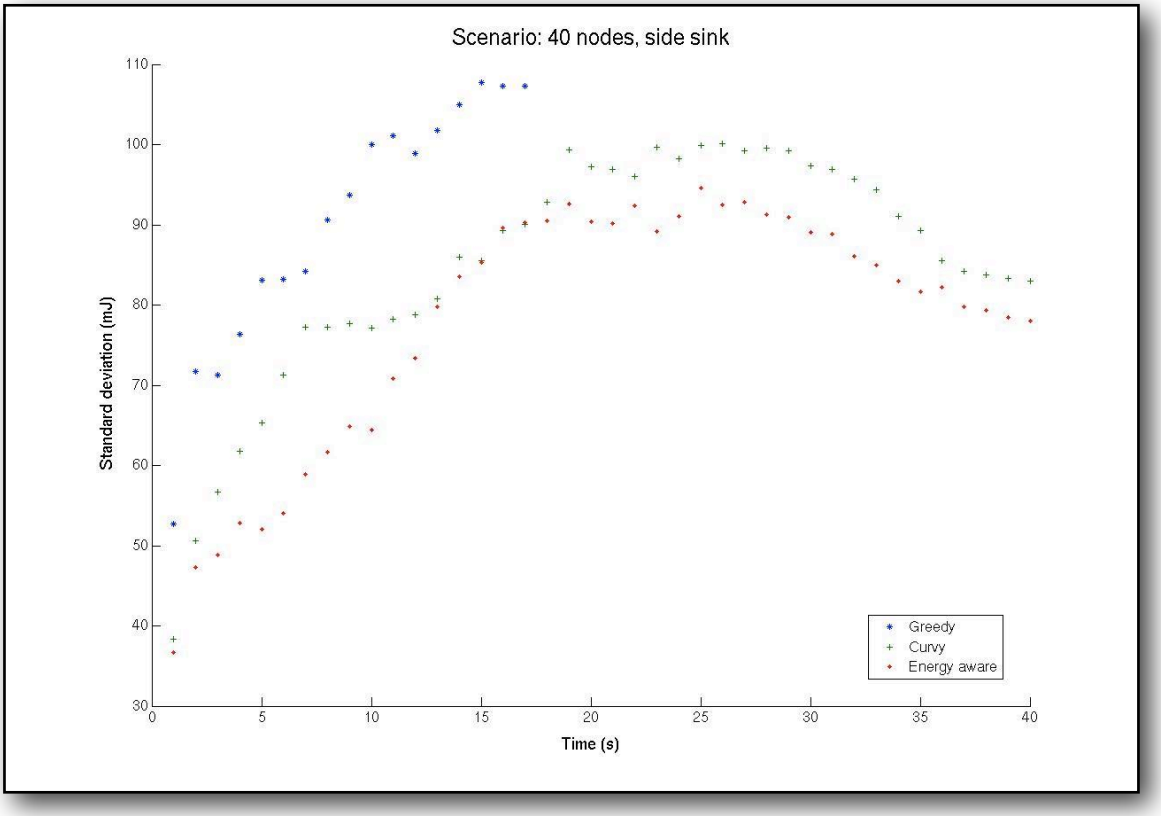


Figure 6.6b: Standard deviation of energy dissipation vs Time - 40 nodes

Observing the network load, in Greedy routing (Figure 6.7a) it is concentrated in the centre where, consequently, network partition is caused. Curvy routing performs better using more nodes than Greedy in total and spreading the load (Figure 6.7b), while Energy Aware routing moves it further away from the centre of the topology (Figure 6.7c). As can be seen, the sink is placed near the edge of the network and therefore, moving traffic away from it, is not possible. Looking at both Figures 6.7b and c, the left part of the topology is identical in terms of residual energy levels. What makes the difference between the two versions of the algorithm is the activity at the centre and the bottom right part of the topology. Energy Aware routing (Figure 6.7c) identified locally the energy level imbalance and led the paths through the bottom right corner, balancing the load between these nodes and the central ones.

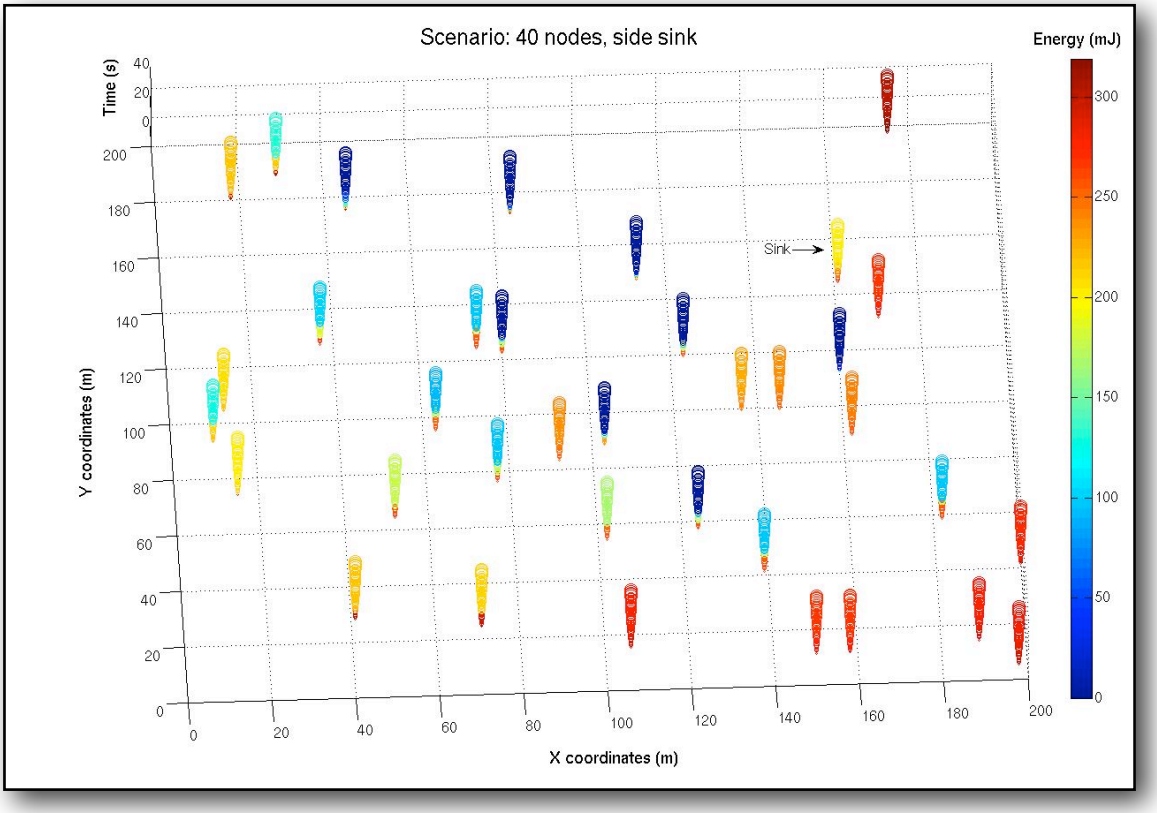


Figure 6.7a: Residual energy map, Greedy routing - 40 nodes

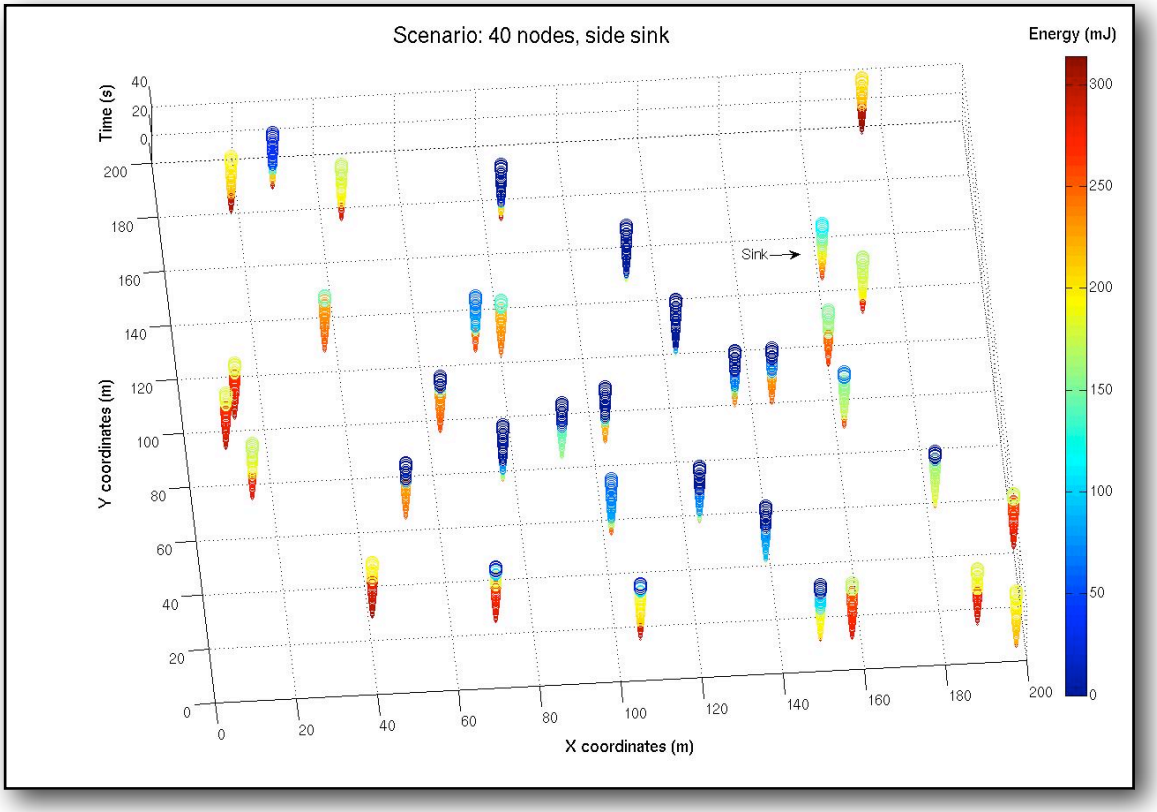


Figure 6.7b: Residual energy map, Curvy routing - 40 nodes

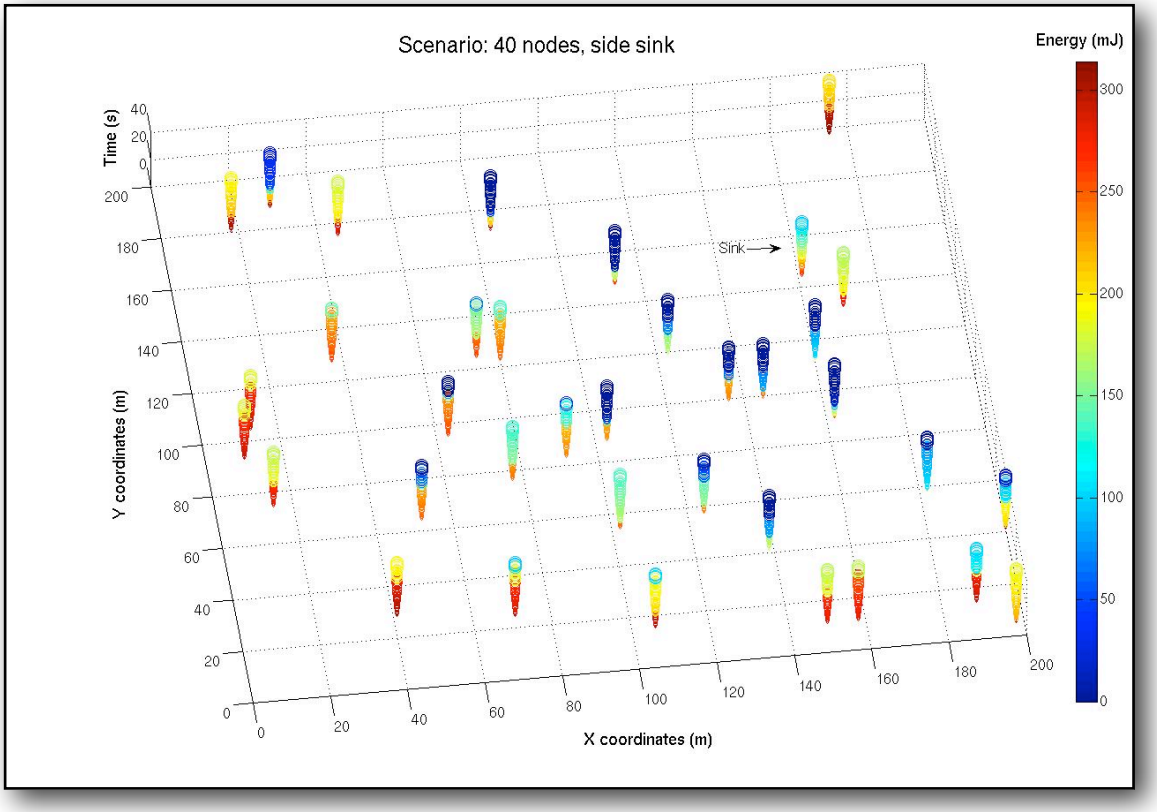


Figure 6.7c: Residual energy map, Energy Aware routing - 40 nodes

The energy level and the refractive index distribution are shown again at 1, 10, 30 and 40 simulation seconds (Figure 6.8). The full size figures can be found in the Appendix C.2.

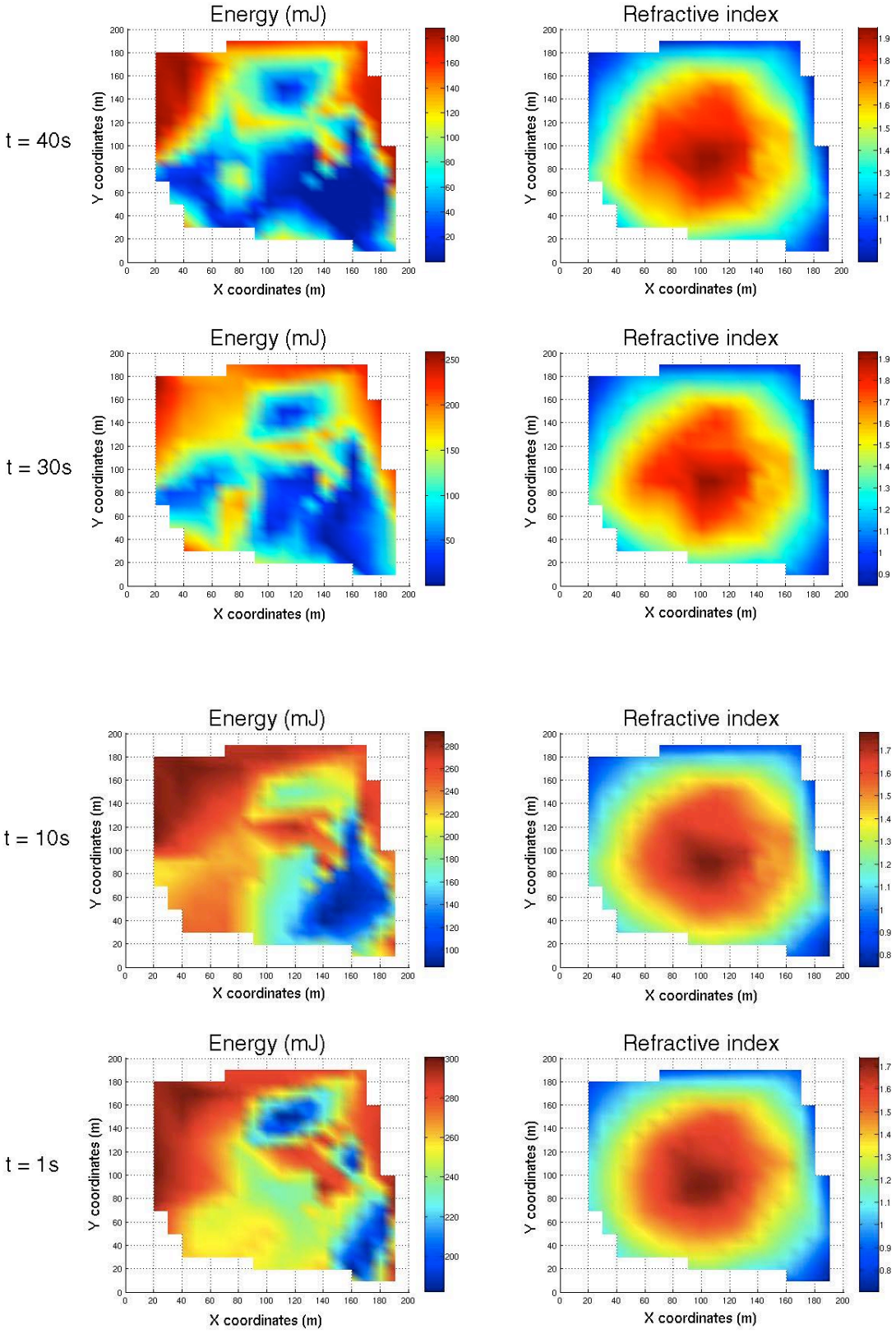


Figure 6.8: Snapshots of Energy and Refractive index distribution - 40 nodes

The initial energy level distribution is shown on the bottom left of Figure 6.8, followed by the refractive index one on the right at the 1st simulation second. Energy level changes are more rapid in this scenario, due to the smaller total number of nodes. Paths have been chosen to pass through the bottom right part of the network and therefore, after 10 simulation seconds, the higher refractive index values are biased towards this region as time progresses. The way the network is adjusting to the available energy distribution becomes a lot clearer at the 30th simulation second, where energy levels are low at the top, the bottom left and the bottom right. The sizeable refractive index can be seen to increase locally in exactly these areas. High amounts of energy have been consumed at the centre of the network as well by the end of the simulation (40 simulation seconds) and hence, the refractive index develops its highest peak right at the centre, while still avoiding the top right corner, where there is plenty of energy left and can be used for relaying messages.

6.1.3 35-node scenario

A number of simulations were run in a 35-node network with density of 0.875×10^{-03} nodes per square metre and the purpose is to examine the algorithm's behaviour while dealing with minimum neighbourhood size. This is the sparsest topology that guarantees connectivity and is represented in Figure 6.9. To provide comparison with theory, the theoretical lower bound for a connected network has been derived by Gupta & Kumar (1998) and, solved for the network parameters used here, it corresponds to ($31.831 \simeq$) 32 nodes in the area.

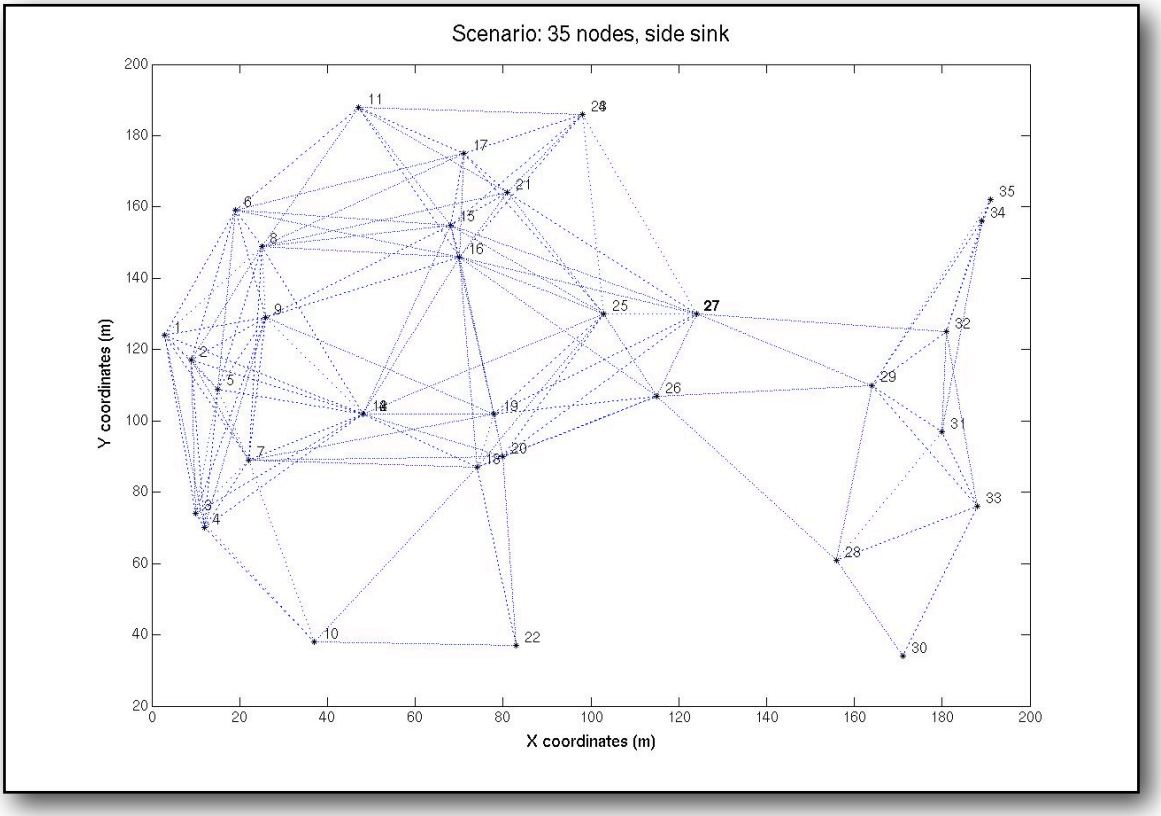


Figure 6.9: Topology with 35 nodes, side sink (node 27)

Figures 6.10a and b compare the three protocols in terms of average energy levels (mJ) and standard deviation respectively. Greedy routing fails again, as expected, causing network partition after 15 simulation seconds. The average energy level is kept a little higher for Curvy routing, but that is again because of the slightly shorter local path selection. The number of nodes in the network is as small as it can get without partitioning, which causes the Energy Aware routing to struggle in the middle of the simulation run with finding alternative routes to avoid low-energy areas.

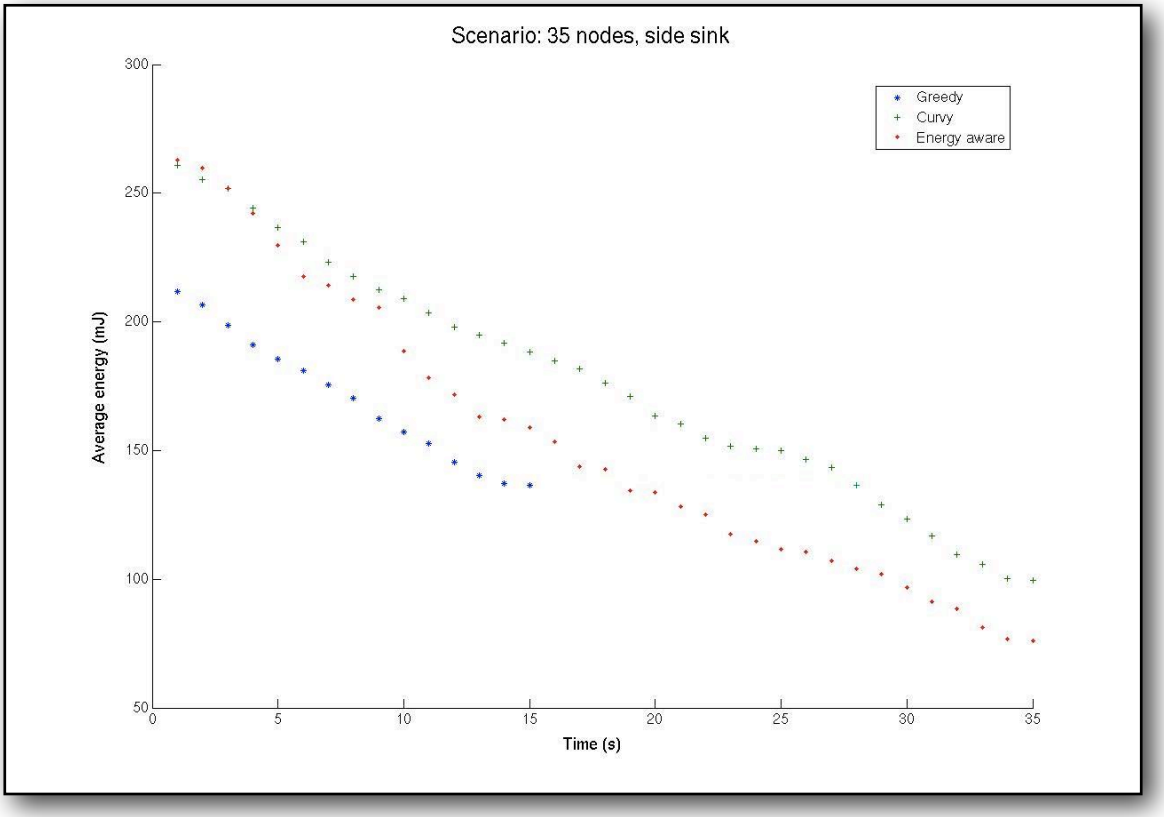


Figure 6.10a: Average energy vs Time - 35 nodes

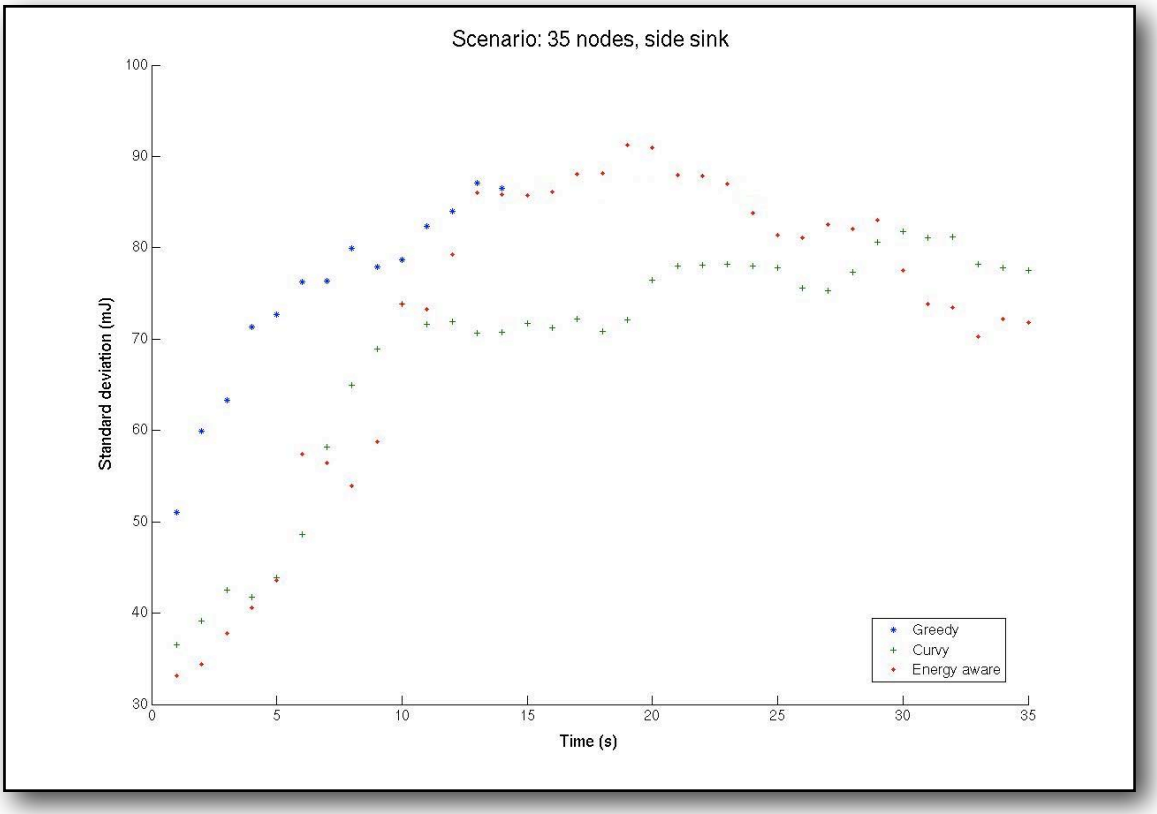


Figure 6.10b: Standard deviation of energy dissipation vs Time - 35 nodes

Looking at the network load distribution, the results are as expected; Greedy routing depletes all the energy at the centre of the network where network partition is caused (Figure 6.11a), while Curvy routing spreads it more evenly than Greedy across the network and uses the energy of more nodes, although without easing the traffic at the centre (Figure 6.11b). Energy Aware routing moves the load away from central locations, reserving energy levels and keeping the network alive for longer (Figure 6.11c). Due to the shape of the topology, it is hard to maintain balance across the network. However, Energy Aware routing utilises the nodes at the left part of the network area with the cost of increasing the path length.

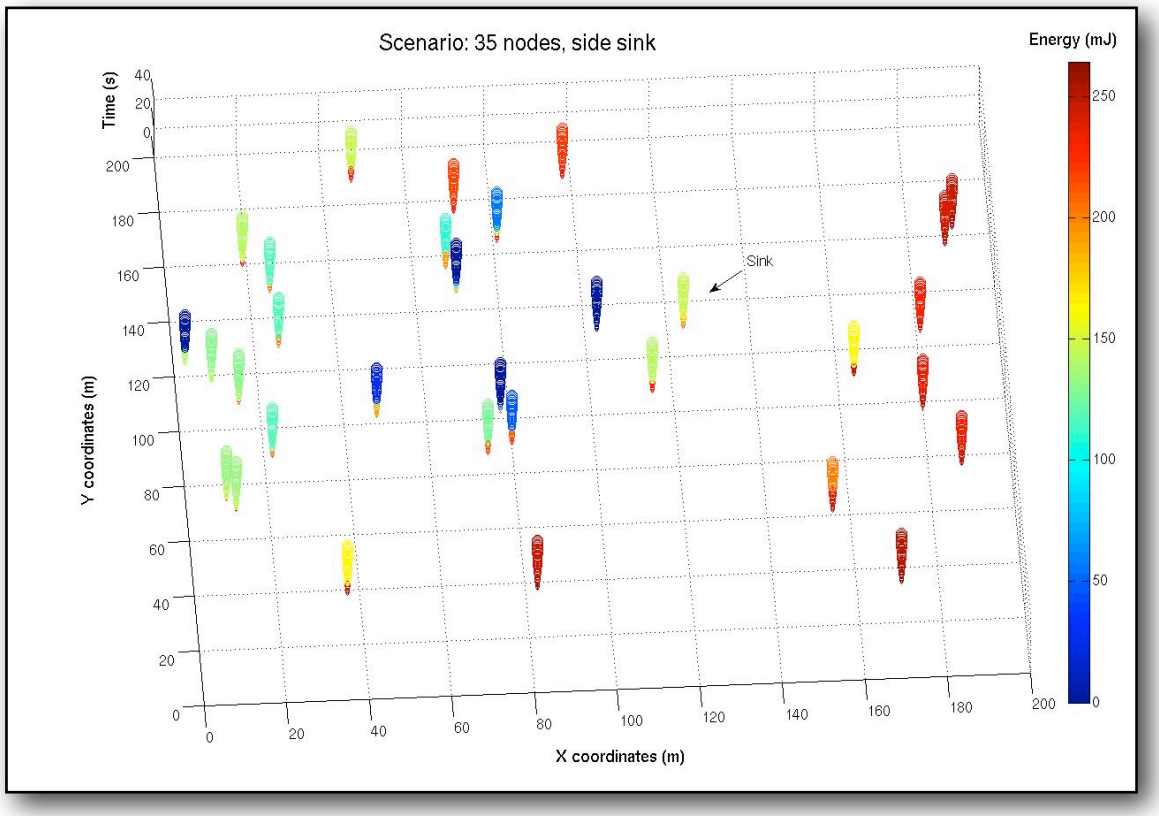


Figure 6.11a: Residual energy map, Greedy routing - 35 nodes

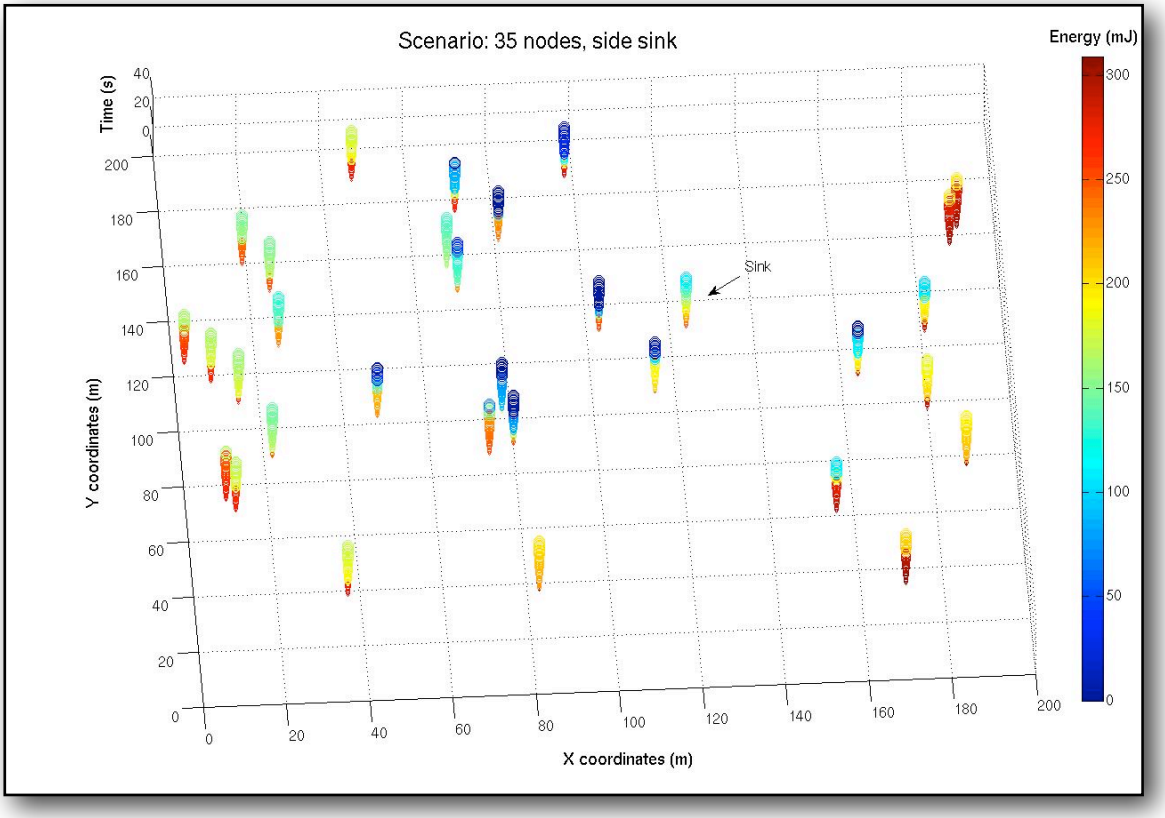


Figure 6.11b: Residual energy map, Curvy routing - 35 nodes

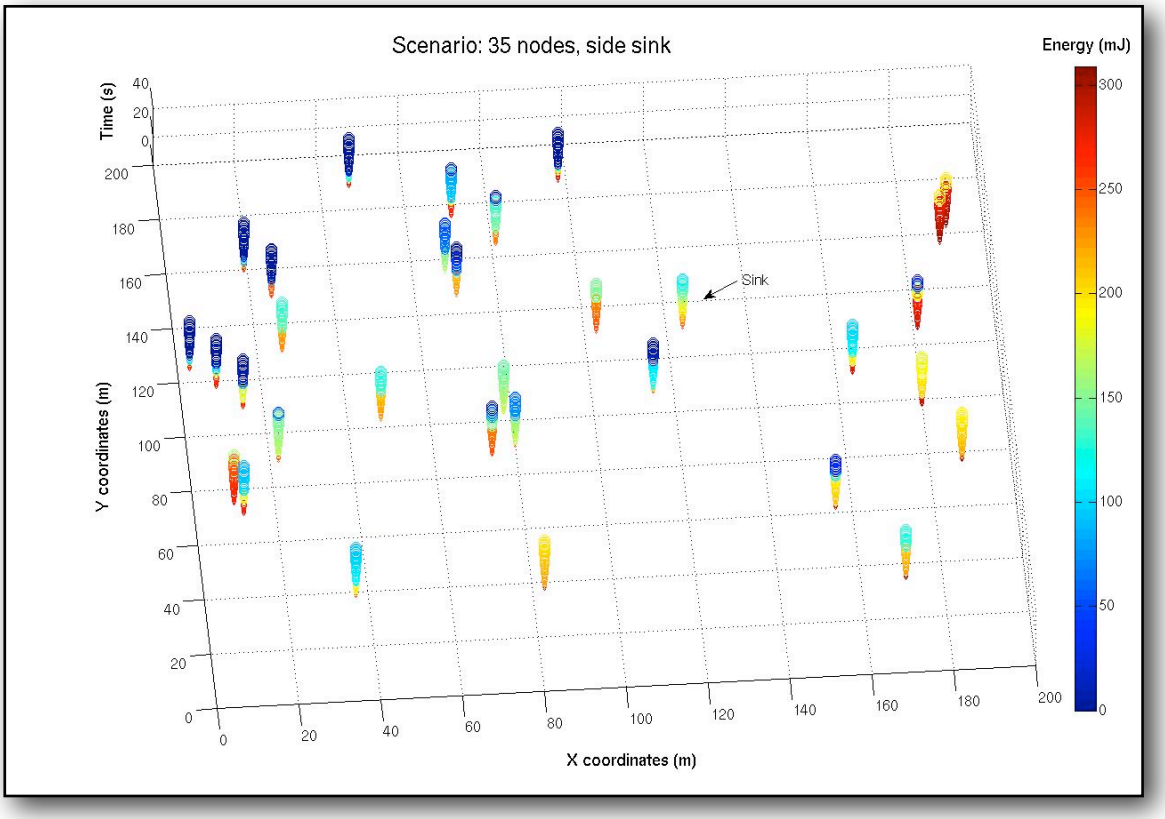


Figure 6.11c: Residual energy map, Energy Aware routing - 35 nodes

Once again, the protocol's behaviour can be verified by observing the energy level and refractive index distribution at specific simulation seconds (Figure 6.12). Full size figures of this scenario are available in Appendix C.3.

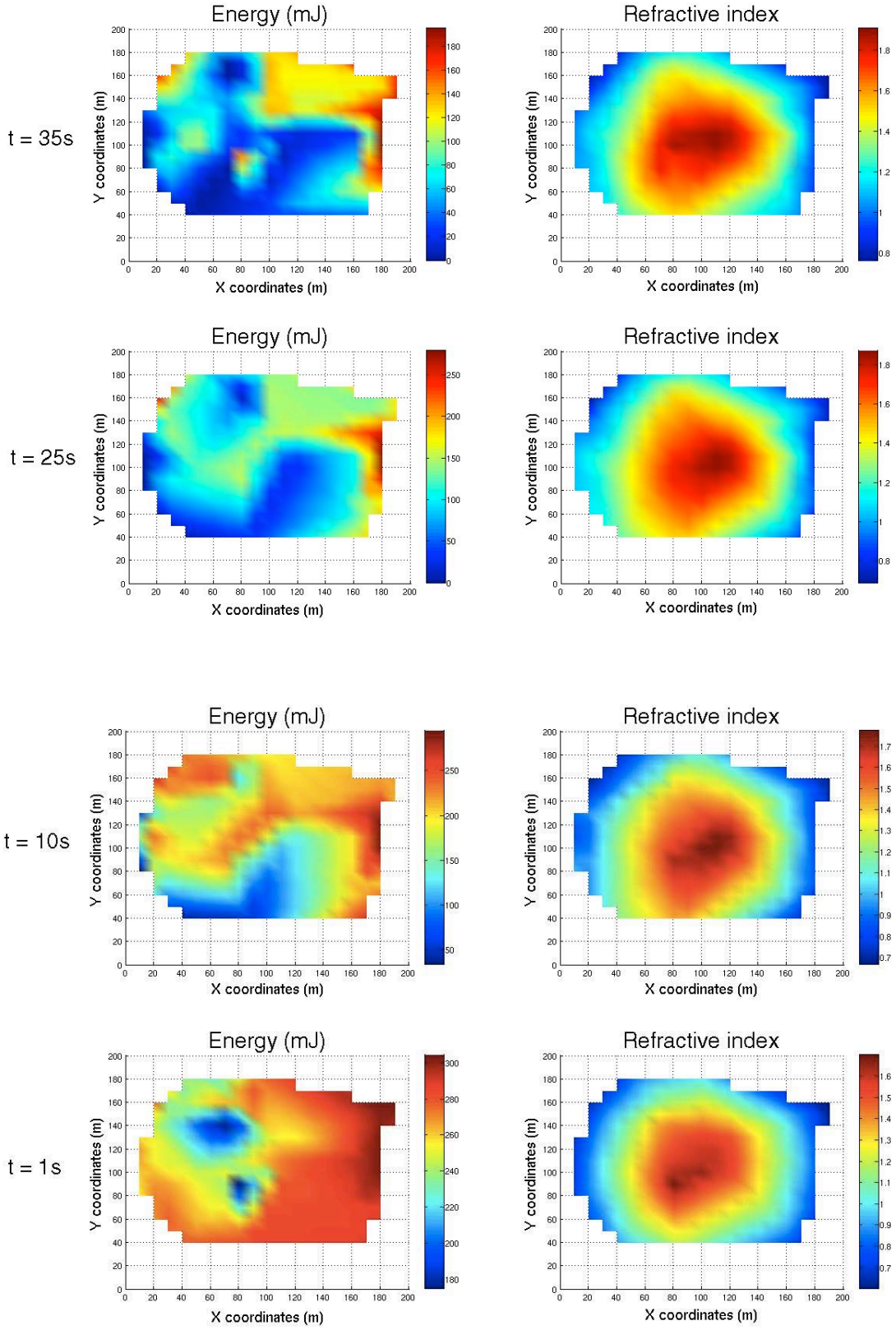


Figure 6.12: Snapshots of Energy and Refractive index distribution - 35 nodes

Using the sparsest scenario possible (while maintaining connectivity), makes more visible the way the algorithm behaves. Figure 6.12 (bottom first) shows the energy and refractive index distribution left and right respectively at the beginning of the simulation. As routing is designed to use curves and avoid the centre, energy levels are lower around it and at the bottom of the network after 10 simulation seconds. The refractive index acts on the change by moving towards that area while increasing its peak. As energy consumption surrounds the centre of the network (25 simulation seconds), the refractive index keeps the previous area covered, but moving the peak slightly and expanding to the top as well. By the end of the simulation (35 seconds), low energy levels are experienced all over the network apart from the top right corner and the refractive index is "expanding" its peak to protect the centre and keep the network alive.

6.1.4 100-node scenario

The transmission range was, then, targeted to experiment with, since it is one of the factors to determine the proposed solution's behaviour and performance. The transmission range also defines the network size, provided the network area and the size of the neighbourhood are kept constant. The transmission range in all previous scenarios was set to 62 m, as introduced in Chapter 4. In this scenario the range is decreased by 1/3, namely set to 41 m, in the same area of 200m x 200m with the same neighbourhood size as in the 35-node scenario 6.1.3. This yields network sizes of 100 nodes and one of those chosen to be presented here is illustrated in Figure 6.13 below.

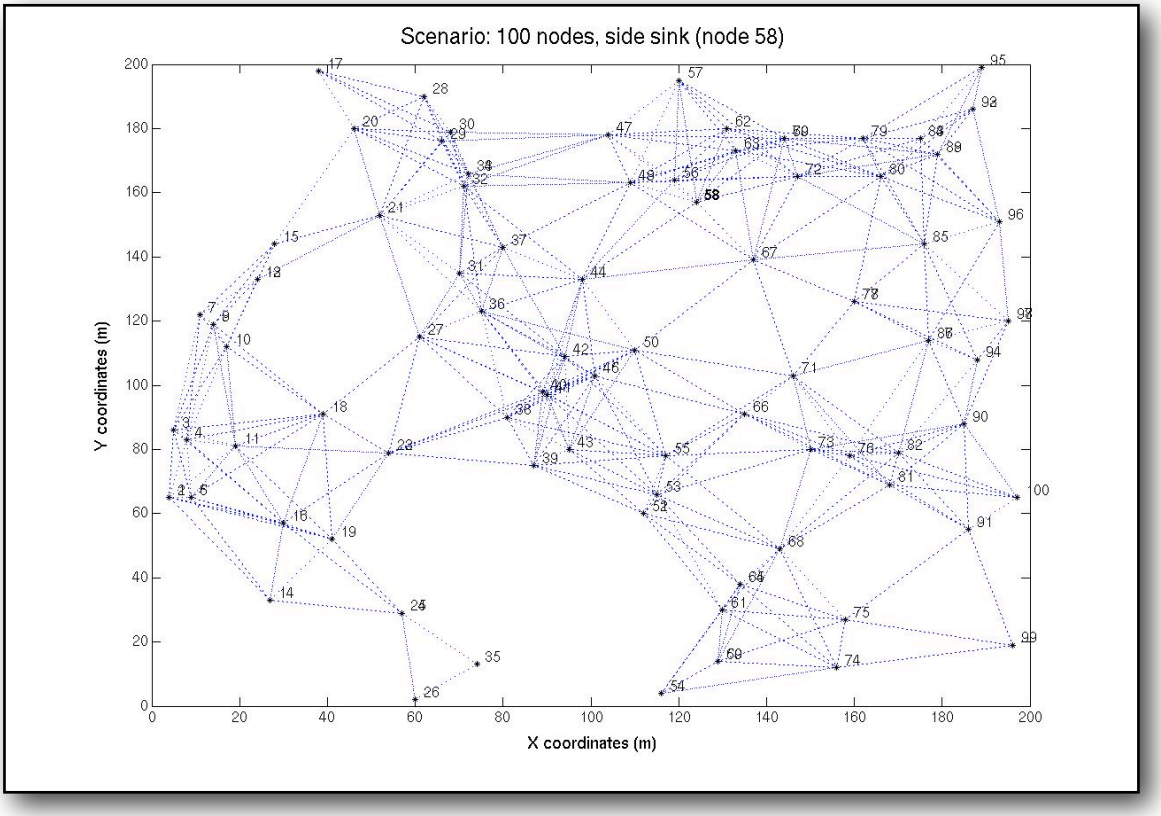


Figure 6.13: Topology with 100 nodes, side sink (node 58)

The average energy of nodes vs time in simulation seconds is shown in Figure 6.14a and it is clear that Greedy routing faced network partition after 59 seconds, where Curvy routing and Energy Aware routing keep the average energy steady and quite high during the simulation. The average energy level is initially lower in Greedy routing and the reason is the invocation of face routing right after the first hello broadcast, which led to enhanced energy consumption. It is also worth mentioning that this is the first scenario where Energy Aware routing has no path length disadvantage against Curvy routing, proved by the nearly identical average energy slope. This effect is also visible in the standard deviation of the energy dissipation in Figure 6.14b, which shows that Energy Aware routing keeps it lower than Curvy routing during the whole simulation run.

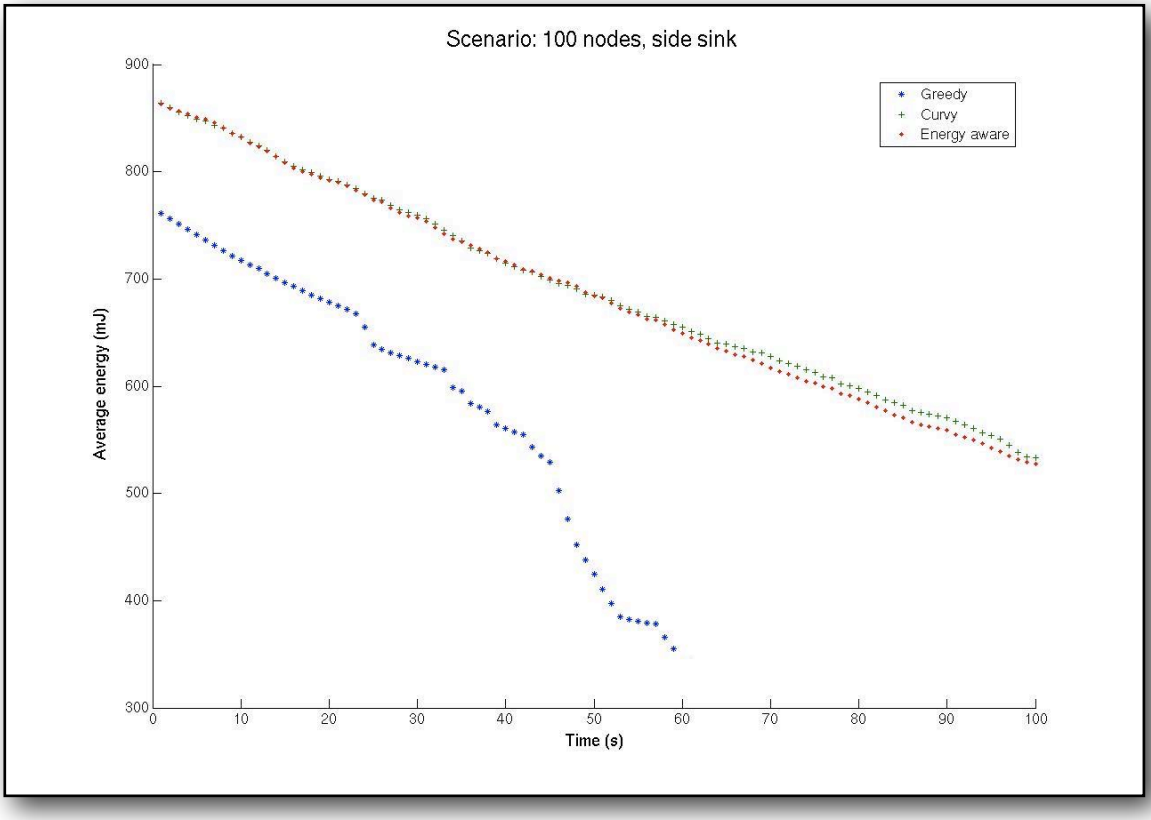


Figure 6.14a: Average energy vs Time - 100 nodes

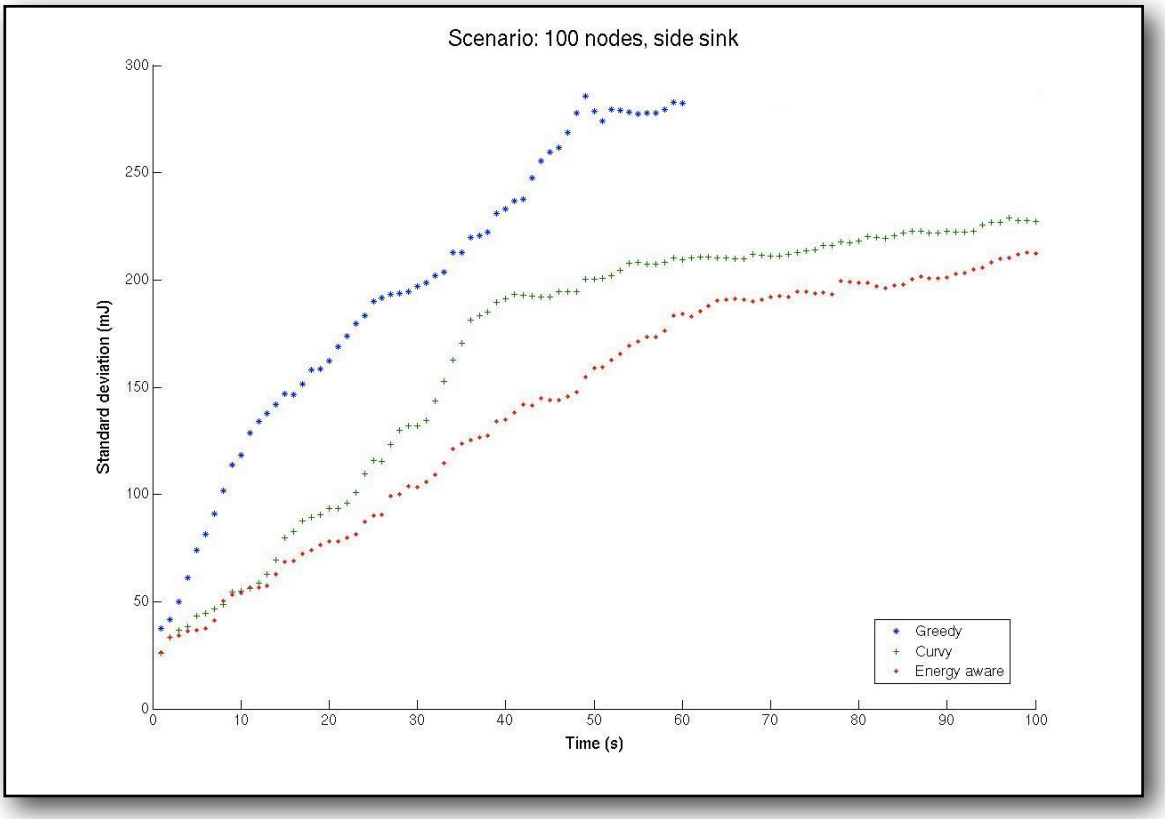


Figure 6.14b: Standard deviation of energy dissipation vs Time - 100 nodes

Observing Figures 6.15a, b and c, the residual energy map for Greedy routing, Curvy routing and Energy Aware routing is represented respectively. As expected, the centre of the network suffers from early energy depletion in Greedy routing and key relay nodes run out of battery prematurely, leaving no forwarding choices to the algorithm. The bulk distribution of Curvy routing does not meet any difficulty in establishing the curved paths and completing the simulation, while the load is spread across the topology. The intelligence of Energy Aware routing is brought out by moving the network load away from the centre and spreading it around a larger area at both top and bottom part of the network.

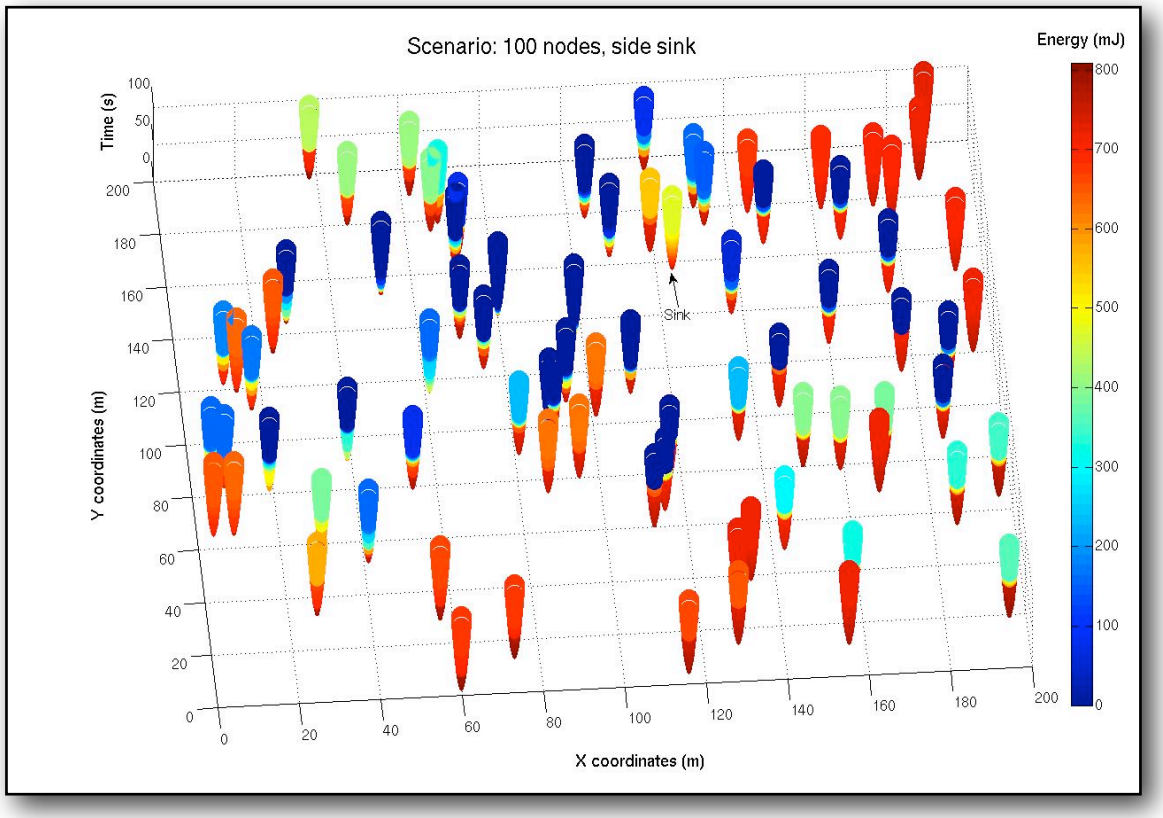


Figure 6.15a: Residual energy map, Greedy routing - 100 nodes

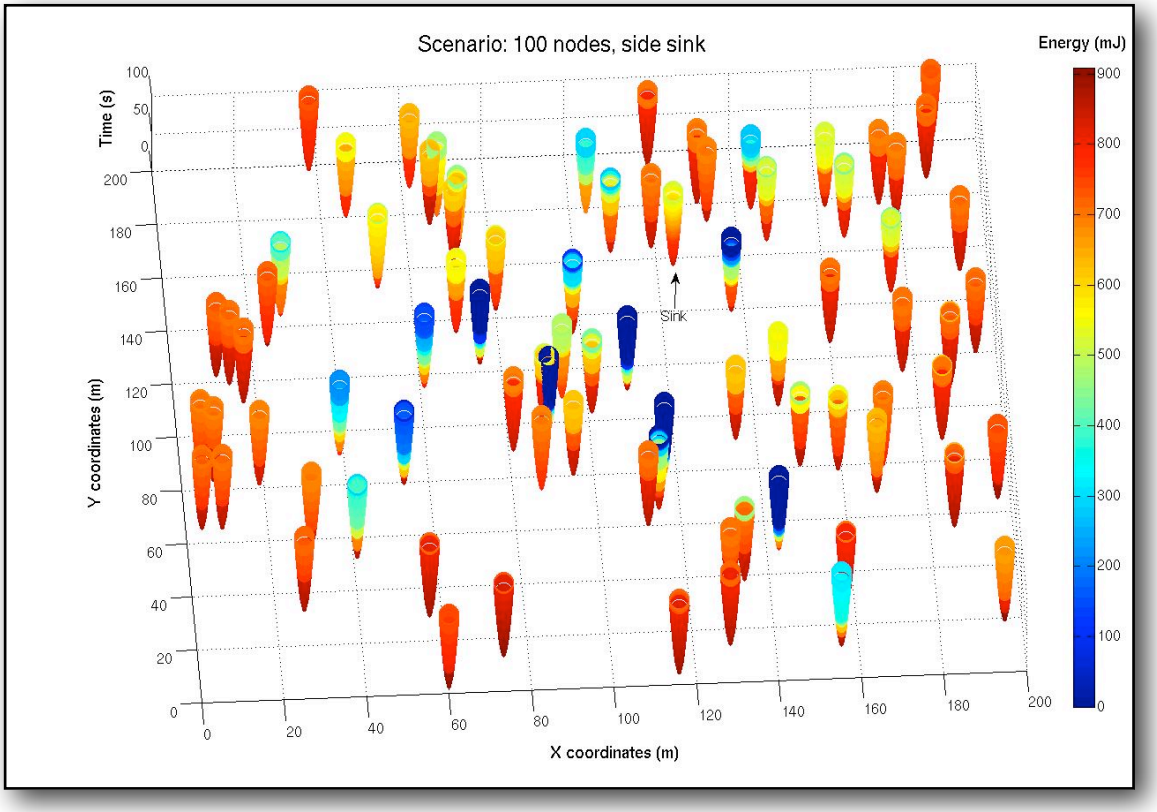


Figure 6.15b: Residual energy map, Curvy routing - 100 nodes

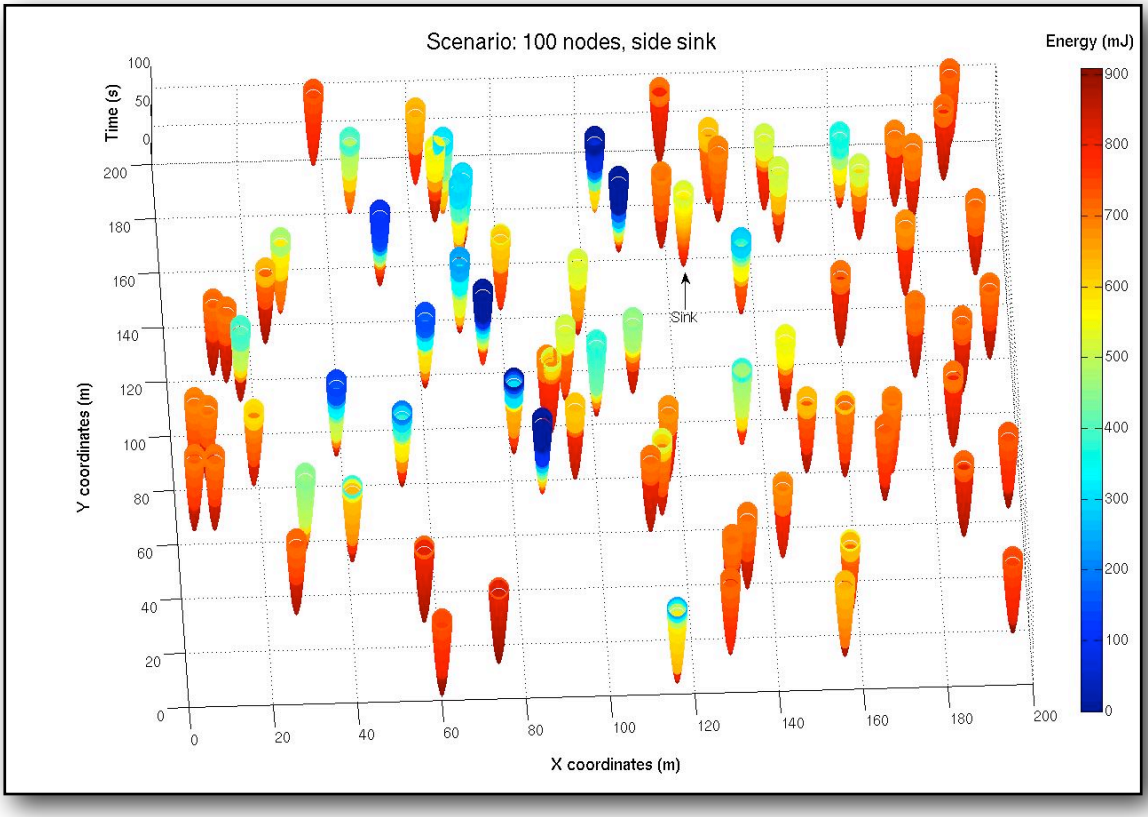
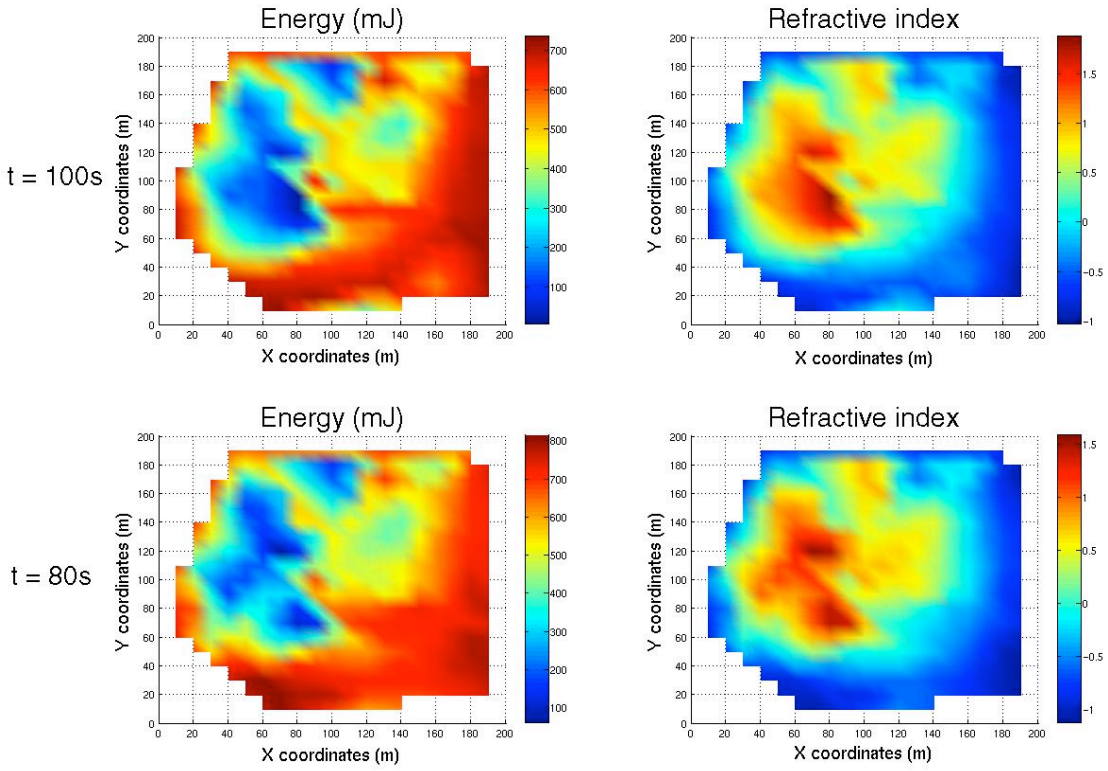


Figure 6.15c: Residual energy map, Energy Aware routing - 100 nodes

Figure 6.16 below (continued in two pages) shows the energy and refractive index distribution at chosen simulation seconds. The full size figures are available in Appendix C.4.



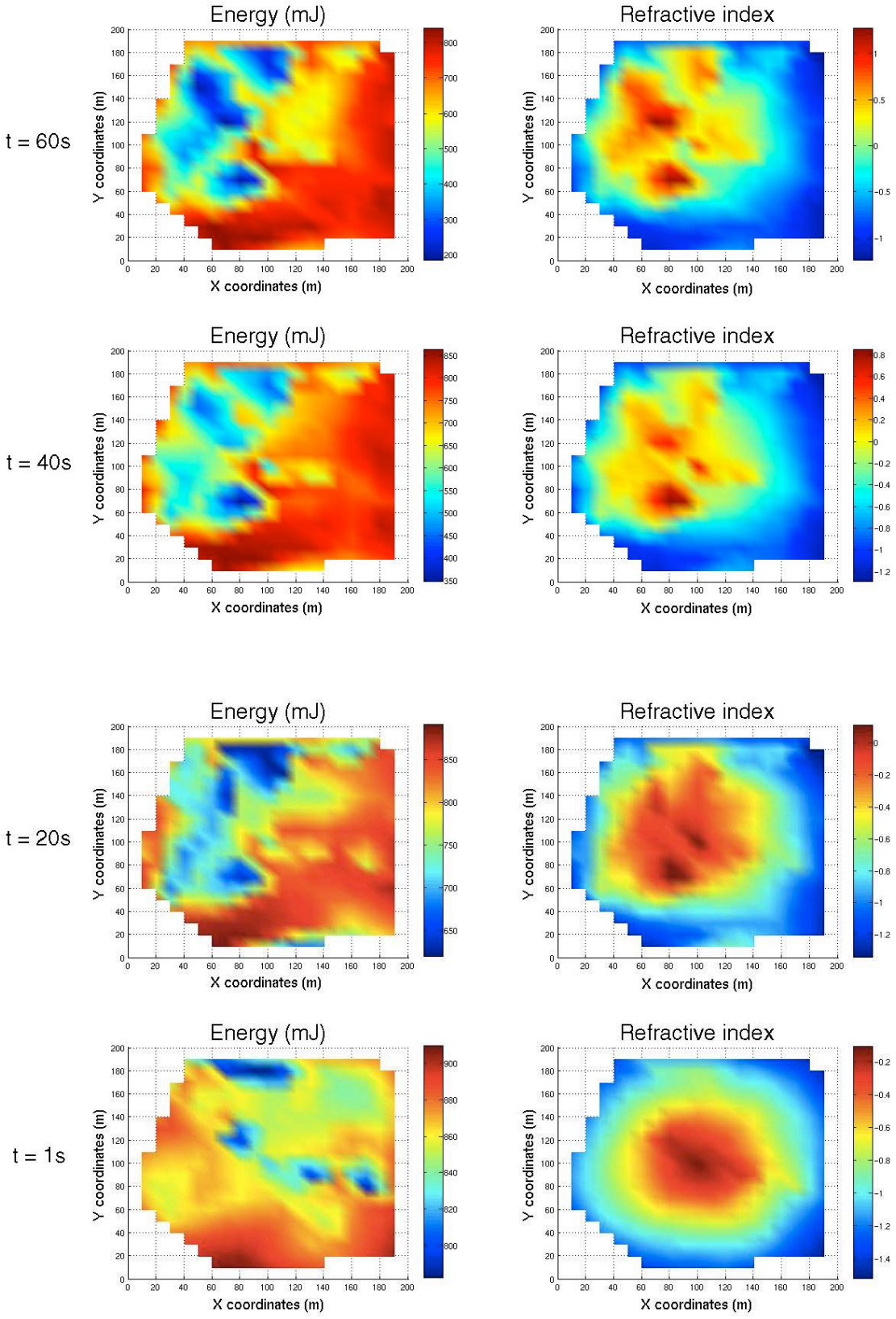


Figure 6.16: Snapshots of Energy and Refractive index distribution - 100 nodes

The initial refractive index distribution at the right bottom of Figure 6.16 is centred on the topology by default, waiting for the initial energy distribution after the first hello broadcast. After 20 simulation seconds low energy regions appear at the top of the network and below its centre, making the refractive index respond to them. At the 40th simulation second, low-energy areas are more distinct at the left part of the network, where the refractive index develops its peaks. At the 60th second, these areas are enlarged while energy depletion is also expanded to the upper right part of the network. The refractive index distribution has reflected the rise of these areas and is expanded to cover them. Approaching the end of the simulation at the 80th second, the low-energy regions seem deeper and the centre of the network suffers from energy depletion as well. Looking at the refractive index distribution on the right, it has increased its value where those areas exist. Finally, the end of the simulation finds the two distributions approximately complementary, with the refractive index reflecting the topological energy alterations.

6.1.5 140-node scenario

Keeping the network parameters the same as in the previous scenario and decreasing the transmission range even more to 31 m, the network size is now increased to 140 nodes. These topologies find it harder to establish connectivity as well as to discover path routes to the sink node. An example of those is presented below and depicted in Figure 6.17.

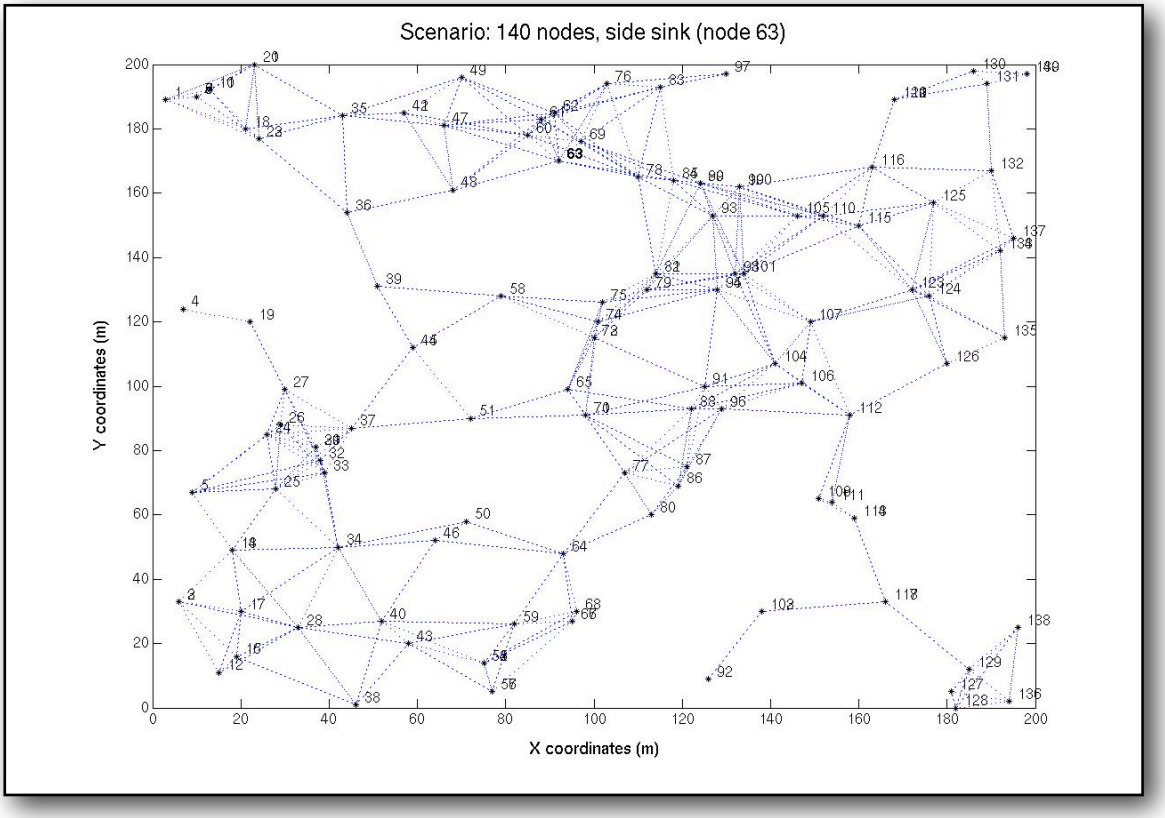


Figure 6.17: Topology with 140 nodes, side sink (node 63)

Figure 6.18a represents the average energy of the nodes throughout the simulation. Due to the extreme topology of the network, the difference between the behaviour of the routing protocols is visible. Greedy routing fails to complete the simulation and network partition is caused very early after 41 seconds. Both Curvy routing and Energy Aware routing control the energy depletion very well with an advantage to the latter. The standard deviation of the energy dissipation is depicted in Figure 6.18b, where clearly Greedy routing suffers from high values (due to lack of forwarding choices), while the other two keep it significantly lower.

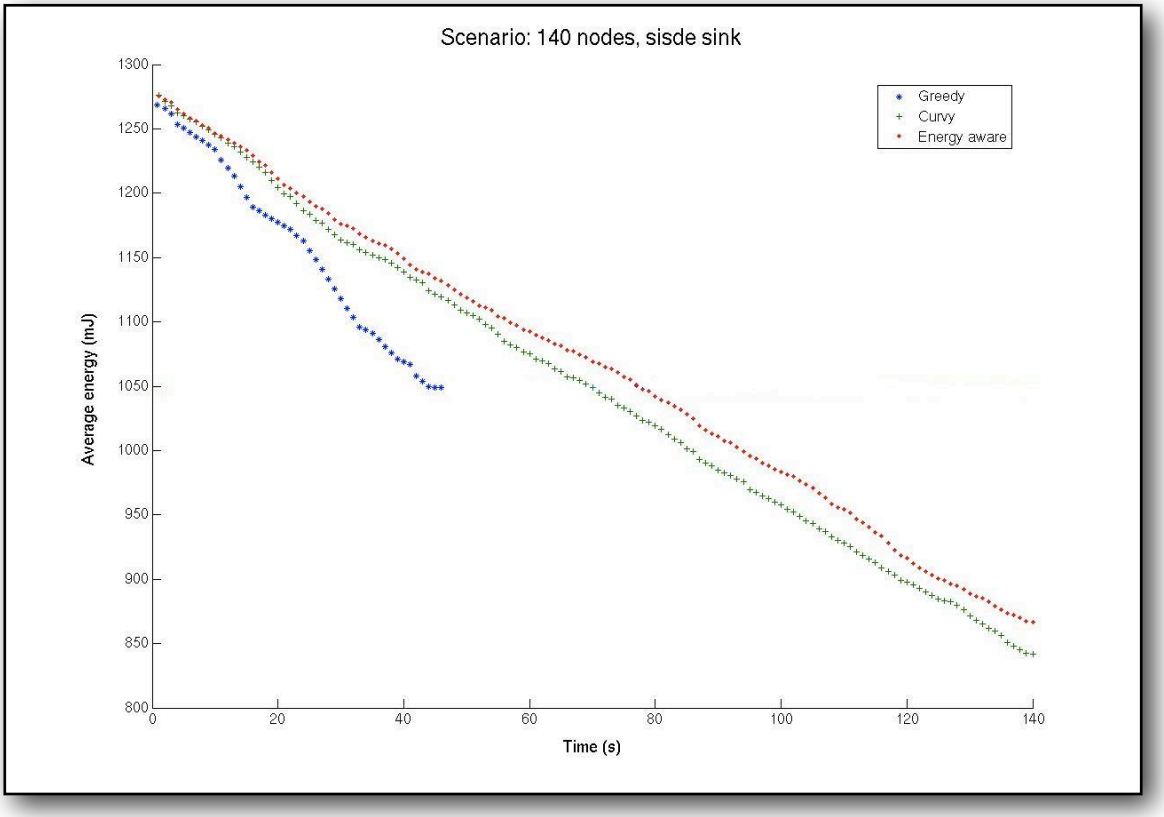


Figure 6.18a: Average energy vs Time - 140 nodes

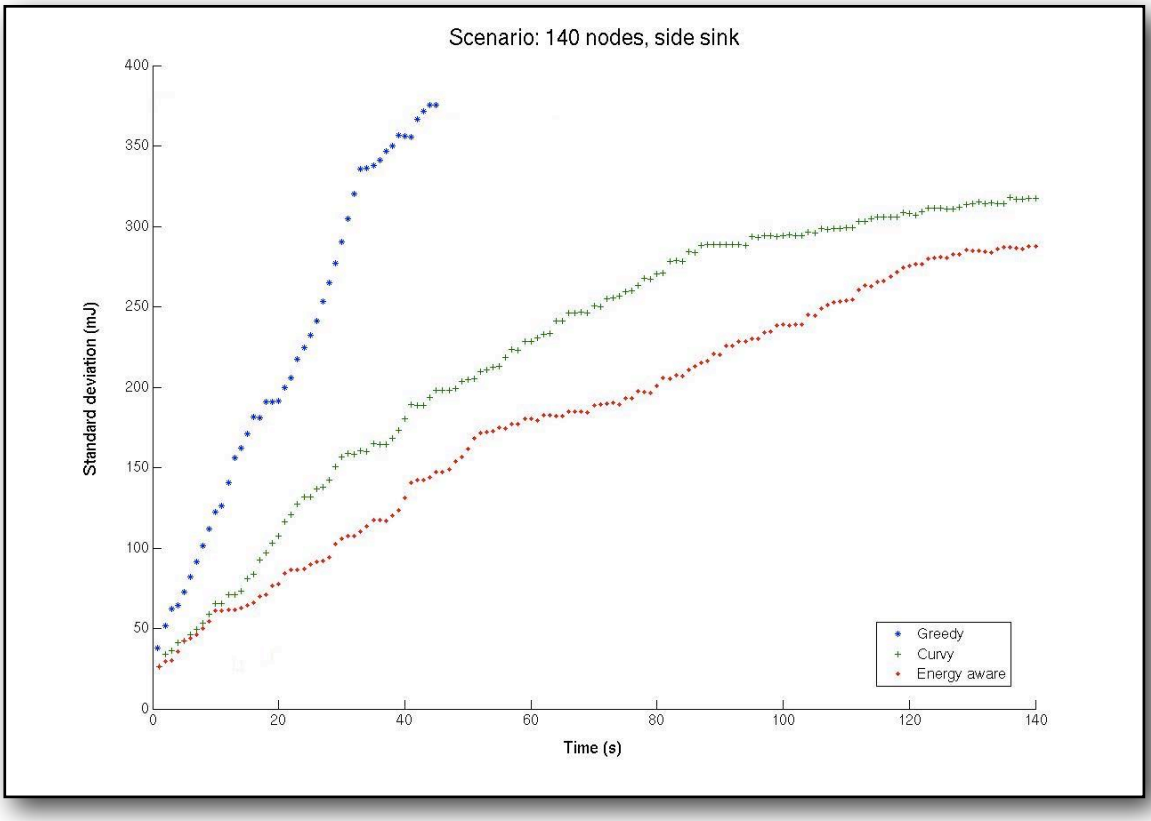


Figure 6.18b: Standard deviation of energy dissipation vs Time - 140 nodes

As the residual energy map shows, the load in Greedy routing (Figure 6.19a) is spread very unevenly causing nodes at key locations to suffer. The crowded centre effect here has not appeared by the time network partition occurred, due to the topological structure. By looking at the links forming neighbourhoods in Figure 6.17, it can be seen that network partition has been caused, because of the battery exhaustion of certain nodes holding together different parts of the network. Curvy routing in Figure 6.19b and Energy Aware routing in Figure 6.19c behave as predicted, spreading the load across the network. In the case of the latter, the network load is efficiently balanced locally and moved away from the centre of the topology. This can be explained as part of the topology that favours the curved paths, since many nodes are located at the periphery of the network and the path length is not an issue anymore.

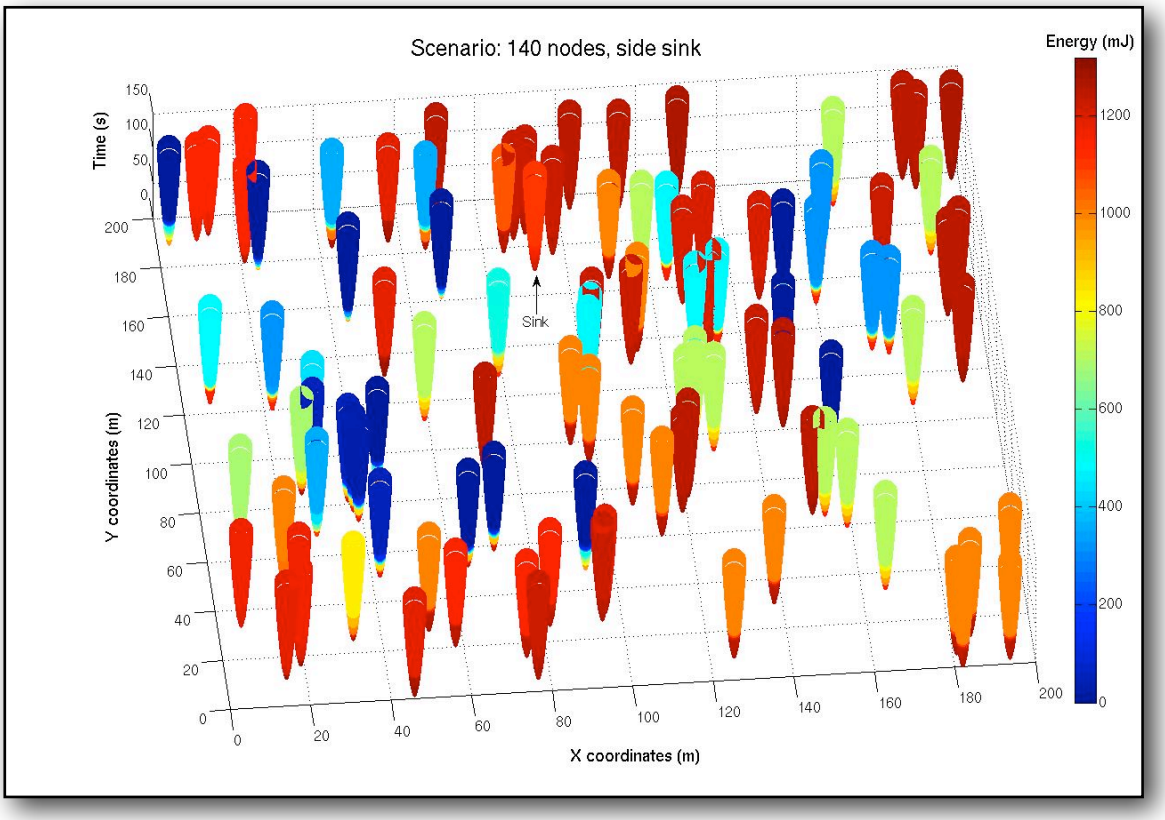


Figure 6.19a: Residual energy map, Greedy routing - 140 nodes

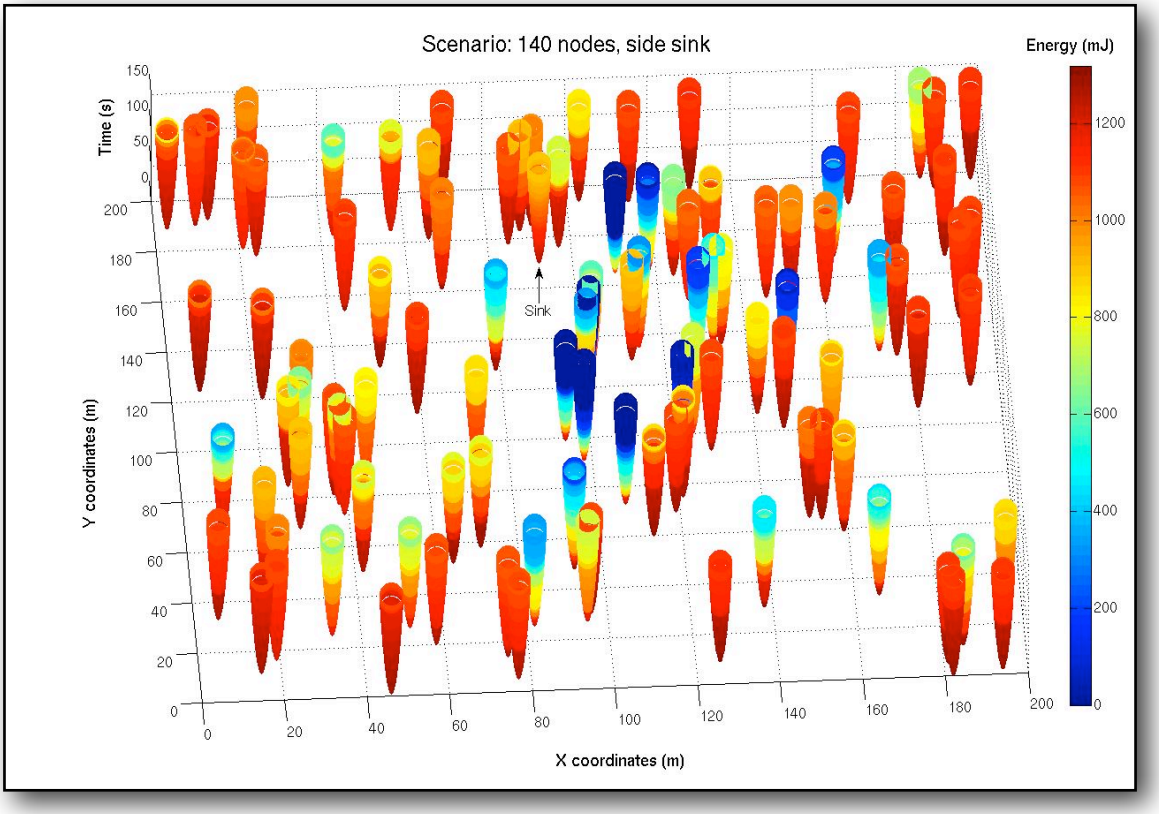


Figure 6.19b: Residual energy map, Curvy routing - 140 nodes

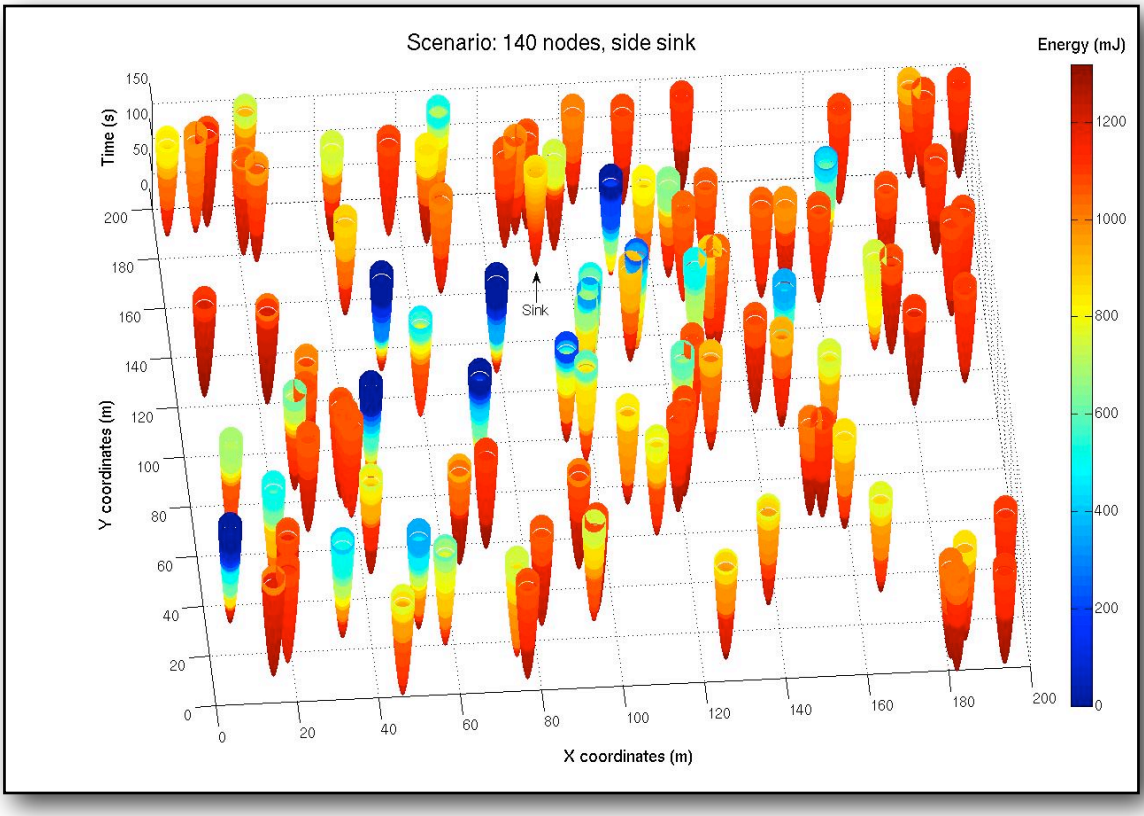
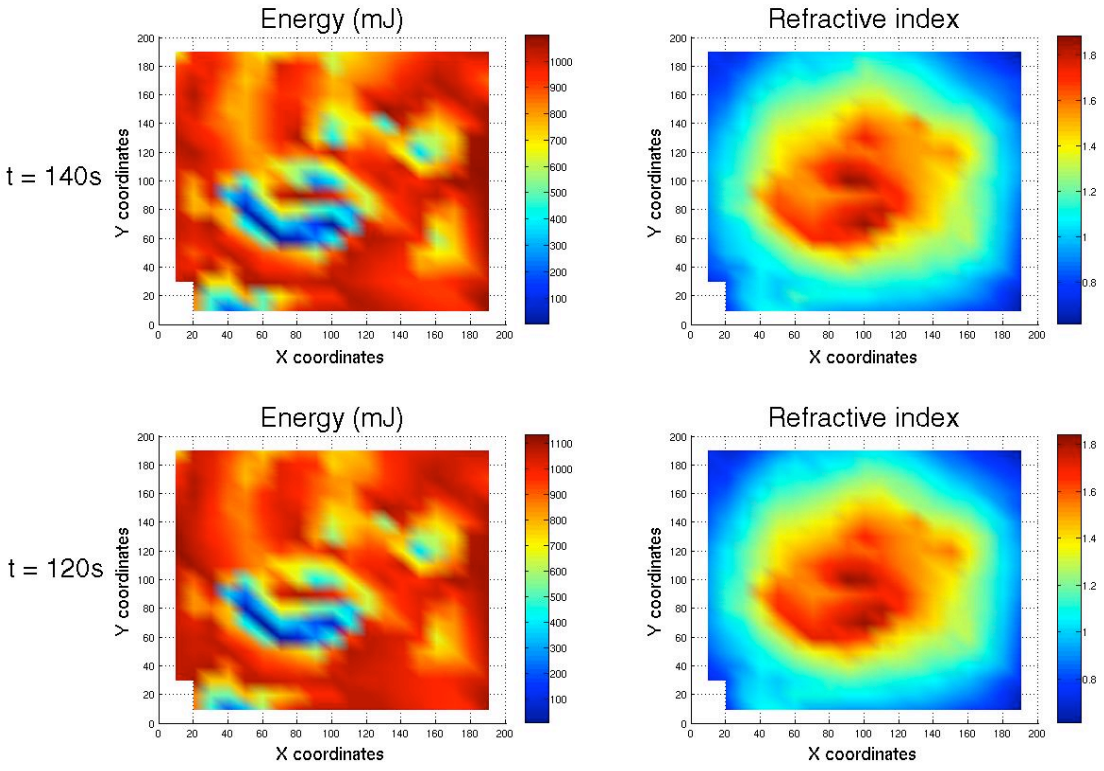


Figure 6.19c: Residual energy map, Energy Aware routing - 140 nodes

The energy and the refractive index distribution in the network are depicted below in Figure 6.20 (continues in the next page). Appendix C.5 contains all of them in full size.



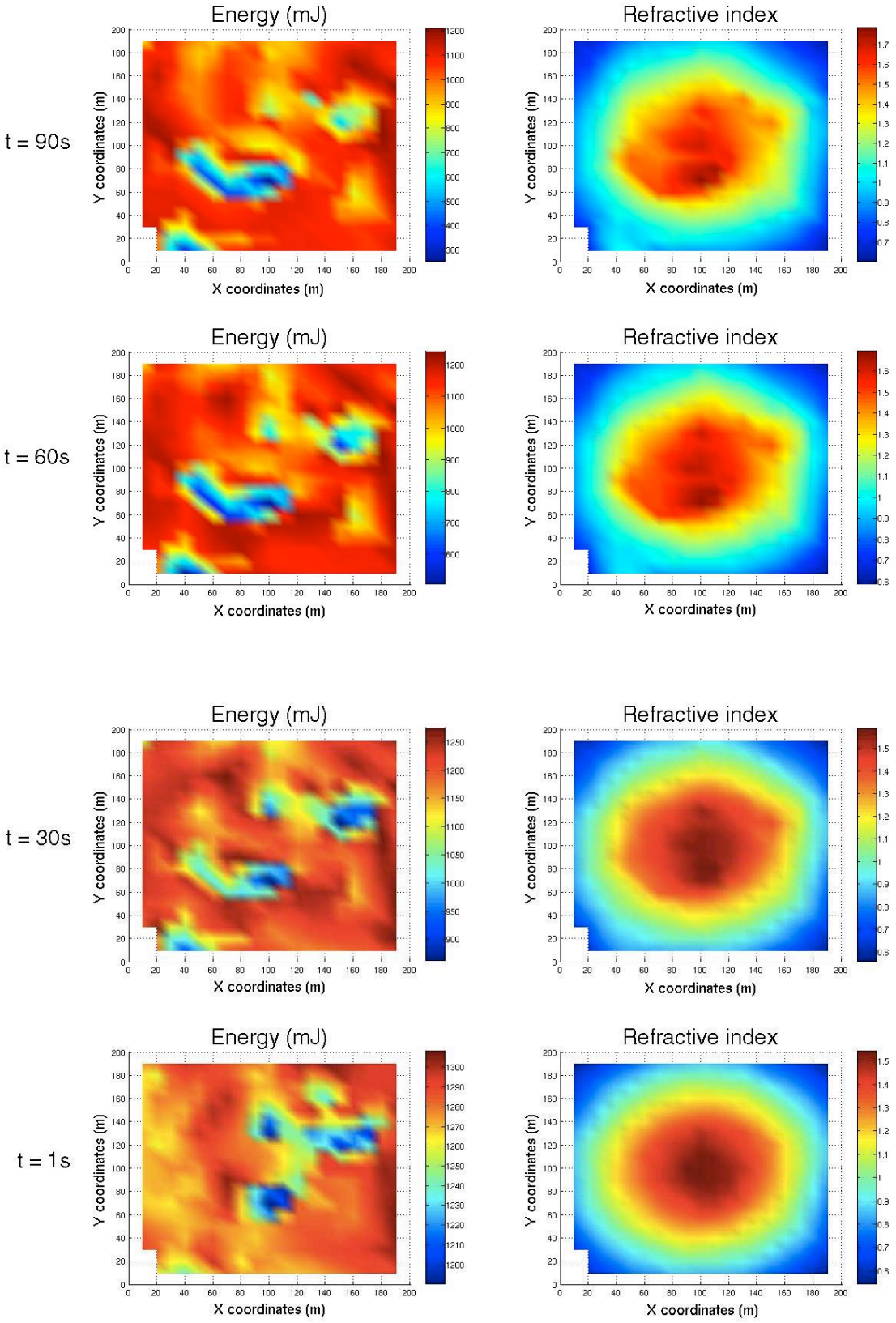


Figure 6.20: Snapshots of Energy and Refractive index distribution - 140 nodes

The refractive index distribution at the beginning of the simulation shows its spherical symmetry (bottom right of Figure 6.20) and the initial energy distribution on the left reveals that the load after the first hello broadcast is quite balanced. After 30 seconds of simulation, the residual energy decreases in the centre and the central left part of the topology, but not by much. The refractive index seems to expand a little to the upper left and also to separate its peak. The energy depletion has not changed much at the 60th nor at the 90th simulation second and the peaks of the refractive index are just a little more distinct. Towards the end of the simulation, at the 120th and the 140th second, energy depletion is observed at the top and the bottom right side of the topology. As can be seen, the refractive index acts accordingly to cover the low-energy areas. The fact that the energy distribution has not altered much throughout the simulation run depends mostly on the path order randomly chosen between each node and the destination.

6.1.6 Random s-d pairs

Finally, instead of having a unique sink node, source-destination (s-d) pairs at random location have been simulated to provide a different approach of perceiving how next-hop decisions are made. The difference is made clear in a 75-node scenario (Figure 6.21), where Greedy routing chooses the shortest path (in black), Curvy routing uses curved based on a fixed refractive index distribution (in green), while Energy Aware routing avoids low-energy areas by calculating the refractive index locally for each neighbourhood (in red).

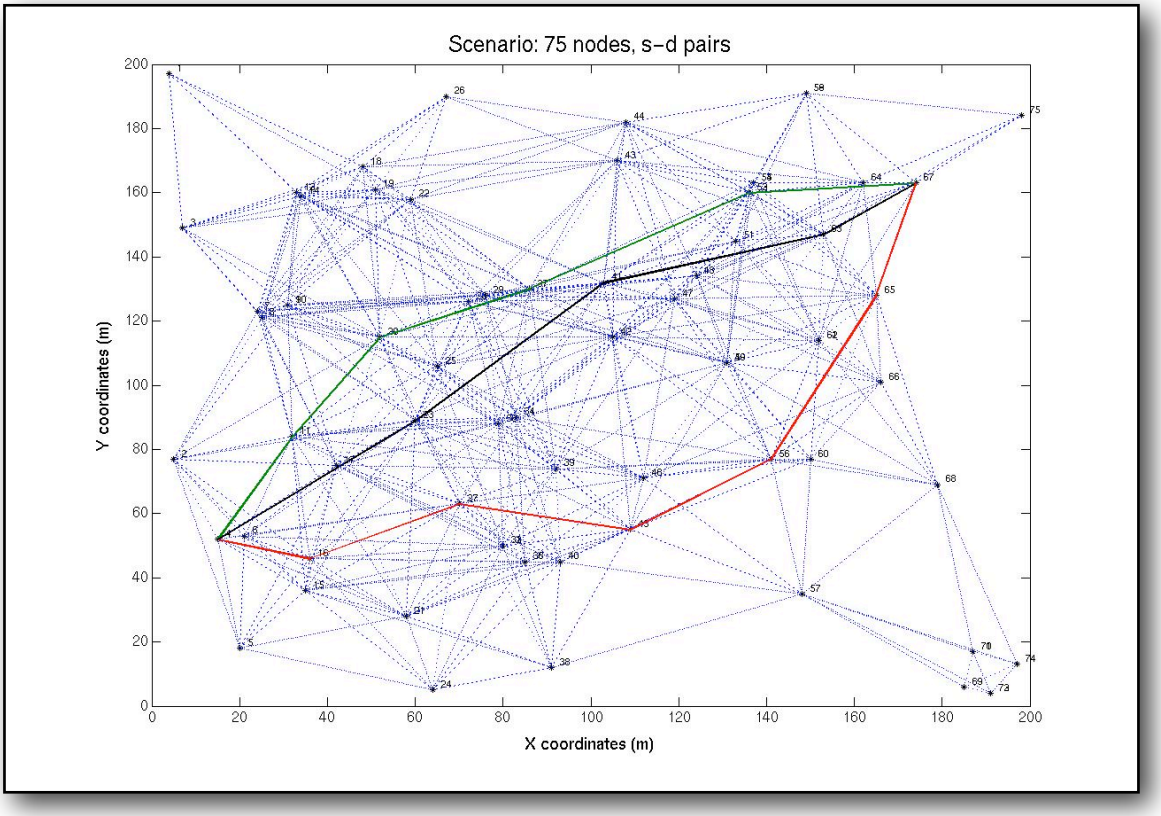


Figure 6.21: Random s-d pair path choice

As sparser networks are the centre of this investigation, 50-node scenarios have been chosen to be simulated, achieving a balance between density and sufficient path choices for the routing protocol. Other network sizes were considered and created, but the purpose of simulating random s-d pairs can not be fully exploited or highlighted. Denser topologies like the one in Figure 6.21 and above are not of particular interest in the context of this investigation, while the lower density ones do not provide enough task-forwarding choices to make a difference in path construction between Greedy, Curvy and Energy Aware routing. The simulation results of one of these scenarios are presented here, whose topology is depicted in Figure 6.22.

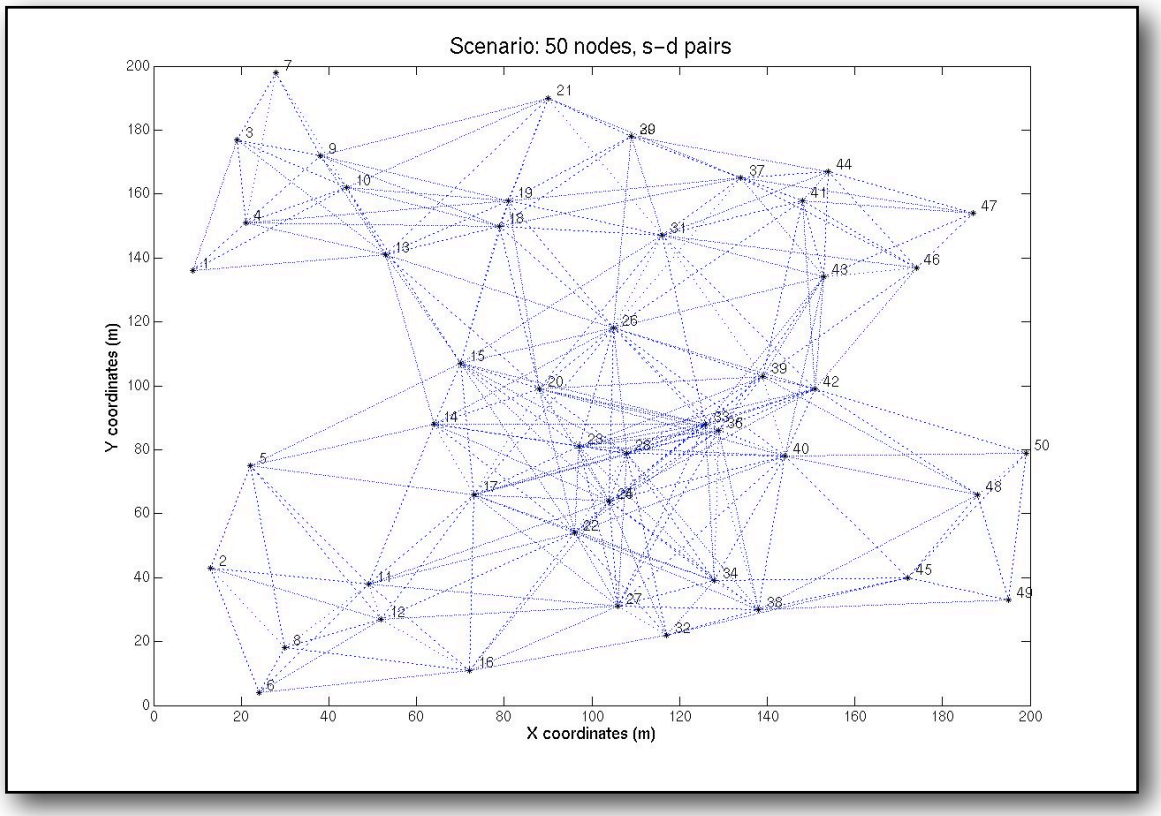


Figure 6.22: Topology with 50 nodes (s-d pairs)

As previously, the overall average energy per simulation second is shown for all versions of the algorithm in Figure 6.23a and the standard deviation of the average energy is demonstrated in Figure 6.23b. Curvy routing performs well, as it initially matches Energy Aware, but during the second half of the simulation its energy depletion rate further improves. This is due to the choice of shorter paths, as discussed in all previous scenarios, but also depends on the s-d pairs chosen, as well as the order in which they are selected. In the standard deviation graph (Figure 6.23b), Greedy routing can be seen to perform well until network partition occurs. Looking at Curvy and Energy Aware routing now, random s-d pairs make a difference in the way routing decisions are handled and "smart" next-hop calculations are recommended. Utilising Energy Aware routing, the standard deviation was managed in a better manner and was kept at lower levels. It has to be mentioned that in all random s-d pairs scenarios the standard deviation in general has evidently lower values compared to the scenarios with a single sink. A unique sink always depletes the nodes in its vicinity, because by necessity all paths end up

to neighbouring nodes before terminating at the sink. Here, the paths are randomly generated and both source and destination nodes are different each time, providing automatically a more even traffic flow across the network and making energy levels easier to manage.

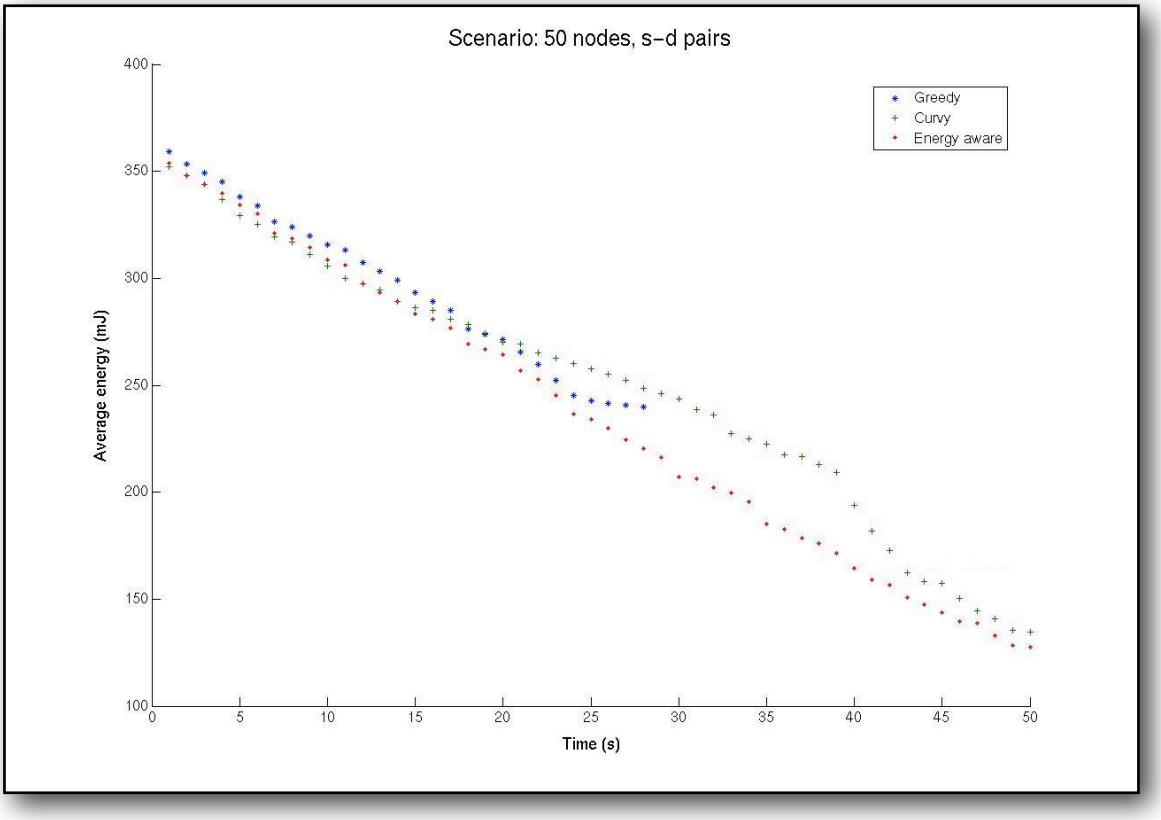


Figure 6.23a: Average energy vs Time - 50 nodes (s-d pairs)

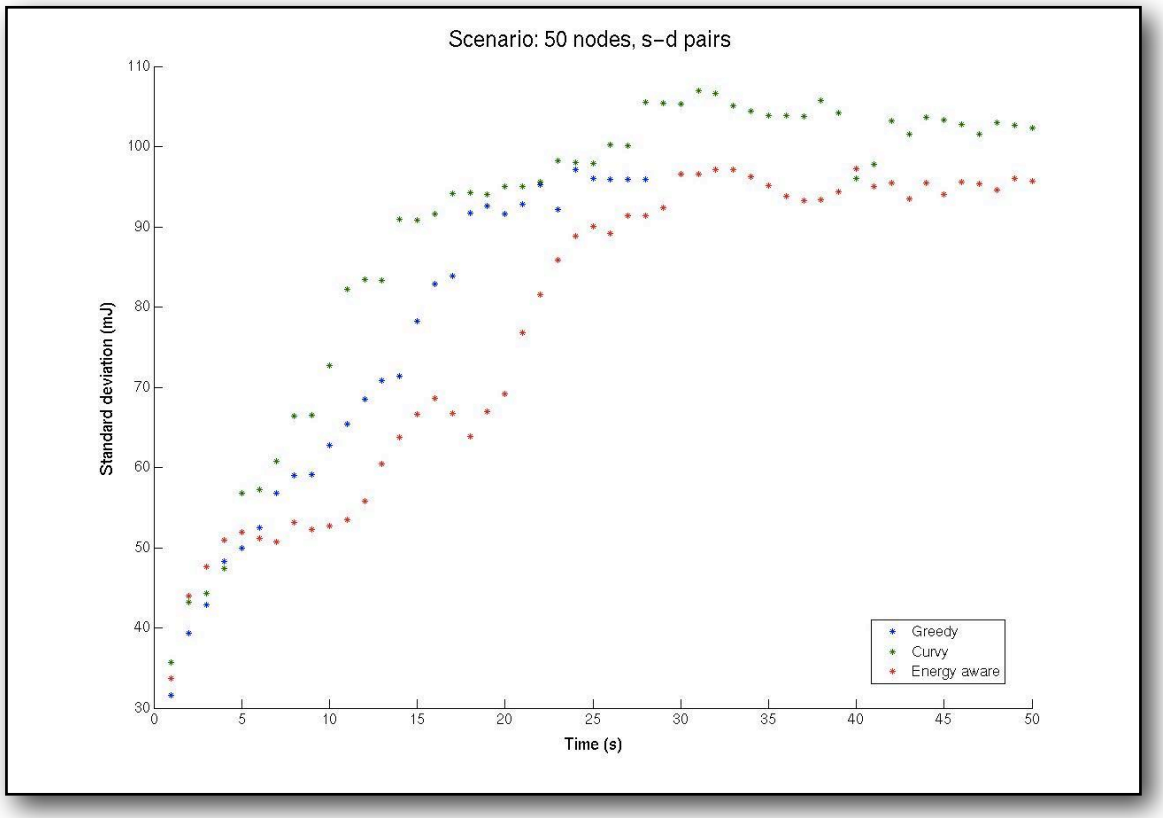


Figure 6.23b: Standard deviation vs Time - 50 nodes (s-d pairs)

Observing the overall network load during the simulation run, the residual energy map shows the crowded centre effect (introduced in Chapter 1) caused by Greedy routing (Figure 6.24a) and eventually leads to partition. Curvy routing exhausts a few nodes in the centre of the network, but spreads the load around a larger area closer to the periphery of the network centre, as shown in Figure 6.24b, which keeps the network alive for longer. As for Energy Aware routing, the energy map follows the definition of lifetime given in Chapter 4 and justifies the objective of this investigation; almost all nodes are used nearly equally in the selection of the paths, causing their energy to deplete gradually and the network to eventually deplete as a whole (Figure 6.24c).

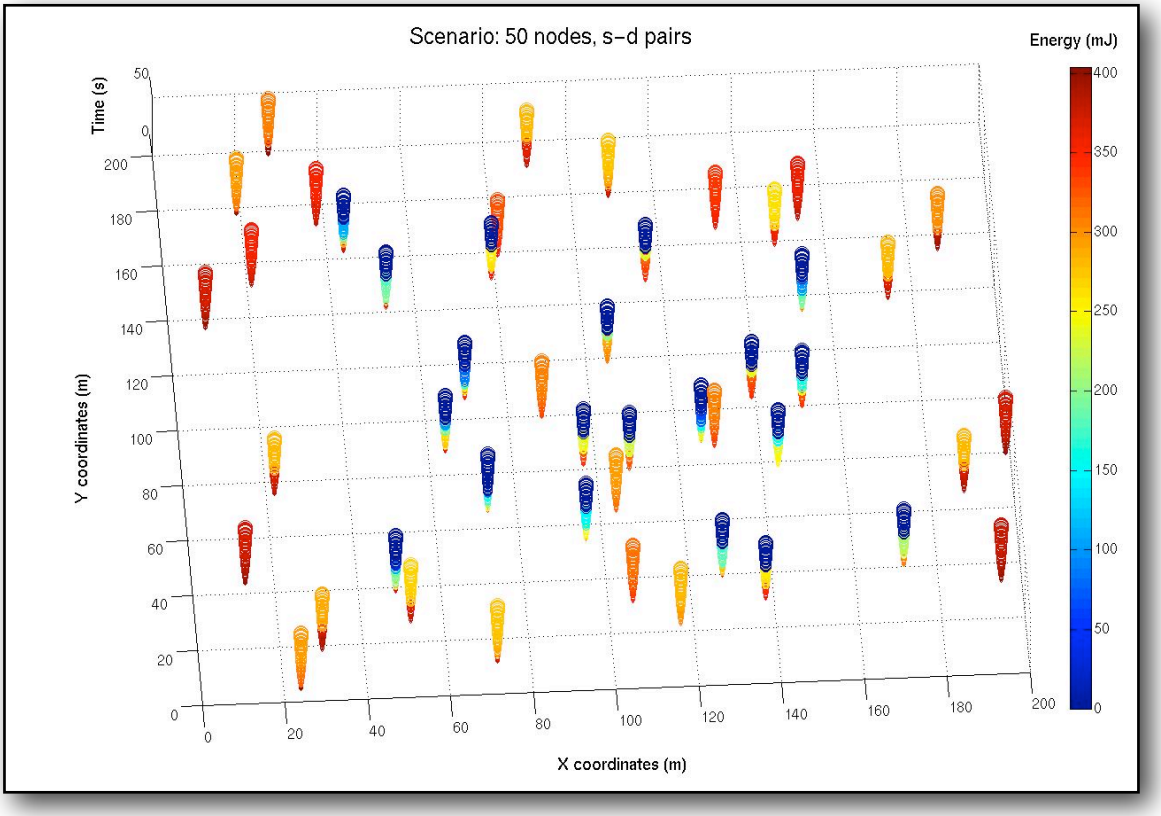


Figure 6.24a: Residual energy map, Greedy routing - 50 nodes (s-d pairs)

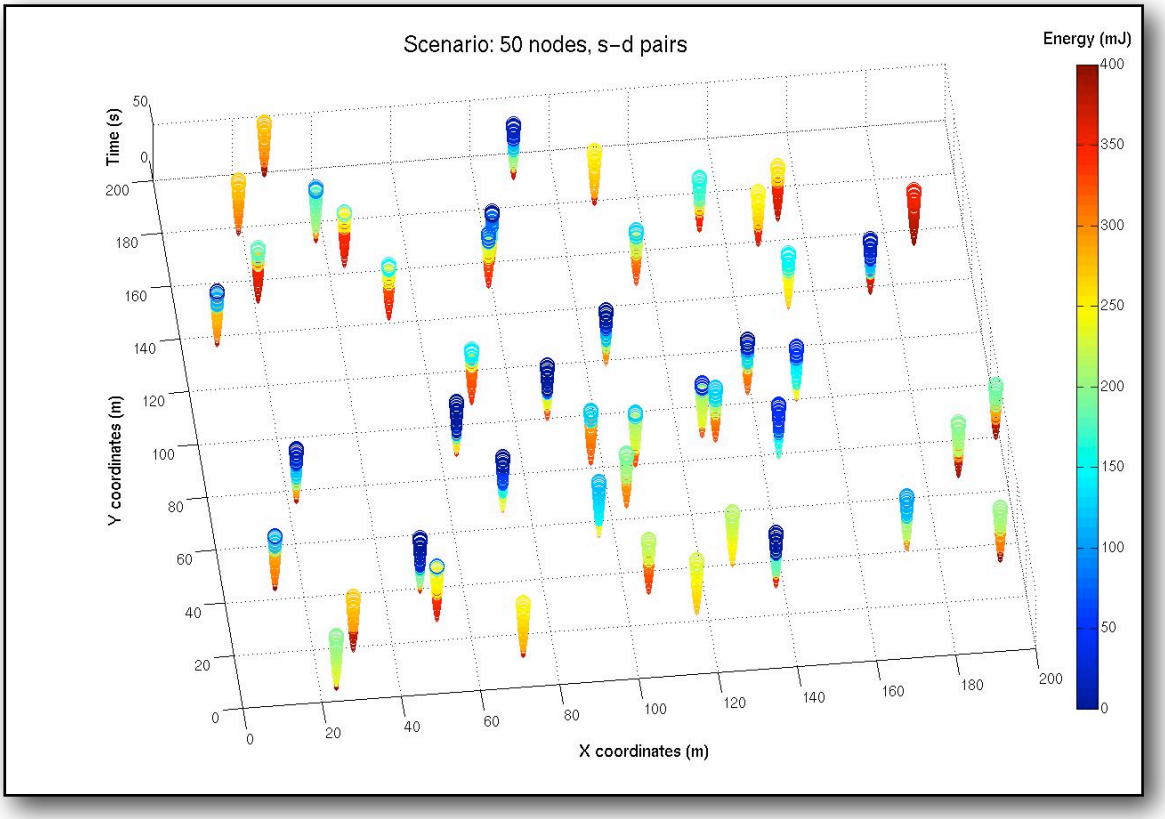


Figure 6.24b: Residual energy map, Curvy routing - 50 nodes (s-d pairs)

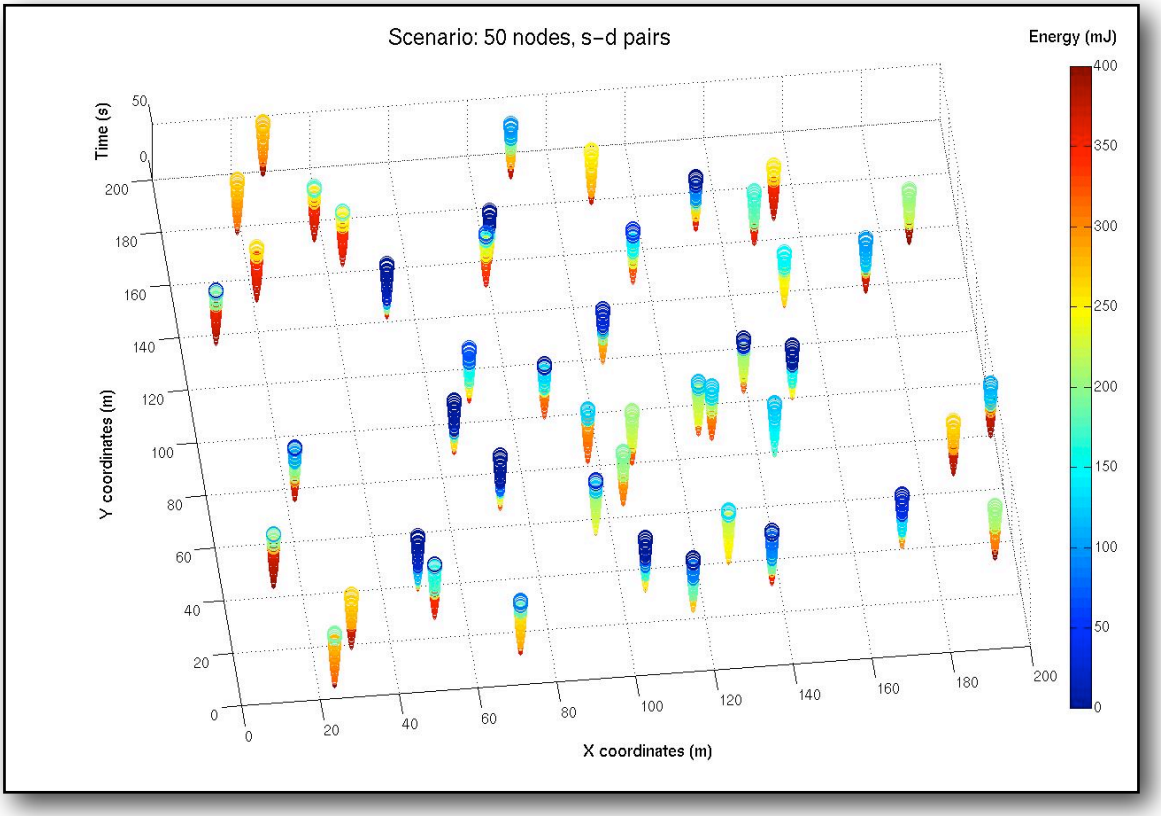
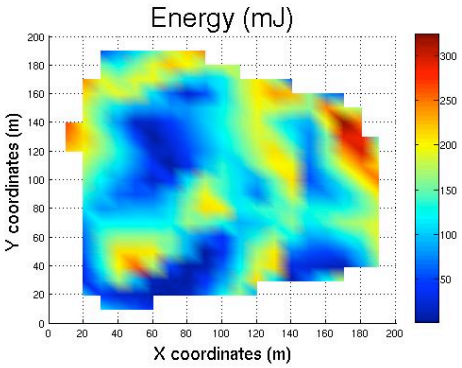
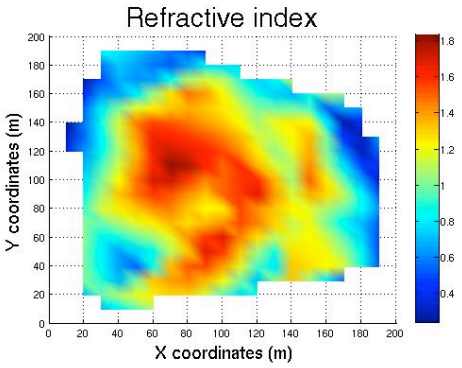


Figure 6.24c: Residual energy map, Energy Aware routing - 50 nodes (s-d pairs)

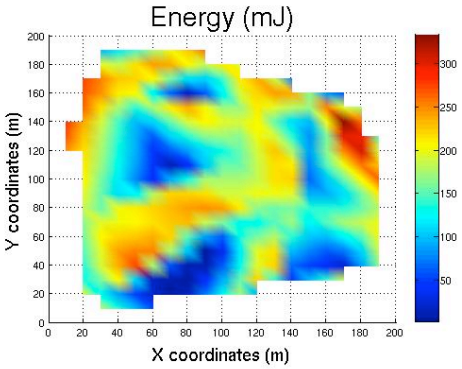
Figure 6.25 (continued in two pages) depicts screenshots of the energy and refractive index distributions at various simulation times, showing the changes in the network node energy distribution and the adaptation of the refractive index. The full size version of the figures presented below are given in Appendix C.6.

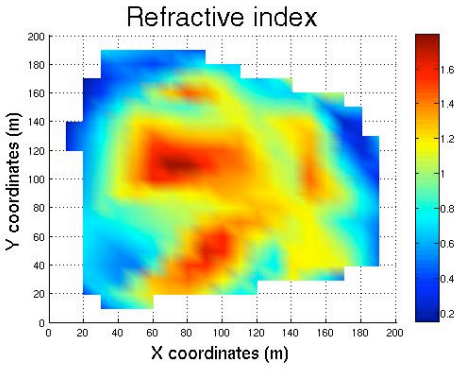
t = 50s



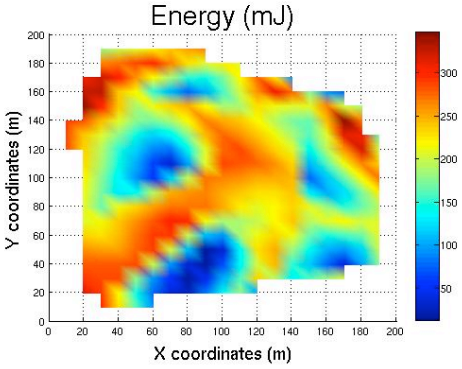


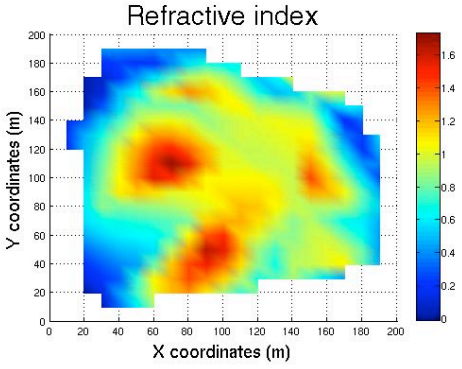
t = 40s



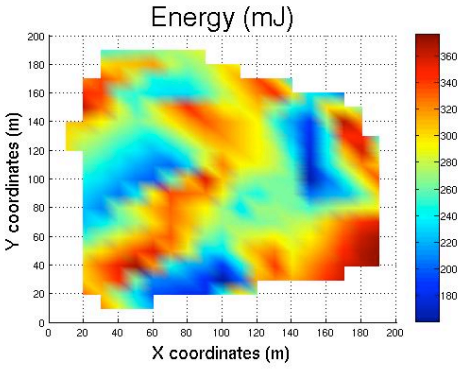


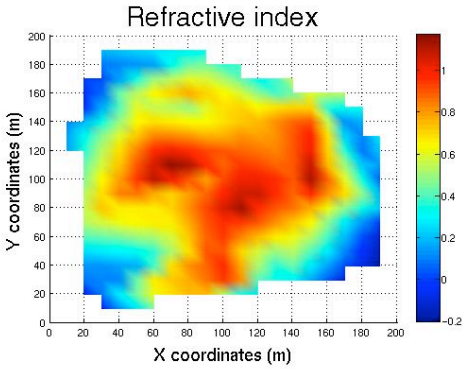
t = 30s





t = 15s





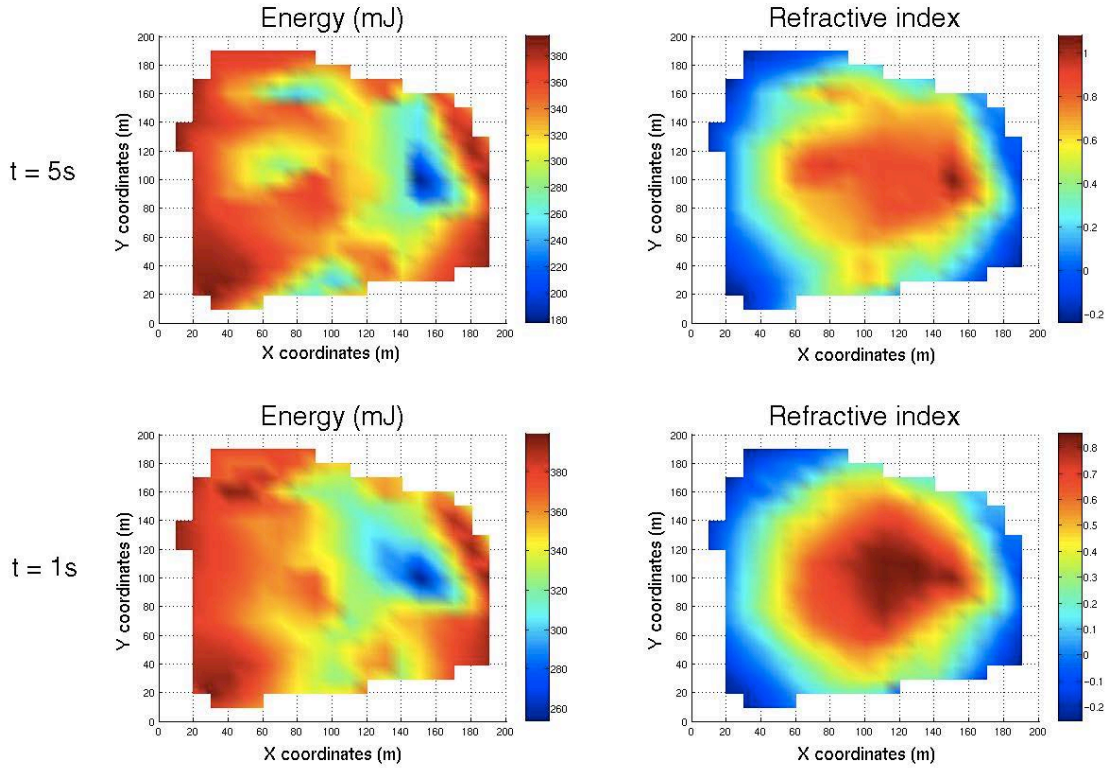


Figure 6.25: Snapshots of Energy and Refractive index distribution - 50 nodes (s-d pairs)

Starting from bottom to top in Figure 6.25, the energy levels are lower at the right-centre of the network at the beginning of the simulation and the peak of the refractive index is displaced to cover that area. After 5 simulation seconds, energy consumption has expanded to the left of the topology as well, causing individual low energy regions. It is very clear that the refractive index can rapidly compensate for those areas and reduce the frequency with which nodes in those areas are chosen by their neighbours. Later on, the residual energy decreases more, covering a larger area of the network and the refractive index manifests its local its peaks near the centre of those areas. By the 30th simulation second, there are distinct areas of low energy and the refractive index mirrors them closely with its local maxima. As time passes by and towards the end of the simulation, the energy levels are equalised with larger vulnerable areas. The refractive index is expanded to protect them and balance the load as designed to do so.

It is worth pointing out the accuracy of the refractive index distribution and its sensitivity to the rapid topological changes. The representation of both the energy level and the refractive index distribution in Figure 6.25 exceed the initial expectations and confirm the concept of using only local information to achieve a global view. The desired information was retrieved for each neighbourhood at times shown on the figure and, put together, made the overall illustration. The response of the refractive index to the energy depletion is visible in the rest of the scenarios presented, but in the s-d scenarios it is extremely responsive forming almost a "negative image" of the energy distribution.

6.2 Summary

The scope of the simulation scenarios presented and analysed was to build a network protocol that works on relatively sparse networks. Through the different densities of the topologies examined, satisfactory performance has been achieved in even very sparse networks of as little as 35 nodes in a 200m x 200m area. However, the best performance has been observed at the 40-node scenarios, where there are enough neighbours to overcome low-energy local areas; otherwise, Curvy routing yields better results due to shorter curved paths. This is verified by looking at the simulation trace files and specifically comparing the paths hop by hop between the exact same pair of source and destination nodes. Figure 6.26 below summarises the performance of Greedy, Curvy, Energy Aware and Curveball (Popa et al., 2007) routing depicting the figure of merit in s/mJ (lifetime over mean residual energy per node) versus the network density in mean number of neighbours per node.

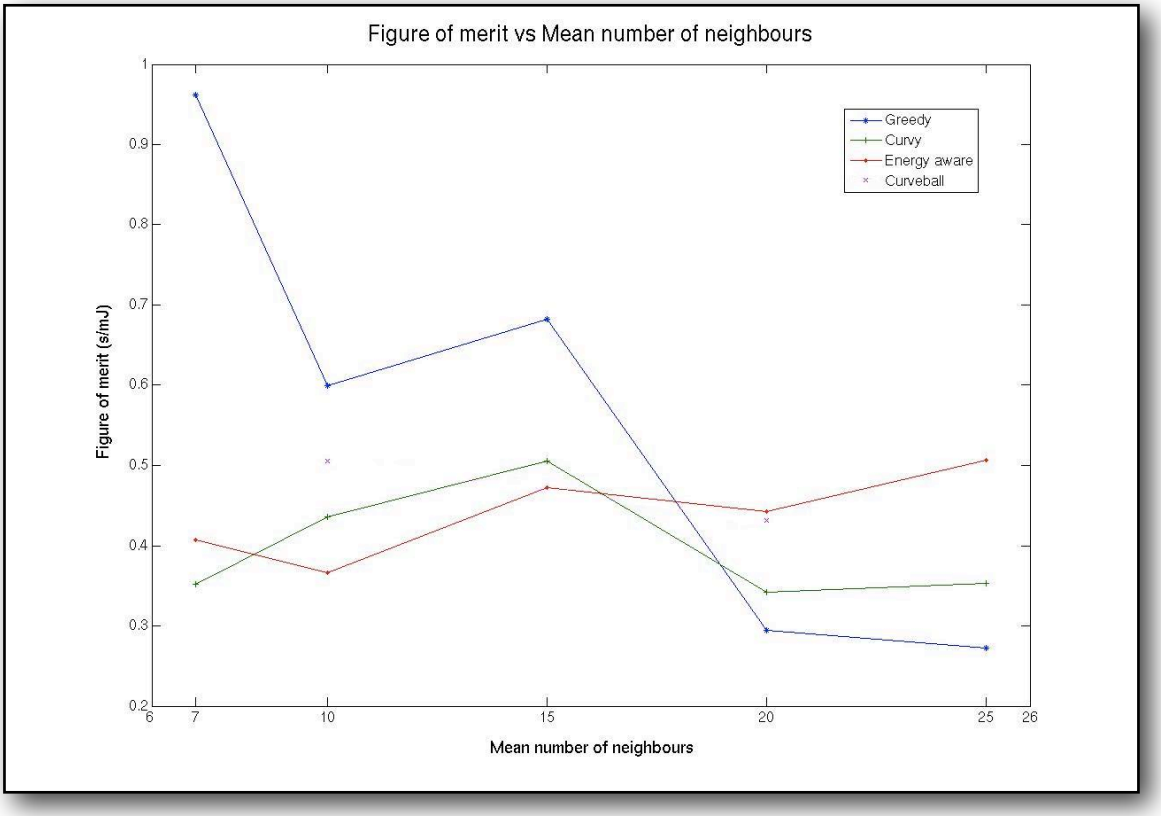


Figure 6.26: Figure of merit vs Mean number of neighbours

Nevertheless, regardless of the network density, though, all scenarios have shown that Energy Aware routing moves the network load away from the geographical centre, leading to further extension of the overall network lifetime.

Second and equally important, is the construction of a network protocol with plausible/practical sink placement. As discussed in Chapter 2, the location of the data sink is not paid particular attention in the literature, as this is not viewed as a significant issue. Cases of randomly chosen s-d pairs (or groups of s-d nodes) are presented with the aim to show the differences in optimal route selection. The results produced are more than just satisfactory and bring out the capabilities of Energy Aware routing. This happens due to the traffic load being spread out more evenly by default, since random pairs are used instead of a unique data sink placement. Although this may be a more suitable way to present and compare the proposed solutions, in reality it does not represent an actual wireless sensor network where there usually exists only one sink at an accessible location. Scenarios

comprising random s-d paths are included here (Sections 5.2.4 and 6.1.6) only to provide a comparison to the existing literature.

Further investigation on the presented network characteristics and more specifically, the transmission range allocated for each node has been carried out. Keeping the node density constant and enlarging the network area results into an increased number of nodes over 300. Unfortunately, due to machine memory limitations, such experiments could not be realised. Instead, the network area and the density was kept constant, decreasing the transmission range. The size of the network increased, but not as much, and the simulations ran showed that both proposed solutions perform well against Greedy routing. It is not a surprise that Energy Aware routing outperforms Curvy routing very easily in all scenarios, since simulation runs of this size exploit the proposed routing technique, leaving more space for the curved paths to propagate.

Finally, the issue of added overhead caused by the adaptation method has been monitored. The amount and the type of information exchanged between nodes is exactly the same as in Greedy routing and it does not influent the total overhead. Network size aside, it only comes down to additional computations towards the next hop selection. These have shown that they do not cause any extra delay at all and they can be perfectly be handled by Matlab. In the case of face routing (explained in Chapter 4) adopted in all versions of the algorithm including Greedy routing, minor delays in constructing the Gabriel graph have been noticed, but those have mostly to do with the graphical representation of it rather than the computations themselves. When network size is the point of view, minor delays have been experienced during the initial hello broadcast and neighbourhood establishment. Beyond that point, the path discovery takes as long as it does with smaller scenarios and the additional simulation time required is only due to the increased number of nodes and hence the number of paths initiated to the sink by default. The length of the paths as a tradeoff has been discussed in Chapter 3 and has an impact on the performance because of the increased energy consumption. However, it does not affect the total overhead by more than a few nanoseconds, since the length is

averagely extended by two or three nodes in comparison with the paths of Greedy routing. In excessively large scenarios of more than 1000 nodes, the path length is expected to be an issue and cause visible delays regardless of the difference between the methods.

CHAPTER 7

Conclusions and Future Work

An introduction to Wireless Sensor Networks (WSNs) was given in Chapter 1, outlining their features and raising open research questions in order to be utilised more efficiently. A review of existing literature on the subject has been presented in Chapter 2 with emphasis on energy management and load balancing issues in WSNs, underlining the shortcoming of existing research and the associated underlying assumptions. In Chapter 3 the theory and the methodology used towards the proposed solution were introduced, including the detailed formulae and calculations used in the proposed algorithm. The custom-built simulator was described in Chapter 4, evaluating the performance of the protocol and verifying the simulation results. In Chapter 5, the results of the Curvy routing algorithm were compared to Greedy routing and the Curveball routing (Popa et al., 2007), while in Chapter 6 a further comparison and results analysis were included involving Greedy routing, Curvy routing and Energy Aware routing, drawing conclusions on the conditions in which the algorithm achieves best performance. These conclusions as well as future work to complement the proposed solution are discussed in this chapter.

7.1 Research Contributions

First and foremost, a novel solution for one of the most vitally important research questions in WSNs was developed. A model inspired by nature and based on geometrical optics was expressed mathematically, modified to reflect the exact desired behaviour of the refractive index and tailored to fit the needs of energy

management across the network. It was implemented as programming code and embodied into the algorithm's sequence.

Two distinct versions of an algorithm were created and can be used to serve different purposes in WSN routing. Curvy routing just uses node location, gross network topography and node-residual energy information, and bends the paths away from the centre of the network based on the initial fixed refractive index distribution. Energy Aware routing, on the contrary, is more sophisticated and takes into account rapid changes of the mean residual energy in the one-hop neighbourhood of each node, updating constantly the refractive index and avoiding explicitly vulnerable, low-energy areas.

A complete network simulator was built in Matlab, capable of evaluating the routing protocol and exploiting the algorithm's advantages. It includes all the necessary features and properties required to perform simulations of that level and to provide extensive numerical analysis on the protocol's performance. Additionally, due to its functional structure and class organisation, it is easy to modify and adjust to demanding environments and exceptional requirements.

7.2 Simulation Results Observations

Concrete conclusions can be drawn by observing the large sample of simulation runs attempted under different circumstances each time. The behaviour and performance of the presented algorithms are analysed before alterations on the topology, network density, sink node location or path length. The detailed network configuration is also examined when network partition occurred.

Topology: Changes in the topology affect the performance of the algorithm to a great extent. A large number of different topologies were simulated per scenario in order to be in a position to extract accurate conclusions. Disk-shaped topologies favour all Greedy routing, Curvy routing and Energy Aware routing performances, but less circular ones do not produce dramatically worse results; only paths

including the edge nodes are affected. Greedy routing obviously performs well when there are sufficient choices in the centre of the topology, namely large concentration of nodes close to the centre to keep the network alive. Large void areas in the topology seem to not affect Curvy routing and Energy Aware routing, since they can be easily circumvented and data delivery is guaranteed. Uneven node concentration (especially on the edges) still does not seem to influence neither of the two proposed versions of the algorithm, in contrary with Greedy routing which struggles to discover alternative routes. In conclusion, the ideal topology comprises of a disc shape and sufficient next-hop selections around the periphery of the network.

Network Density: The network density is another important parameter which affects the performance of the algorithm. While keeping the network dimensions fixed at a 200m x 200m area, the density was controlled by the node population in the network and is measured in number of nodes per square metre or in average number of neighbours per node. Greedy routing performs well in abundant networks with densities higher than 1.875×10^{-03} nodes per square metre, namely around 20 - 25 neighbours per node in a total of 75 nodes. Lower densities favour Curvy routing and Energy Aware routing, which both perform as designed and even touch the limit of network connectivity at 0.95×10^{-3} nodes per square metre, which corresponds to an average number of 7 - 10 neighbours per node in networks as sparse as just 35 nodes in total.

Sink node location: The position of the sink node played a big role in the simulation results presented in previous chapters, as initially it was placed at the centre of the topology. Greedy outperformed Curvy routing and Energy Aware routing, as expected, since the location of the destination was convenient for shortest-path construction. Several different positions were tried afterwards to the edge of the network, as they represent a more realistic and practical structure. The large sample of simulation runs showed that the proposed algorithm performs better when the sink node is placed away from the geographical centre of the network and in positions where there are sufficient number of neighbours to provide easy access

via more than one routes. When random source-destination (s-d) pairs are deployed, the performance difference is more visible and shows that Greedy routing cannot cope at all with demanding environments. Curvy routing experiences some early battery depletion on some key-positioned nodes, but in general has no difficulty in completing the longest simulation runs. The intelligence of Energy Aware routing proves essential in the s-d pairs scenarios, maintaining a balanced overall residual energy and allowing more available options.

Transmission range: Scenarios of reduced transmission range have been simulated and the results have shown dramatic improvement in the proposed algorithm's performance. Initially, it was set to be fairly large covering one third of the network area (62 m) in order to keep the network size conveniently small. Although it produced satisfactory results, it did not fully bring out the advantage of performing clever local routing decisions. Examples of transmission range at 41 m and 31 m were presented in Chapter 6, covering one fifth and one sixth of the area respectively. The outcome justifies the predictions and shows that Energy Aware routing can handle any type of network configuration in a sophisticated manner and outperform Curvy routing's bulk distribution over the network. It keeps the average energy depletion under control and the standard deviation of the energy at lower levels, while the network load is shifted away from the centre of the topology.

Network size: The size of the network is inversely proportional to the transmission range, as it is defined by varying the radius of each node while maintaining the same density (0.95×10^{-3} nodes per square metre) in the same area (200m x 200m). Following the decrease of the transmission range, the network size increases as much as 100 nodes and 140 nodes respectively in the area. Simulating networks of larger scale has generated excellent results, and provided the ability to have a "higher resolution" view of the locality determining the routing paths. Greedy routing has not shown any improvement, since the centre of the network still suffers and network partition occurs depending on the topology. The effect of network size has greater impact on Energy Aware routing when compared to Curvy routing. Forwarding decisions are made faster and less biased due to the alternative

path availability, which helps to keep a balance between path length and energy consumption. In particular, the average node energy and the standard deviation of the energy dissipation are a lot smoother than for Curvy routing.

Path length: After experimenting with the shape and type of different topologies, results observations showed that the performance of the proposed algorithm in comparison with Greedy routing is directly dependent on the length of the paths selected during next-hop calculation. This is clearly visible on the graphs representing the standard deviation of the average energy per simulation second. The phenomenon is also visible when the sink node is placed in the centre and explains the reason why Greedy routing performs well in certain topologies. When it comes to comparing Curvy routing and Energy Aware routing with each other, path length is key and due to the latter's sophisticated way of avoiding low-energy areas, the standard deviation is seriously affected. Only the energy map over the network shows the capability of Energy Aware routing to shift the load away from central, low-energy locations and to keep those alive.

Network partition: In many cases network partition has been experienced, mostly by Greedy routing, and it has been perceived that the structure of the topology is the primary reason of it occurring at early simulation time. Void areas at the centre of the topology affect Greedy routing paths and, thus, lead to premature partition; that does not affect Curvy routing and Energy Aware routing at all. The position of the sink is an issue, as discussed, but for reasons of node concentration around it; if the number of sink neighbours is limited, then both versions of the proposed algorithm struggle with keeping the network alive for as long as they do otherwise. Path length also affects Greedy routing (in terms of partition), since it does not have any mechanisms for energy conservation.

The above conclusions form strong evidence for the proposed algorithm's efficiency in sparse networks with plausible and realistic structure. In contrary to the existing literature (Chapter 2), in the proposed solution, assumptions such as massively large networks with high density and impractical sink node placement do

not need to be made. Under the conditions mentioned and discussed above, both versions of the algorithm show optimal performance and prove that they can be used accordingly for efficient energy-aware and load-balancing routing. Table 7.1 summarises the set of the ideal network characteristics that need to be present in order to achieve maximum performance, maintaining a tradeoff between them.

Network characteristics		
Topology	**Position of sink**	**Density**
Predominantly convex	Away from the geographical centre of the network	Average number of 7-15 neighbours per node

Table 7.1: Network characteristics resulting best performance of the algorithm

7.3 Future Work

Experimenting more with the transmission range and the network size individually is one of the areas to expand, as it can exploit the features of the algorithm. Due to machine power limitations, this was not possible, as by keeping the density the same (enough to establish connectivity), the size of the simulation increased dramatically and could not be handled by the equipment's available memory. Should there be implemented, it is believed to show the routing capabilities of the proposed solution due to the available space offered for the curved paths to propagate. Large scale simulations will also provide a closer view to the behaviour of the algorithm and are expected to reveal how exceptional network cases are handled by the proposed routing technique.

Additionally, following the results obtained from networks where the sink node was placed in the centre (presented in both Chapters 5 and 6), a different approach providing a solution is suggested. The network can be split into several sub-

networks based on its topology, where the original central sink will now have a position at the edge with reference to each one of these sub-networks. Curvy routing and Energy Aware routing techniques can be executed on each sub-network separately, combining the final results and are expected to still outperform Greedy routing.

As mentioned in Chapter 4, due to the way Curvy and Energy Aware routing are designed to operate, face routing (Bose, Morin, Stojmenovic and Urrutia, 1999) was never needed to be invoked in the scenarios simulated. However, if properly modified, face routing can be utilised to prevent void areas to grow larger. A connected graph of points can be created as before, but the nodes with the maximum residual energy level will be selected to form it instead. When a point is reached where the length of the paths is similar to that of the unmodified face routing (because of these void areas), the above technique can be invoked and contribute to the network's survivability issue. This way, only the most "energetic" nodes will be included in the path route, since a new graph will be created each time face routing is called.

Finally, aiming to integrate the algorithm and improve the way routing is performed, information about traffic demand can be embodied to the task forwarding decision. A third term can be added to the equation (7.1) which will further modify the path deviations to account for the rate of energy depletion based on the recent history of the amount of data traffic forwarded by the nodes. Low-energy / high-traffic information can be combined in such way that the refractive index distribution is as precise as possible and adopts quickly to local changes.

$$n(\mathbf{r},t) = n_{bulk}(\mathbf{r}) + n_{battery}(\mathbf{r},t) + n_{traffic}(\mathbf{r},t) \tag{7.1}$$

APPENDIX A

Network Simulator Source Code

The source code the network simulator is constructed from is presented below. The functions are listed in the order they are called by the program; starting with Greedy routing (Appendix A.1), then Curvy routing (Appendix A.2) and finally Energy Aware routing (Appendix A.3).

A.1 Greedy Routing Source Code

A.1.1 Main.m

```
% Initial setup
num_of_nodes = 50;
area_of_plot = 200; % metres
trans_range = 62; % metres

% Uniformly random distribution of the nodes in the area
x_axis_points = round (area_of_plot *(rand(1,num_of_nodes)));
y_axis_points = round (area_of_plot *(rand(1,num_of_nodes)));

% Plot the graph
plot (x_axis_points, y_axis_points, 'bh');
xlabel('X-axis (distance)');
ylabel('Y-axis (distance)');
title('200 x 200 area of 50 nodes', 'FontSize',12);
figure;

% Position of the nodes in the area
sort_x_points = sort(x_axis_points);

for i = 1:length(sort_x_points)
    temp_var = sort_x_points(i);
    for j = 1:length(x_axis_points)
        if (temp_var == x_axis_points(j))
            sort_y_points(i) = y_axis_points(j);
        end
    end
end

coord_of_nodes = ones(length(sort_x_points),3);
coord_of_nodes(:,1) = 1:num_of_nodes;
coord_of_nodes(:,2) = sort_x_points';
coord_of_nodes(:,3) = sort_y_points';

% UDG
[Path, path_alg, cost, node_density, neighbour_matrix, vicinity_matrix, num_of_neighbours,
energy_matrix, zero_energy] = UDG(num_of_nodes, coord_of_nodes, trans_range);
```

A.1.2 Unit_disc_graph.m

```
function [Path, path_alg, cost, node_density, neighbour_matrix, vicinity_matrix,
num_of_neighbours, energy_matrix, zero_energy] = UDG(num_of_nodes, coord_of_nodes,
trans_range, R)

hold on;

% Distance from first node to last node
for i = 1:num_of_nodes
  x1_point = coord_of_nodes(i,2);
  y1_point = coord_of_nodes(i,3);
  density = 0;

  % Plot randomly chosen points
  plot (coord_of_nodes(i,2), coord_of_nodes(i,3), 'k*');
  text(coord_of_nodes(i,2) + 2, coord_of_nodes(i,3) + 2, num2str(i));
  xlabel('X-axis (distance)');
  ylabel('Y-axis (distance)');
  title('Dijkstra Algorithm', 'FontSize',14);

  for j = 1:num_of_nodes
    x2_point = coord_of_nodes(j,2);
    y2_point = coord_of_nodes(j,3);
    distance_bw_points(i,j) = sqrt((y2_point - y1_point)^2 + (x2_point - x1_point)^2);

      if ((distance_bw_points(i,j) <= trans_range))
       density = density + 1;
       vicinity_matrix(i,j)= 1;
       % Links between nodes
       line([x1_point x2_point], [y1_point y2_point], 'LineStyle', ':');
      else
       vicinity_matrix(i,j)= inf;
      end
  end
node_density(i) = density;

% Find the number of neighbours counted comparing distance_bw_points to trans_range (self node
included)
num_of_neighbours = ones(length(node_density),2);
num_of_neighbours(:,1) = 1:i;
num_of_neighbours(:,2) = node_density';
% Search the vicinity matrix and return the neighbours (ones - self node
%inluded) in row-column, then flip to display column-row
[row, col] = find(vicinity_matrix == 1);
neighbour_matrix = fliplr([row, col])';
end

% Shortest-path and corresponding cost calculation using Dijkstra
[path, cost] = dijkstra(1, num_of_nodes, vicinity_matrix);

% Draw shortest path
if ~isempty(path)
    for i = 1:(length(path)-1)

       switch i
            case 1
              col_scheme = 'k';
            case 2
              col_scheme = 'c';
            case 3
              col_scheme = 'r';
            case 4
              col_scheme = 'g';
            case 5
              col_scheme = 'm';
            case 6
              col_scheme = 'k';
            case 7
              col_scheme = 'r';
              case 8
              col_scheme = 'r';
            case 9
              col_scheme = 'g';
            case 10
              col_scheme = 'm';
```

```
            case 11
              col_scheme = 'k';
            case 12
              col_scheme = 'c';
            case 13
              col_scheme = 'r';
            case 14
              col_scheme = 'g';
            case 15
              col_scheme = 'm';
            otherwise
              col_scheme = 'k';
       end
        line([coord_of_nodes(path(i),2) coord_of_nodes(path(i+1),2)], [coord_of_nodes(path(i),
3) coord_of_nodes(path(i+1),3)], 'Color',col_scheme,'LineWidth', 2, 'LineStyle', '--');
    end
end
legend('Nodes','Paths b/w nodes',2);
box on;

hold off;


% Hello broadcast energy info
[hello_energy_matrix] = hello_energy(num_of_nodes, neighbour_matrix);
energy_matrix = zeros(num_of_nodes,num_of_nodes+1);
zero_energy = 0;

% create Path cell array to hold each path to the sink
Path = cell(num_of_nodes,1);
for ord = 1:num_of_nodes
    m = R(ord);
    if ord == m
        Path{ord} = m;
    else
figure;
% greedy routing
[path_alg, next_hop, k, side_sink] = greedy_alg(coord_of_nodes, num_of_nodes, vicinity_matrix,
trans_range, m);
Path(ord,1) = {path_alg};

% if a node's coordinates are the same as sink's
if (path_alg(numel(path_alg)) ~= side_sink) && (coord_of_nodes(path_alg(numel(path_alg)),2) ==
coord_of_nodes(side_sink,2)) && (coord_of_nodes(path_alg(numel(path_alg)),3) ==
coord_of_nodes(side_sink,3))
  Path{ord} = [Path{ord}, side_sink];
elseif next_hop == 0
  Path{ord} = [Path{ord}];
% include all different cases of route discovery
elseif (path_alg(numel(path_alg)) ~= side_sink) && ((next_hop ~= k) || (next_hop ~=
side_sink))
    Path{ord} = [Path{ord}, Path{next_hop}];
    S = intersect(Path{next_hop}, zero_energy);
  % greedy routing for second time if the path does not exist or its nodes
  % are dead
  if (isempty(Path{next_hop}) || ~isempty(S))
      %find Path{next_hop} if not empty and remove, to add the new path_2
      if (~isempty(S))
        y = numel(Path{next_hop});
        z = find(Path{ord}, y, 'last');
        Path{ord}(z) = [];
      end
      n = next_hop;
      [path_2, next_2] = greedy_2(coord_of_nodes, num_of_nodes, vicinity_matrix, trans_range,
n);
      if next_2 == 0
      Path{ord} = [Path{ord}];
      end
      Path{ord} = [Path{ord}, path_2];
      % if sink is not reached after greedy again
      if (path_2(numel(path_2)) ~= side_sink)
          P = path_2(numel(path_2));
          % perform face routing
```

```
          [next_node] = face_routing(coord_of_nodes, num_of_nodes, neighbour_matrix,
side_sink, P);
          Path{ord} = [Path{ord}, next_node];
           while (next_node ~= side_sink)
           P = next_node;
           [next_node] = face_routing(coord_of_nodes, num_of_nodes, neighbour_matrix,
side_sink, P);
           Path{ord} = [Path{ord}, next_node];
           if (next_node == side_sink) | (isempty(next_node))
               break
           end
           end % while
           if (isempty(next_node))
               Q = P;
               % external face routing
               [next_face_2] = face_2(coord_of_nodes, num_of_nodes, neighbour_matrix,
side_sink, Q);
               Path{ord} = [Path{ord}, next_face_2];
               while (next_face_2 ~= side_sink)
                    Q = next_face_2;
                    [next_face_2] = face_2(coord_of_nodes, num_of_nodes, neighbour_matrix,
side_sink, Q);
                    Path{ord} = [Path{ord}, next_face_2];
                    if (next_face_2 == side_sink)
                        break
                    elseif (isempty(next_face_2))
                        disp('network partition');
                        break
                    end
               end % while_2
           end % if next_node empty
      end % if numel(path_2)
  end % if next_hop empty
end % if numel(path_alg)

%celldisp(Path);
%close figure 1;

% energy_matrix
energy_matrix(:,1) = hello_energy_matrix(:,end);
energy_matrix(Path{ord}(1:end-1),ord) = energy_matrix(Path{ord}(1:end-1),ord) - 75; % energy
to send -75 mJ
energy_matrix(Path{ord}(2:end),ord) = energy_matrix(Path{ord}(2:end),ord) - 5; % energy to
receive -5 mJ
energy_matrix(Path{ord},ord) = energy_matrix(Path{ord},ord) + 1; % no idle for Path nodes
energy_matrix(:,ord+1) = energy_matrix(:,ord) - 1; % idle energy depletion -1 mJ

% list of dead nodes
zero_energy = find(energy_matrix(:,ord) <= 0);
% case if a node is dead
if ismember(m, zero_energy)
    Path{ord} = -m;
end

%disp(Path{ord});
celldisp(Path);

% update vicinity and neighbour matrices
for a = 1:numel(zero_energy);
vicinity_matrix(zero_energy(a),:) = Inf;
vicinity_matrix(:,zero_energy(a)) = Inf;
end
[row, col] = find(vicinity_matrix == 1);
neighbour_matrix = fliplr([row, col])';

% if sink is dead stop the simulation
if ismember(side_sink, zero_energy)
    disp('sink is dead');
    break
end
    end % if whole path
end % ord ends

celldisp(Path);
```

```
disp(energy_matrix);
disp(zero_energy);

%Energy graph
energy_matrix(:,1) = [];
energy_matrix(energy_matrix < 0) = 0;
X = coord_of_nodes(:,2);
Y = coord_of_nodes(:,3);
for h = 1:num_of_nodes
    Z = repmat(h,num_of_nodes,1);
    scatter3 (X, Y, Z, 4*Z, energy_matrix(:,h));
    hold on;
end
```

A.1.3 Dijkstra.m

```
% The Dijkstra's algorithm, Implemented by Yi Wang, 2005
% USAGE: [path, cost]= dijkstra(pathStart, pathEnd,  transMatrix)
% PARAMETERS:
%   pathS : the index of start node, indexing from 1
%   pathE : the index of end node, indexing from 1
% transmat: the transition matrix, or adjacent matrix

function [r_path, r_cost] = dijkstra(pathS, pathE, transmat)

if (size(transmat,1) ~= size(transmat,2))
    error('detect_cycles:Dijkstra_SC', ...
     'transmat has different width and heights');
end

% Initialization:
%  noOfNode     : nodes in the graph
%  parent(i)    : record the parent of node i
%  distance(i)  : the shortest distance from i to pathS
%  queue        : for width-first traveling of the graph
noOfNode = size(transmat, 1);

for i = 1:noOfNode
  parent(i) = 0;    %all parents node [0 0 0 0 0 0 0 0 0 0]
  distance(i) = Inf; %all distances[inf ..inf]
end

queue = [];

% Start from pathS
for i=1:noOfNode % [1 - 10]
  if transmat(pathS, i)~=Inf
    distance(i) = transmat(pathS, i); %[1 1 1 inf inf inf inf inf inf inf ]
    parent(i)   = pathS;              %[1 1 1 0 0 0 0 0 0 0 ]
    queue       = [queue i];          %[1 2 3]
  end
end

% Width-first exploring the whole graph
while length(queue) ~= 0          %3
  hopS  = queue(1);               %7
  queue = queue(2:end);           %[]

  for hopE = 1:noOfNode           %[1 - 10] [1 1 1 2 2 3 4 5 5 5 ]
    if distance(hopE) > distance(hopS) + transmat(hopS,hopE) %2 > 5 + 1
      distance(hopE) = distance(hopS) + transmat(hopS,hopE); %[1 1 1 2 2 3 4 5 5 5]
      parent(hopE)   = hopS;     % [1 1 1 2 3 5 6 7 7 7]
      queue          = [queue hopE]; %[8 ]
    end
  end

end

% Back-trace the shortest-path
r_path = [pathE];
i = parent(pathE);
```

```
while i~=pathS && i~=0
  r_path = [i r_path];
  i = parent(i);
end

if i == pathS
  r_path = [i r_path];
else
  r_path = [];
end

% Return cost
r_cost = distance(pathE);
```

A.1.4 Hello_energy.m

```
function [hello_energy_matrix] = hello_energy(num_of_nodes, neighbour_matrix)

nm = flipud(rot90(neighbour_matrix));
initial_energy = 500;
t = 1;

hello_energy_matrix = zeros(num_of_nodes,num_of_nodes+1) + initial_energy;

for n = 1:num_of_nodes
t = t + 1;
ind = nm((nm(:,1) == n) & (nm(:,2) ~= n),2);
hello_energy_matrix(:,t) = hello_energy_matrix(:,t-1);
hello_energy_matrix(ind,t) = hello_energy_matrix(ind,t-1) - 5;
hello_energy_matrix(n,t) = hello_energy_matrix(n,t-1) - 75;
end
```

A.1.5 Greedy_algorithm.m

```
function [path_alg, next_hop, k, side_sink] = greedy_alg(coord_of_nodes, num_of_nodes,
vicinity_matrix, trans_range, m)

hold on;

side = zeros(1,3);
side(1,2) = 150;
side(1,3) = 150;
side_sink = dsearchn(coord_of_nodes, side);

for i=1:num_of_nodes
  x1_point = coord_of_nodes(i,2);
  y1_point = coord_of_nodes(i,3);

  % Plot randomly chosen points
  plot (coord_of_nodes(i,2), coord_of_nodes(i,3), 'k*');
  text(coord_of_nodes(i,2) + 2, coord_of_nodes(i,3) + 2, num2str(i));
  xlabel('X-axis (distance)');
  ylabel('Y-axis (distance)');
  title('Localised Greedy Algorithm', 'FontSize',12);

  for j=1:num_of_nodes
    x2_point = coord_of_nodes(j,2);
    y2_point = coord_of_nodes(j,3);
    graph_dist = sqrt((y2_point - y1_point)^2 + (x2_point - x1_point)^2);

    if (graph_dist <= trans_range)
      line([x1_point x2_point], [y1_point y2_point], 'LineStyle', ':');
    end

    if (vicinity_matrix(i,j) == 1)
      full_distance = sqrt((coord_of_nodes(i,2)- coord_of_nodes(side_sink,2))^2 +
(coord_of_nodes(i,3)- coord_of_nodes(side_sink,3))^2);
      part_distance = sqrt((coord_of_nodes(j,2)- coord_of_nodes(side_sink,2))^2 +
(coord_of_nodes(j,3)- coord_of_nodes(side_sink,3))^2);
```

```
      progress = full_distance - part_distance;

      if (progress >= 0)
      prr_avg(i,j) = progress;
      end
    end
  end % j ends
end % i ends


if (m <= side_sink)
for k = m:side_sink
  if (k == m)
    [max_value, next_hop] = max(prr_avg(k,:));
    counter = 1;
    path_alg(counter) = k;
  elseif (k == next_hop)
    counter = counter + 1;
    path_alg(counter) = k;
    [max_value, next_hop] = max(prr_avg(k,:));
  end
end % k ends
elseif (m > side_sink)
for k = m:-1:side_sink
  if (k == m)
    [max_value, next_hop] = max(prr_avg(k,:));
    counter = 1;
    path_alg(counter) = k;
  elseif (k == next_hop)
    counter = counter + 1;
    path_alg(counter) = k;
    [max_value, next_hop] = max(prr_avg(k,:));
  end
end % k ends
% if progress is 0, next_hop would still be 1 and it is set to 0 when there are no neighbours
forward
if ((next_hop == 1) && (vicinity_matrix(next_hop,k) == Inf) && (vicinity_matrix(k,next_hop) ==
Inf))
    next_hop = 0;
end
end % m ends


if ~isempty(path_alg)
    for i = 1:(length(path_alg) - 1)
       switch i
            case 1
              col_scheme = 'k';
            case 2
              col_scheme = 'c';
            case 3
              col_scheme = 'r';
            case 4
              col_scheme = 'g';
            case 5
              col_scheme = 'm';
            case 6
              col_scheme = 'k';
            case 7
              col_scheme = 'c';
            case 8
              col_scheme = 'r';
            case 9
              col_scheme = 'g';
            case 10
              col_scheme = 'm';
            case 11
              col_scheme = 'k';
            case 12
              col_scheme = 'c';
            case 13
              col_scheme = 'r';
            case 14
              col_scheme = 'g';
```

```
            case 15
              col_scheme = 'm';
            otherwise
              col_scheme = 'k';
       end
    line([coord_of_nodes(path_alg(i),2) coord_of_nodes(path_alg(i+1),2)],
[coord_of_nodes(path_alg(i),3) coord_of_nodes(path_alg(i+1),3)],
'Color',col_scheme,'LineWidth', 2, 'LineStyle', '--');
    end
end
legend('Nodes','Paths b/w nodes',2);
box on;
hold off;
```

A.1.6 Face_routing.m

```
function [next_node] = face_routing(coord_of_nodes, num_of_nodes, neighbour_matrix, side_sink,
P)

% Gabriel graph
GG = flipud(rot90(neighbour_matrix));
for u = 1:num_of_nodes
 Nu = GG((GG(:,1)==u) & (GG(:,2)~=u),2); % neighbours of u
    for v = Nu'
        for w = Nu'
 wx = coord_of_nodes(w,2);
 wy = coord_of_nodes(w,3);
 vx = coord_of_nodes(v,2);
 vy = coord_of_nodes(v,3);
 mx = (vx + coord_of_nodes(u,2))/2; % midpoint
 my = (vy + coord_of_nodes(u,3))/2;
 dmw = sqrt((mx - wx)^2 + (my - wy)^2);
 dmu = sqrt((coord_of_nodes(u,2) - mx)^2 + (coord_of_nodes(u,3) - my)^2);
         if ((w ~= v) && (dmw < dmu))
             uv = (GG(:,1) == u) & (GG(:,2)==v);
             GG(uv,:) = []; % delete uv
         end % end if w~=v
         end %end for w
     end % end for v
end % end for u
%figure;
%line(reshape(coord_of_nodes(GG',2),2,size(GG,1)), reshape(coord_of_nodes(GG',3),2,size(GG,
1)), 'LineStyle', ':');

% perimeter routing
hold all
line([coord_of_nodes(P,2)  coord_of_nodes(side_sink,2)], [coord_of_nodes(P,3)
coord_of_nodes(side_sink,3)],'Color','y','LineWidth', 2, 'LineStyle', '--');
hold off
intlines = 0;
for g = 1:num_of_nodes
    G_line_2 = GG((GG(:,1)==g) & (GG(:,2)~=g),2);

  % coordinates of drawn line(s) and Gabriel graph
    for c = 1:numel(G_line_2)
  X1 = coord_of_nodes(g,2);
  Y1 = coord_of_nodes(g,3);
  X2 = coord_of_nodes(G_line_2(c),2);
  Y2 = coord_of_nodes(G_line_2(c),3);
  X3 = coord_of_nodes(P,2);
  Y3 = coord_of_nodes(P,3);
  X4 = coord_of_nodes(side_sink,2);
  Y4 = coord_of_nodes(side_sink,3);

  a = (X4-X3)*(Y1-Y3) - (Y4-Y3)*(X1-X3);
  b = (X2-X1)*(Y1-Y3) - (Y2-Y1)*(X1-X3);
  A = (Y4-Y3)*(X2-X1) - (X4-X3)*(Y2-Y1);

  % calculation of cross product vectors
  Ua = a/A;
  Ub = b/A;
```

```
      if ((0<Ua) && (Ua<1) && (0<Ub) && (Ub<1))
          intlines = intlines + 1;

  % list of intersecting GG lines
  intersection_matrix = zeros(length(intlines),4);
  intersection_matrix(:,1) = X1;
  intersection_matrix(:,2) = Y1;
  intersection_matrix(:,3) = X2;
  intersection_matrix(:,4) = Y2;

  coord_of_GG = zeros(length(GG),4);
  coord_of_GG(:,1) = coord_of_nodes((GG(:,1)),2);
  coord_of_GG(:,2) = coord_of_nodes((GG(:,1)),3);
  coord_of_GG(:,3) = coord_of_nodes((GG(:,2)),2);
  coord_of_GG(:,4) = coord_of_nodes((GG(:,2)),3);

  line_1 = zeros(1,4);
  line_1(1,1) = intersection_matrix(1,1);
  line_1(1,2) = intersection_matrix(1,2);
  line_1(1,3) = coord_of_nodes(P,2);
  line_1(1,4) = coord_of_nodes(P,3);

  line_2 = zeros(1,4);
  line_2(1,1) = intersection_matrix(1,3);
  line_2(1,2) = intersection_matrix(1,4);
  line_2(1,3) = coord_of_nodes(P,2);
  line_2(1,4) = coord_of_nodes(P,3);

  % check if line exists in GG before choosing to transmit
  l1 = ismember(line_1, coord_of_GG, 'rows');
  l2 = ismember(line_2, coord_of_GG, 'rows');

  % different cases of lines coming out from one node (defining face to traverse and left hand
rule)
          if ((l1 == 1) && (l2 == 0))
              next_node = find((coord_of_nodes(:,2)==line_1(1,1)) & (coord_of_nodes(:,
3)==line_1(1,2)));
          elseif ((l1 == 0) && (l2 == 1))
              next_node = find((coord_of_nodes(:,2)==line_2(1,1)) & (coord_of_nodes(:,
3)==line_2(1,2)));
          elseif ((( l1 == 0) && (l2 == 0)) || ((l1 == 1) && (l2 == 1)))
              P_edges = GG((GG(:,1)==P) & (GG(:,2)~=P),2);
              for r = 1:numel(P_edges)
                % cross product check to discard the right and keep the
                % left edges
                x1 = coord_of_nodes(side_sink,2);
                y1 = coord_of_nodes(side_sink,3);
                x2 = coord_of_nodes(P,2);
                y2 = coord_of_nodes(P,3);
                x3 = coord_of_nodes((P_edges(r)),2);
                y3 = coord_of_nodes((P_edges(r)),3);
                ax = x1 - x2;
                ay = y1 - y2;
                bx = x3 - x2;
                by = y3 - y2;
                check(r) = (ax * by) - (ay * bx);
              end % for r
              % choose the left ones (if more than one) and calculate angle
              % to determine the most left
              M = find(check>0);
              N = find(check == 0); % if lines coincide, sink is a neighbour
              if (numel(N) > 0) && (ismember(side_sink, P_edges))
                  next_node = side_sink;
              elseif numel(M) > 1
                  for s = 1:numel(M)
                      x4 = coord_of_nodes(side_sink,2);
                      y4 = coord_of_nodes(side_sink,3);
                      x5 = coord_of_nodes(P,2);
                      y5 = coord_of_nodes(P,3);
                      x6 = coord_of_nodes((P_edges(M(s))),2);
                      y6 = coord_of_nodes((P_edges(M(s))),3);
                      cx = x4 - x5;
                      cy = y4 - y5;
                      dx = x6 - x5;
```

```
                dy = y6 - y5;
                Cross(s) = (cx * dy) - (cy * dx);
                Dot(s) = (cx * cy) + (dx * dy);
                Magnitude(s) = sqrt(cx^2 + cy^2) * sqrt(dx^2 + dy^2);
                Thetasin(s) = asin(Cross(s)/Magnitude(s));
                Thetacos(s) = pi - (acos(Dot(s)/Magnitude(s)));
            end % for s
            % if both sin calculations are positive, no choice can be
            % made, so cos calculation is in need
            if sign(Thetasin) == 1
                theta = abs(Thetacos);
            else
                theta = abs(Thetasin);
            end
            % choose max angle
            [C,I] = max(theta);
            next_node = P_edges(M(I));
        % if only one edge to the left
        elseif numel(M) == 1
            next_node = P_edges(M);
        elseif numel(M) == 0;
            next_node = [];
        end % M
    end % if l1, l2

else % if the line does not intersect
    P_edges = GG((GG(:,1)==P) & (GG(:,2)~=P),2);
    if ~isempty(P_edges)
        for r = 1:numel(P_edges)
          % cross product check to discard the right and keep the
          % left edges
          x1 = coord_of_nodes(side_sink,2);
          y1 = coord_of_nodes(side_sink,3);
          x2 = coord_of_nodes(P,2);
          y2 = coord_of_nodes(P,3);
          x3 = coord_of_nodes((P_edges(r)),2);
          y3 = coord_of_nodes((P_edges(r)),3);
          ax = x1 - x2;
          ay = y1 - y2;
          bx = x3 - x2;
          by = y3 - y2;
          check(r) = (ax * by) - (ay * bx);
        end % for r
        % choose the left ones (if more than one) and calculate angle
        % to determine the most left
        M = find(check>0);
        N = find(check == 0); % if lines coincide, sink is a neighbour
        if (numel(N) > 0) && (ismember(side_sink, P_edges))
            next_node = side_sink;
        elseif numel(M) > 1
            for s = 1:numel(M)
                x4 = coord_of_nodes(side_sink,2);
                y4 = coord_of_nodes(side_sink,3);
                x5 = coord_of_nodes(P,2);
                y5 = coord_of_nodes(P,3);
                x6 = coord_of_nodes((P_edges(M(s))),2);
                y6 = coord_of_nodes((P_edges(M(s))),3);
                cx = x4 - x5;
                cy = y4 - y5;
                dx = x6 - x5;
                dy = y6 - y5;
                Cross(s) = (cx * dy) - (cy * dx);
                Dot(s) = (cx * cy) + (dx * dy);
                Magnitude(s) = sqrt(cx^2 + cy^2) * sqrt(dx^2 + dy^2);
                Thetasin(s) = asin(Cross(s)/Magnitude(s));
                Thetacos(s) = pi - (acos(Dot(s)/Magnitude(s)));
            end % for s
            % if both sin calculations are positive, no choice can be
            % made, so cos calculation is in need
            if sign(Thetasin) == 1
                theta = abs(Thetacos);
            else
                theta = abs(Thetasin);
            end
```

```
                % choose max angle
                [C,I] = max(theta);
                next_node = P_edges(M(I));
            % if only one edge to the left
            elseif numel(M) == 1
                next_node = P_edges(M);
            elseif numel(M) == 0;
                next_node = [];
            end % M
        end
    end % if Ua, Ub
  end % for c
end % for g
```

A.2 Curvy Routing Source Code

A.2.1 Main.m

```
% load the topology
load Data/Topology.mat;
%R = randperm(num_of_nodes);
load R.mat;

% Initial setup
num_of_nodes = 50;
area_of_plot = 200; % metres
trans_range = 62; % metres

%find central node and use as sink
side = zeros(1,3);
side(1,2) = 150;
side(1,3) = 150;
side_sink = dsearchn(coord_of_nodes, side);

% transformation to obtain new coordinates
[T_coords, f] = transformation(num_of_nodes, coord_of_nodes);

% sort T_coords for routing and create index map for old names
sorted_T_coords = sortrows(T_coords);
[tf, name_index] = ismember(sorted_T_coords, T_coords, 'rows');
new_side_sink = find(name_index == side_sink);

% UDG
[Path, path_alg, node_density, T_neighbour_matrix, vicinity_matrix, num_of_neighbours,
energy_matrix, zero_energy, sorted_energy_matrix] = UDG(coord_of_nodes, new_side_sink,
num_of_nodes, trans_range, sorted_T_coords, name_index, R);
```

A.2.2 Transformation.m

```
function [T_coords, f] = transformation(num_of_nodes, coord_of_nodes)

% Rotating the network to fit the ellipse
% edge points of the graph
exX_1 = 0;
exY_1 = 0;
exX_2 = 200;
exY_2 = 0;
% first and last node coordinates
ExX_1 = coord_of_nodes(1,2);
ExY_1 = coord_of_nodes(1,3);
ExX_2 = coord_of_nodes(num_of_nodes,2);
ExY_2 = coord_of_nodes(num_of_nodes,3);

% calculation of rotation angle
nom = (exX_2 - exX_1) * (ExX_2 - ExX_1) + (exY_2 - exY_1) * (ExY_2 - ExY_1);
```

```
denom = sqrt((exX_2 - exX_1)^2 + (exY_2 - exY_1)^2) * sqrt((ExX_2 - ExX_1)^2 + (ExY_2 -
ExY_1)^2);
cosg = nom/denom;
sing = cos(pi/2 - acos(cosg));
% rotation matrix
R_matrix = [cosg sing; -sing cosg];

% multiply the two matrices to obtain the middle coordinates
deleted_coords = coord_of_nodes;
deleted_coords(:,1) = [];
d_coords = deleted_coords - 100;
mid_coords = d_coords * R_matrix;

% calculation of the stretch factor
f = (max(mid_coords(:,2)) - min(mid_coords(:,2))) / (max(mid_coords(:,1)) - min(mid_coords(:,
1)));
% stretch factor matrix
f_matrix = [1 0; 0 f];

% multiply the new matrices to obtain the transformed coordinates
T_coords = mid_coords * f_matrix;
```

A.2.3 Unit_disc_graph.m

```
function [Path, path_alg, node_density, T_neighbour_matrix, vicinity_matrix,
num_of_neighbours, energy_matrix, zero_energy, sorted_energy_matrix] = UDG(coord_of_nodes,
new_side_sink, num_of_nodes, trans_range, sorted_T_coords, name_index, R)

hold on;

% Distance from first node to last node
for i = 1:num_of_nodes
  x1_point = sorted_T_coords(i,1);
  y1_point = sorted_T_coords(i,2);
  density = 0;

  % Plot randomly chosen points
  plot (sorted_T_coords(i,1), sorted_T_coords(i,2), 'k*');
  text(sorted_T_coords(i,1) + 2, sorted_T_coords(i,2) + 2, num2str(i));
  xlabel('X-axis (distance)');
  ylabel('Y-axis (distance)');
  title('Dijkstra Algorithm', 'FontSize',14);

  for j = 1:num_of_nodes
    x2_point = sorted_T_coords(j,1);
    y2_point = sorted_T_coords(j,2);
    distance_bw_points(i,j) = sqrt((y2_point - y1_point)^2 + (x2_point - x1_point)^2);

      if ((distance_bw_points(i,j) <= trans_range))
       density = density + 1;
       vicinity_matrix(i,j)= 1;
       % Links between nodes
       line([x1_point x2_point], [y1_point y2_point], 'LineStyle', ':');
      else
       vicinity_matrix(i,j)= inf;
      end
  end
node_density(i) = density;

% Find the number of neighbours counted comparing distance_bw_points to trans_range (self node
included)
num_of_neighbours = ones(length(node_density),2);
num_of_neighbours(:,1) = 1:i;
num_of_neighbours(:,2) = node_density';
% Search the vicinity matrix and return the neighbours (ones - self node
%inluded) in row-column, then flip to display column-row
[row, col] = find(vicinity_matrix == 1);
T_neighbour_matrix = fliplr([row, col])';
end

% Shortest-path and corresponding cost calculation using Dijkstra
[path] = dijkstra(1, num_of_nodes, vicinity_matrix);
```

```
% Draw shortest path
if ~isempty(path)
    for i = 1:(length(path)-1)

       switch i
            case 1
              col_scheme = 'k';
            case 2
              col_scheme = 'c';
            case 3
              col_scheme = 'r';
            case 4
              col_scheme = 'g';
            case 5
              col_scheme = 'm';
            case 6
              col_scheme = 'k';
            case 7
              col_scheme = 'r';
              case 8
              col_scheme = 'r';
            case 9
              col_scheme = 'g';
            case 10
              col_scheme = 'm';
            case 11
              col_scheme = 'k';
            case 12
              col_scheme = 'c';
            case 13
              col_scheme = 'r';
            case 14
              col_scheme = 'g';
            case 15
              col_scheme = 'm';
            otherwise
              col_scheme = 'k';
       end
        line([sorted_T_coords(path(i),1) sorted_T_coords(path(i+1),1)],
[sorted_T_coords(path(i),2) sorted_T_coords(path(i+1),2)], 'Color',col_scheme,'LineWidth', 2,
'LineStyle', '--');
    end
end
legend('Nodes','Paths b/w nodes',2);
box on;

hold off;

%figure;
%UDG_circles(path, trans_range, sorted_T_coords, num_of_nodes, area_of_plot);

% Hello broadcast energy info
[hello_energy_matrix] = hello_energy(num_of_nodes, T_neighbour_matrix);
energy_matrix = zeros(num_of_nodes,num_of_nodes+1);

Path = cell(num_of_nodes,1);
for ord = 1:num_of_nodes
    m = R(ord);
figure;
if ~ismember(m, T_neighbour_matrix)
    Path{ord} = m;
else
[path_alg, next_hop, k] = greedy_alg(new_side_sink, num_of_nodes, trans_range, m,
T_neighbour_matrix, sorted_T_coords);
%path_alg
Path(ord,1) = {path_alg};

if (path_alg(numel(path_alg)) ~= new_side_sink) && (sorted_T_coords(path_alg(numel(path_alg)),
1) == sorted_T_coords(new_side_sink,1)) && (sorted_T_coords(path_alg(numel(path_alg)),2) ==
sorted_T_coords(new_side_sink,2))
  Path{ord} = [Path{ord}, new_side_sink];
elseif (path_alg(numel(path_alg)) ~= new_side_sink) && ((next_hop < k) || (next_hop <
new_side_sink))
```

```
  Path{ord} = [Path{ord}, Path{next_hop}];
  S = intersect(Path{next_hop}, zero_energy);
  if (isempty(Path{next_hop})) || ~isempty(S)
      %find Path{next_hop} if not empty and remove, to add the new path_2
      if (~isempty(S))
        y = numel(Path{next_hop});
        z = find(Path{ord}, y, 'last');
        Path{ord}(z) = [];
      end
      n = next_hop;
      [path_2, next_2] = greedy_2(new_side_sink, num_of_nodes, trans_range, n,
T_neighbour_matrix, sorted_T_coords);
      Path{ord} = [Path{ord}, path_2];
      S = intersect(path_2, zero_energy);
      if (path_2(numel(path_2)) ~= new_side_sink) || ~isempty(S)
        %find path_2 if not empty and remove, to add the new path_2
        if (~isempty(S))
           y = numel(path_2);
           z = find(Path{ord}, y, 'last');
           Path{ord}(z) = [];
        end
      mn = next_2;
      [path_3, next_3] = greedy_3(new_side_sink, num_of_nodes, trans_range, mn,
T_neighbour_matrix, sorted_T_coords);
      Path{ord} = [Path{ord}, path_3];
      end % if numel(path_2)
  end % if next_hop empty
elseif (path_alg(numel(path_alg)) ~= new_side_sink) && ((next_hop > k) && (next_hop >
new_side_sink))
  n = next_hop;
  [path_2, next_2] = greedy_2(new_side_sink, num_of_nodes, trans_range, n, T_neighbour_matrix,
sorted_T_coords);
  Path{ord} = [Path{ord}, path_2];
  S = intersect(path_2, zero_energy);
  if (path_2(numel(path_2)) ~= new_side_sink) || ~isempty(S)
      %find path_2 if not empty and remove, to add the new path_2
      if (~isempty(S))
         y = numel(path_2);
         z = find(Path{ord}, y, 'last');
          Path{ord}(z) = [];
      end
      mn = next_2;
      [path_3, next_3] = greedy_3(new_side_sink, num_of_nodes, trans_range, mn,
T_neighbour_matrix, sorted_T_coords);
      Path{ord} = [Path{ord}, path_3];
      S = intersect(path_3, zero_energy);
      if (path_3(numel(path_3)) ~= new_side_sink) || ~isempty(S)
          %find path_3 if not empty and remove, to add the new path_2
          if (~isempty(S))
              y = numel(path_3);
              z = find(Path{ord}, y, 'last');
              Path{ord}(z) = [];
          end
          nm = next_3;
          [path_4, next_4] = greedy_4(new_side_sink, num_of_nodes, trans_range, nm,
T_neighbour_matrix, sorted_T_coords);
          Path{ord} = [Path{ord}, path_4];
      end % if numel(path_3)
  end % if numel(path_2)
end % if numel(path_alg)
end % if m is dead

if Path{ord}(numel(Path{ord})) == 12 || Path{ord}(numel(Path{ord})) == 13
    if ~ismember(15, zero_energy)
        Path{ord} = [Path{ord}, 15, new_side_sink];
    else
        Path{ord} = [Path{ord}, 16, new_side_sink];
    end
end

% energy_matrix
energy_matrix(:,1) = hello_energy_matrix(:,end);
energy_matrix(Path{ord}(1:end-1),ord) = energy_matrix(Path{ord}(1:end-1),ord) - 75; % energy
to send -75 mJ
```

```
energy_matrix(Path{ord}(2:end),ord) = energy_matrix(Path{ord}(2:end),ord) - 5; % energy to
receive -5 mJ
energy_matrix(Path{ord},ord) = energy_matrix(Path{ord},ord) + 1; % no idle for Path nodes
energy_matrix(:,ord+1) = energy_matrix(:,ord) - 1; % idle energy depletion -1 mJ

% list of dead nodes
zero_energy = find(energy_matrix(:,ord) <= 0);
% case if a node is dead
if ismember(m, zero_energy)
    Path{ord} = -m;
end

disp(Path{ord});

% update vicinity and neighbour matrices
for a = 1:numel(zero_energy);
vicinity_matrix(zero_energy(a),:) = Inf;
vicinity_matrix(:,zero_energy(a)) = Inf;
end
[row, col] = find(vicinity_matrix == 1);
T_neighbour_matrix = fliplr([row, col])';

% if sink is dead stop the simulation
if ismember(new_side_sink, zero_energy)
    disp('sink is dead');
    break
end
end % ord ends

celldisp(Path);
disp(energy_matrix);
disp(zero_energy);

%Energy graph
energy_matrix(:,1) = [];
energy_matrix(energy_matrix < 0) = 0;
X = coord_of_nodes(:,2);
Y = coord_of_nodes(:,3);
% retrieve original node names before plotting energy
index_energy_matrix = [name_index energy_matrix];
sorted_energy_matrix = sortrows(index_energy_matrix);
sorted_energy_matrix(:,1) = [];
for h = 1:num_of_nodes
    Z = repmat(h,num_of_nodes,1);
    scatter3 (X, Y, Z, 4*Z, sorted_energy_matrix(:,h));
    hold on;
end
```

A.2.4 Dijkstra.m

Dijkstra's algorithm has not been modified (same as in A.1.3)

A.2.5 Hello_energy.m

Hello_energy.m file has remained the same as in A.1.4

A.2.6 Greedy_algorithm.m

```
function [path_alg, next_hop, k] = greedy_alg(new_side_sink, num_of_nodes, trans_range, m,
T_neighbour_matrix, sorted_T_coords)

hold on;

for i=1:num_of_nodes
```

```
  x1_point = sorted_T_coords(i,1);
  y1_point = sorted_T_coords(i,2);

  % Plot randomly chosen points
  plot (sorted_T_coords(i,1), sorted_T_coords(i,2), 'k*');
  text(sorted_T_coords(i,1) + 2, sorted_T_coords(i,2) + 2, num2str(i));
  xlabel('X-axis (distance)');
  ylabel('Y-axis (distance)');
  title('Localised Greedy Curved Algorithm', 'FontSize',12);

  for j=1:num_of_nodes
    x2_point = sorted_T_coords(j,1);
    y2_point = sorted_T_coords(j,2);
    graph_dist = sqrt((y2_point - y1_point)^2 + (x2_point - x1_point)^2);

    if (graph_dist <= trans_range)
      line([x1_point x2_point], [y1_point y2_point], 'LineStyle', ':');
    end
  end % j ends
end % i ends


if (m == new_side_sink)
    k = new_side_sink;
    next_hop = k;
    path_alg = next_hop;
elseif (m < new_side_sink)
for k = m:new_side_sink
  if (k == m)
    % calculate angles of all eligible neighbours and retrieve next hop
    [next_hop] = greedy_curve(k, num_of_nodes, T_neighbour_matrix, new_side_sink,
sorted_T_coords);
    counter = 1;
    path_alg(counter) = k;
  elseif (k == next_hop)
    counter = counter + 1;
    path_alg(counter) = k;
    % calculate angles of all eligible neighbours and retrieve next hop
    [next_hop] = greedy_curve(k, num_of_nodes, T_neighbour_matrix, new_side_sink,
sorted_T_coords);
  end
end % k ends
elseif (m > new_side_sink)
for k = m:-1:new_side_sink
  if (k == m)
    % calculate angles of all eligible neighbours and retrieve next hop
    [next_hop] = greedy_curve(k, num_of_nodes, T_neighbour_matrix, new_side_sink,
sorted_T_coords);
    counter = 1;
    path_alg(counter) = k;
  elseif (k == next_hop)
    counter = counter + 1;
    path_alg(counter) = k;
    % calculate angles of all eligible neighbours and retrieve next hop
    [next_hop] = greedy_curve(k, num_of_nodes, T_neighbour_matrix, new_side_sink,
sorted_T_coords);
  end
end % k ends
end % m ends


if ~isempty(path_alg)
    for i = 1:(length(path_alg) - 1)
       switch i
            case 1
              col_scheme = 'k';
            case 2
              col_scheme = 'c';
            case 3
              col_scheme = 'r';
            case 4
              col_scheme = 'g';
            case 5
              col_scheme = 'm';
```

```
            case 6
              col_scheme = 'k';
            case 7
              col_scheme = 'c';
            case 8
              col_scheme = 'r';
            case 9
              col_scheme = 'g';
            case 10
              col_scheme = 'm';
            case 11
              col_scheme = 'k';
            case 12
              col_scheme = 'c';
            case 13
              col_scheme = 'r';
            case 14
              col_scheme = 'g';
            case 15
              col_scheme = 'm';
            otherwise
              col_scheme = 'k';
       end
    line([sorted_T_coords(path_alg(i),1) sorted_T_coords(path_alg(i+1),1)],
[sorted_T_coords(path_alg(i),2) sorted_T_coords(path_alg(i+1),2)],
'Color',col_scheme,'LineWidth', 2, 'LineStyle', '--');
    end
end
legend('Nodes','Paths b/w nodes',2);
box on;
hold off;
```

A.2.7 Curvy.m

```
function [next_hop] = greedy_curve(k, num_of_nodes, T_neighbour_matrix, new_side_sink,
sorted_T_coords)

Nn = flipud(rot90(T_neighbour_matrix));
NN = Nn((Nn(:,1) == k) & (Nn(:,2) ~= k),2);
% next hop is sink, if neighbours with k
if ismember(new_side_sink,NN)
    next_hop = new_side_sink;
else
    %find next hop using curves

% get polar coordinates
[P_coords, theta, r] = polar(num_of_nodes, sorted_T_coords);

% calculation of variables needed to obtain psi angle
a = sqrt(((sorted_T_coords(num_of_nodes,1)- sorted_T_coords(1,1))^2 +
(sorted_T_coords(num_of_nodes,2) - sorted_T_coords(1,2))^2))/2;
test_alpha = atan((sin(theta(new_side_sink))*r(new_side_sink)*(r(k)^2 - a^2) -
sin(theta(k))*r(k)*(r(new_side_sink)^2 - a^2))/
(cos(theta(new_side_sink))*r(new_side_sink)*(r(k)^2 - a^2) -
cos(theta(k))*r(k)*(r(new_side_sink)^2 - a^2)));
if ((sin(theta(k) - test_alpha) > 0) && (r(k) < test_alpha))
    alpha_r = test_alpha + pi;
elseif ((sin(theta(k) - test_alpha) < 0) && (r(k) > test_alpha))
    alpha_r = test_alpha - pi;
else
    alpha_r = test_alpha;
end
n0 = 2;
n = n0*(1 /(1 + (r(k)/a)^2));
e = (r(k) - a^2)/(r(k)*sin(theta(k) - alpha_r));
c = (r(k)*n*a^2*sin(theta(k) - alpha_r))/(sqrt((r(k) - a^2)^2 + 4));
r_theta_1 = (-e*sin(theta(k) - alpha_r) + sqrt((e*sin(theta(k) - alpha_r))^2 + 4*a^2))/2;
r_theta_2 = (-e*sin(theta(k) - alpha_r) - sqrt((e*sin(theta(k) - alpha_r))^2 + 4*a^2))/2;
if r_theta_1 >= 0
    r_theta = r_theta_1;
else%if r_theta_2 >= 0
    r_theta = r_theta_2;
```

```
end
% vector calculation
ri = [cos(theta(k)); sin(theta(k))];
rs = r_theta * ri;
tx_1 = (rs(1) * sqrt(n^2*(rs(1)^2 + rs(2)^2) - c^2) - c*rs(2))/n*(rs(1)^2 + rs(2)^2);
tx_2 = (rs(1) * sqrt(n^2*(rs(1)^2 + rs(2)^2) - c^2) - c*rs(2))/(-n*(rs(1)^2 + rs(2)^2));
ty_1 = sqrt(1 - (tx_1)^2);
ty_2 = sqrt(1 - (tx_2)^2);
tangents = [tx_1 ty_1; tx_2 ty_2];
dest = [sorted_T_coords(new_side_sink,1) sorted_T_coords(new_side_sink,2)];
which_tx = dsearchn(tangents,dest);
if which_tx == 1
    tx = tx_1;
    ty = ty_1;
elseif which_tx == 2
    tx = tx_2;
    ty = ty_2;
end
%t = [tx; ty];
v = [-ty; tx];
% gradient in polar coordinates: gradient((log(n))) = -2*r(k)/(a^2 + (r(k))^2)) * ri
G_log_n = (-2*r(k)/(a^2 + (r(k))^2)) * ri;
K = v(1) * G_log_n(1) + v(2) * G_log_n(2);
rho = 1/abs(K);
% find unit vectors and check direction of v with dot product
test_v_unit = v/sqrt(v(1)^2 + v(2)^2);
rs_unit = rs/sqrt(rs(1)^2 + rs(2)^2);
Dot_v_rs = test_v_unit(1) * rs_unit(1) + test_v_unit(2) * rs_unit(2);
if Dot_v_rs > 0
    v_unit = test_v_unit;
else%if Dot_v_rs < 0
    v_unit = -test_v_unit;
end
rc = rs + rho*v_unit;
% find mean distance between each node and its neighbours
for f = 1:numel(NN)
    dist_neig(f) = sqrt((sorted_T_coords(NN(f),1) - sorted_T_coords(k,1))^2 +
(sorted_T_coords(NN(f),2) - sorted_T_coords(k,2))^2);
end % for f
d = mean(dist_neig);
for g = 1:numel(NN)
    rn = [sorted_T_coords(NN(g),1); sorted_T_coords(NN(g),2)];
    rm = rn - rc;
    cross_rsrc = rs(1) * rc(2) - rs(2) * rc(1);
    cross_rcrs = rc(1) * rs(2) - rc(2) * rs(1);
    % check which one to use as first
    if cross_rsrc > cross_rcrs
        % when rs above rc
        rsc = rs - rc;
        rnc = rn - rc;
        psi_angle = asin((rsc(1)*rnc(2) - rsc(2)*rnc(1))/(sqrt(rsc(1)^2 + rsc(2)^2) *
sqrt(rnc(1)^2 + rnc(2)^2)));
    else%if cross_rcrs > cross_rsrc
        % when rc above rs
        rcs = rc - rs;
        rcn = rc - rn;
        psi_angle = asin((rcs(1)*rcn(2) - rcs(2)*rcn(1))/(sqrt(rcs(1)^2 + rcs(2)^2) *
sqrt(rcn(1)^2 + rcn(2)^2)));
    end
    angle_psi(g) = psi_angle;
    % threshold for eligible neighbours
    threshold = abs(sqrt(rm(1)^2 + rm(2)^2) - rho);
    if (threshold <= 2*d) && (sin(psi_angle) >= 0) % choose only positive angles
        psi_n(g) = psi_angle;
    else % if no eligible neighbours
        psi_n(g) = -0;
    end % threshold
end % for g

% choose the largest angle
[psi_value, psi_index] = max(abs(psi_n));
% find which neighbour has the largest angle and set it as the next hop
next_hop = NN(psi_index);
end
```

A.2.8 Polar.m

```
function [P_coords, theta, r] = polar(num_of_nodes, sorted_T_coords)

% find polar coordinates of neighbours
for h = 1:num_of_nodes
    theta(h) = atan2(sorted_T_coords(h,2), sorted_T_coords(h,1));
    r(h) = sqrt((sorted_T_coords(h,1))^2 + (sorted_T_coords(h,2))^2);

    PolarX(h) = r(h)*cos(theta(h));
    PolarY(h) = r(h)*sin(theta(h));
end
% polar coordinates matrix
P_coords = ones(num_of_nodes,2);
P_coords(:,1) = PolarX;
P_coords(:,2) = PolarY;
```

A.2.9 Face_routing.m

Face_routing.m file has remained the same as in A.1.6

A.3 Energy Aware Routing Source Code

A.3.1 Main.m

Main.m file has remained the same as in A.2.1

A.3.2 Transformation.m

Transformation.m file has remained the same as in A.2.2

A.3.3 Unit_disc_graph.m

Hello_energy.m file has remained the same as in A.2.3

A.3.4 Dijkstra.m

Dijkstra's algorithm has not been modified (same as in A.1.3)

A.3.5 Hello_energy.m

Hello_energy.m file has remained the same as in A.1.4

A.3.6 Greedy_algorithm.m

Hello_energy.m file has remained the same as in A.2.6

A.3.7 Curvy.m

```
function [next_hop] = greedy_curve(k, num_of_nodes, T_neighbour_matrix, new_side_sink,
sorted_T_coords, energy_matrix, ord)

Nn = flipud(rot90(T_neighbour_matrix));
NN = Nn((Nn(:,1) == k) & (Nn(:,2) ~= k),2);
% next hop is sink, if neighbours with k
if ismember(new_side_sink,NN)
    next_hop = new_side_sink;
else
    %find next hop using curves

% get polar coordinates
[P_coords, theta, r] = polar(num_of_nodes, sorted_T_coords);

% calculation of variables needed to obtain psi angle
a = sqrt(((sorted_T_coords(num_of_nodes,1)- sorted_T_coords(1,1))^2 +
(sorted_T_coords(num_of_nodes,2) - sorted_T_coords(1,2))^2))/2;
test_alpha = atan((sin(theta(new_side_sink))*r(new_side_sink)*(r(k)^2 - a^2) -
sin(theta(k))*r(k)*(r(new_side_sink)^2 - a^2))/
(cos(theta(new_side_sink))*r(new_side_sink)*(r(k)^2 - a^2) -
cos(theta(k))*r(k)*(r(new_side_sink)^2 - a^2)));
if ((sin(theta(k) - test_alpha) > 0) && (r(k) < test_alpha))
    alpha_r = test_alpha + pi;
elseif ((sin(theta(k) - test_alpha) < 0) && (r(k) > test_alpha))
    alpha_r = test_alpha - pi;
else
    alpha_r = test_alpha;
end
n0 = 2;
old_n = n0*(1 /(1 + (r(k)/a)^2));
e = (r(k) - a^2)/(r(k)*sin(theta(k) - alpha_r));
```

```
c = (r(k)*old_n*a^2*sin(theta(k) - alpha_r))/(sqrt((r(k) - a^2)^2 + 4));
r_theta_1 = (-e*sin(theta(k) - alpha_r) + sqrt((e*sin(theta(k) - alpha_r))^2 + 4*a^2))/2;
r_theta_2 = (-e*sin(theta(k) - alpha_r) - sqrt((e*sin(theta(k) - alpha_r))^2 + 4*a^2))/2;
if r_theta_1 >= 0
    r_theta = r_theta_1;
else%if r_theta_2 >= 0
    r_theta = r_theta_2;
end
% vector calculation
ri = [cos(theta(k)); sin(theta(k))];
rs = r_theta * ri;
tx_1 = (rs(1) * sqrt(old_n^2*(rs(1)^2 + rs(2)^2) - c^2) - c*rs(2))/old_n*(rs(1)^2 + rs(2)^2);
tx_2 = (rs(1) * sqrt(old_n^2*(rs(1)^2 + rs(2)^2) - c^2) - c*rs(2))/(-old_n*(rs(1)^2 +
rs(2)^2));
ty_1 = sqrt(1 - (tx_1)^2);
ty_2 = sqrt(1 - (tx_2)^2);
tangents = [tx_1 ty_1; tx_2 ty_2];
dest = [sorted_T_coords(new_side_sink,1) sorted_T_coords(new_side_sink,2)];
which_tx = dsearchn(tangents,dest);
if which_tx == 1
    tx = tx_1;
    ty = ty_1;
elseif which_tx == 2
    tx = tx_2;
    ty = ty_2;
end
old_t = [tx; ty];
old_v = [-ty; tx];
% gradient in polar coordinates: gradient((log(n))) = -2*r(k)/(a^2 + (r(k))^2)) * ri
G_log_n = (-2*r(k)/(a^2 + (r(k))^2)) * ri;
old_K = old_v(1) * G_log_n(1) + old_v(2) * G_log_n(2);

%%%%%%%%%%%%% Local energy new variables calculation %%%%%%%%%%%%%%%%%%%%%%%
initial_energy = 500; % mJ
ksi = old_n/initial_energy;
%creation of A and b matrices for the singular value decomposition (svd)
for ii = 1:numel(NN)
```

```
    A_1(ii) = sorted_T_coords(NN(ii),1) - sorted_T_coords(k,1);
    A_2(ii) = sorted_T_coords(NN(ii),2) - sorted_T_coords(k,2);
    b(ii) = ksi*(energy_matrix(NN(ii),ord) - energy_matrix(k,ord));
end
A = [A_1' A_2'];
% svd to find the gradient of ne
[U,S,V] = svd(A,0);
x = V*S*U'*b';
G_ne = [x(1); x(2)];
ne = ksi * energy_matrix(k,ord);
% calculation of the new d_K components
d_Ky = (G_ne(1)^2*(-old_t(1)) - G_ne(1)*old_t(1)*ne*old_K - G_ne(1)*old_K - G_ne(1)*old_n -
ne*old_K^2 - ne*old_K*old_n)/(G_ne(1)^2*old_t(1)^2 + old_K^2 + 2*old_K*old_n + old_n^2);
d_Kx = (d_Ky^2*(-G_ne(2)-G_log_n(2)) + d_Ky*(G_log_n(2)*old_t(2) + G_log_n(2)*old_t(2) +
G_ne(2)*old_t(2)) + G_log_n(2) - ne*old_K)/(old_K + old_n);
% new vectors
d_K = [d_Kx(1); d_Ky];
d_t = [-d_Ky; d_Kx(1)];
K = old_K + d_K;
t = old_t + d_t;
v = [-t(2); t(1)];
%%%%%%%%%%%% end of new variables calculation %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


rho = 1/(sqrt(K(1)^2 + K(2)^2));
% find unit vectors and check direction of v with dot product
test_v_unit = v/sqrt(v(1)^2 + v(2)^2);
rs_unit = rs/sqrt(rs(1)^2 + rs(2)^2);
Dot_v_rs = test_v_unit(1) * rs_unit(1) + test_v_unit(2) * rs_unit(2);
if Dot_v_rs > 0
    v_unit = test_v_unit;
else%if Dot_v_rs < 0
    v_unit = -test_v_unit;
end
rc = rs + rho*v_unit;
% find mean distance between each node and its neighbours
for f = 1:numel(NN)
```

```
    dist_neig(f) = sqrt((sorted_T_coords(NN(f),1) - sorted_T_coords(k,1))^2 +
(sorted_T_coords(NN(f),2) - sorted_T_coords(k,2))^2);
end % for f
d = mean(dist_neig);
for g = 1:numel(NN)
    rn = [sorted_T_coords(NN(g),1); sorted_T_coords(NN(g),2)];
    rm = rn - rc;
    cross_rsrc = rs(1) * rc(2) - rs(2) * rc(1);
    cross_rcrs = rc(1) * rs(2) - rc(2) * rs(1);
    % check which one to use as first
    if cross_rsrc > cross_rcrs
        % when rs above rc
        rsc = rs - rc;
        rnc = rn - rc;
        psi_angle = asin((rsc(1)*rnc(2) - rsc(2)*rnc(1))/(sqrt(rsc(1)^2 + rsc(2)^2) *
sqrt(rnc(1)^2 + rnc(2)^2)));
    else%if cross_rcrs > cross_rsrc
        % when rc above rs
        rcs = rc - rs;
        rcn = rc - rn;
        psi_angle = asin((rcs(1)*rcn(2) - rcs(2)*rcn(1))/(sqrt(rcs(1)^2 + rcs(2)^2) *
sqrt(rcn(1)^2 + rcn(2)^2)));
    end
    % threshold for eligible neighbours
    threshold = abs(sqrt(rm(1)^2 + rm(2)^2) - rho);
    if (threshold <= 2*d) && (sin(psi_angle) >= 0) % choose only positive angles
        psi_n(g) = psi_angle;
    else % if no eligible neighbours
        psi_n(g) = -0;
    end % threshold
end % for g
% choose the largest angle
[psi_value, psi_index] = max(abs(psi_n));
% find which neighbour has the largest angle and set it as the next hop
next_hop = NN(psi_index);
end
```

A.3.8 Polar.m

Polar.m file has remained the same as in A.2.8

A.3.9 Face_routing.m

Face_routing.m file has remained the same as in A.2.9

APPENDIX B

Simulation Topologies

B.1 140 nodes

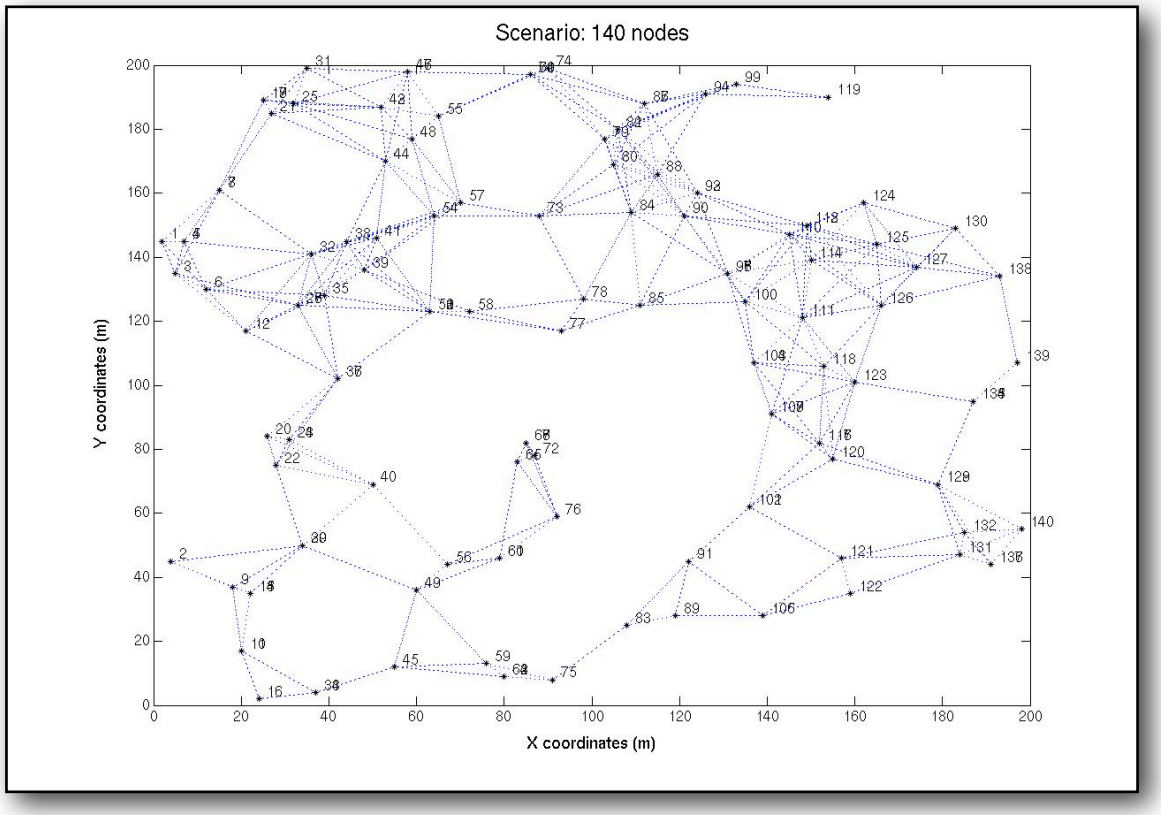


Figure B.1: Topology 1 with 140 nodes

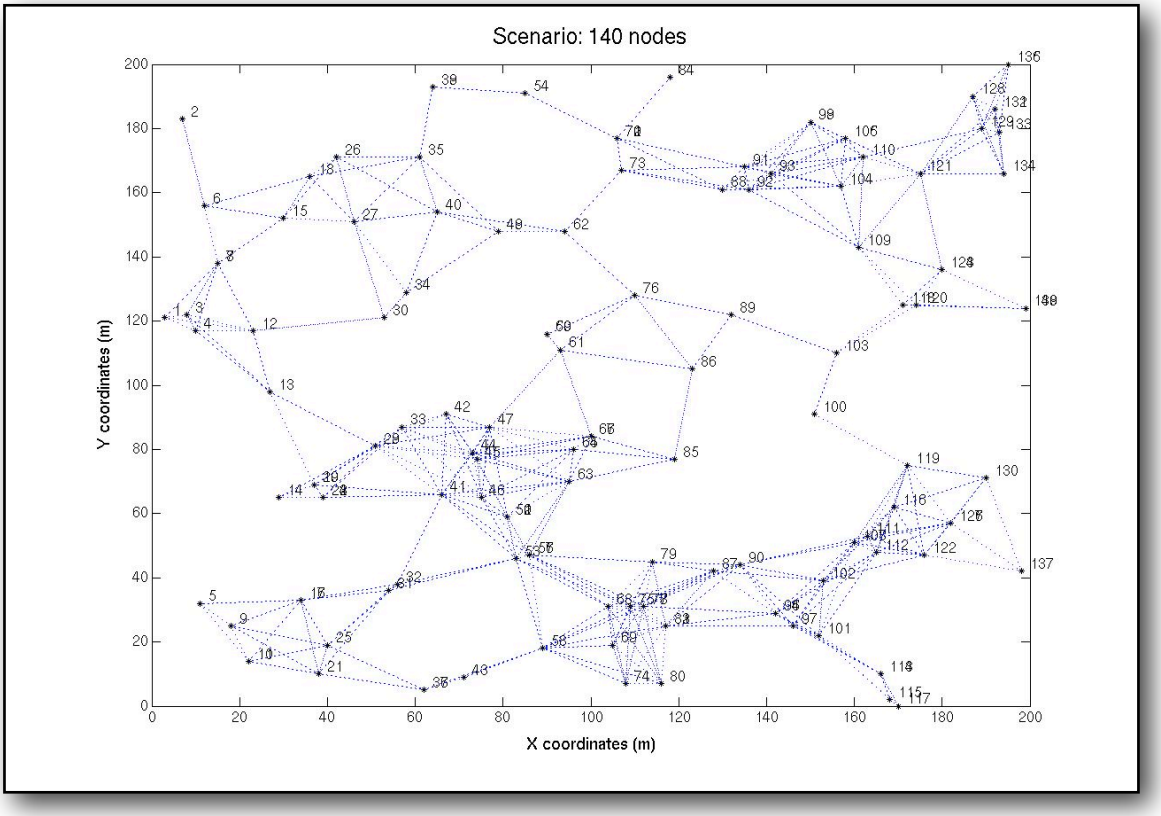


Figure B.2: Topology 2 with 140 nodes

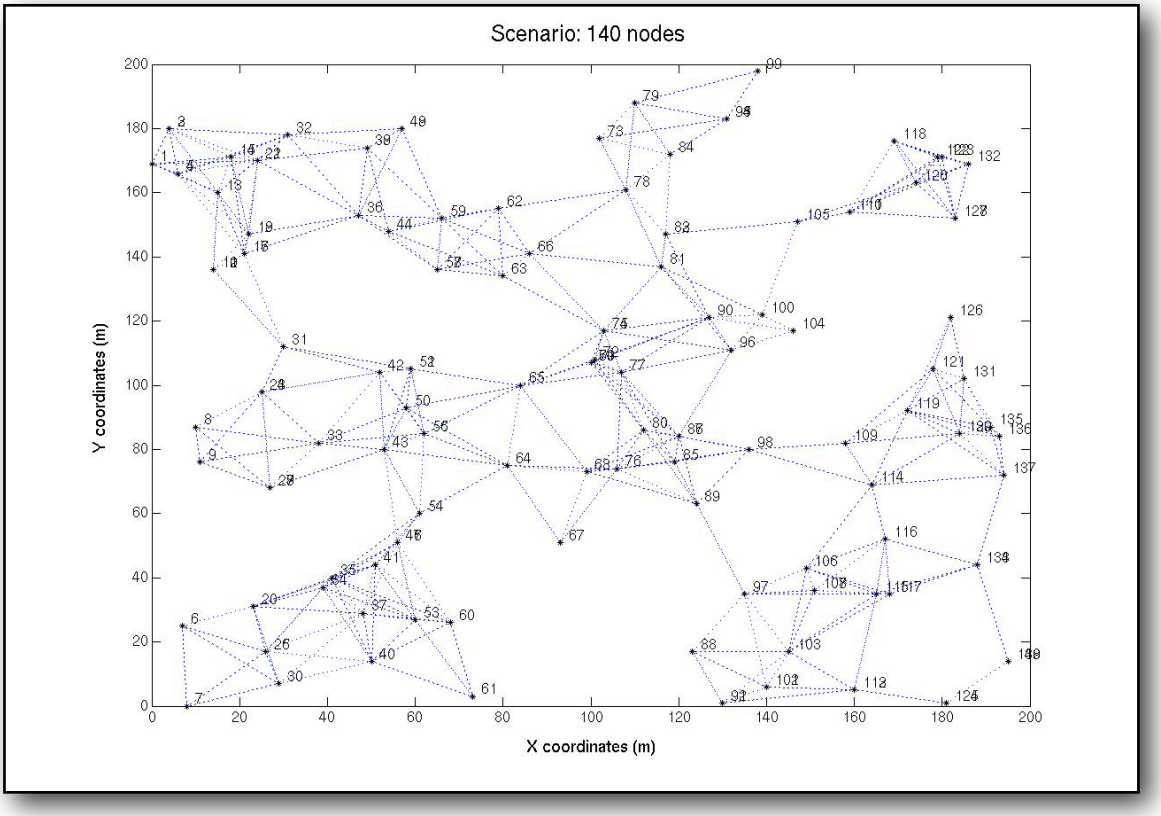


Figure B.3: Topology 3 with 140 nodes

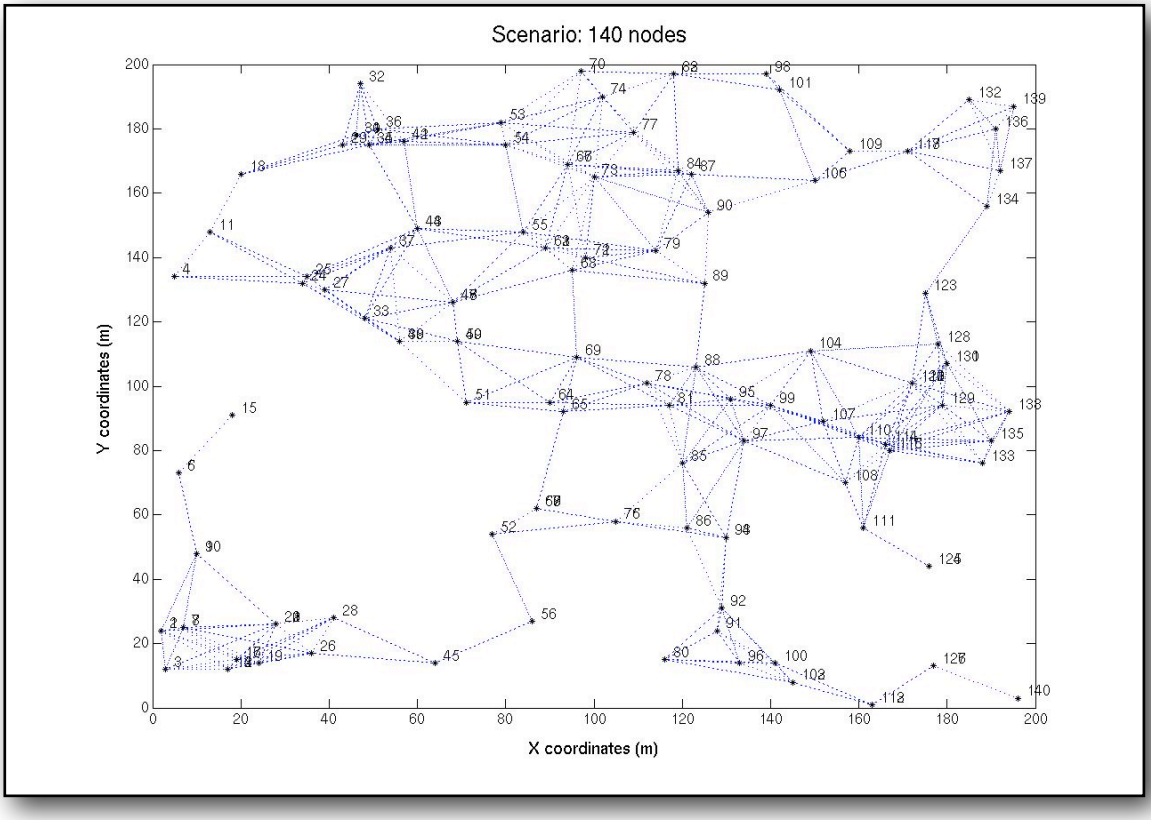


Figure B.4: Topology 4 with 140 nodes

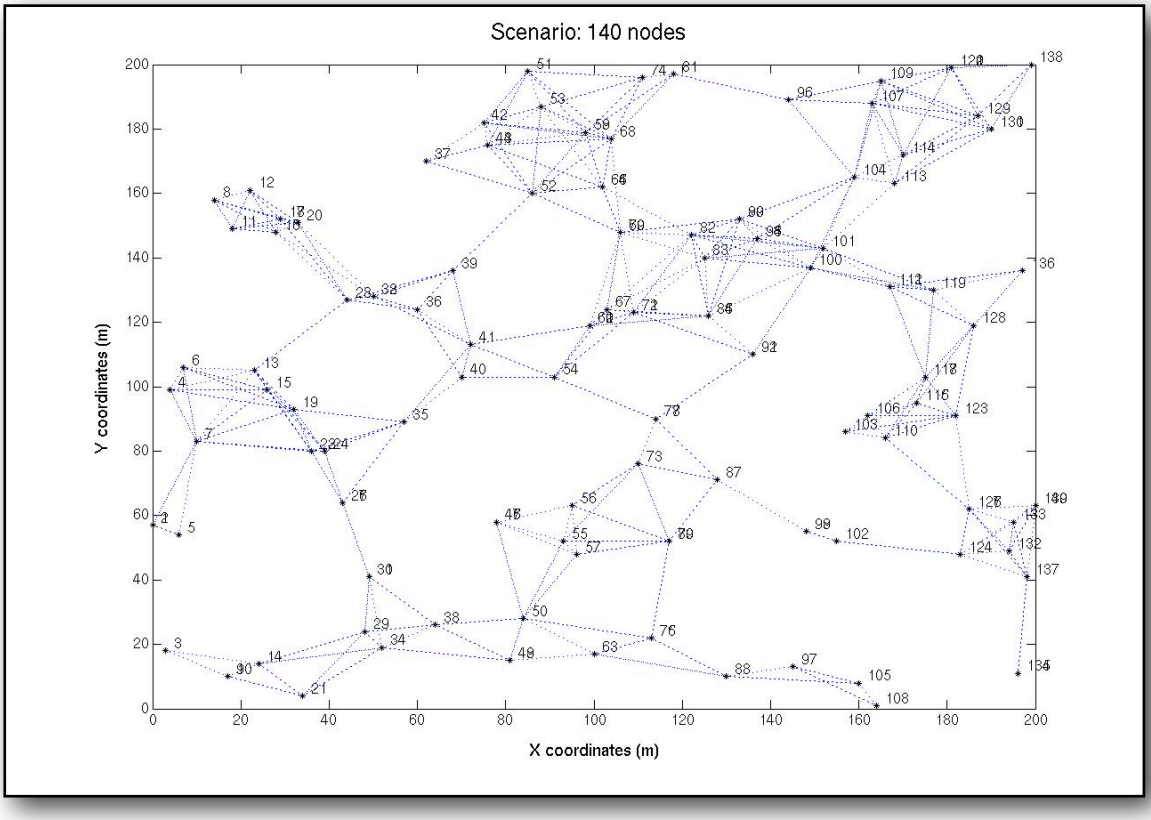


Figure B.5: Topology 5 with 140 nodes

B.2 100 nodes

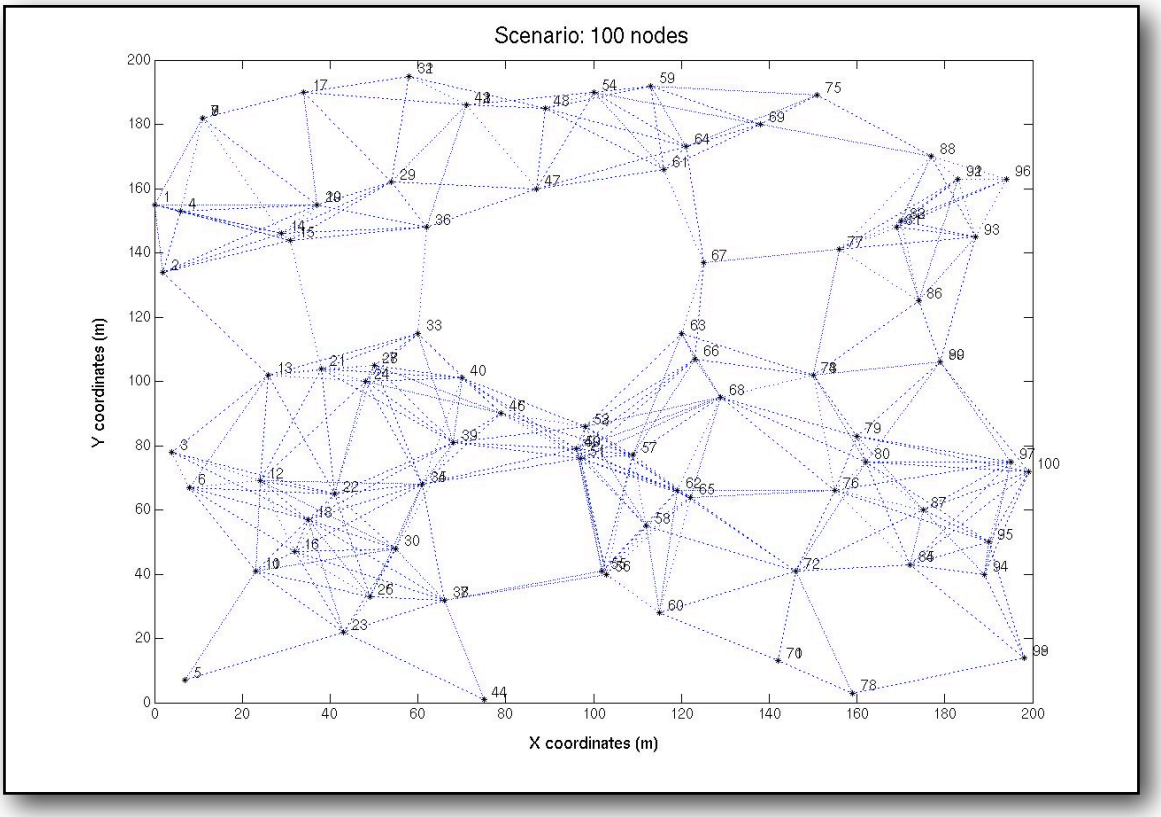


Figure B.6: Topology 1 with 100 nodes

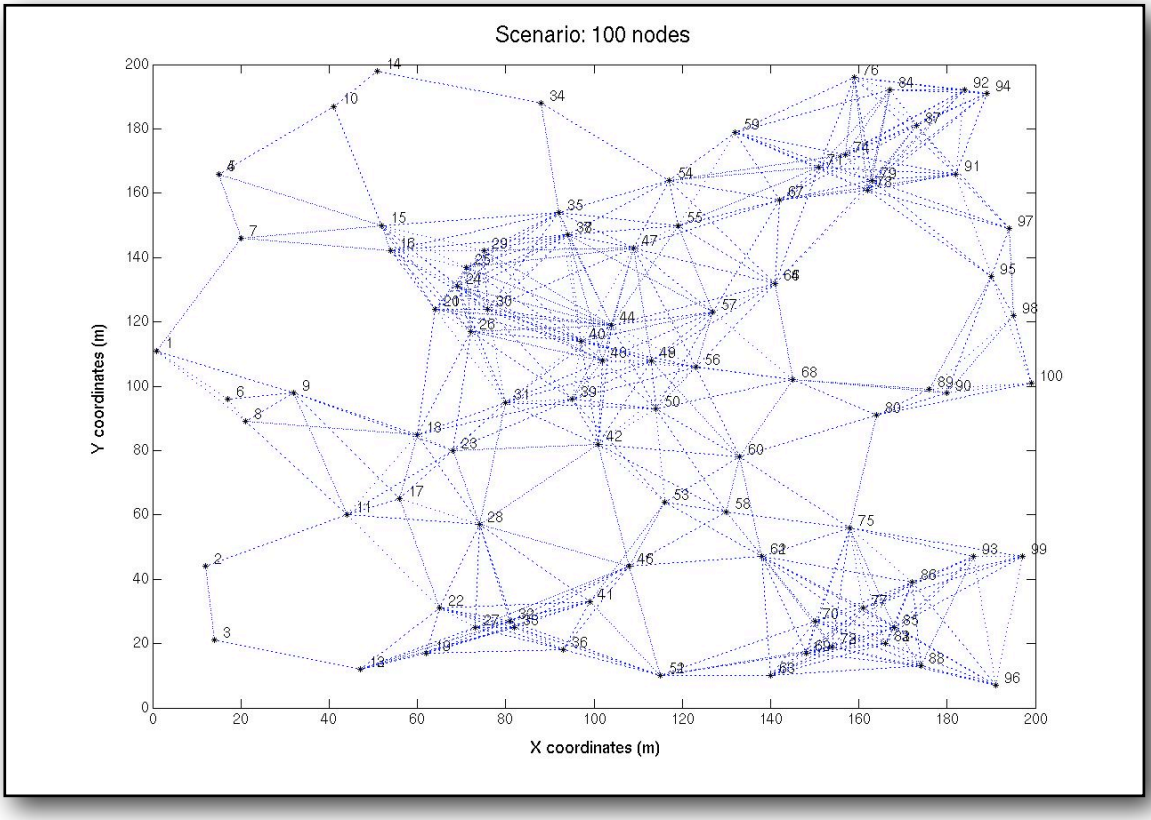


Figure B.7: Topology 2 with 100 nodes

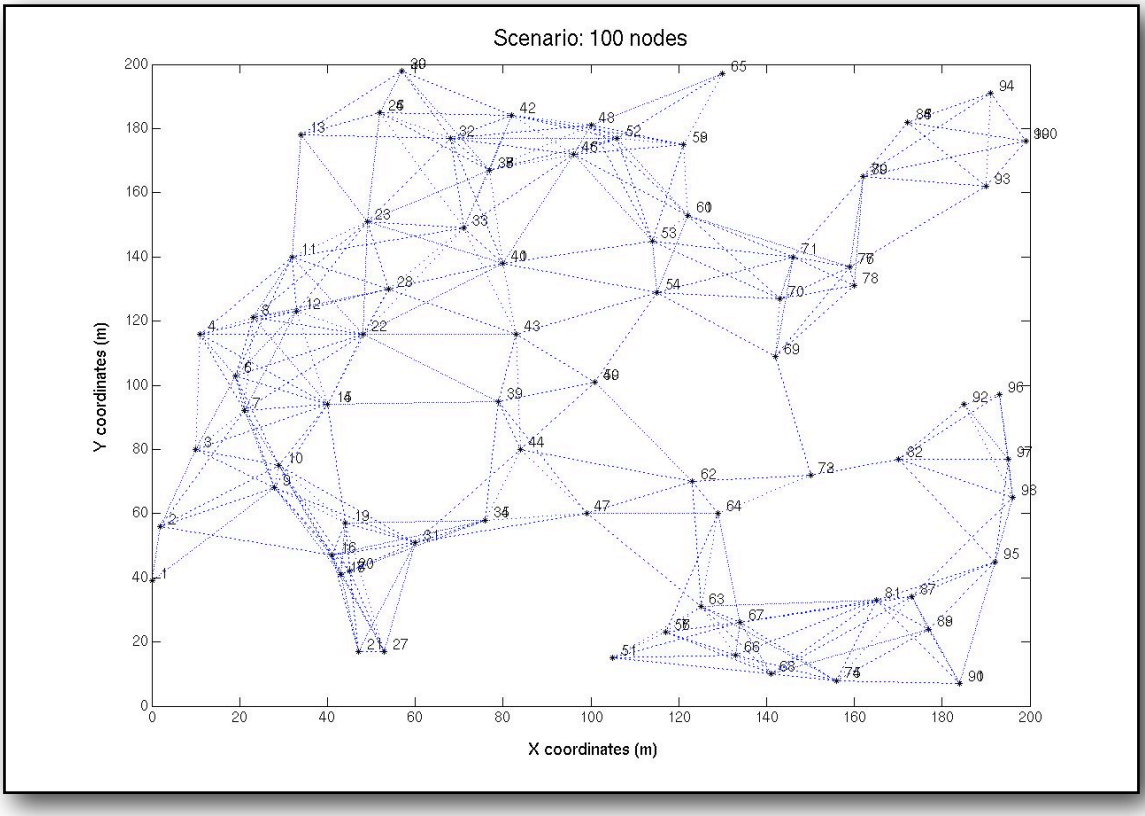


Figure B.8: Topology 3 with 100 nodes

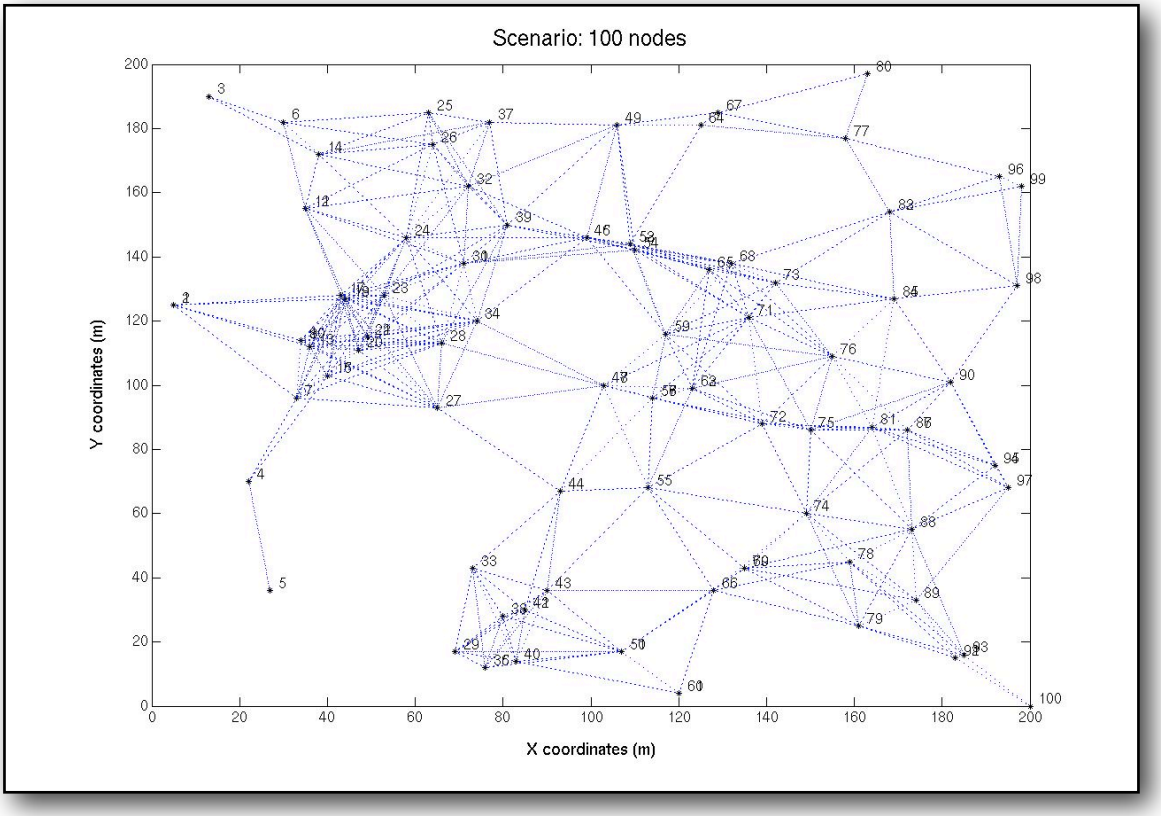


Figure B.9: Topology 4 with 100 nodes

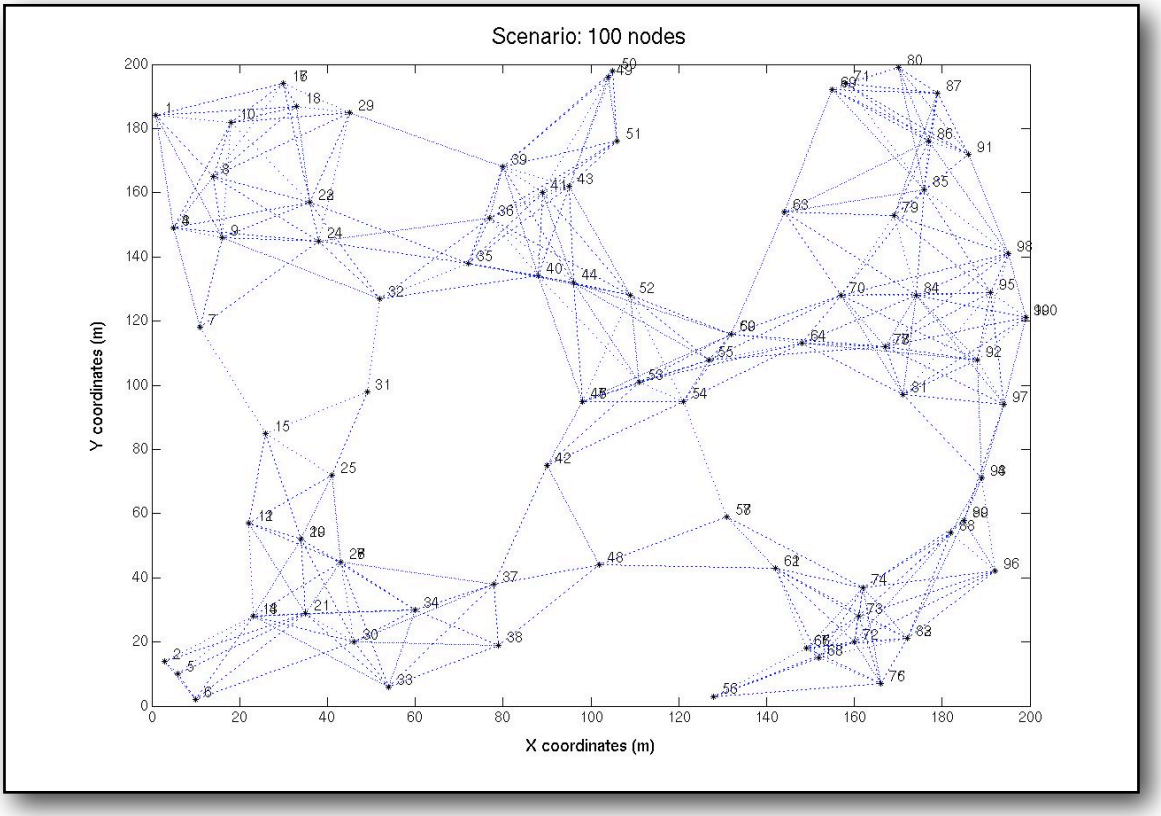


Figure B.10: Topology 5 with 100 nodes

B.3 75 nodes

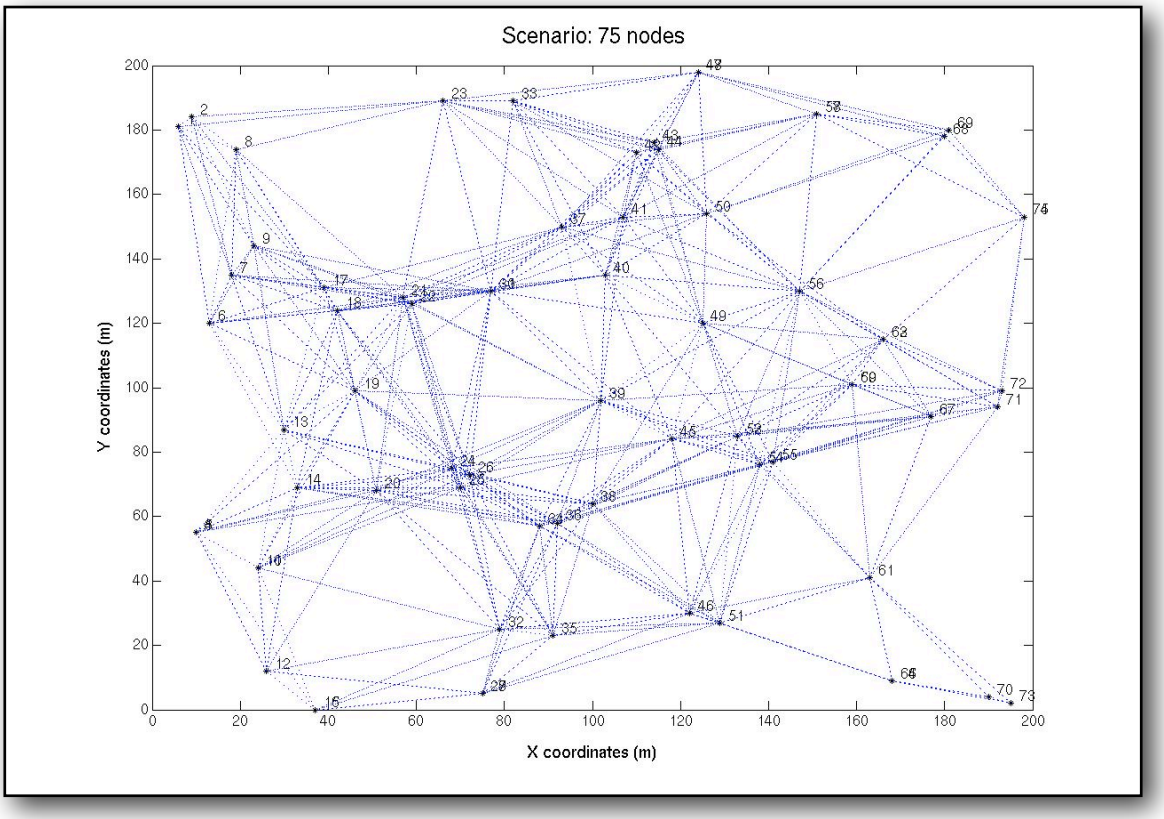


Figure B.11: Topology 1 with 75 nodes

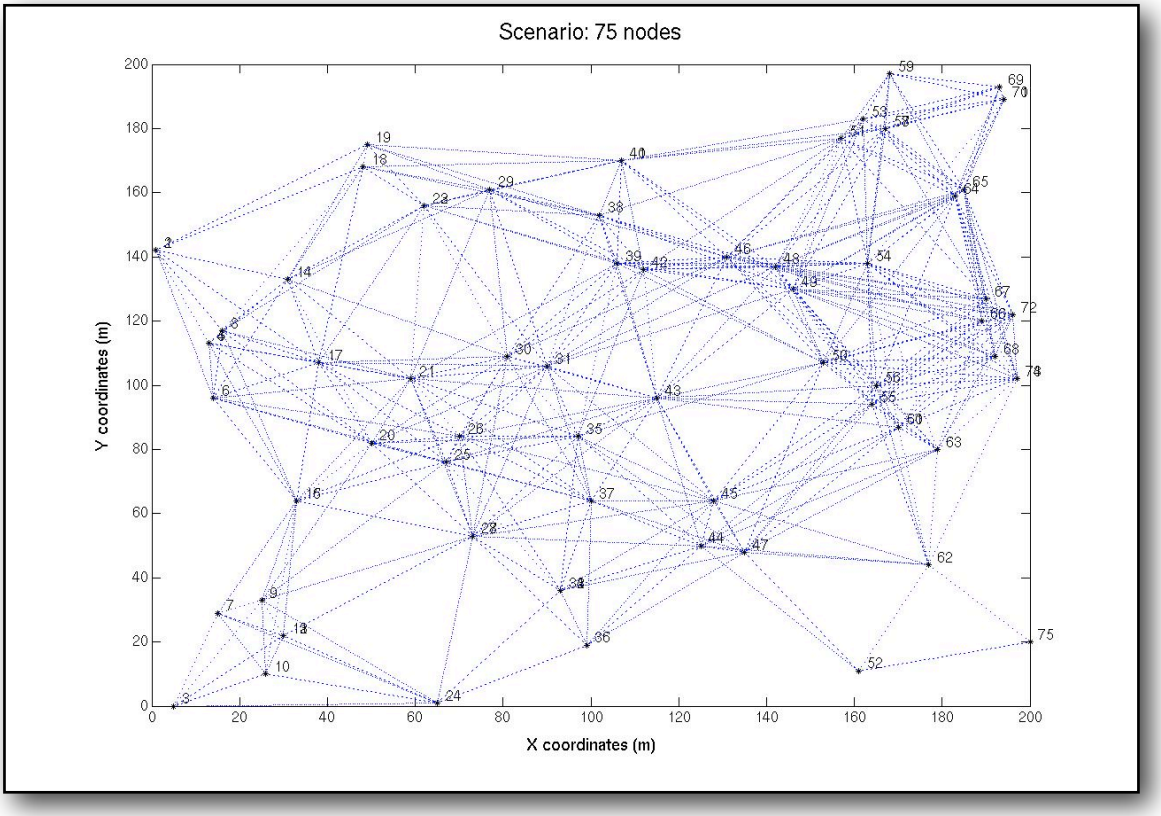


Figure B.12: Topology 2 with 75 nodes

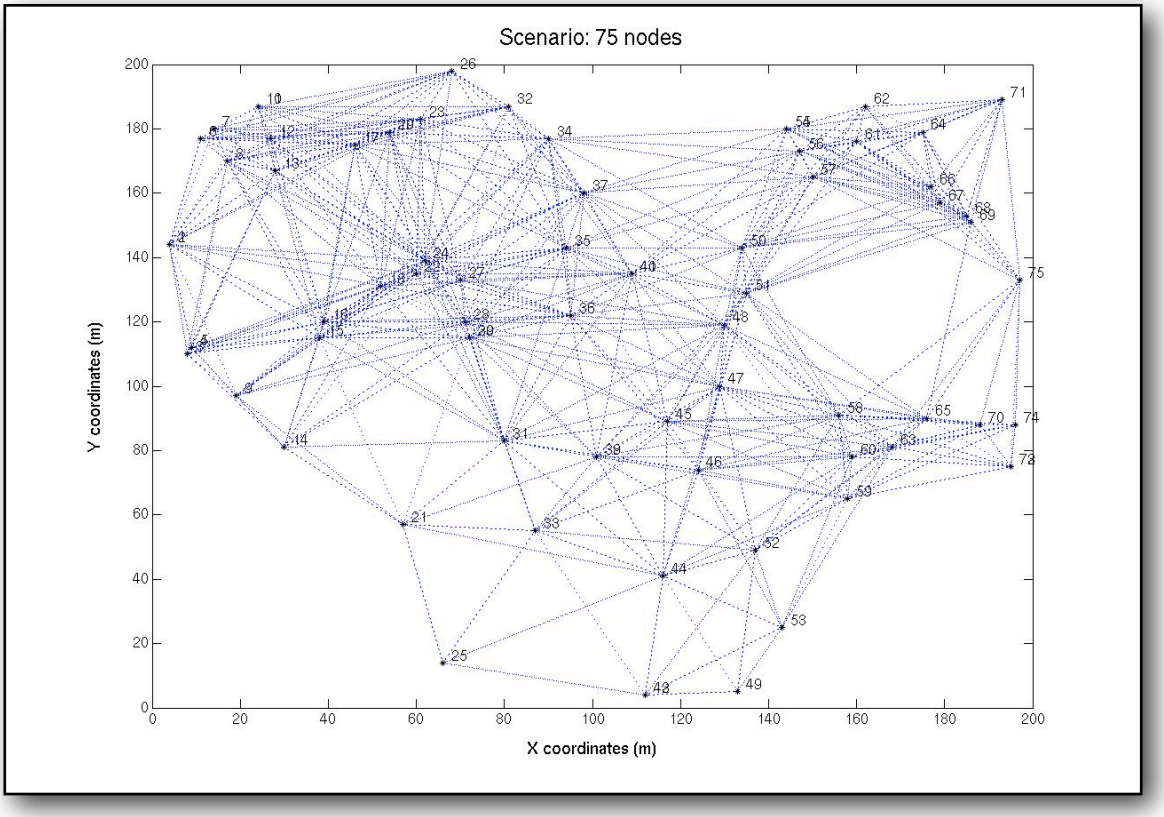


Figure B.13: Topology 3 with 75 nodes

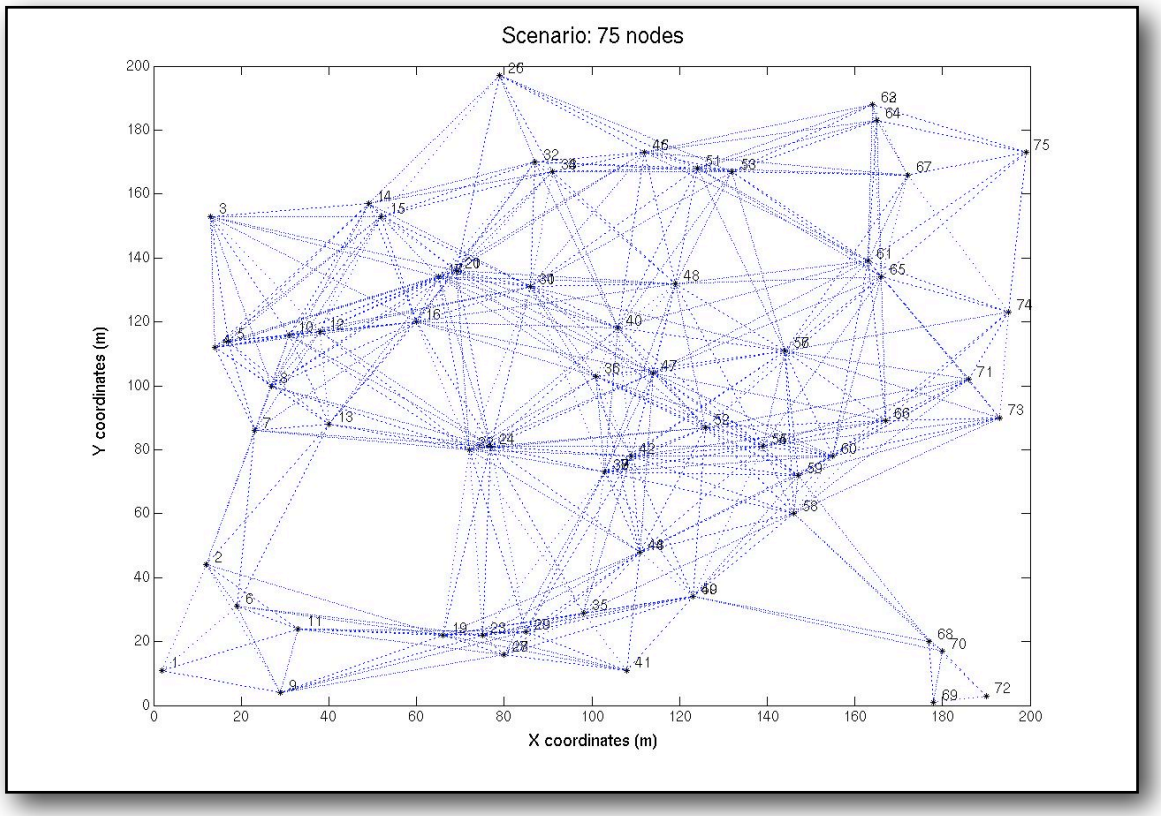


Figure B.14: Topology 4 with 75 nodes

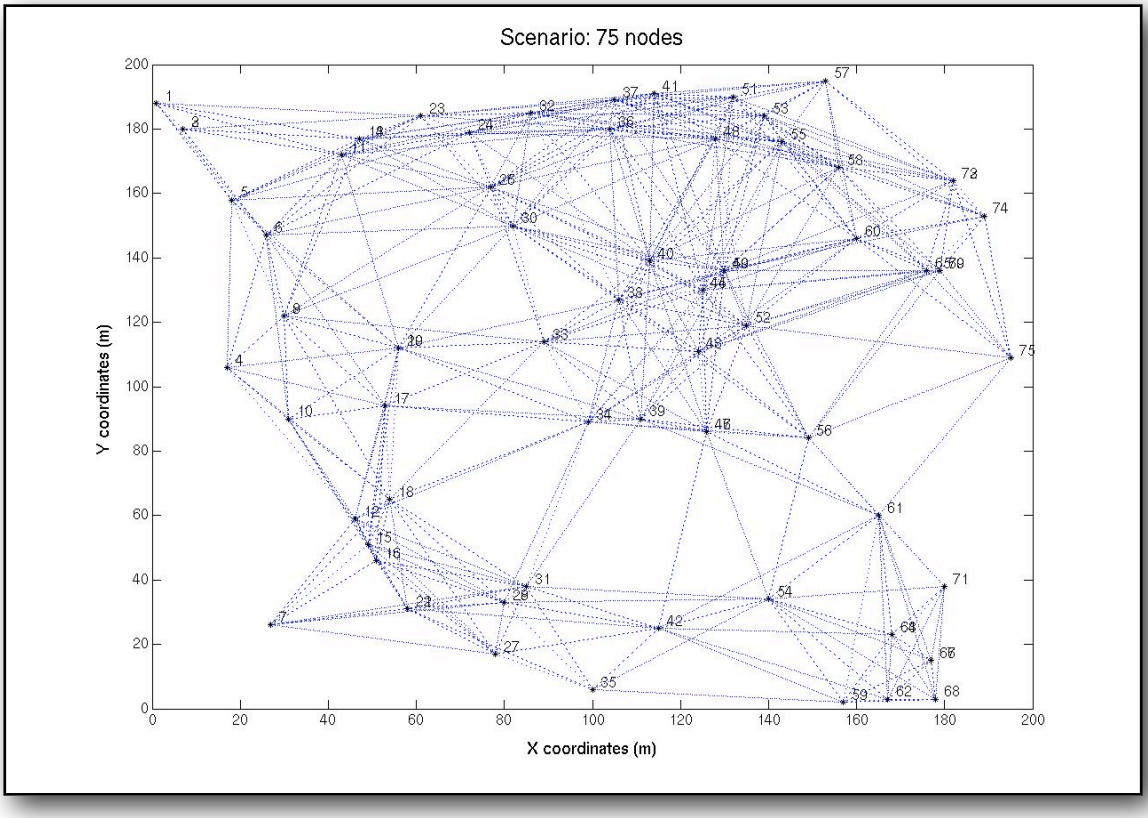


Figure B.15: Topology 5 with 75 nodes

B.4 50 nodes

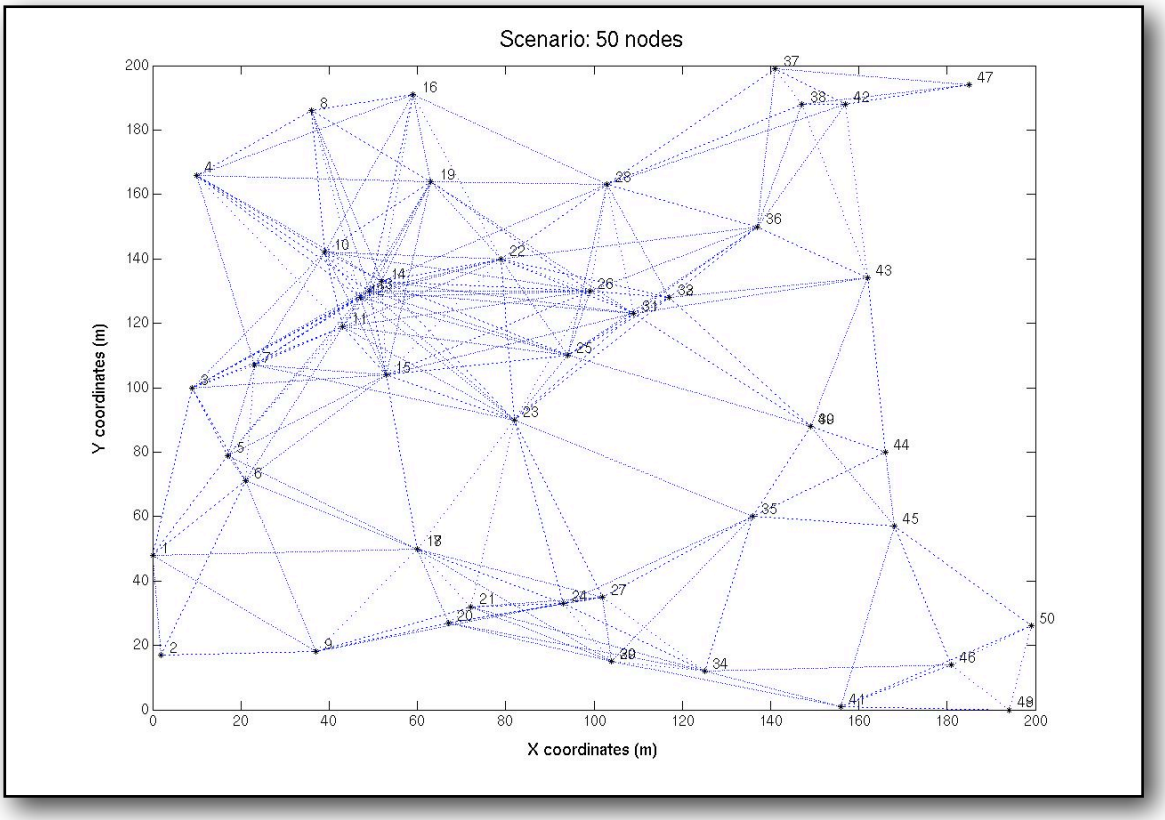


Figure B.16: Topology 1 with 50 nodes

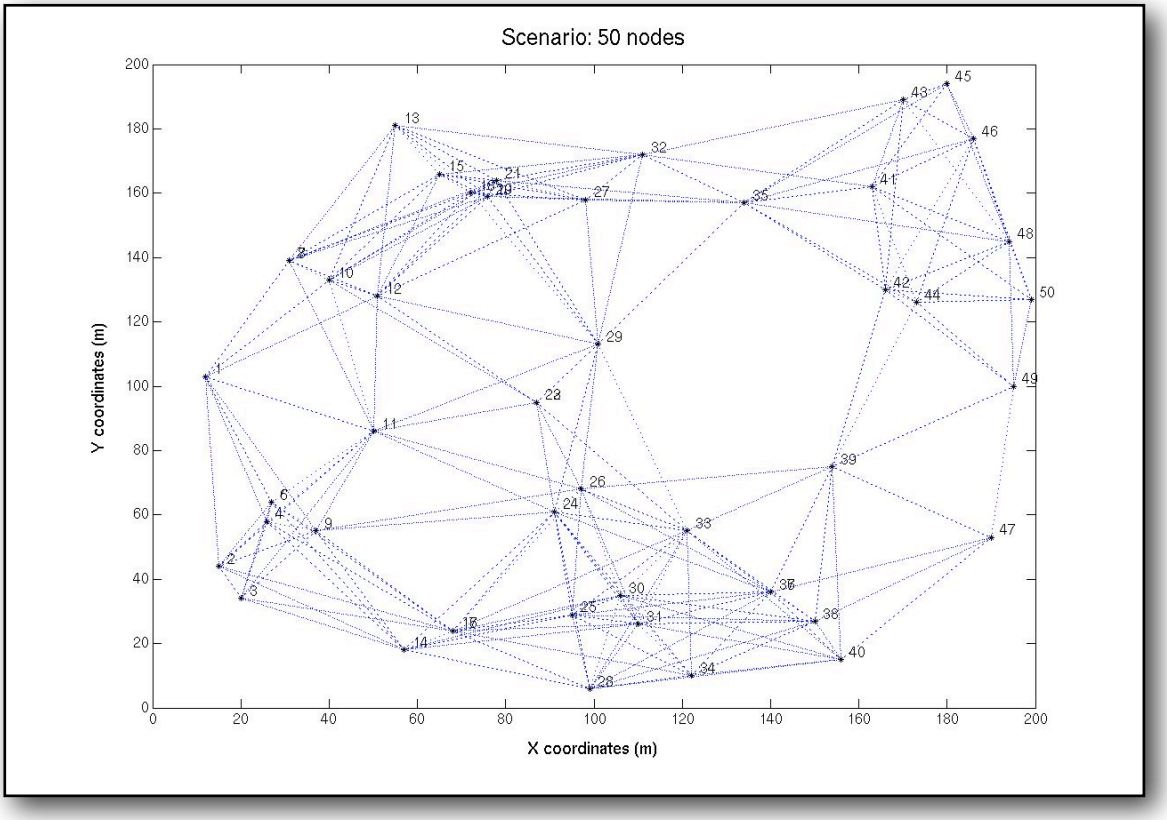


Figure B.17: Topology 2 with 50 nodes

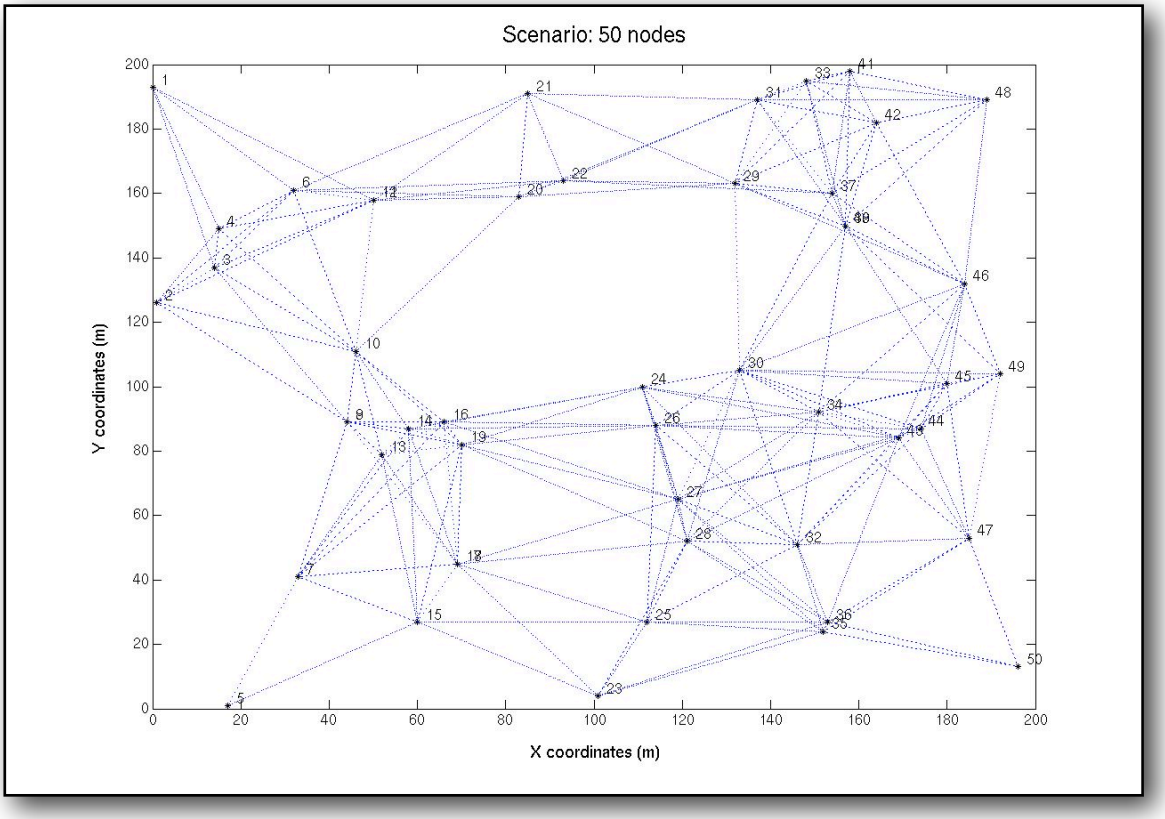


Figure B.18: Topology 3 with 50 nodes

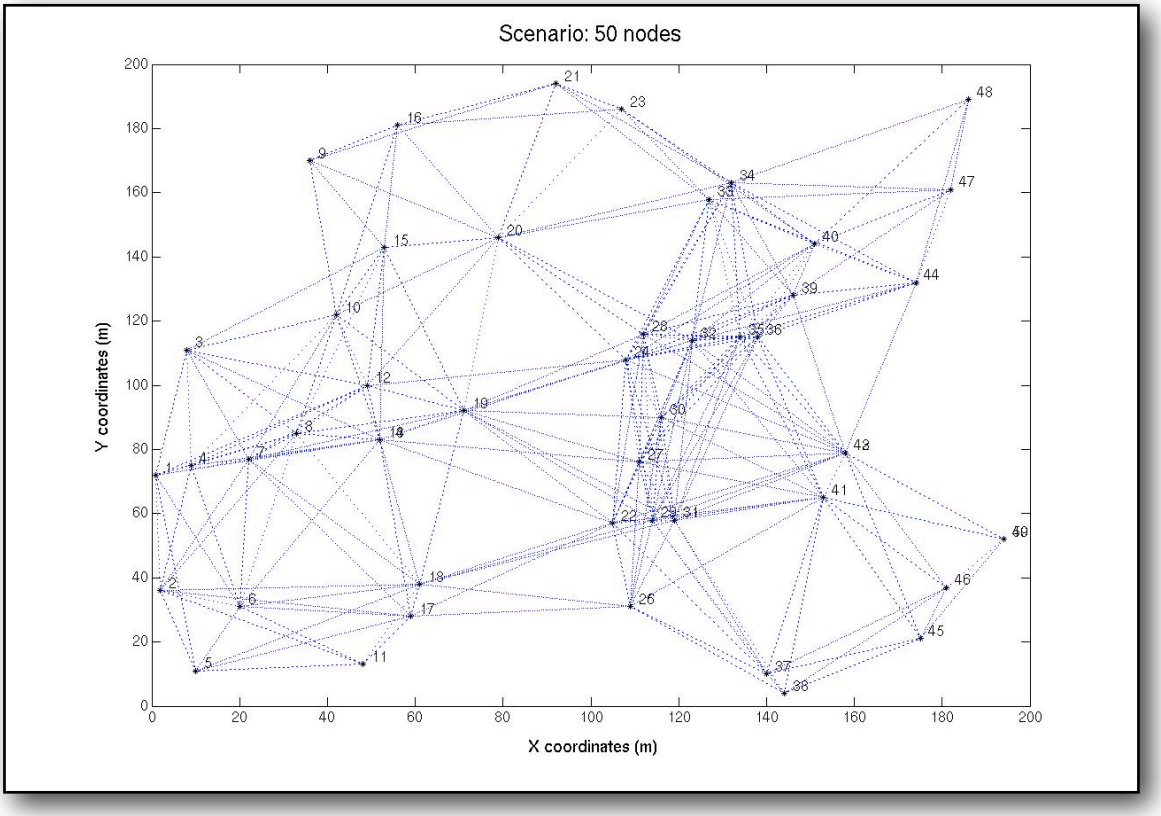


Figure B.19: Topology 4 with 50 nodes

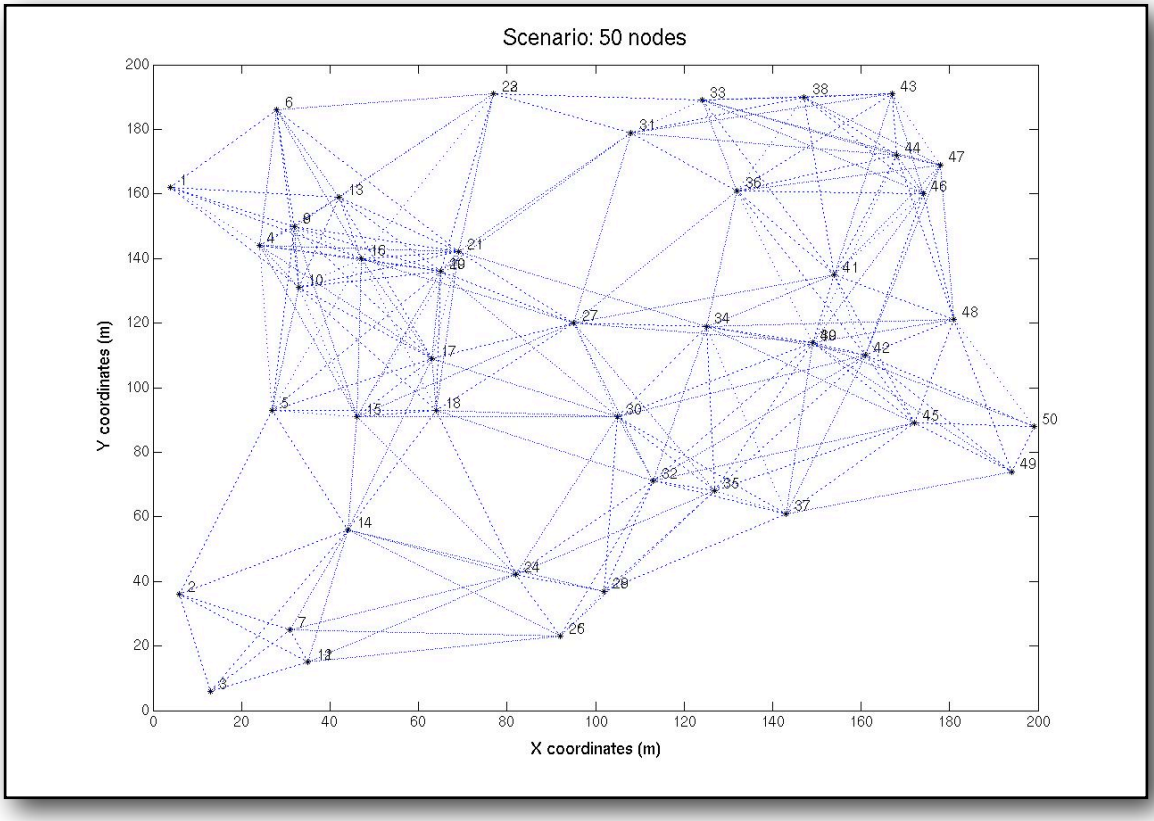


Figure B.20: Topology 5 with 50 nodes

B.5 40 nodes

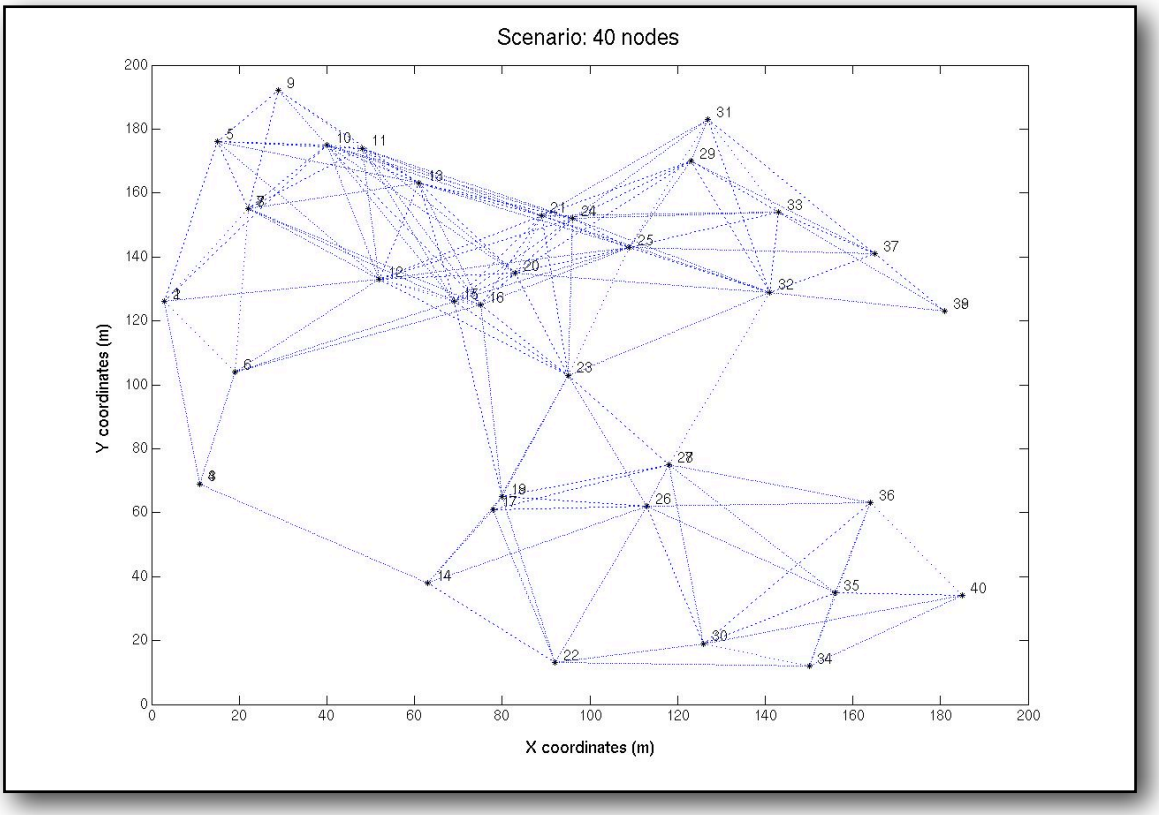


Figure B.21: Topology 1 with 40 nodes

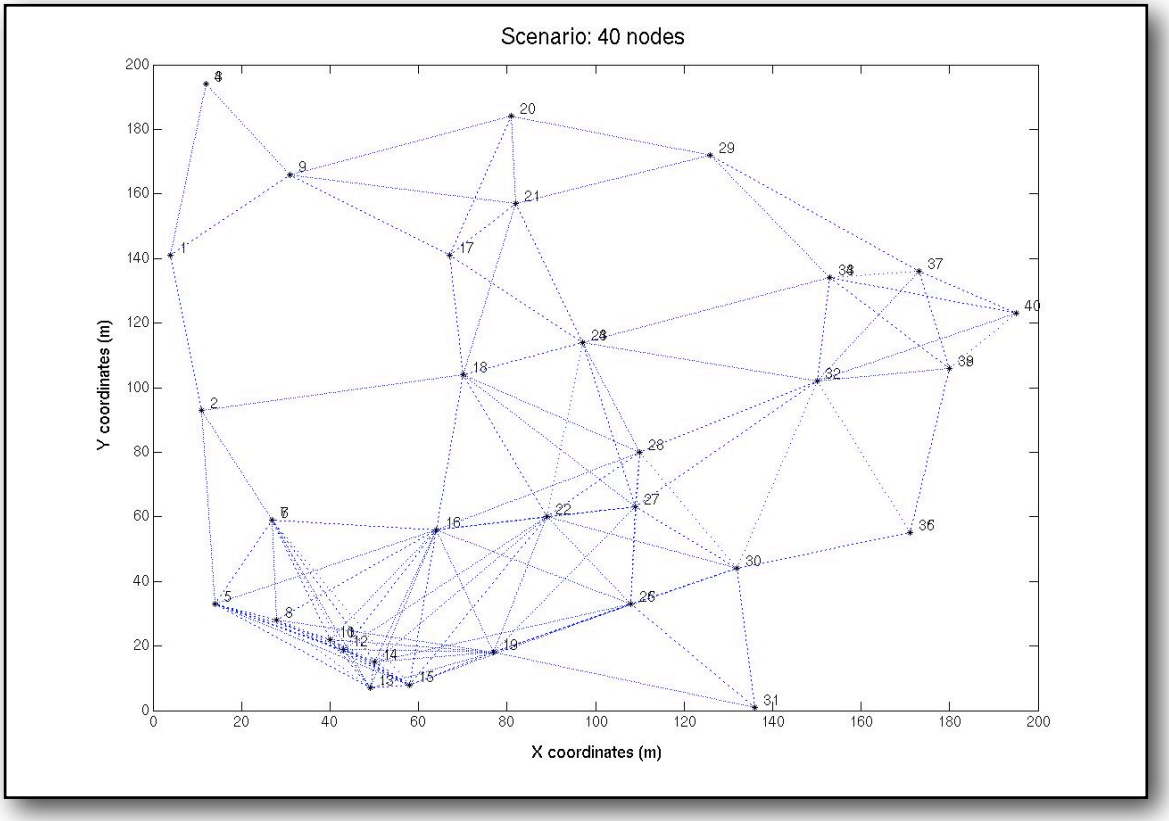


Figure B.22: Topology 2 with 40 nodes

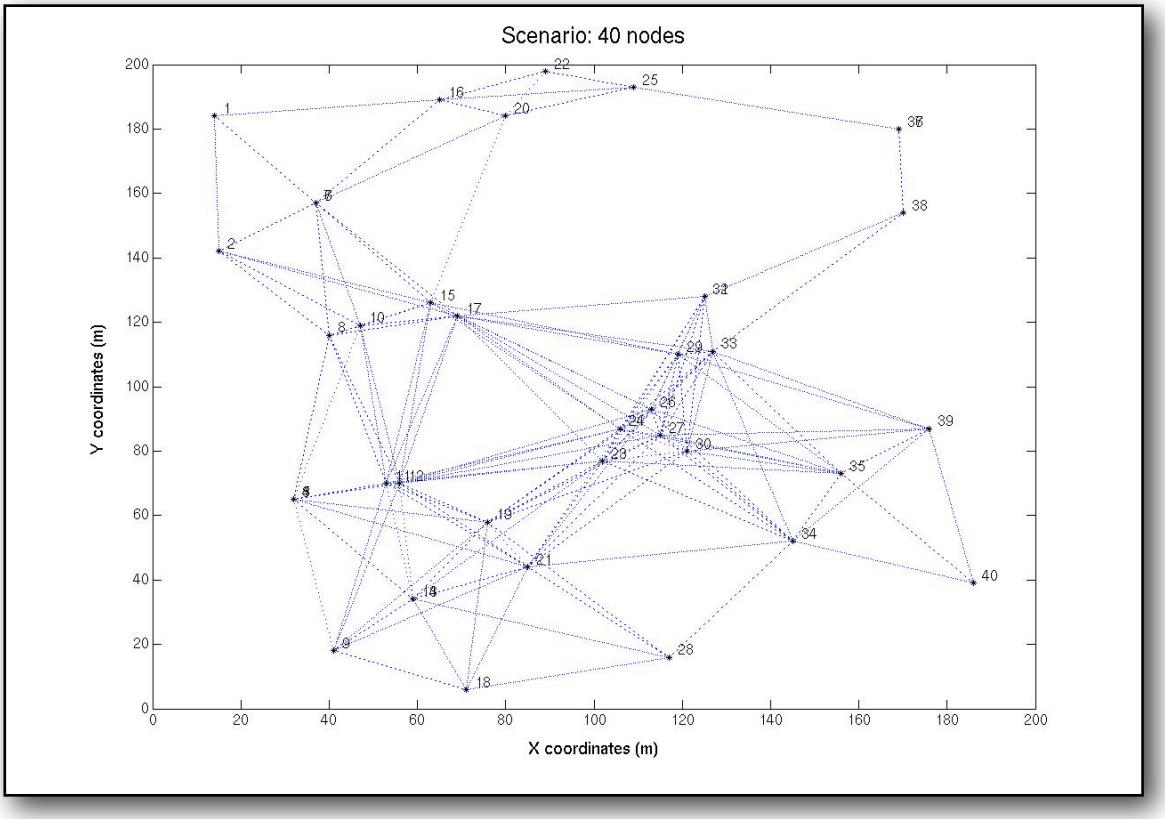


Figure B.23: Topology 3 with 40 nodes

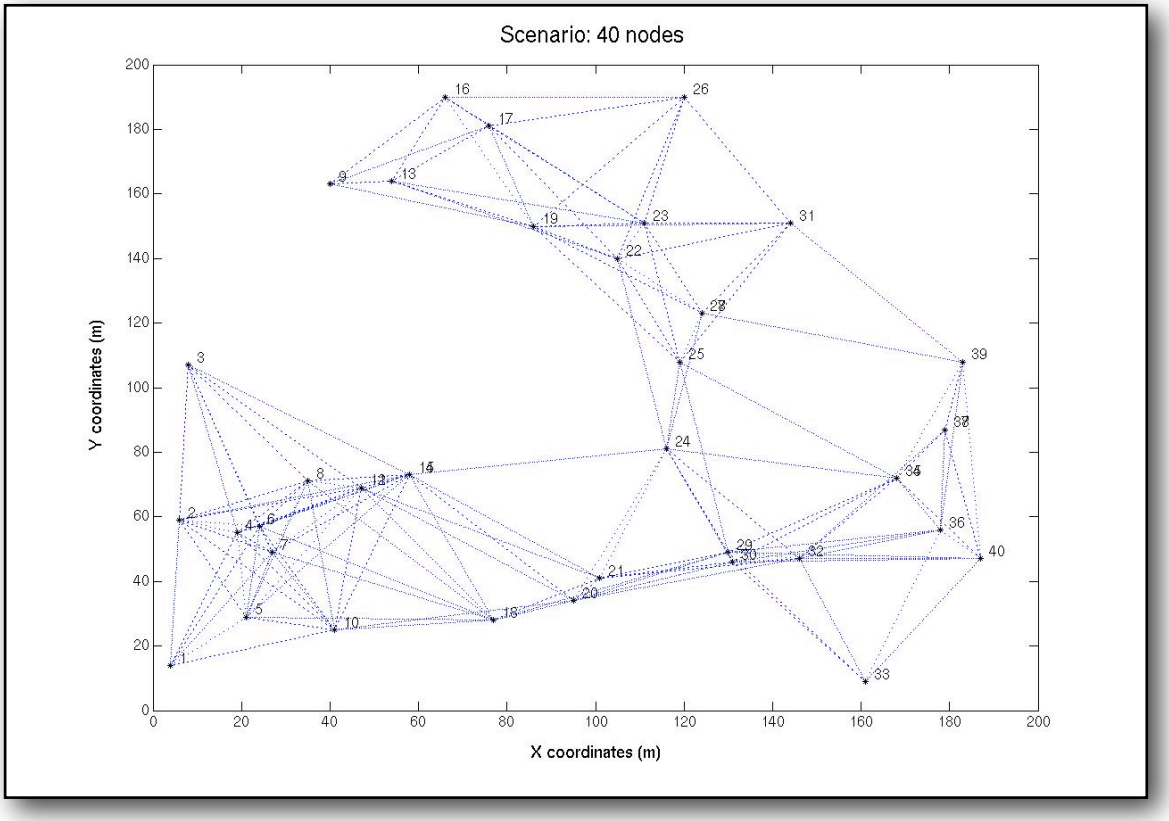


Figure B.24: Topology 4 with 40 nodes

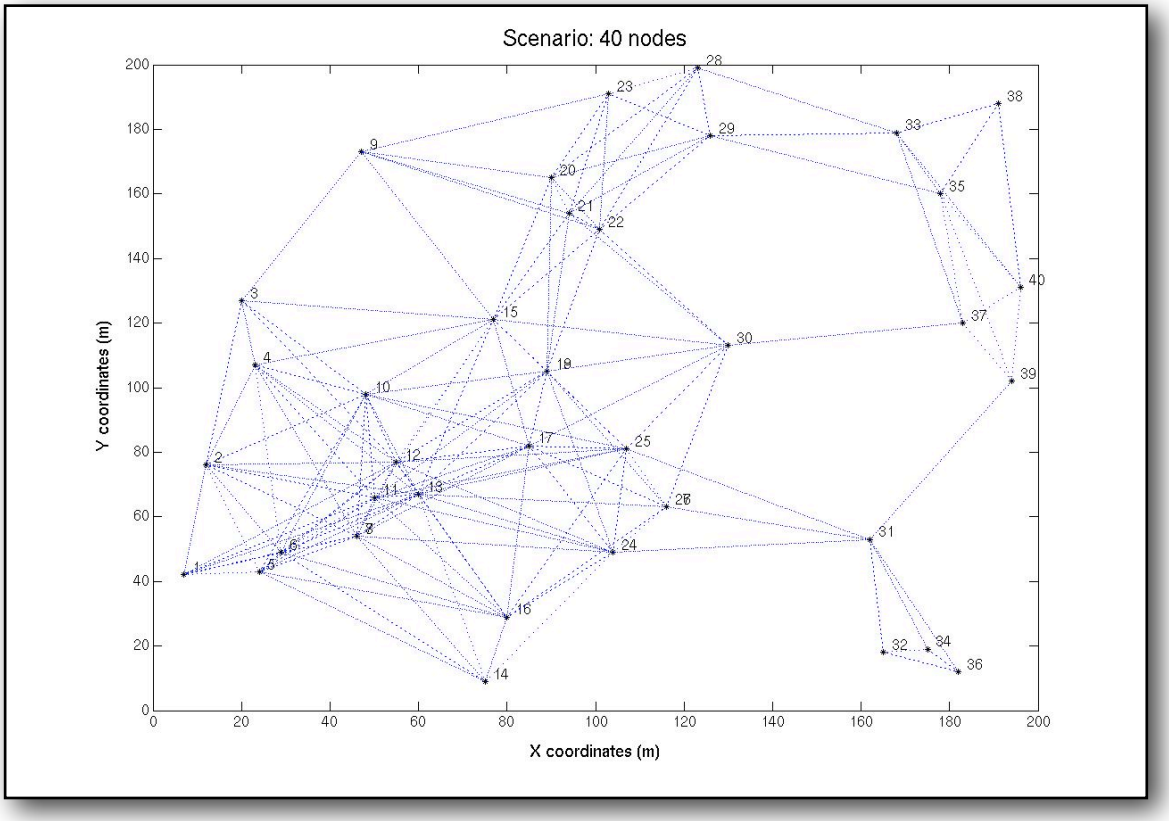


Figure B.25: Topology 5 with 40 nodes

B.6 38 nodes

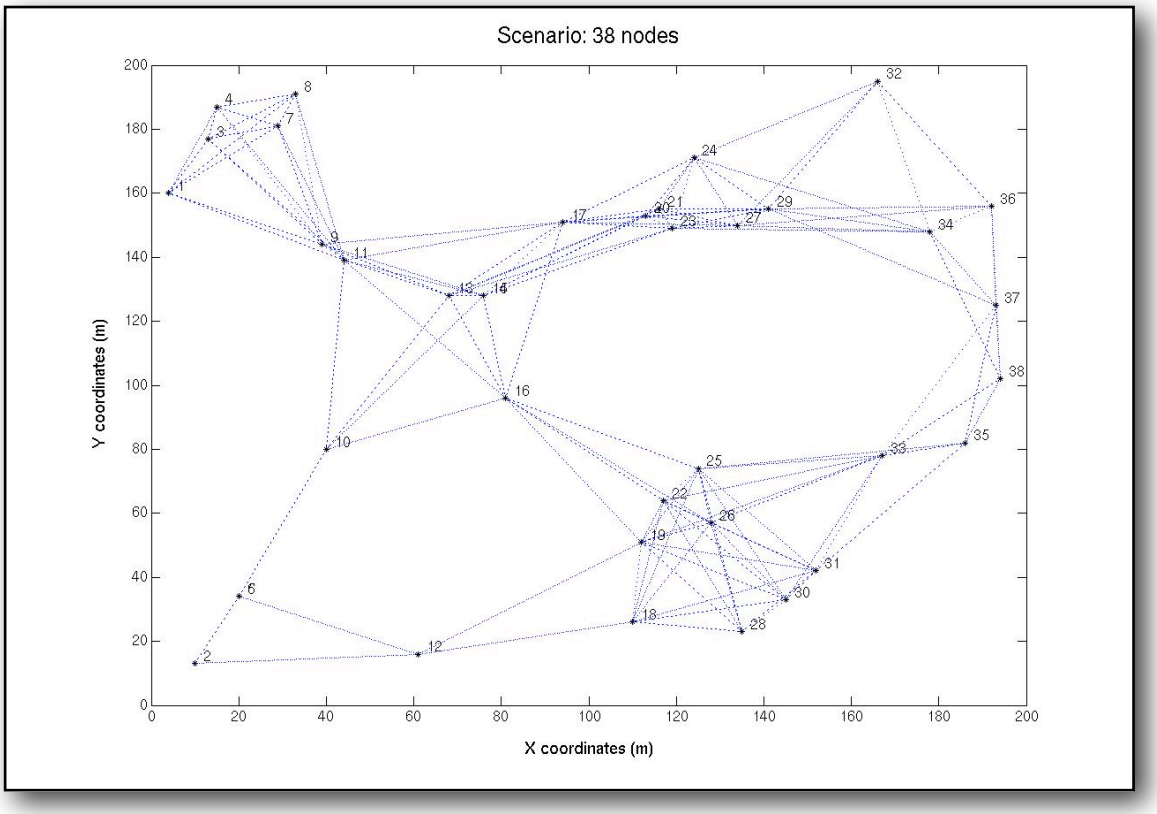


Figure B.26: Topology 1 with 38 nodes

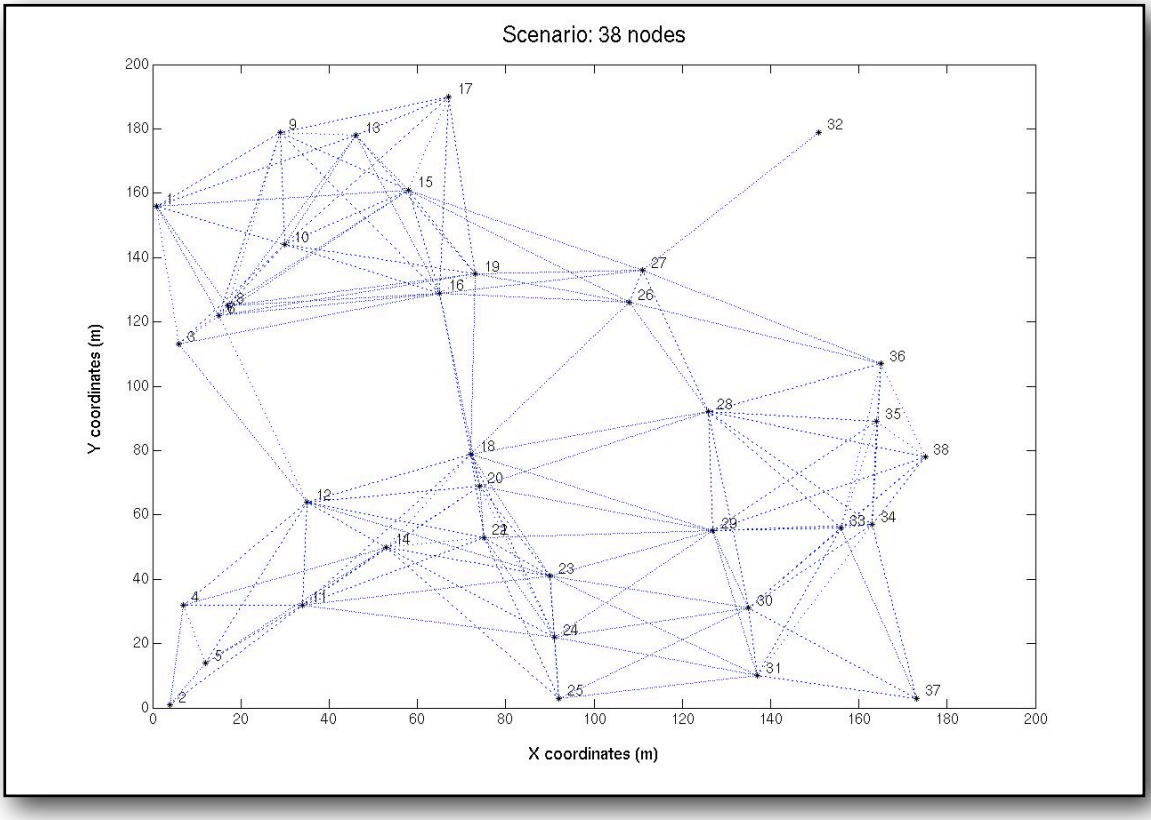


Figure B.27: Topology 2 with 38 nodes

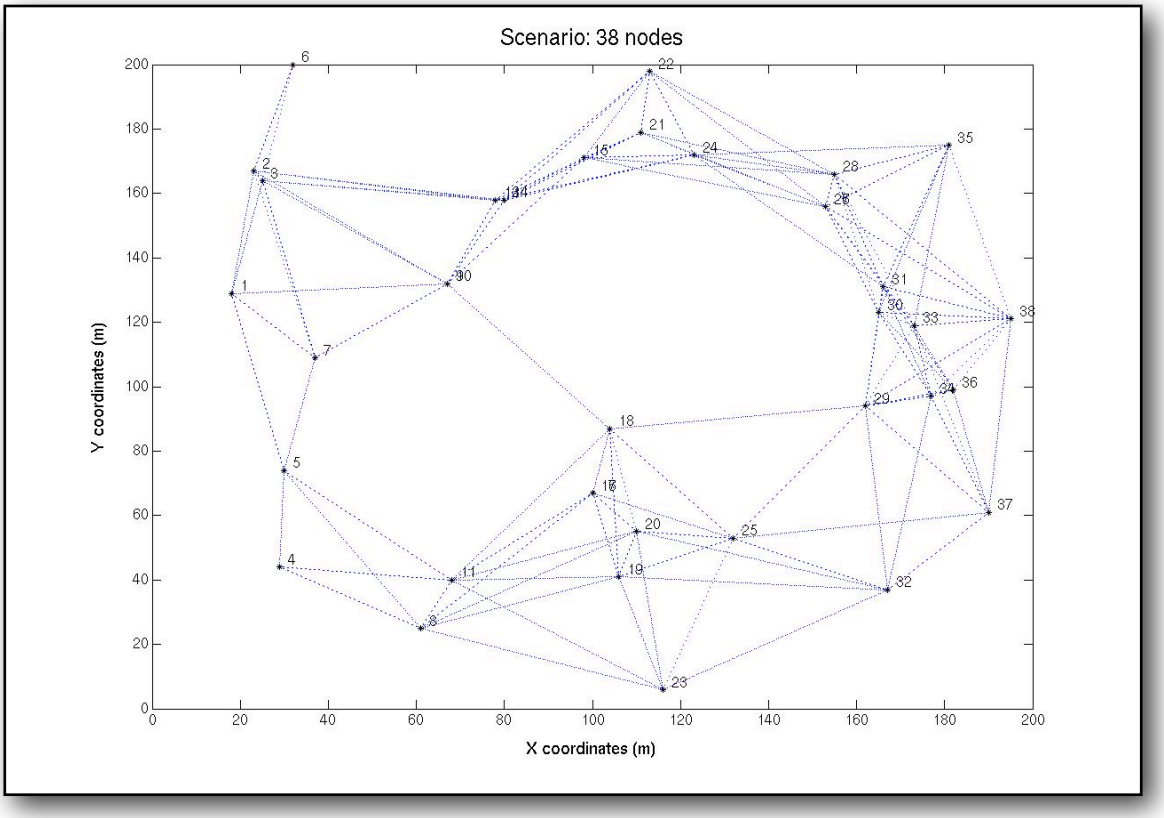


Figure B.28: Topology 3 with 38 nodes

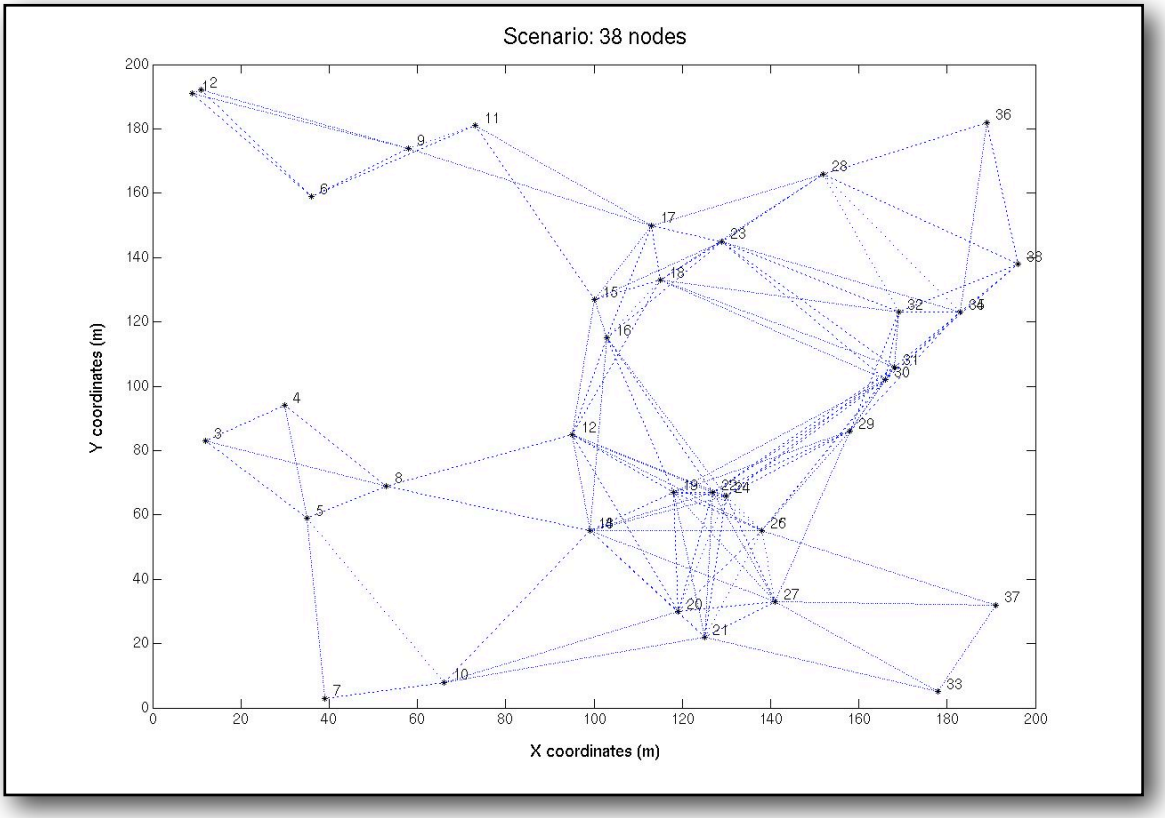


Figure B.29: Topology 4 with 38 nodes

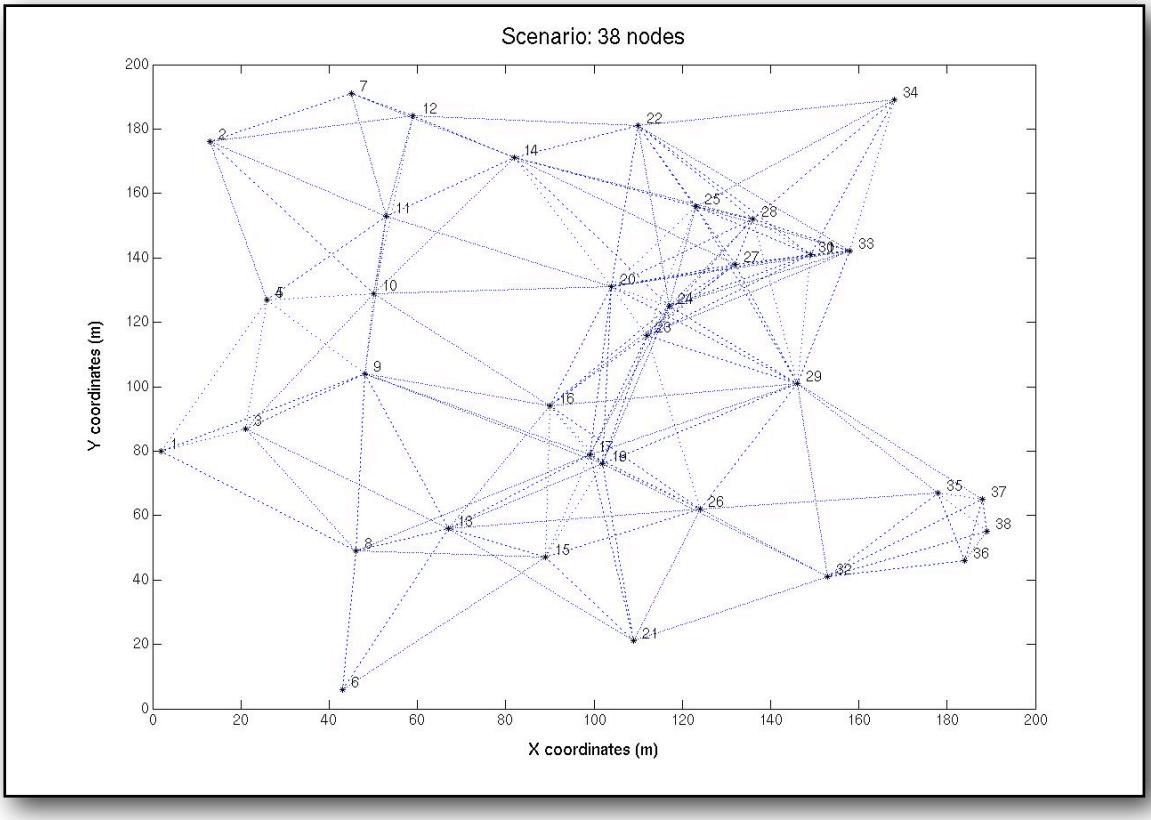


Figure B.30: Topology 5 with 38 nodes

B.7 35 nodes

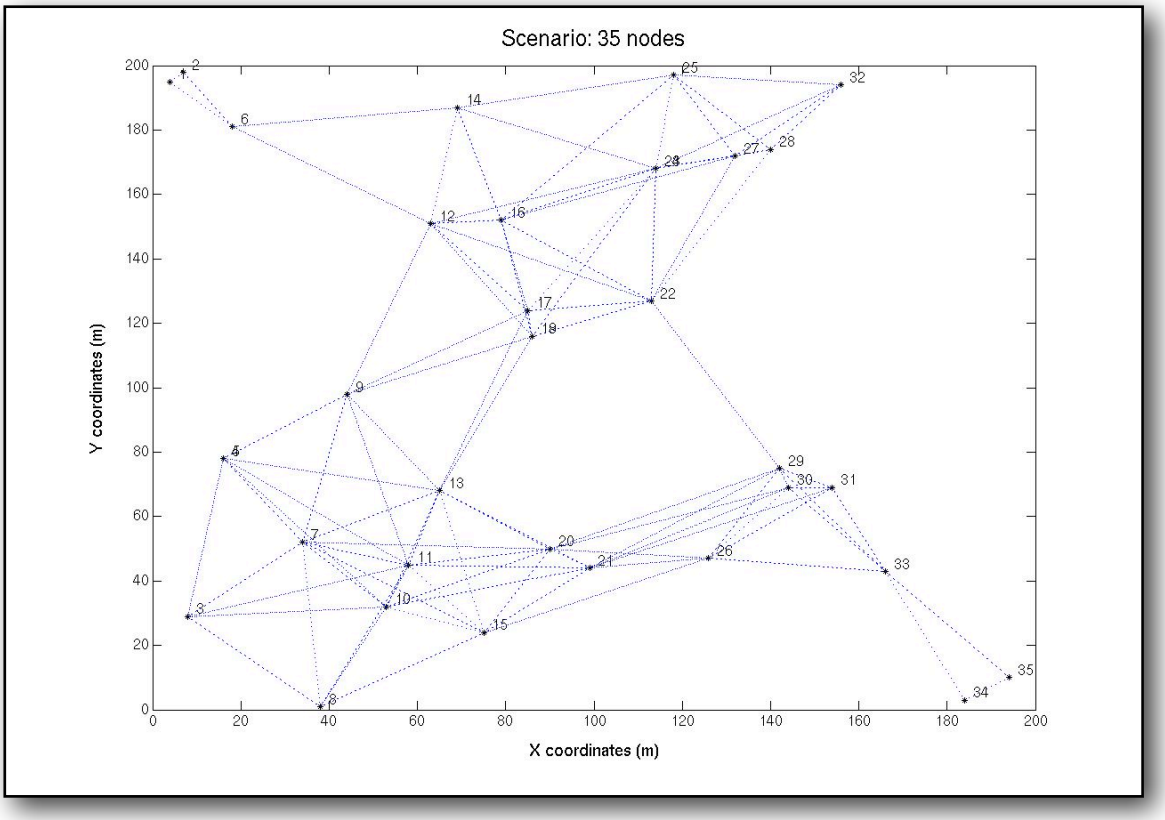


Figure B.31: Topology 1 with 35 nodes

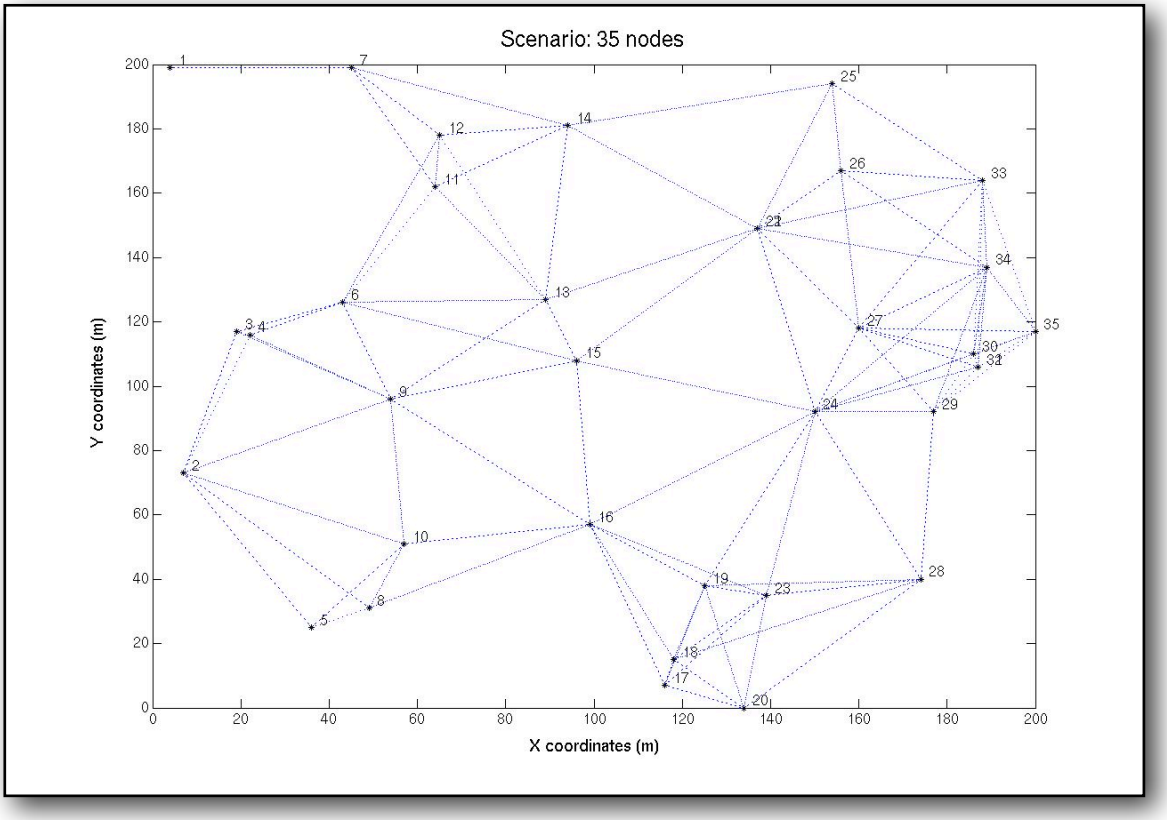


Figure B.32: Topology 2 with 35 nodes

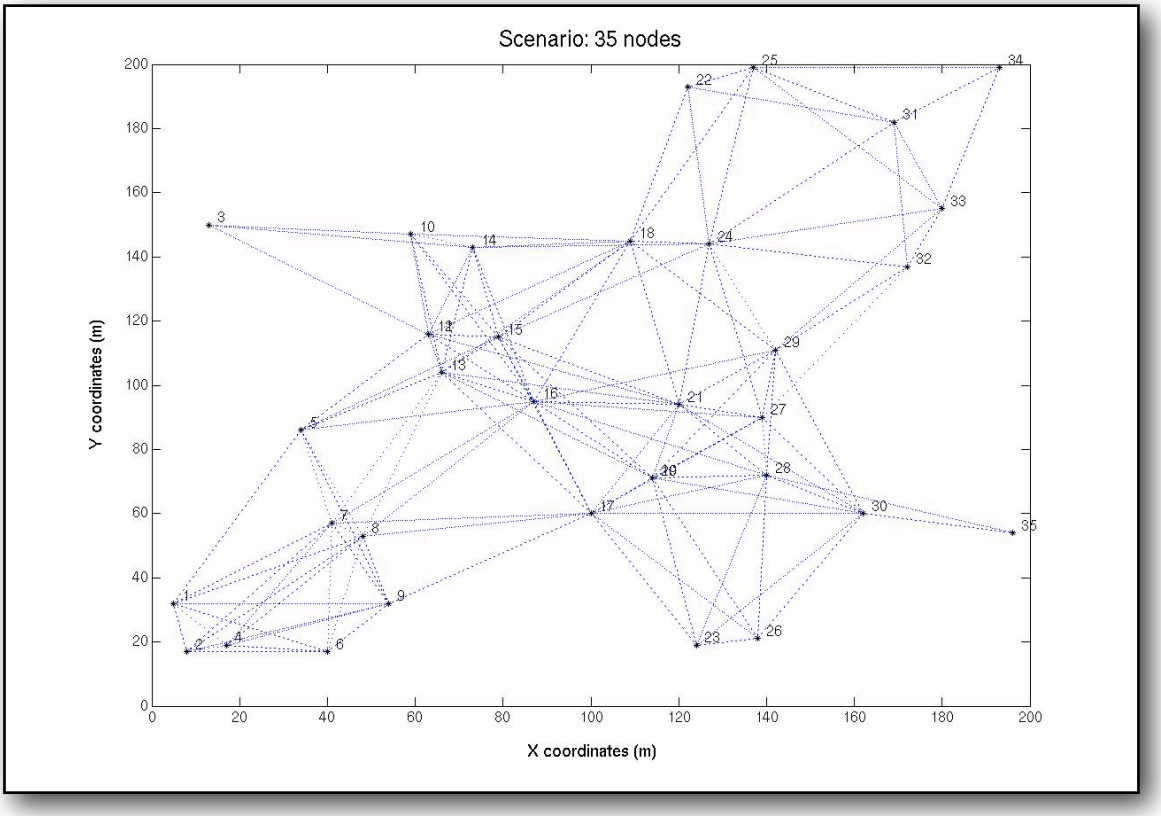


Figure B.33: Topology 3 with 35 nodes

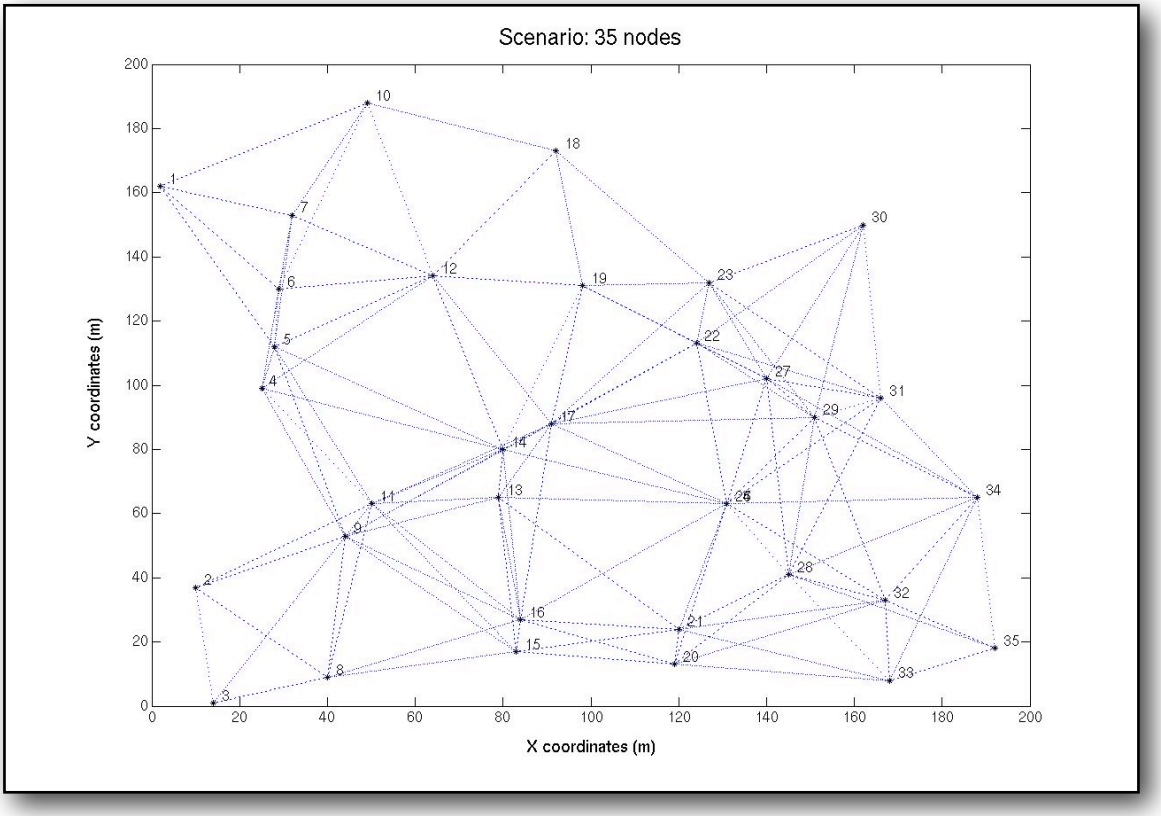


Figure B.34: Topology 4 with 35 nodes

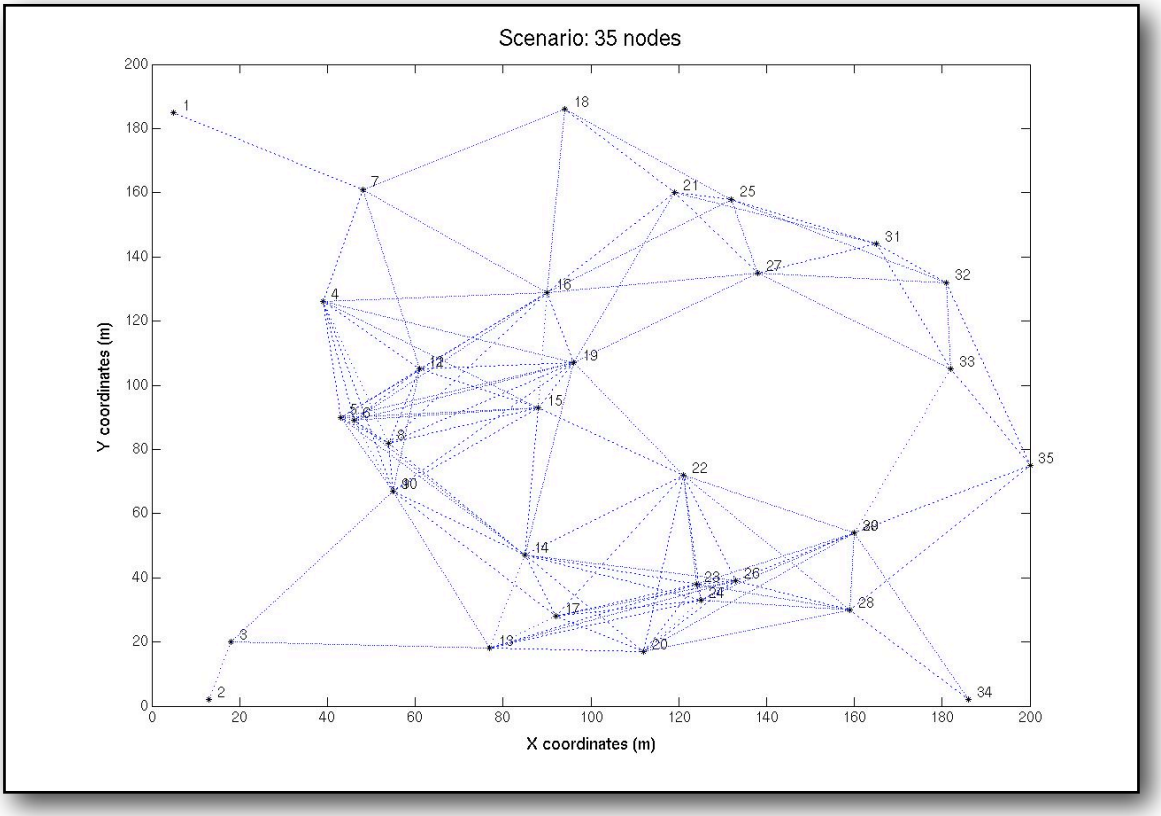


Figure B.35: Topology 5 with 35 nodes

APPENDIX C

Energy and Refractive Index Distribution Screenshots

C.1 50-node scenario

C.1.1 Energy distribution

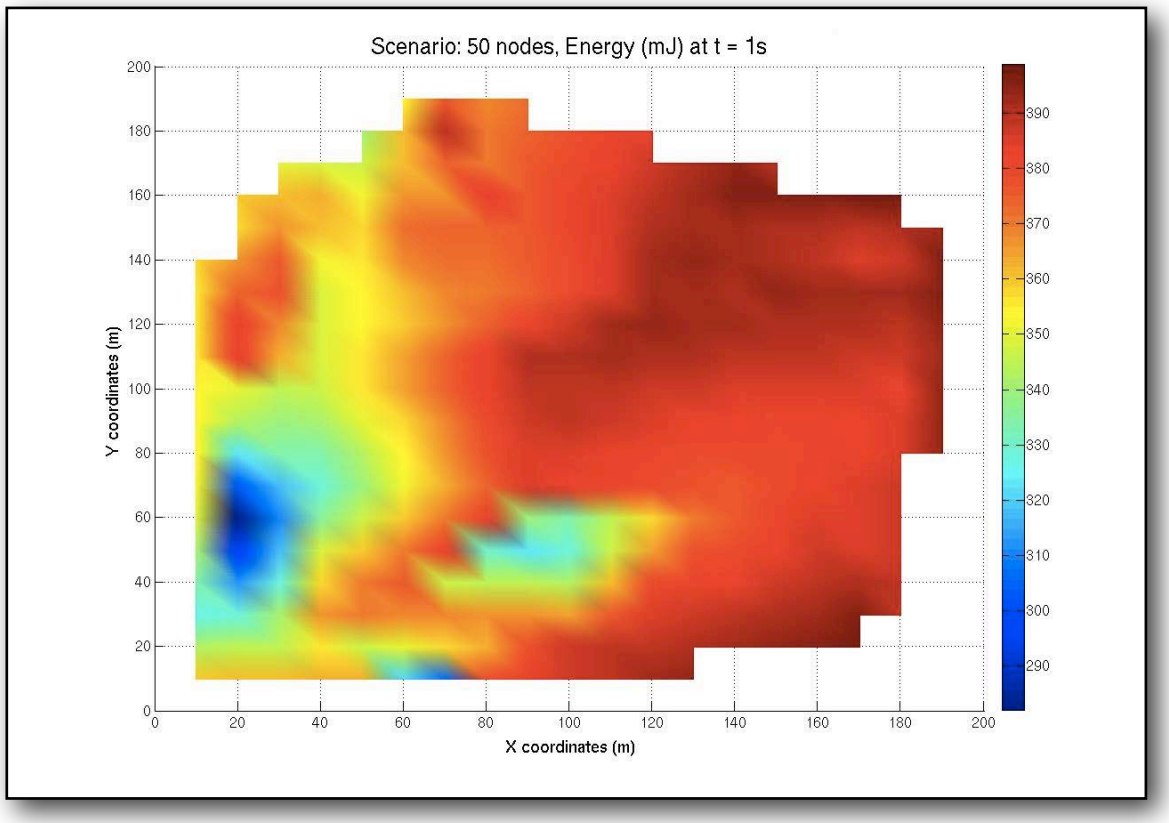


Figure C.1: Energy distribution at 1 simulation seconds - 50 nodes

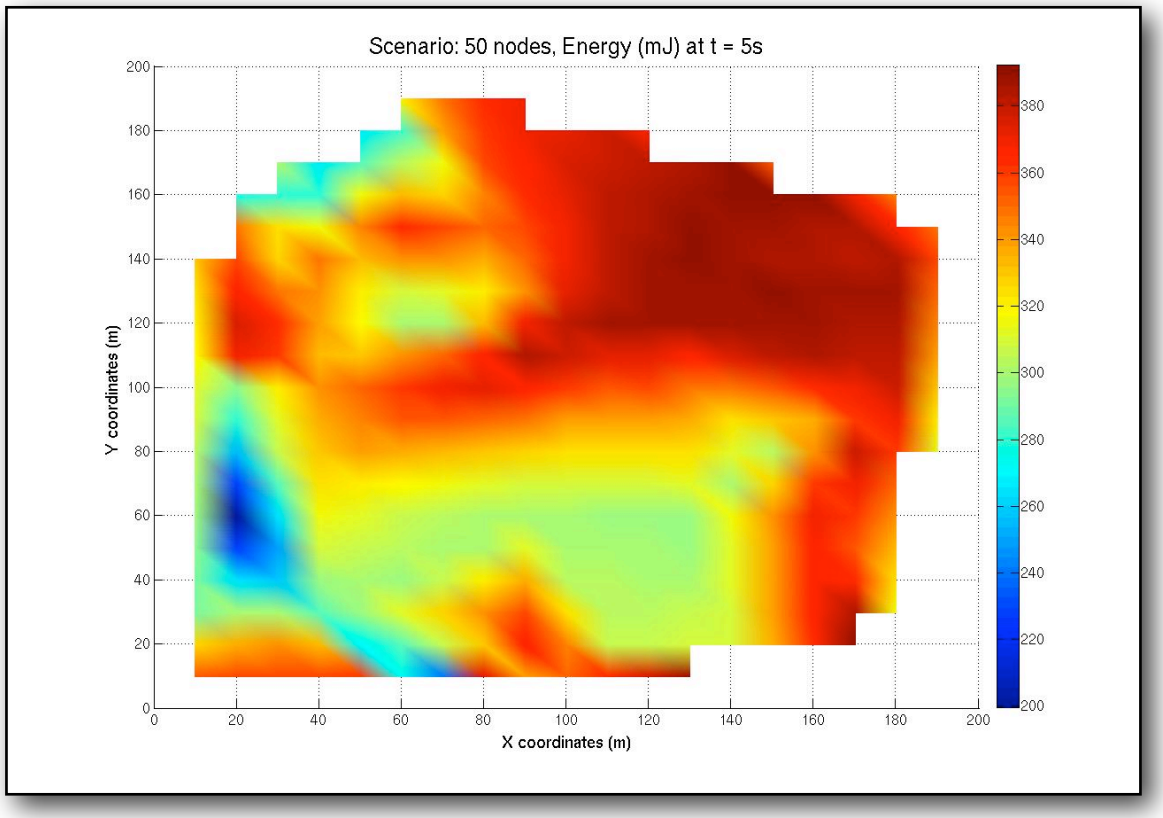


Figure C.2: Energy distribution at 5 simulation seconds - 50 nodes

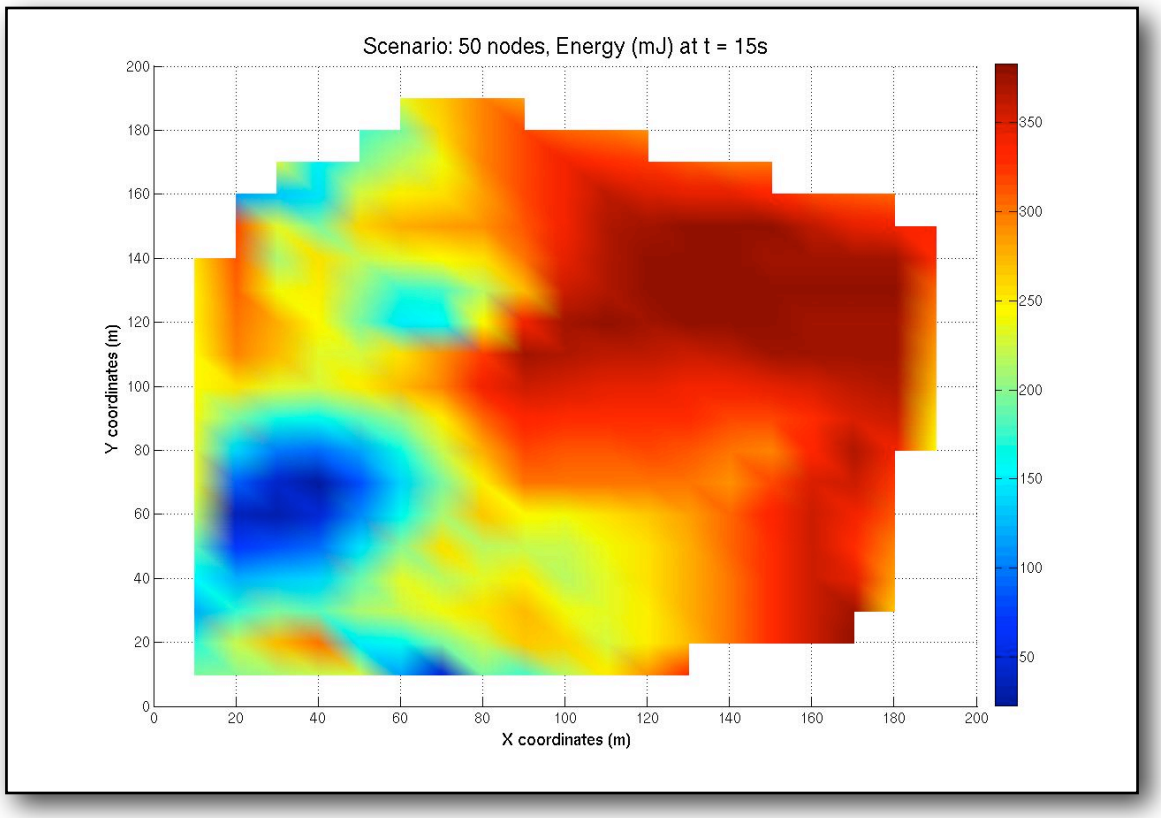


Figure C.3: Energy distribution at 15 simulation seconds - 50 nodes

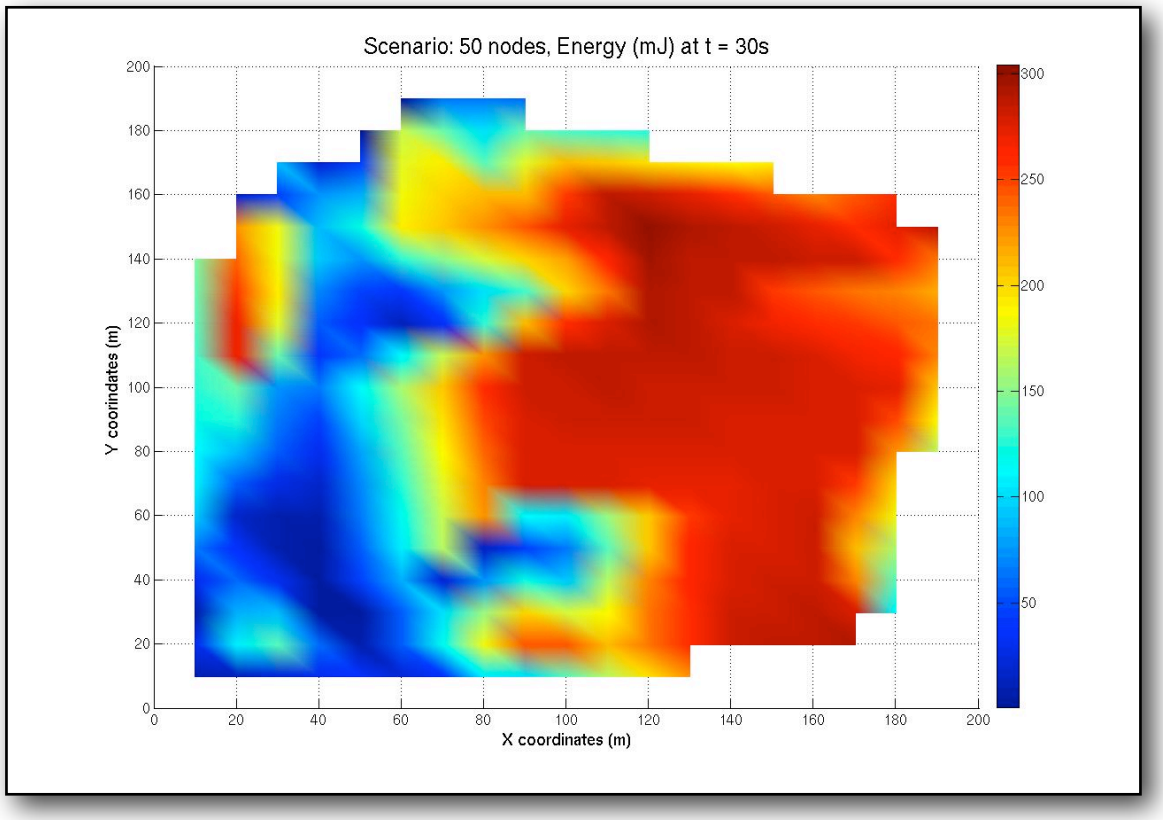


Figure C.4: Energy distribution at 30 simulation seconds - 50 nodes

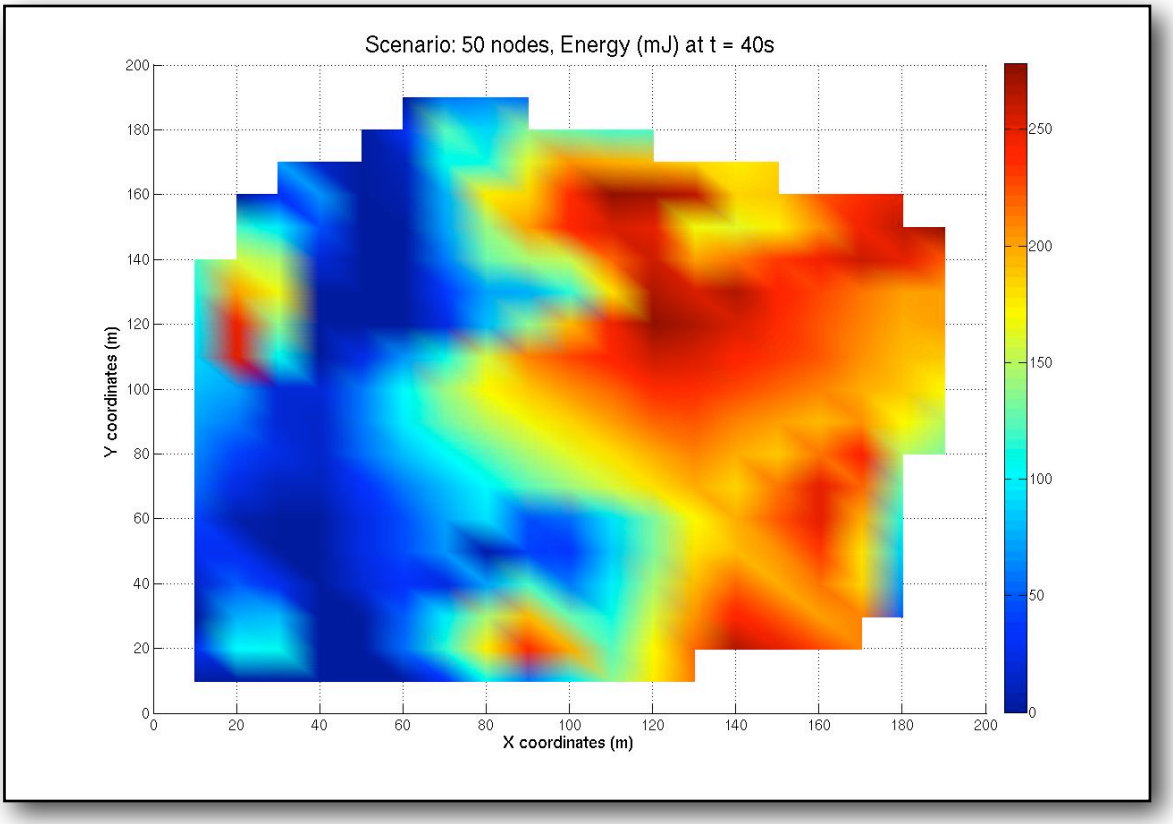


Figure C.5: Energy distribution at 40 simulation seconds - 50 nodes

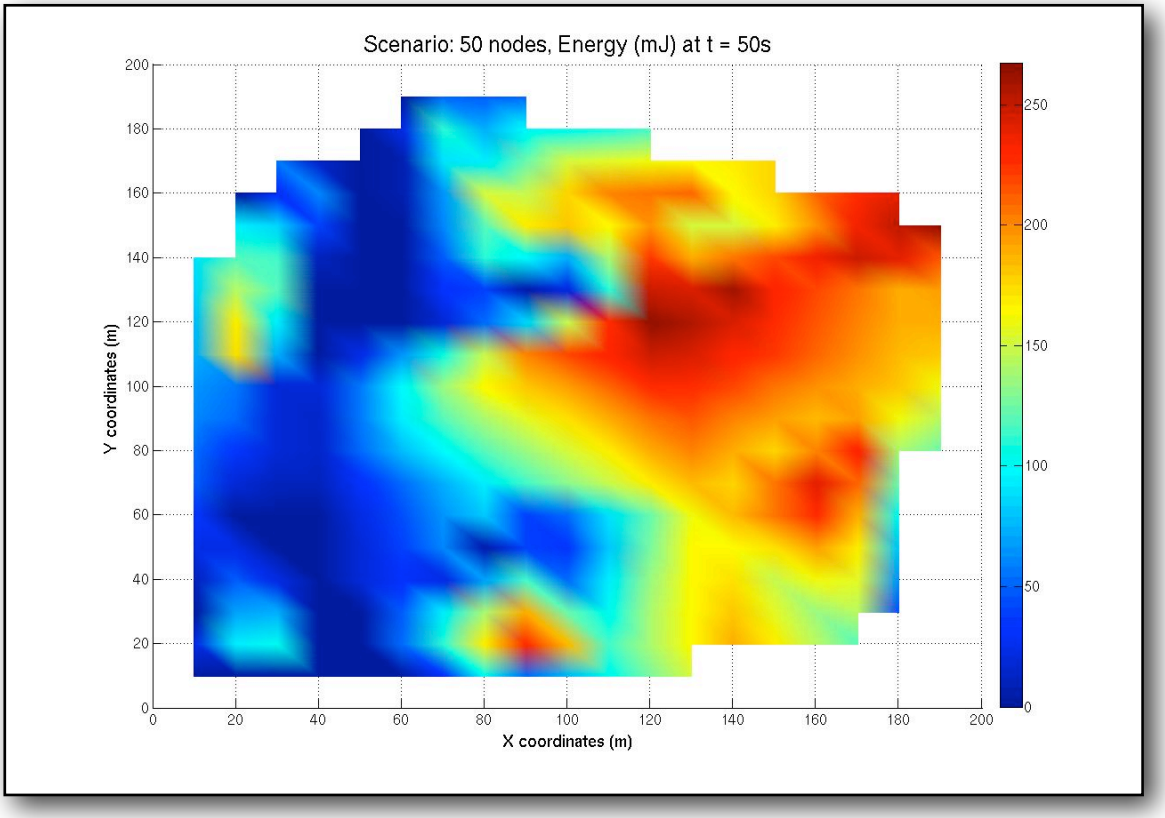


Figure C.6: Energy distribution at 50 simulation seconds - 50 nodes

C.1.2 Refractive index distribution

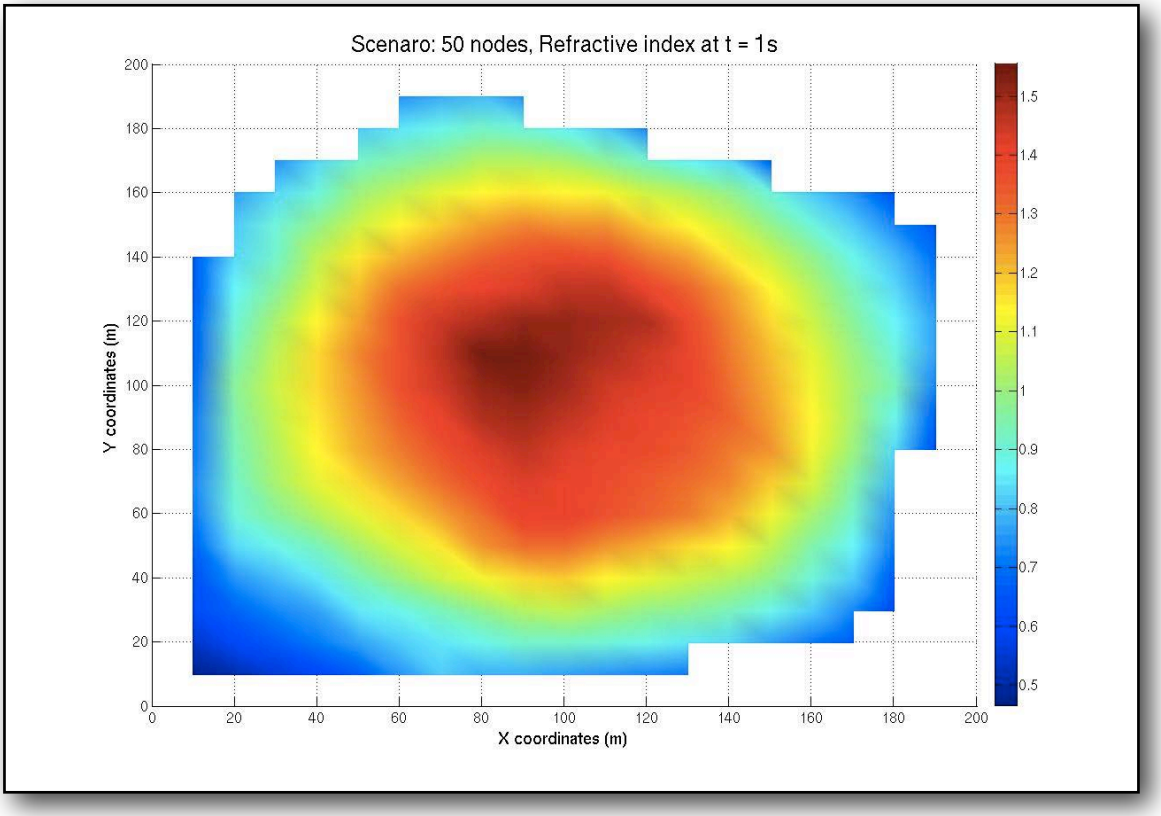


Figure C.7: Refractive index distribution at 1 simulation seconds - 50 nodes

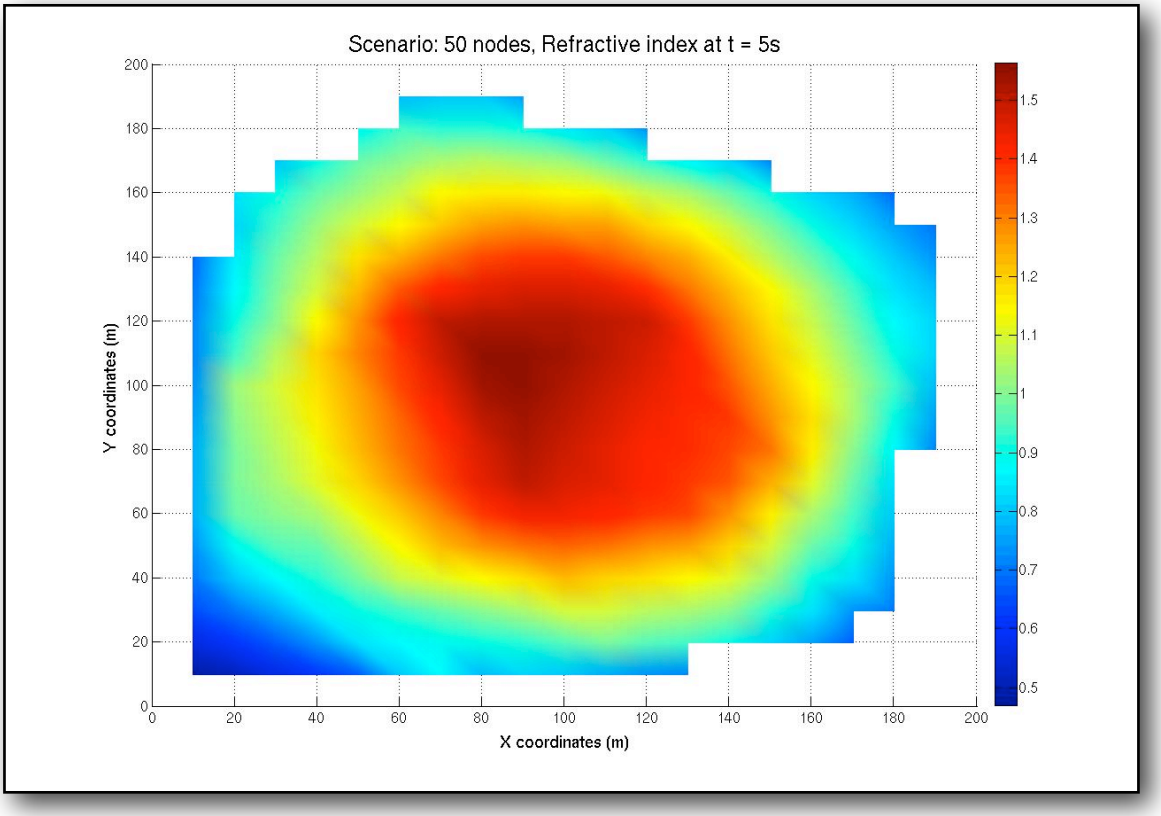


Figure C.8: Refractive index distribution at 5 simulation seconds - 50 nodes

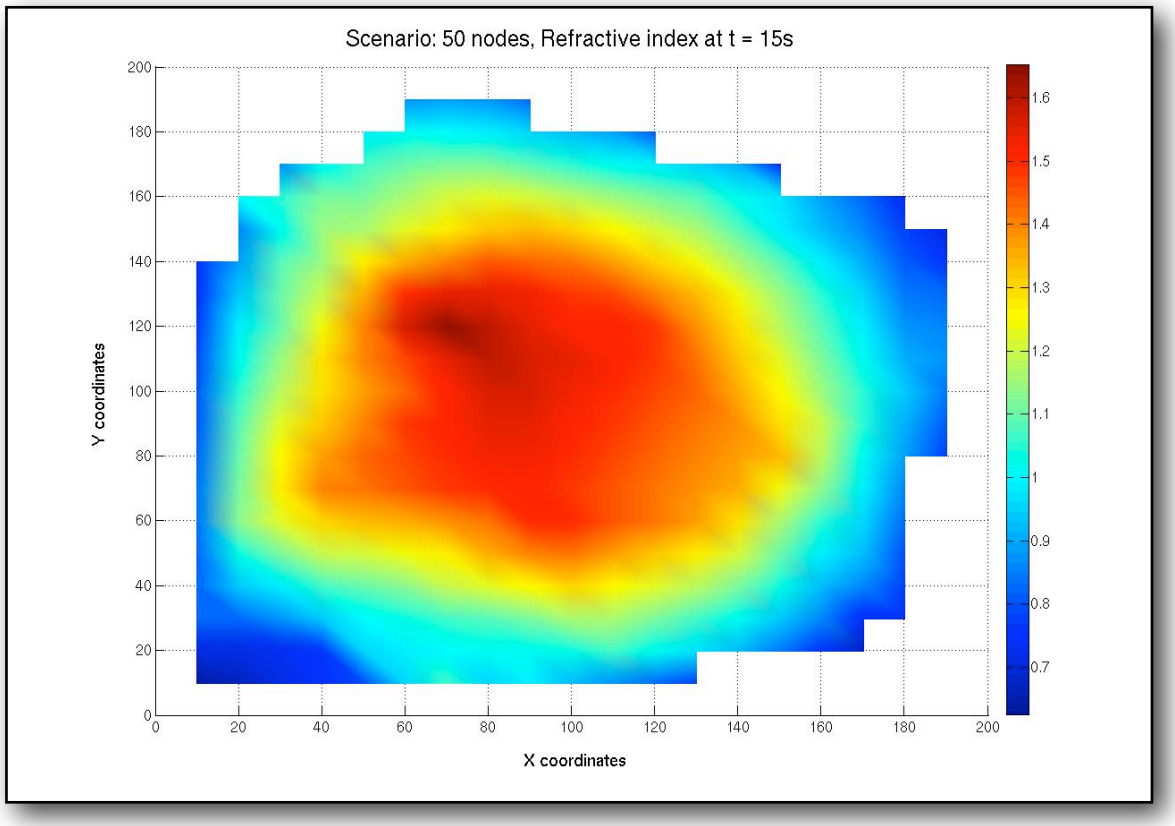


Figure C.9: Refractive index distribution at 15 simulation seconds - 50 nodes

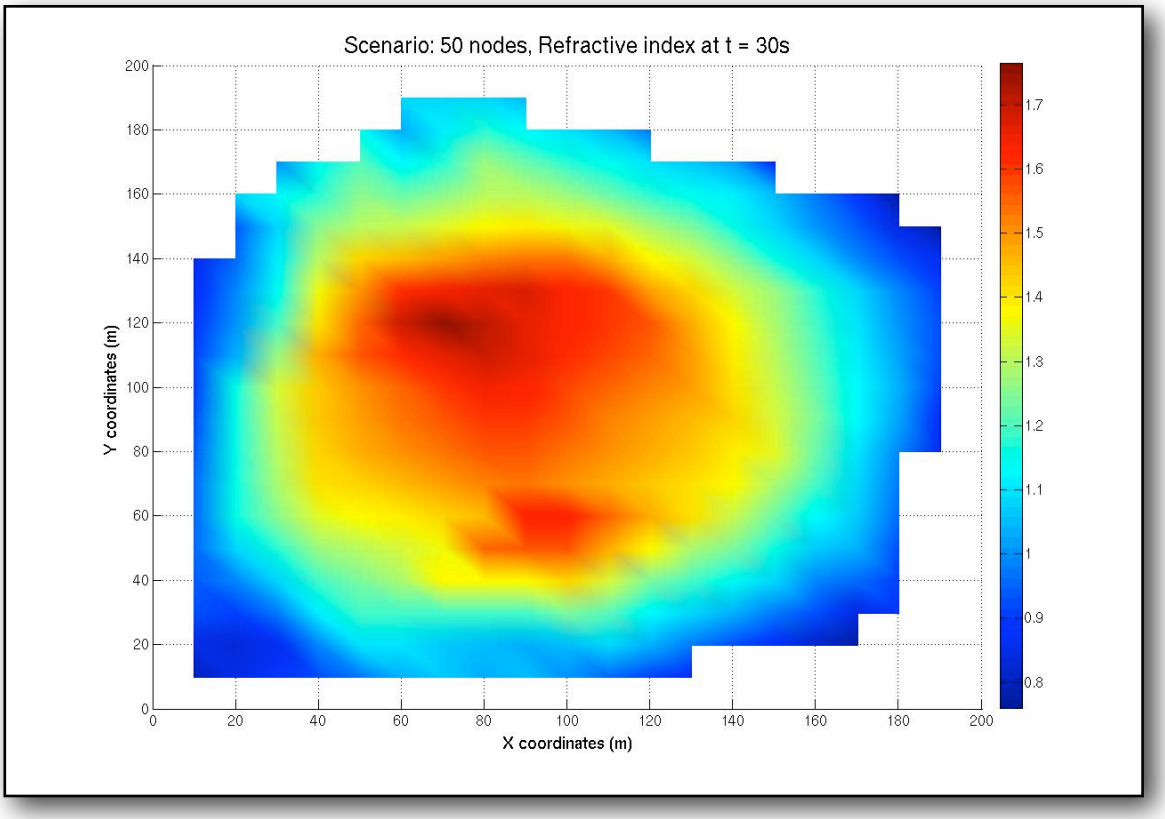


Figure C.10: Refractive index distribution at 30 simulation seconds - 50 nodes

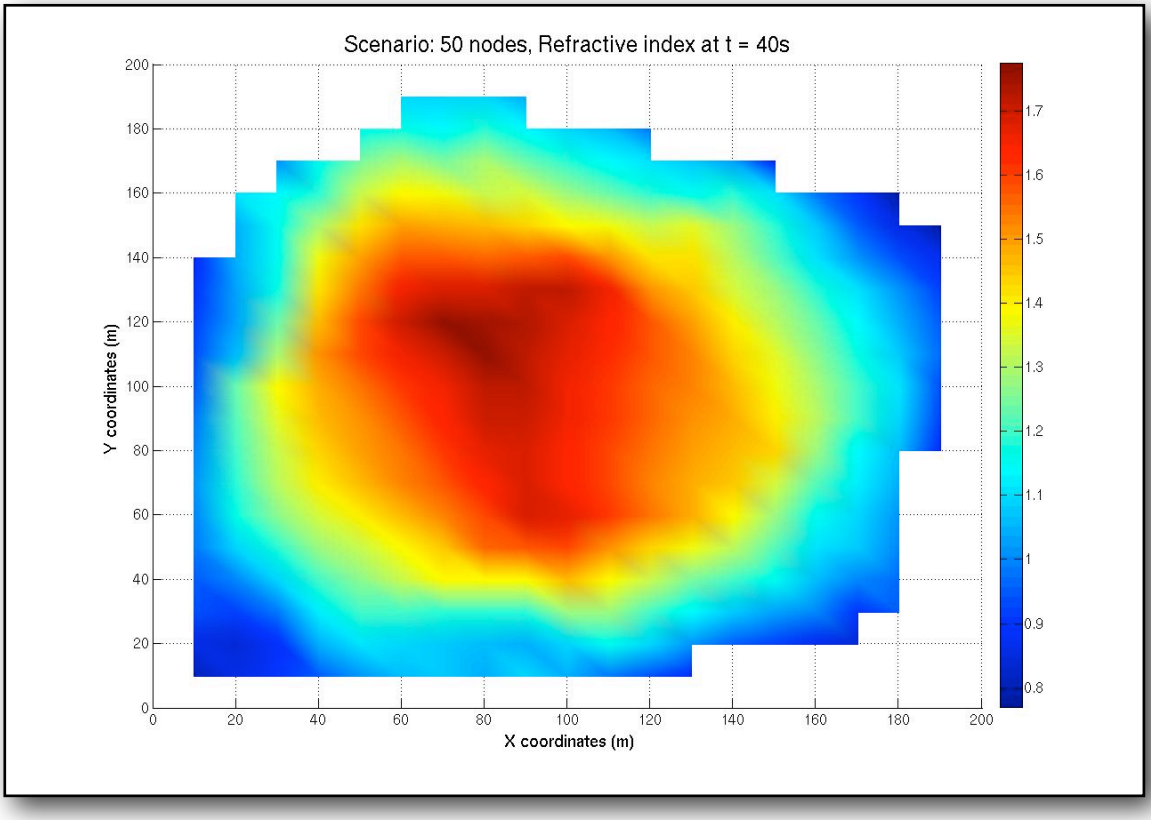


Figure C.11: Refractive index distribution at 40 simulation seconds - 50 nodes

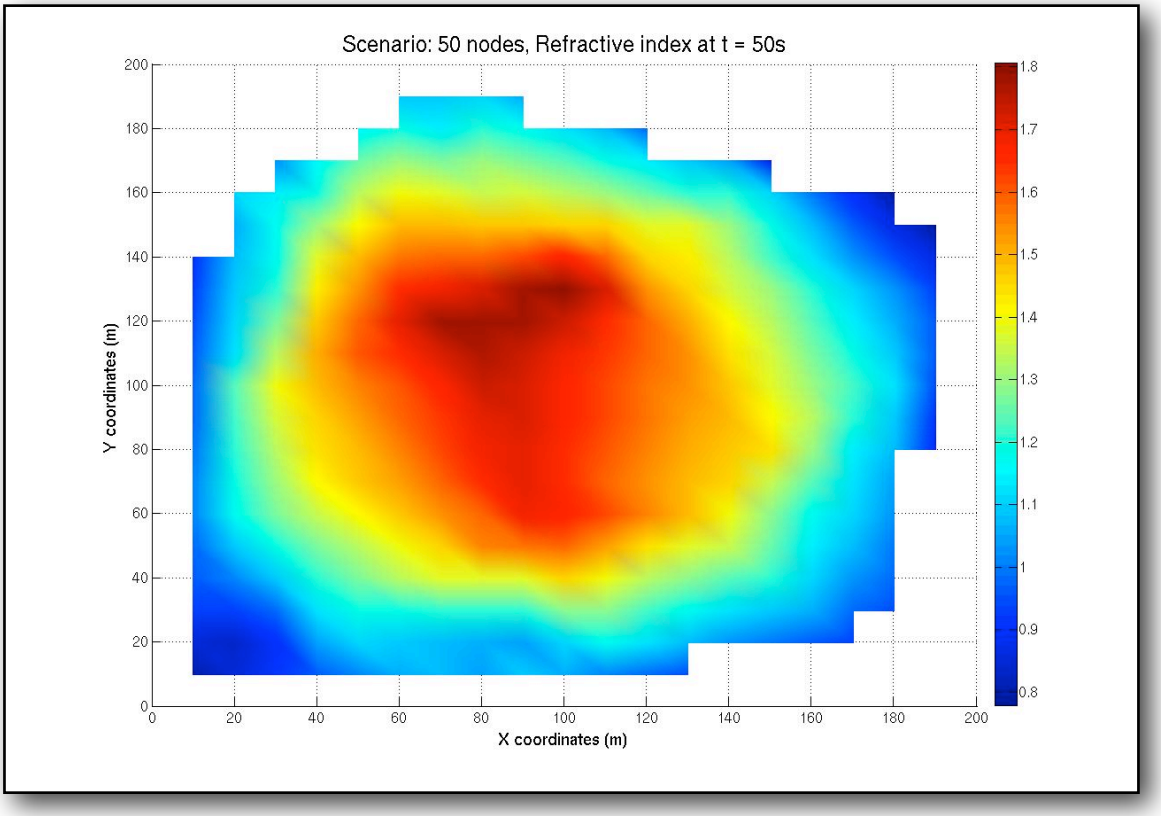


Figure C.12: Refractive index distribution at 50 simulation seconds - 50 nodes

C.2 40-node scenario

C.2.1 Energy distribution

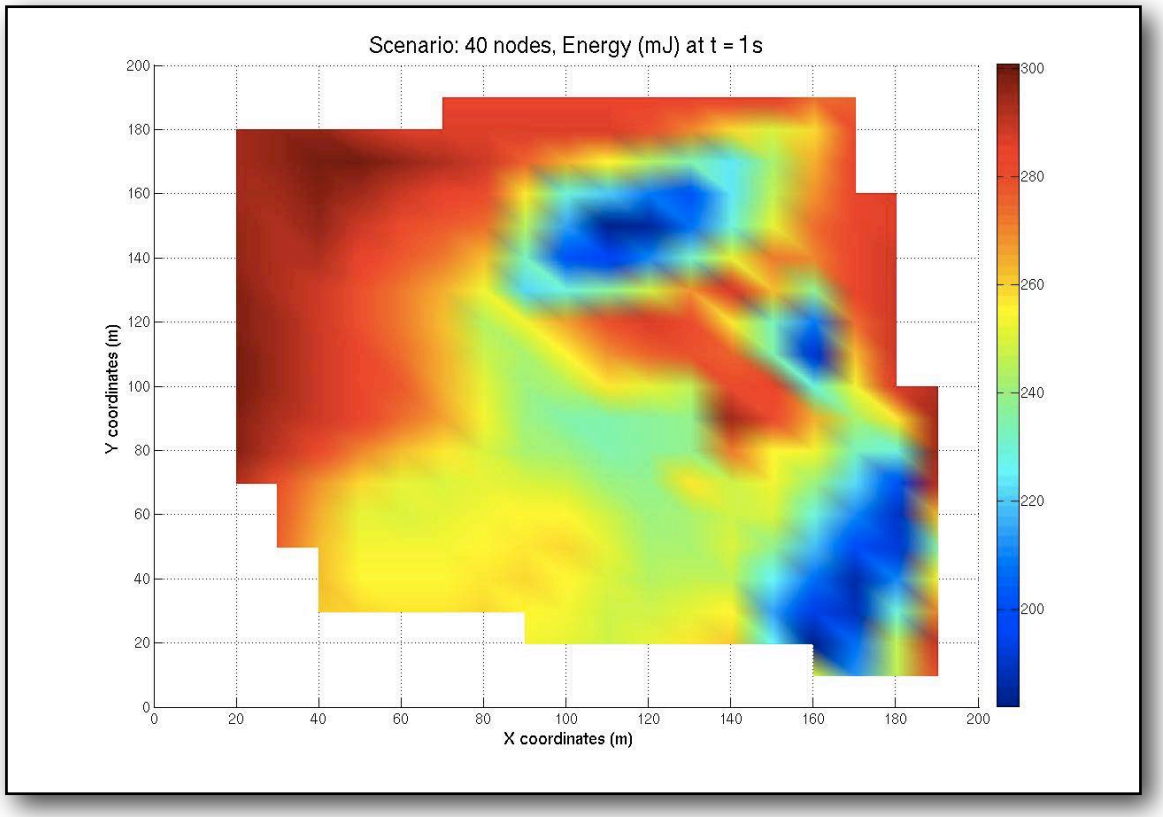


Figure C.13: Energy distribution at 1 simulation seconds - 40 nodes

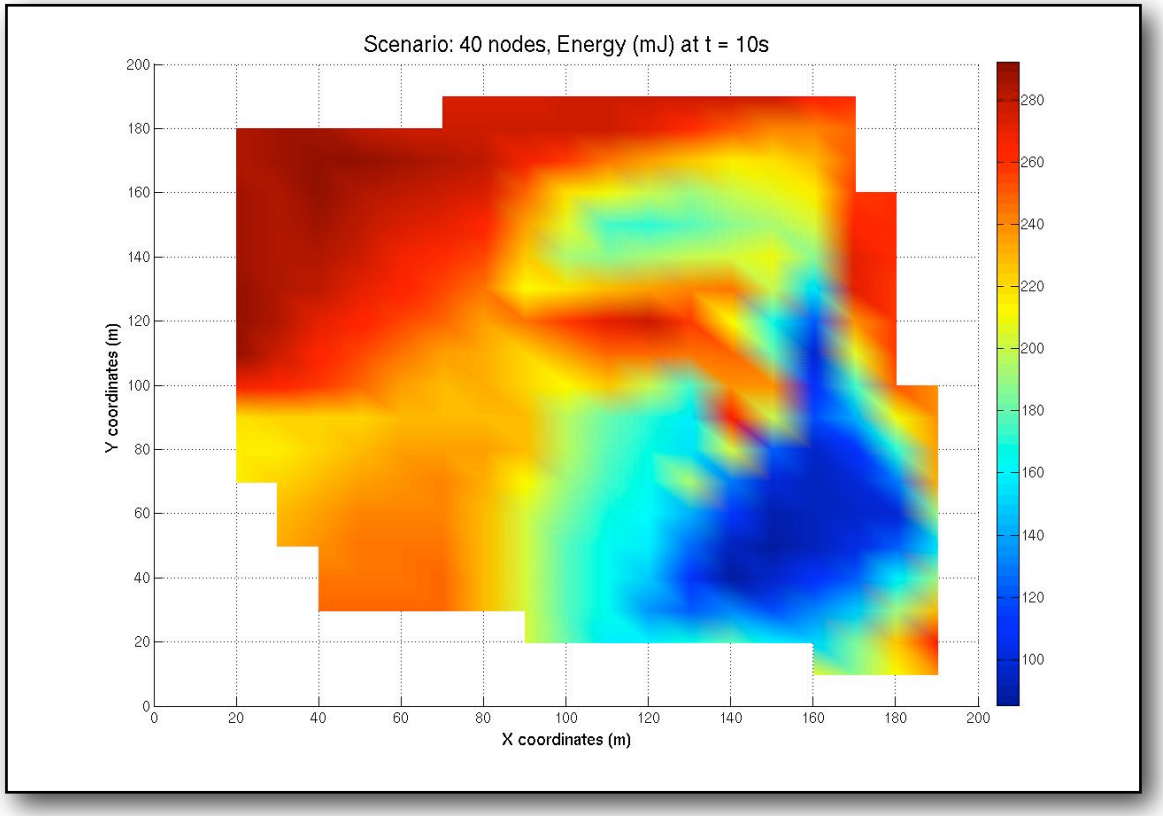


Figure C.14: Energy distribution at 10 simulation seconds - 40 nodes

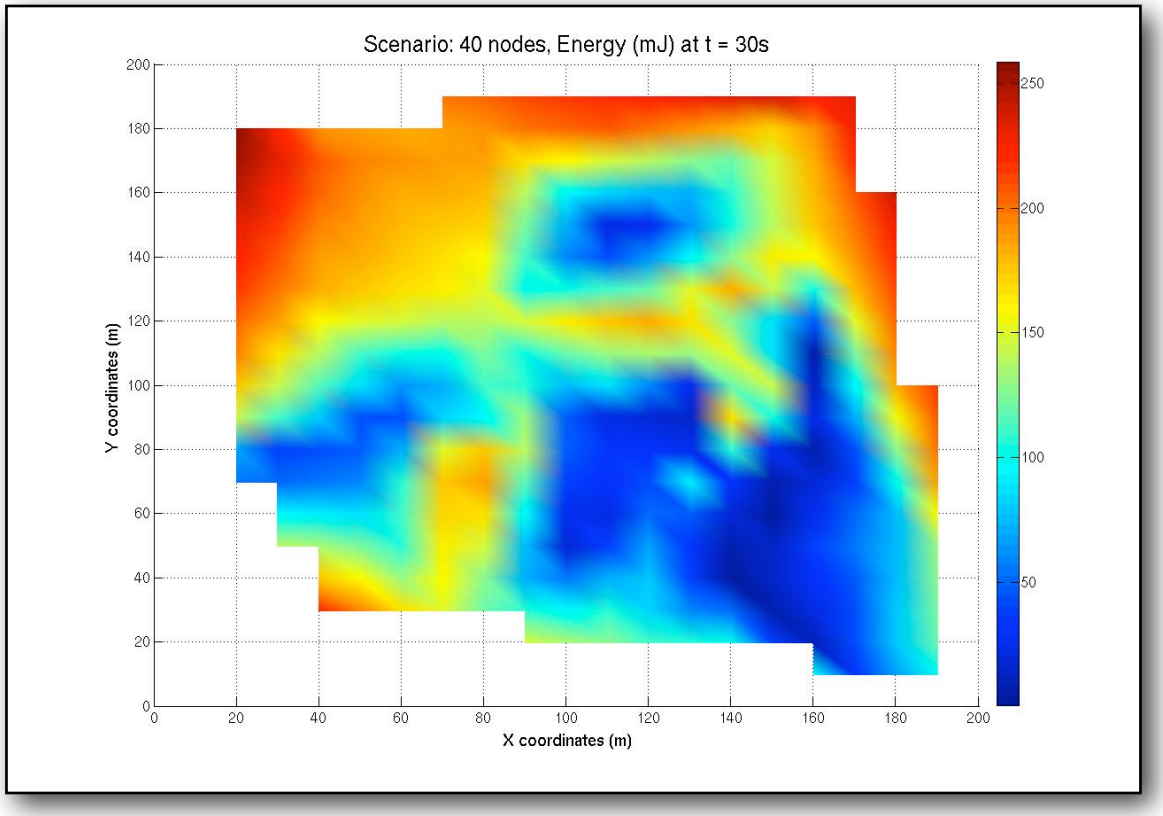


Figure C.15: Energy distribution at 30 simulation seconds - 40 nodes

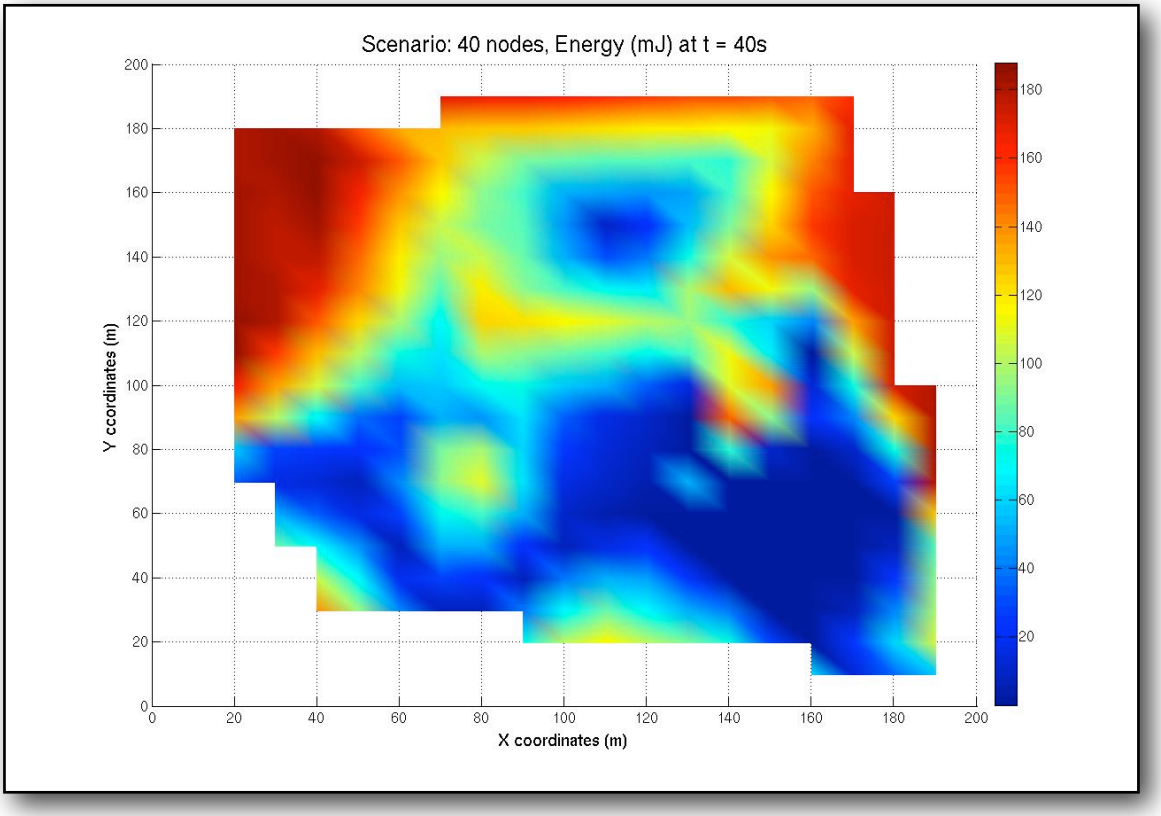


Figure C.16: Energy distribution at 40 simulation seconds - 40 nodes

C.2.2 Refractive index distribution

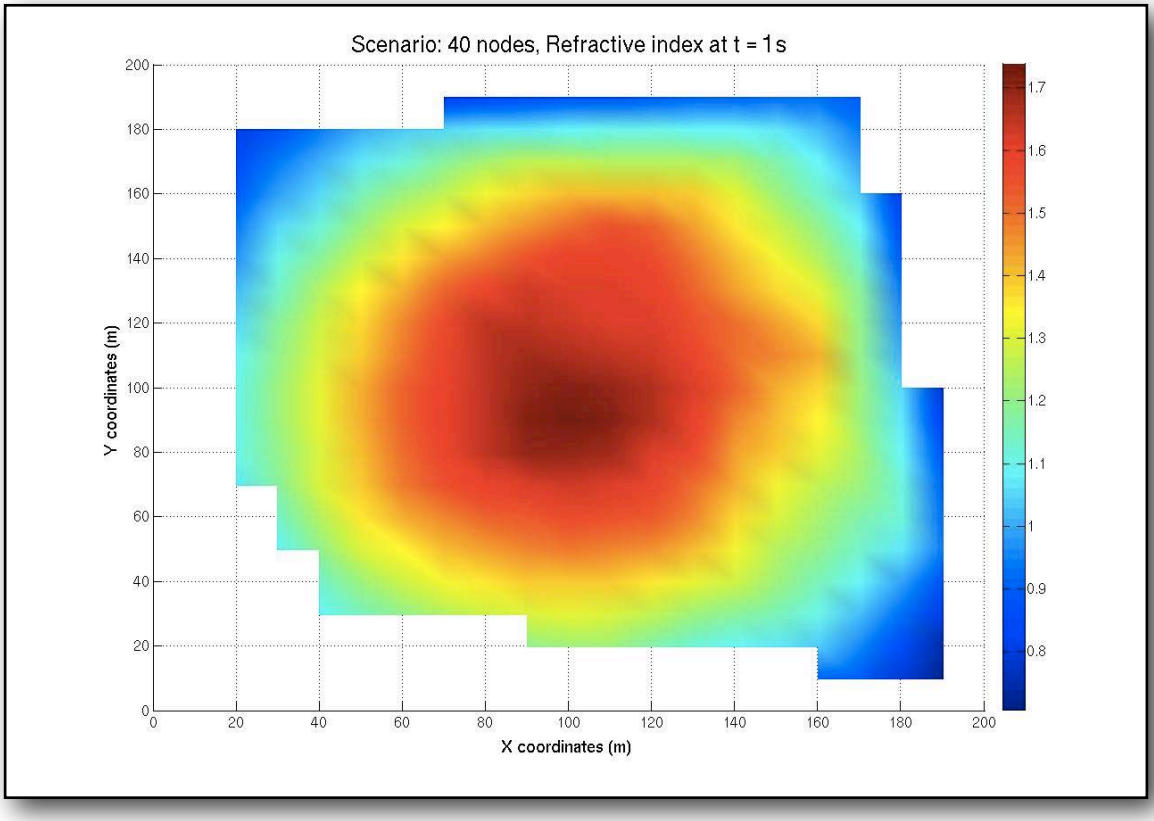


Figure C.17: Refractive index distribution at 1 simulation seconds - 40 nodes

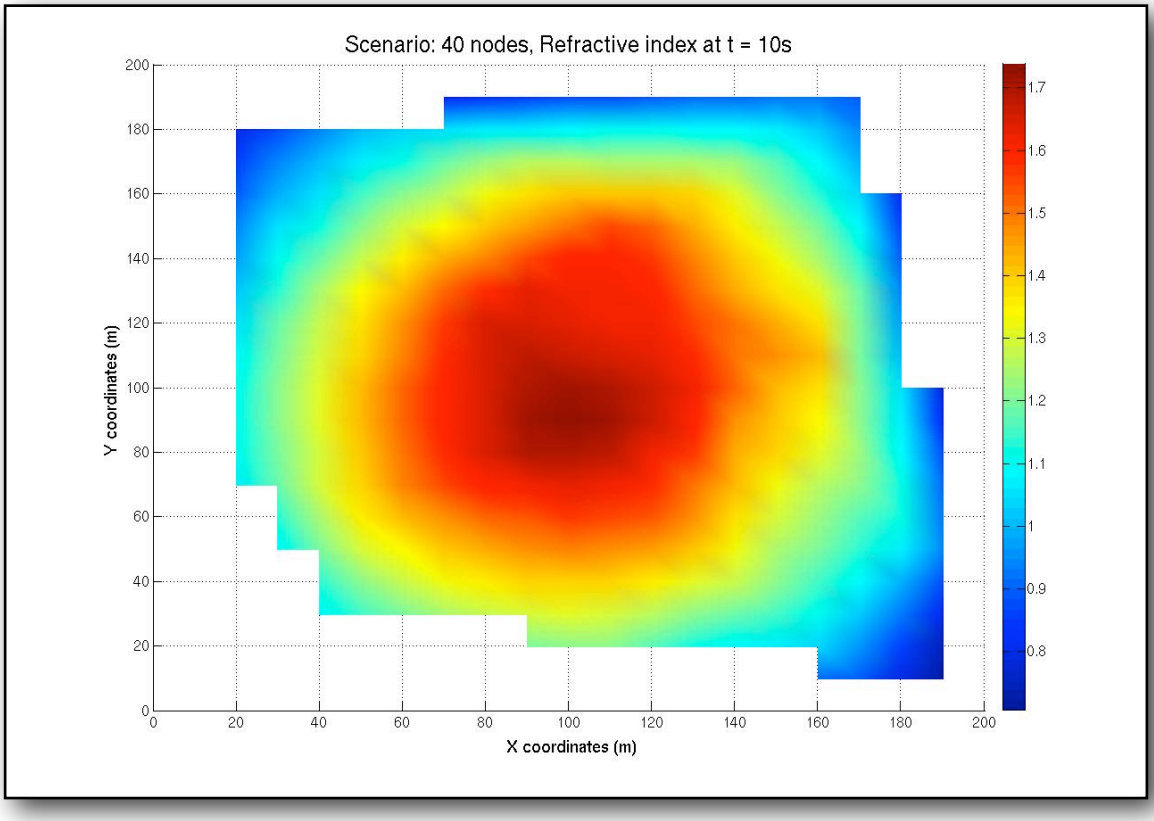


Figure C.18: Refractive index distribution at 10 simulation seconds - 40 nodes

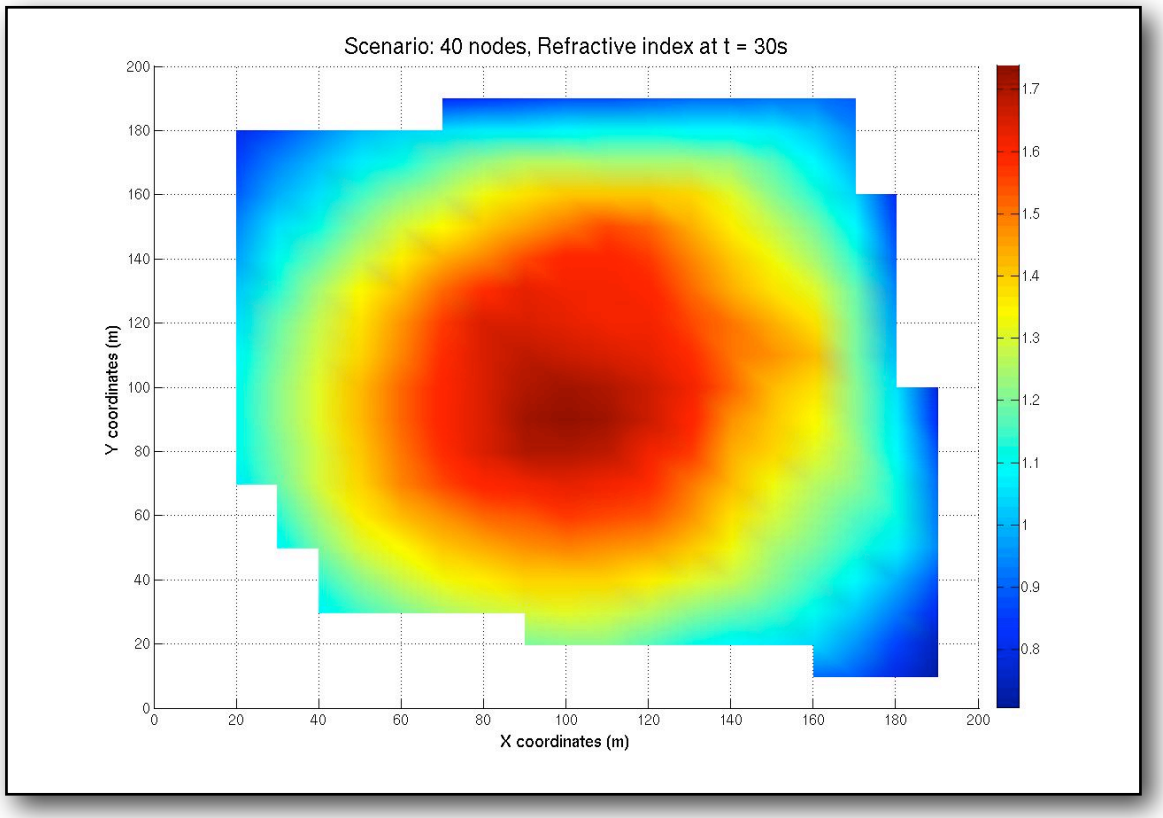


Figure C.19: Refractive index distribution at 30 simulation seconds - 40 nodes

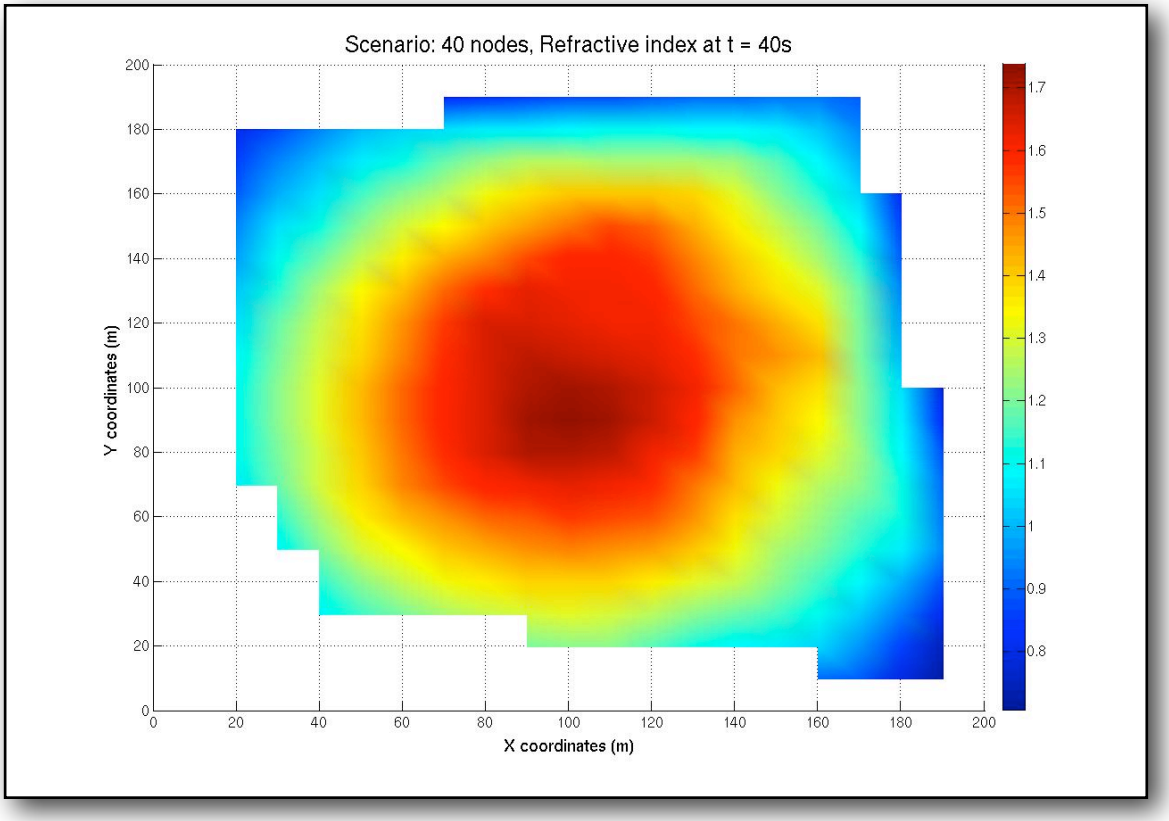


Figure C.20: Refractive index distribution at 40 simulation seconds - 40 nodes

C.3 35-node scenario

C.3.1 Energy distribution

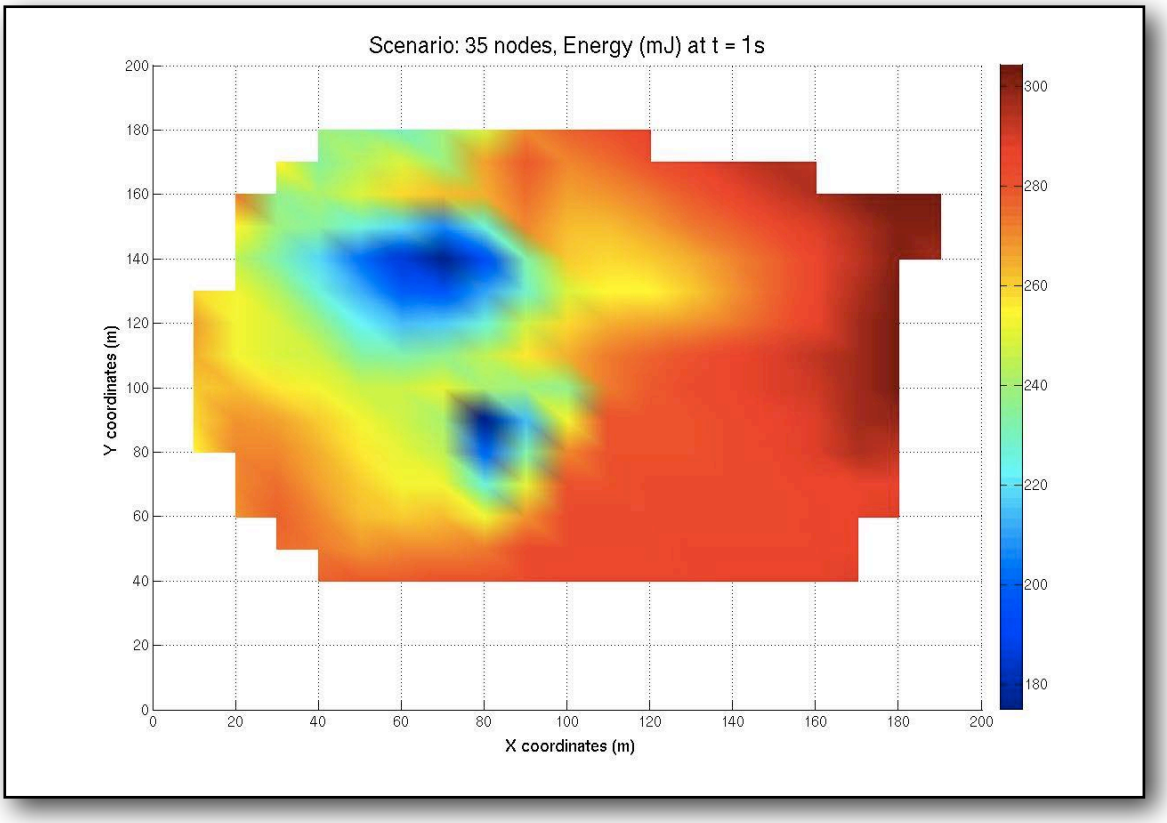


Figure C.21: Energy distribution at 1 simulation seconds - 35 nodes

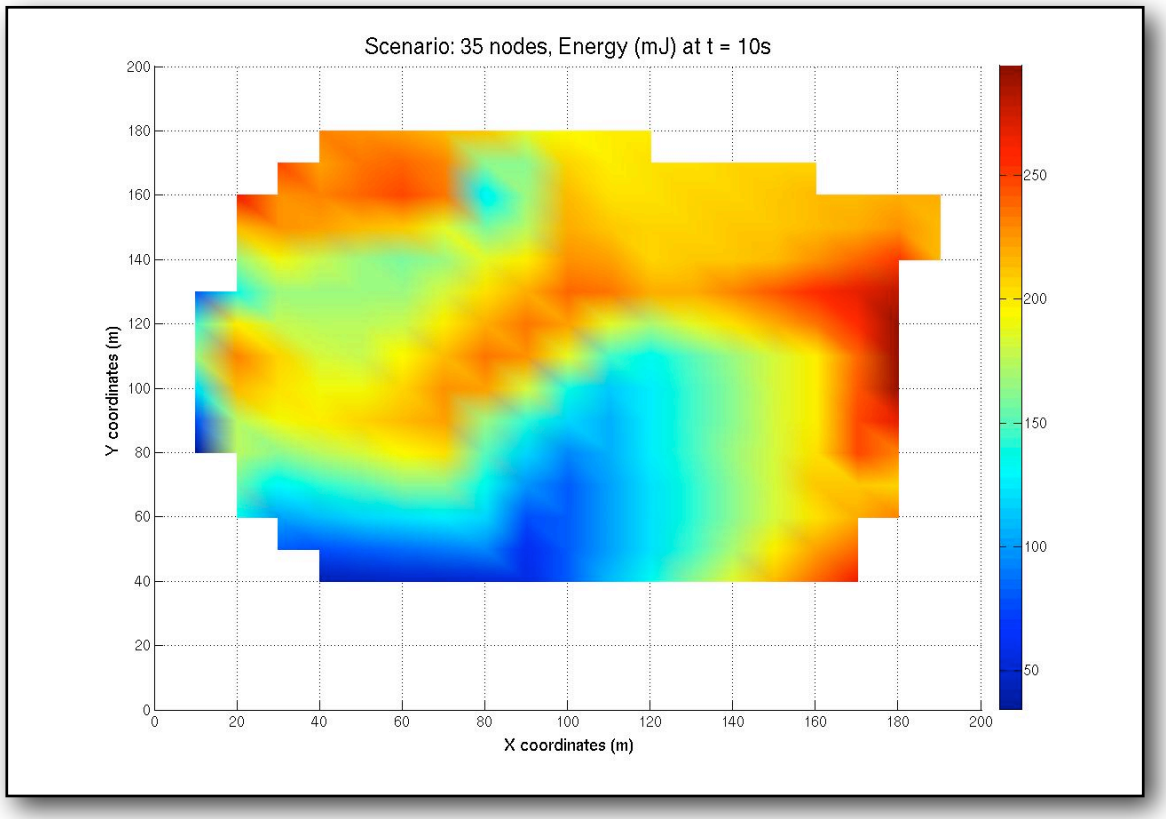


Figure C.22: Energy distribution at 10 simulation seconds - 35 nodes

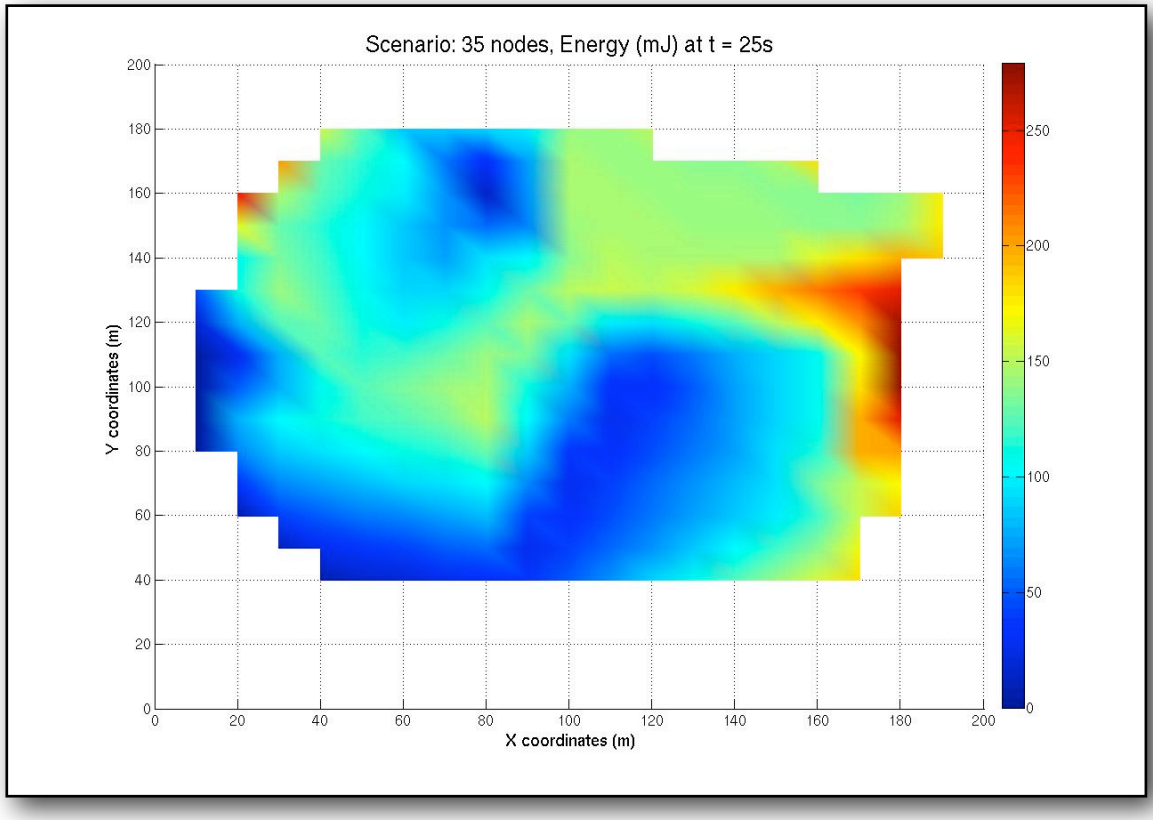


Figure C.23: Energy distribution at 25 simulation seconds - 35 nodes

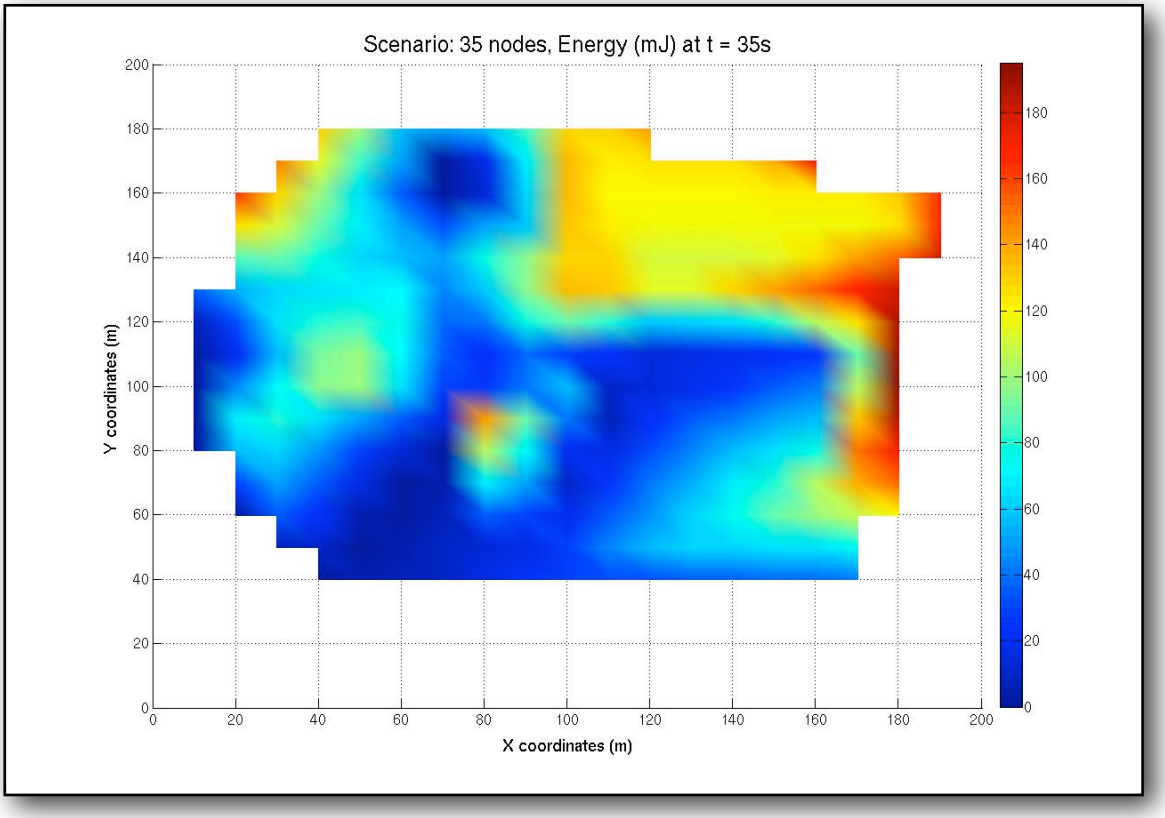


Figure C.24: Energy distribution at 35 simulation seconds - 35 nodes

C.3.2 Refractive index distribution

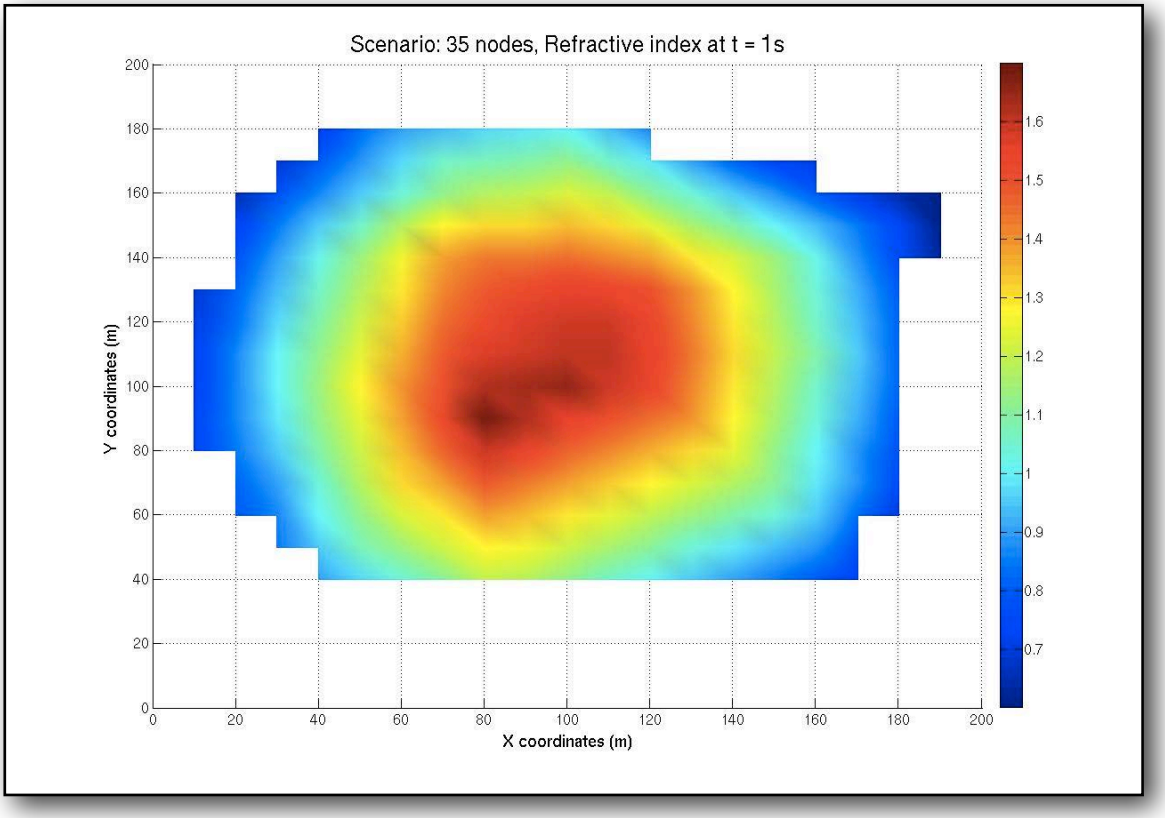


Figure C.25: Refractive index distribution at 1 simulation seconds - 35 nodes

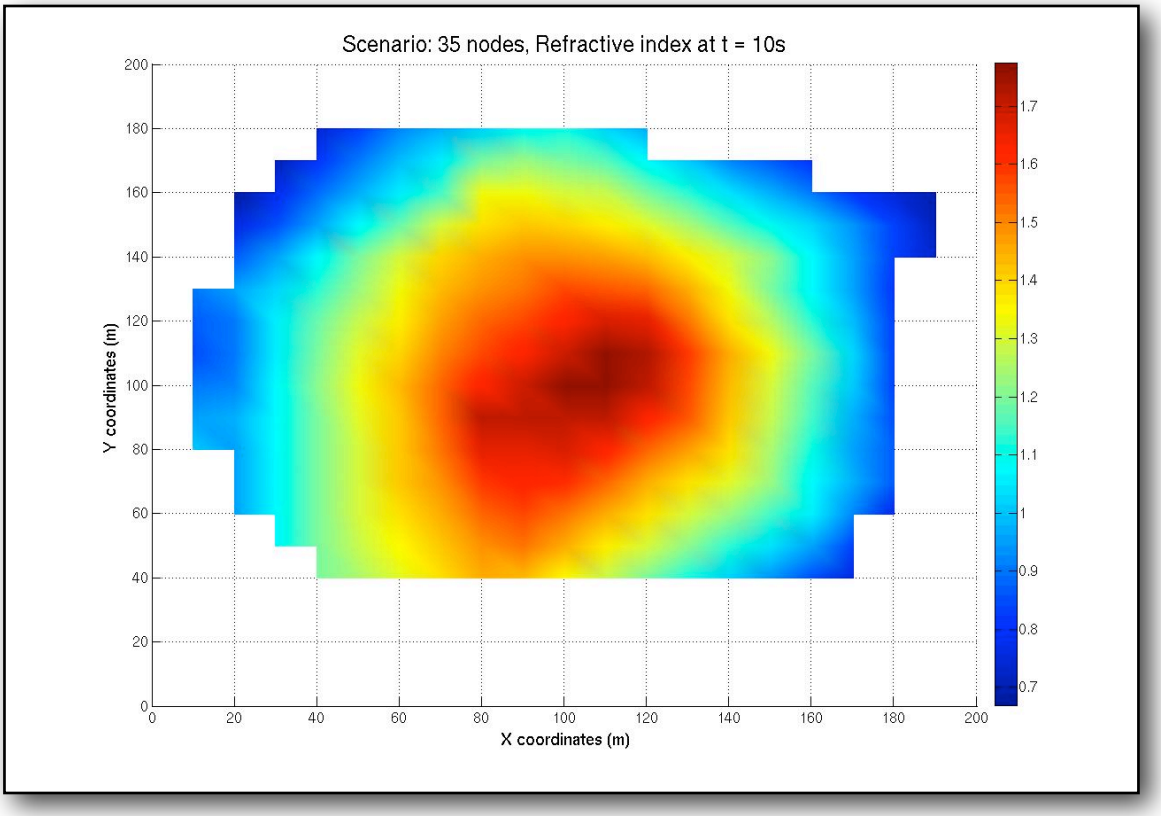


Figure C.26: Refractive index distribution at 10 simulation seconds - 35 nodes

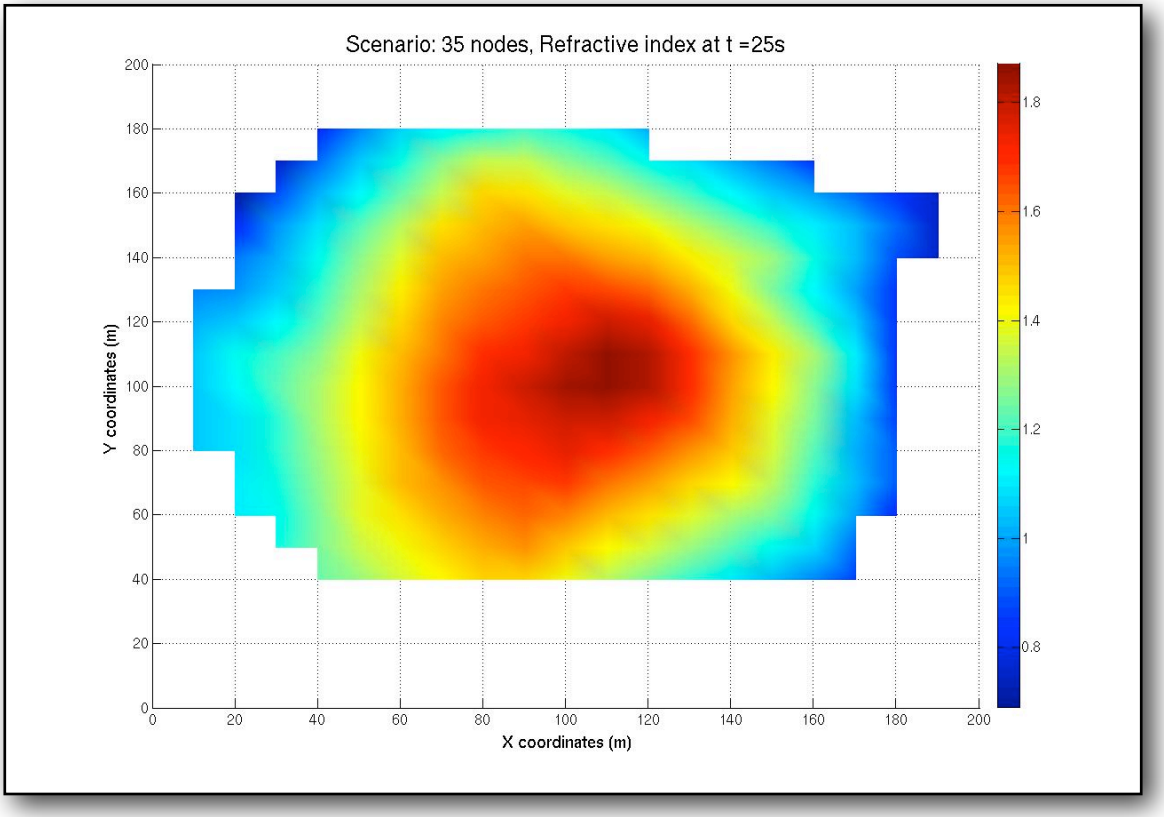


Figure C.27: Refractive index distribution at 25 simulation seconds - 35 nodes

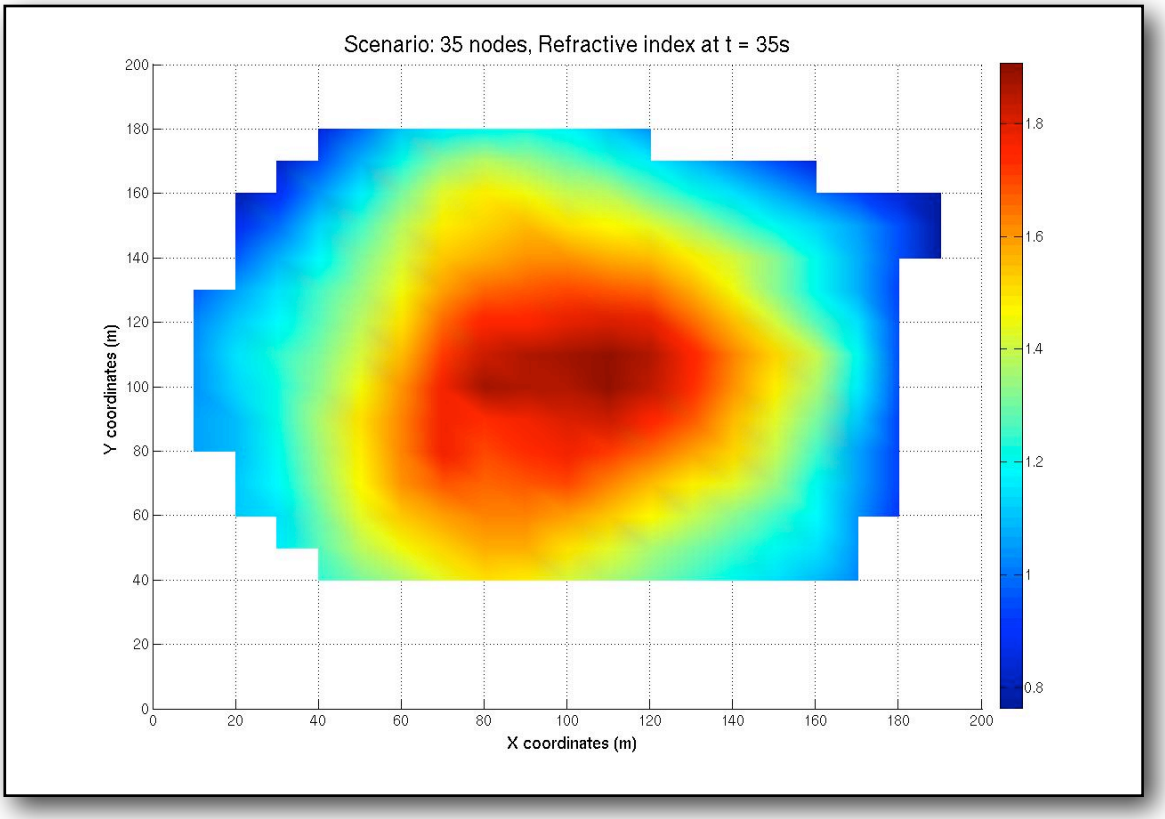


Figure C.28: Refractive index distribution at 35 simulation seconds - 35 nodes

C.4 100-node scenario

C.4.1 Energy distribution

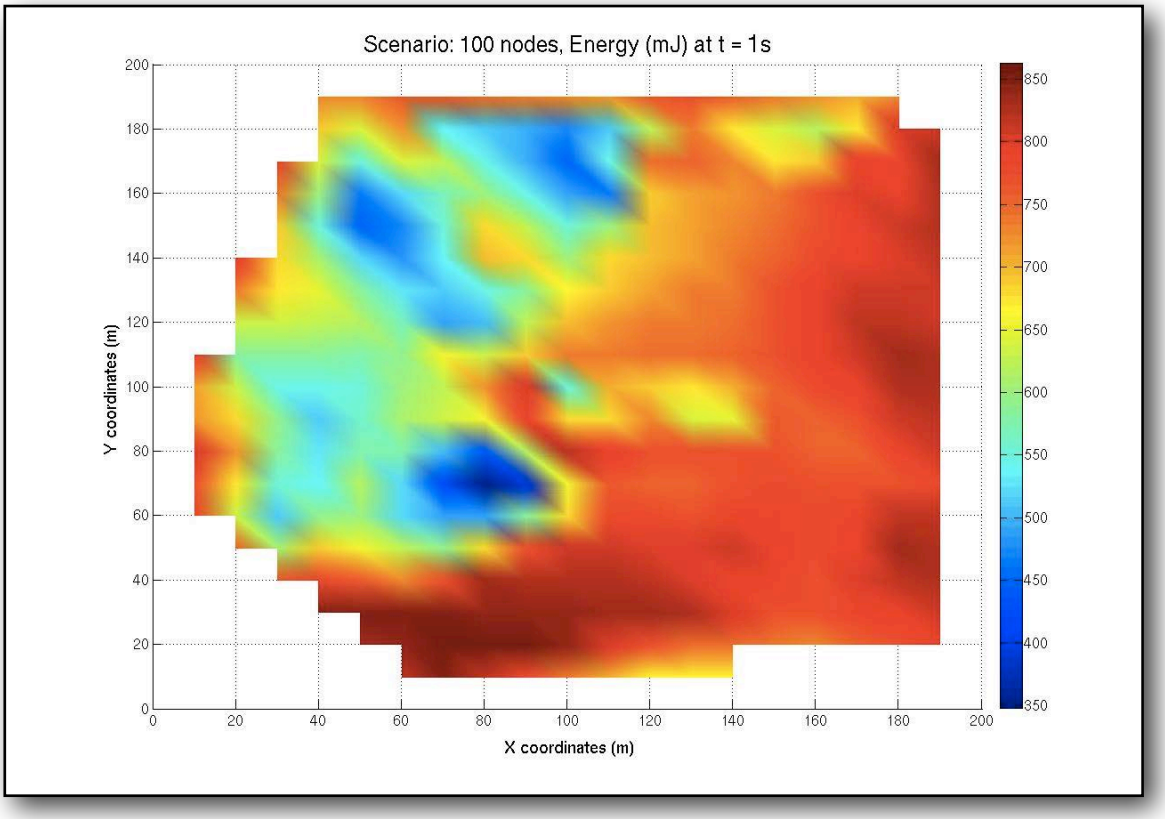


Figure C.29: Energy distribution at 1 simulation seconds - 100 nodes

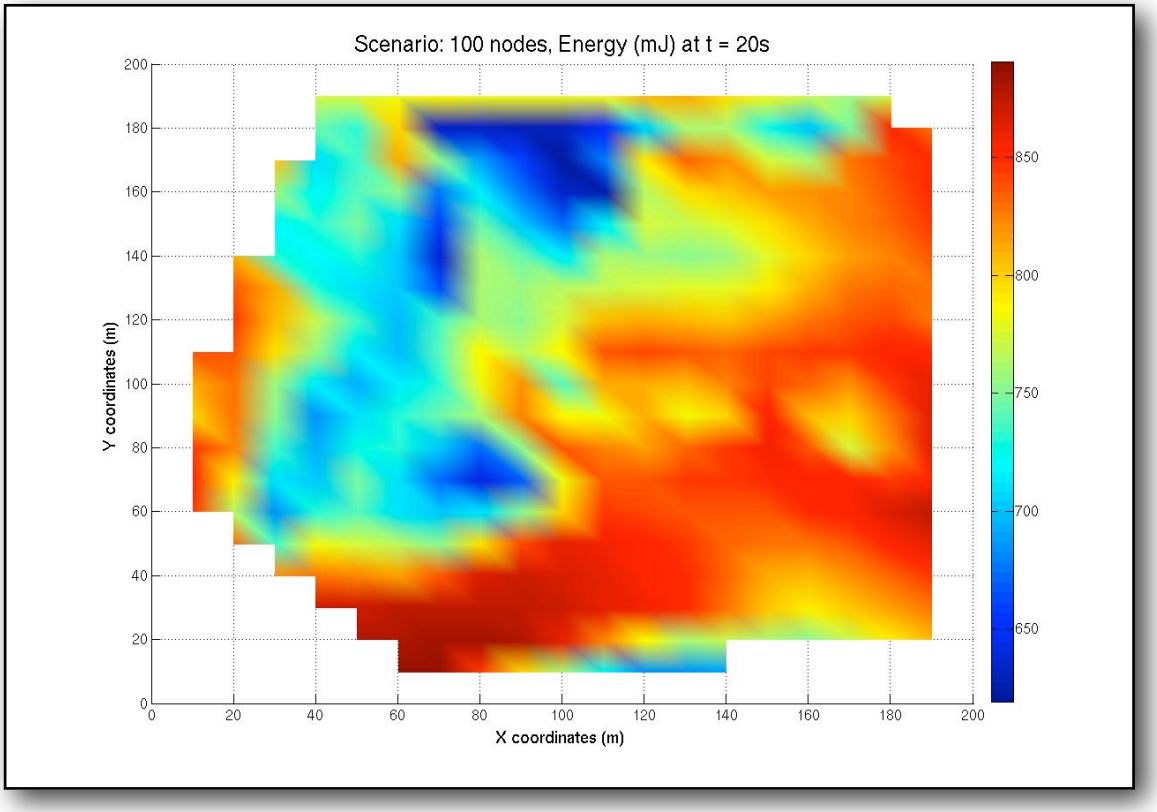


Figure C.30: Energy distribution at 20 simulation seconds - 100 nodes

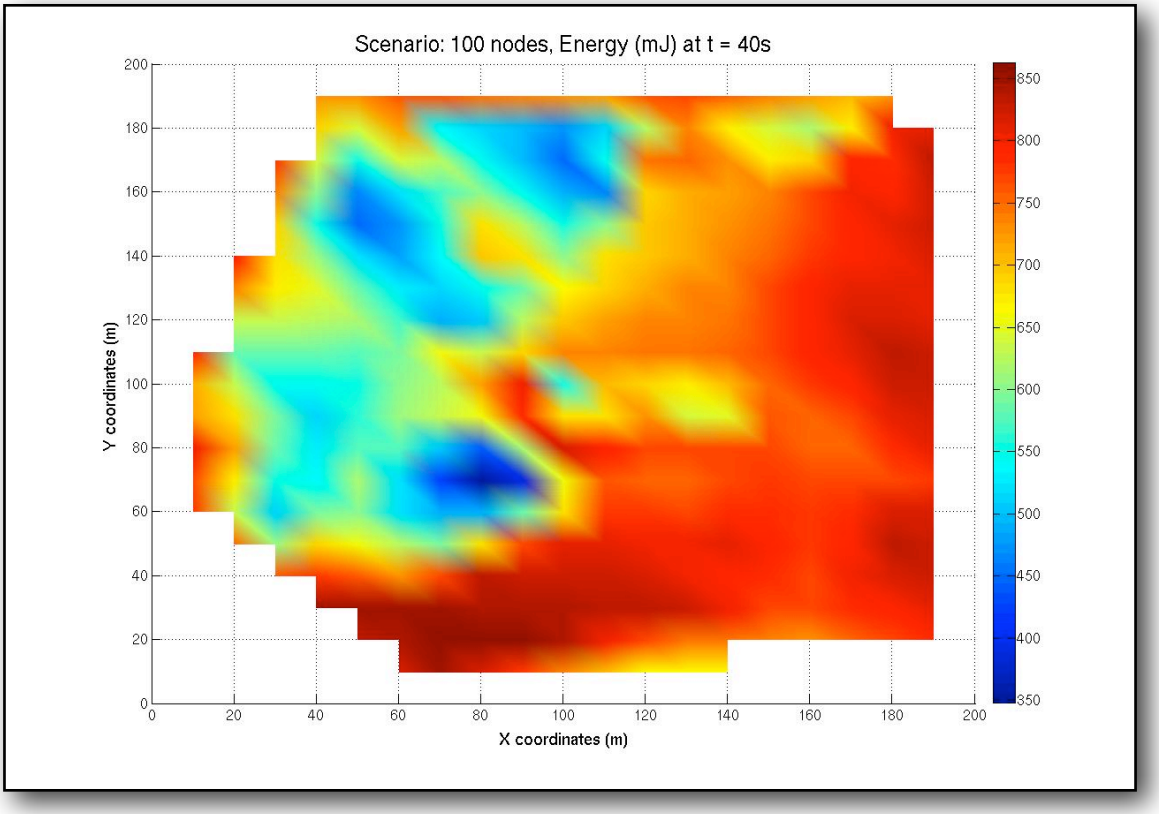


Figure C.31: Energy distribution at 40 simulation seconds - 100 nodes

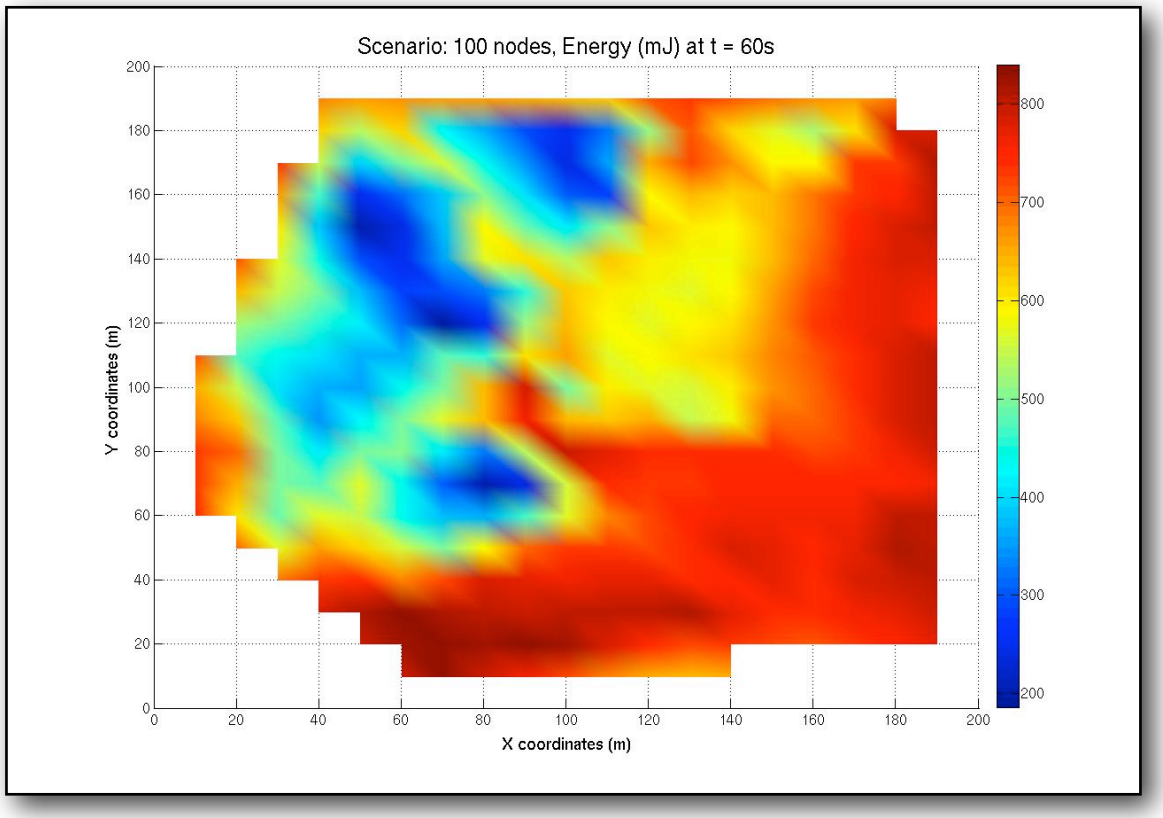


Figure C.32: Energy distribution at 60 simulation seconds - 100 nodes

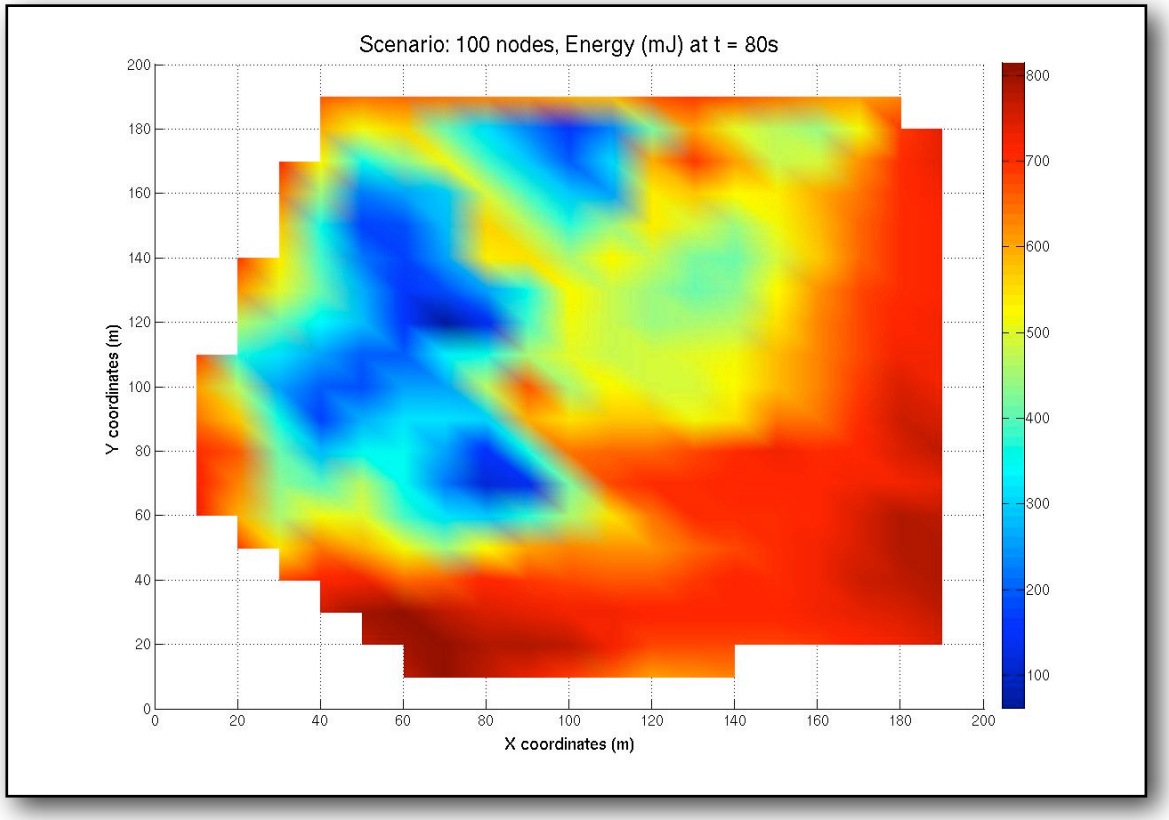


Figure C.33: Energy distribution at 80 simulation seconds - 100 nodes

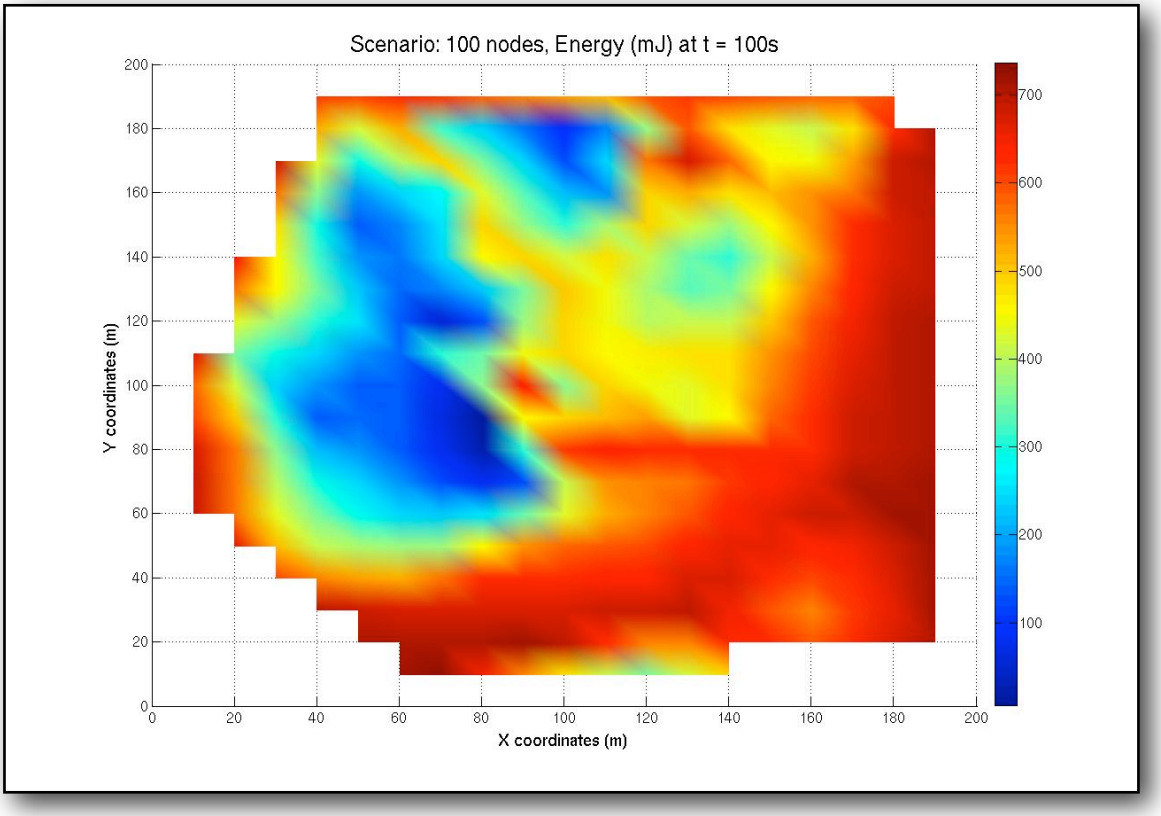


Figure C.34: Energy distribution at 100 simulation seconds - 100 nodes

C.4.2 Refractive index distribution

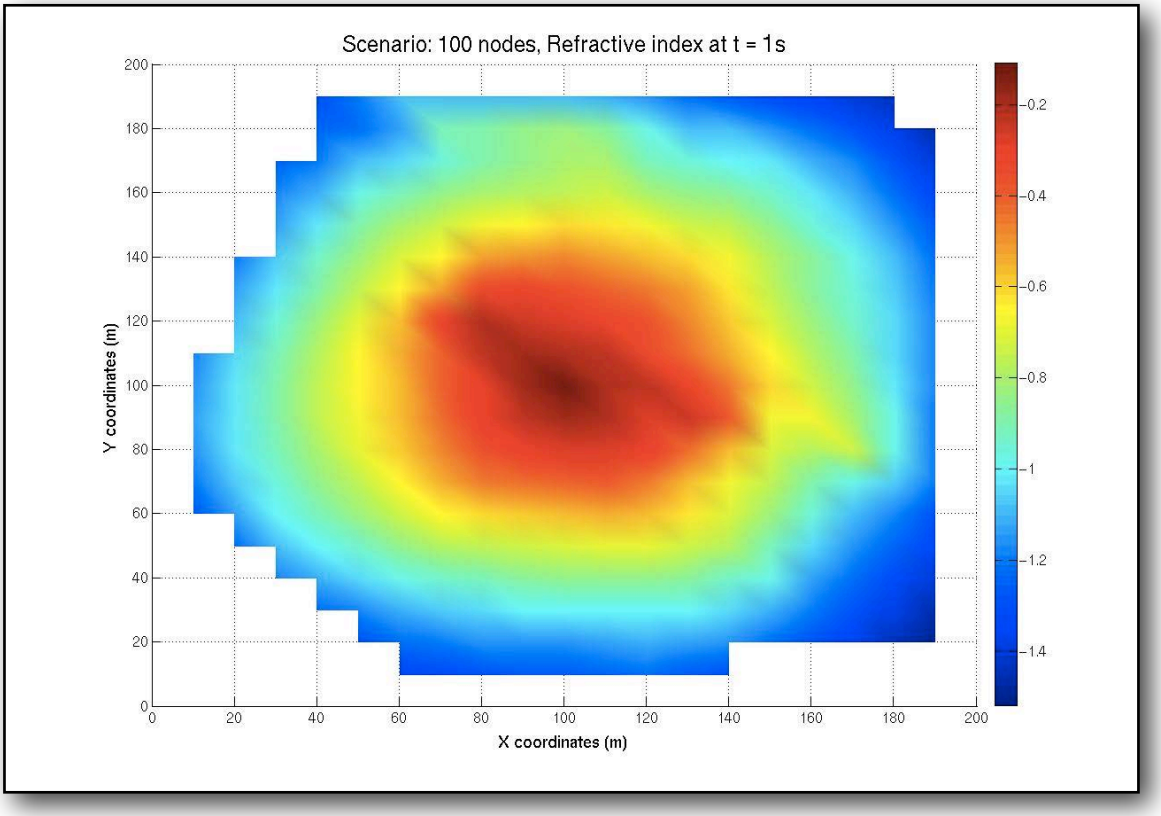


Figure C.35: Refractive index distribution at 1 simulation seconds - 100 nodes

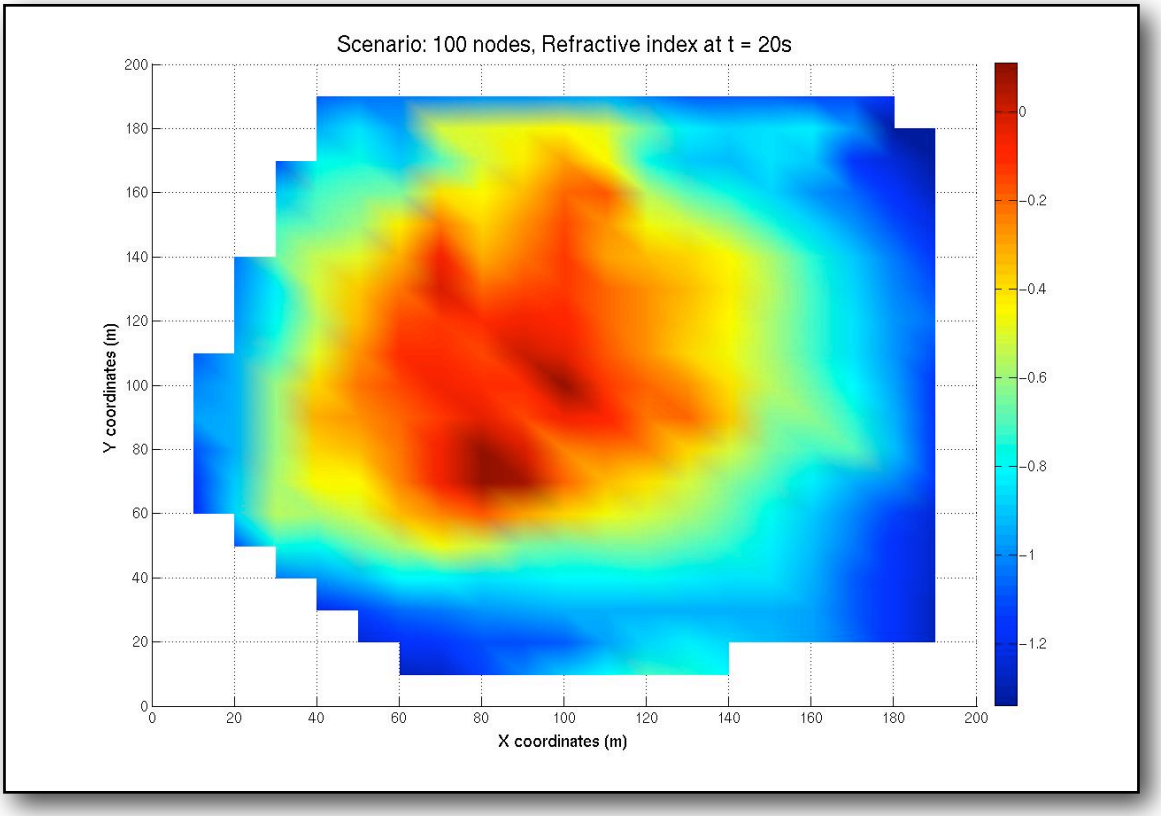


Figure C.36: Refractive index distribution at 20 simulation seconds - 100 nodes

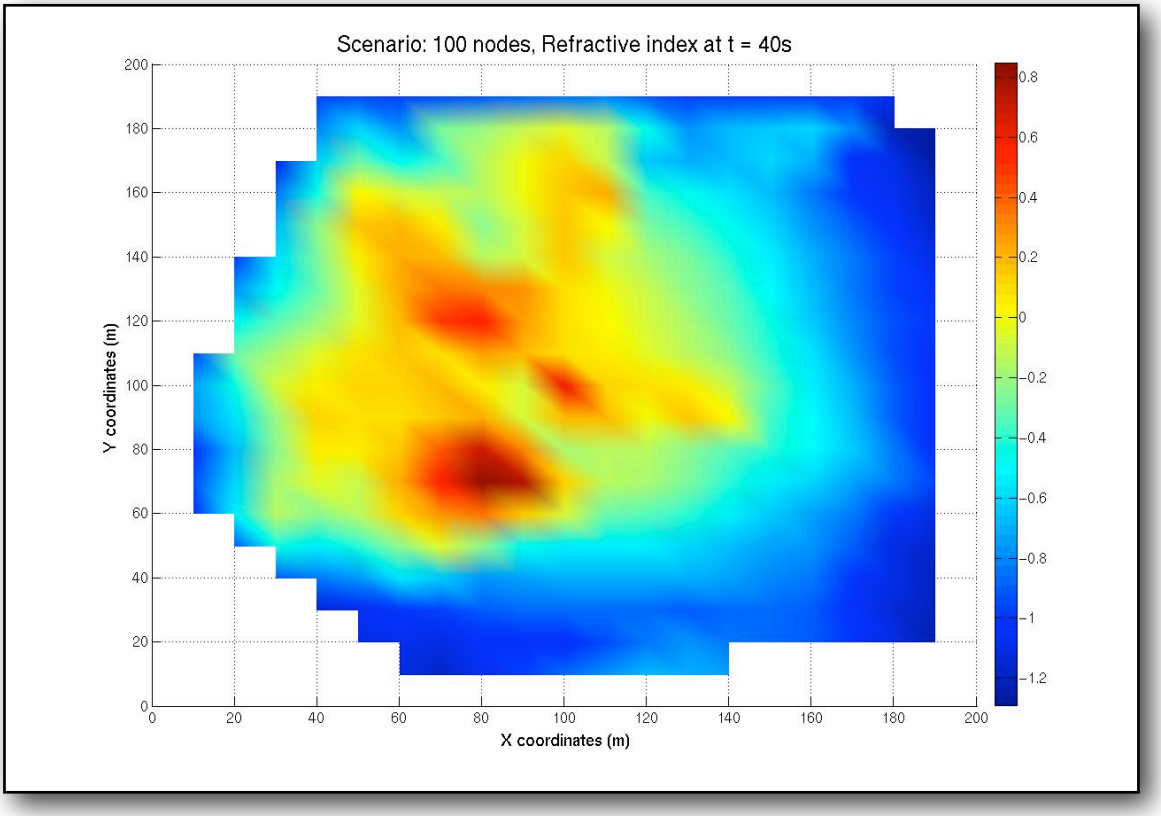


Figure C.37: Refractive index distribution at 40 simulation seconds - 100 nodes

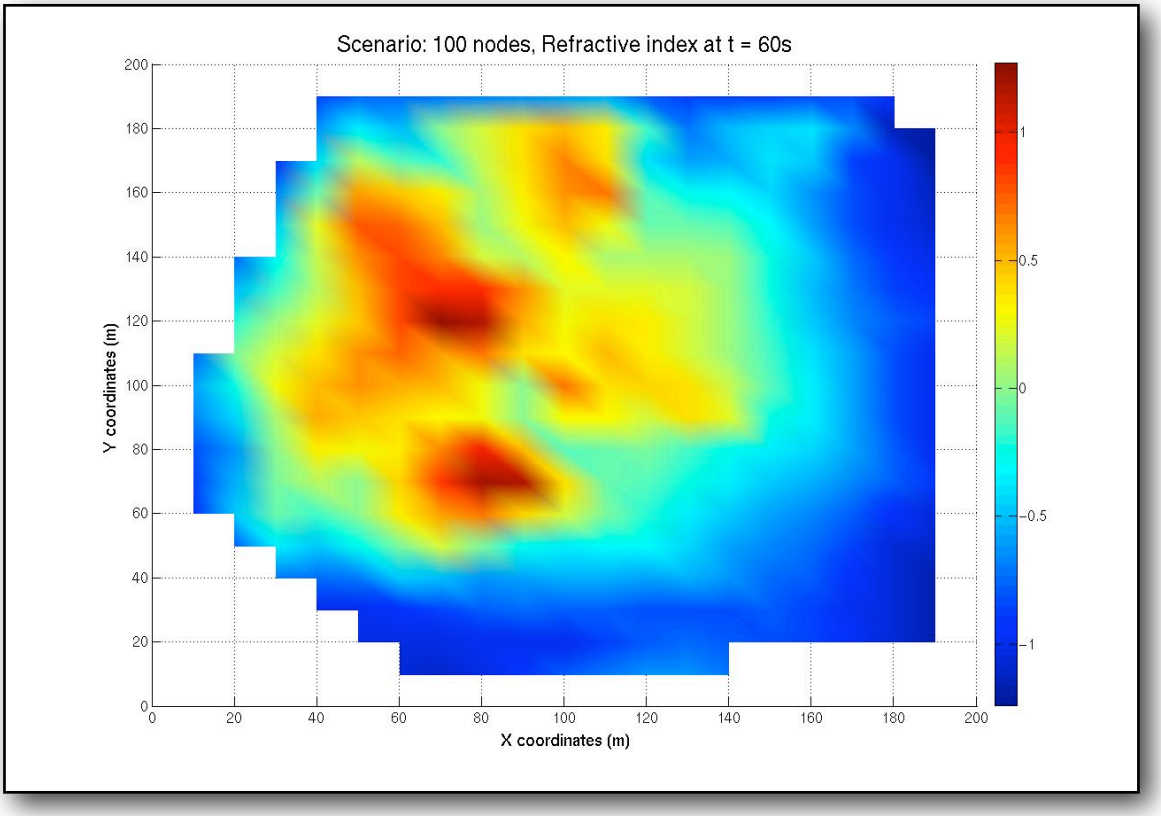


Figure C.38: Refractive index distribution at 60 simulation seconds - 100 nodes

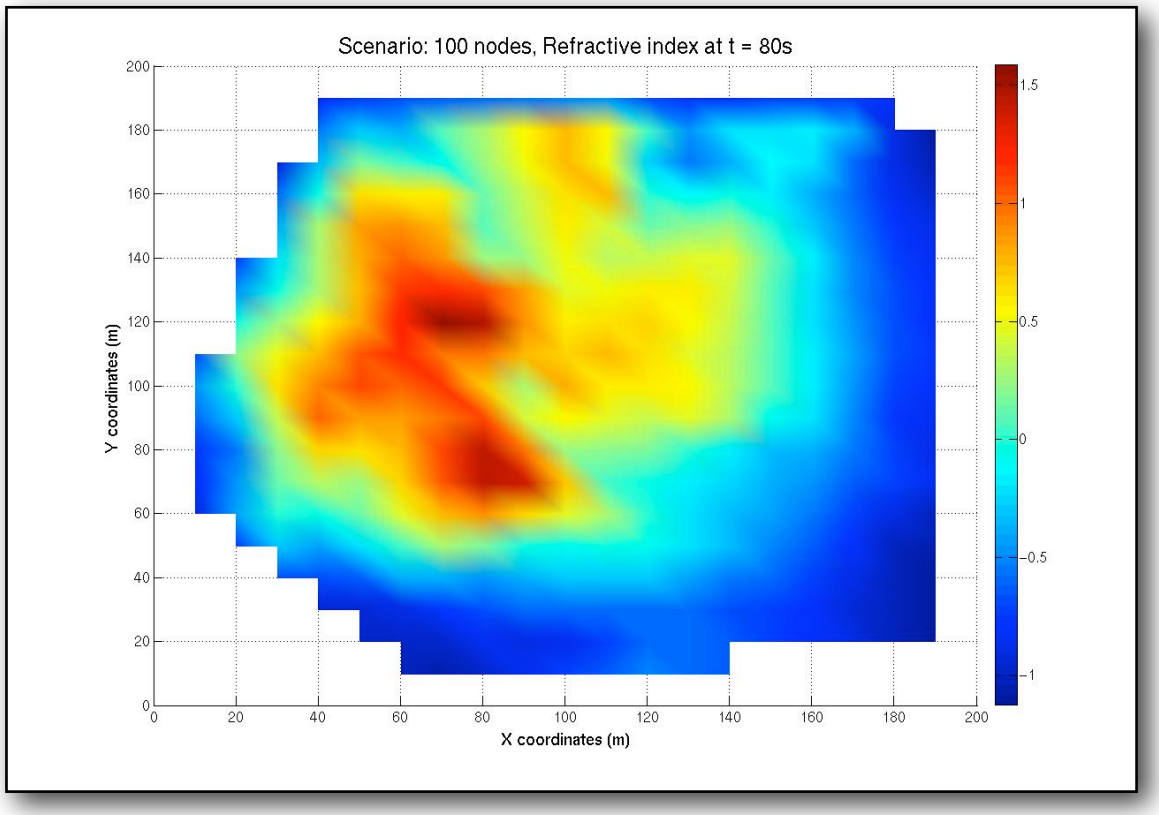


Figure C.39: Refractive index distribution at 80 simulation seconds - 100 nodes

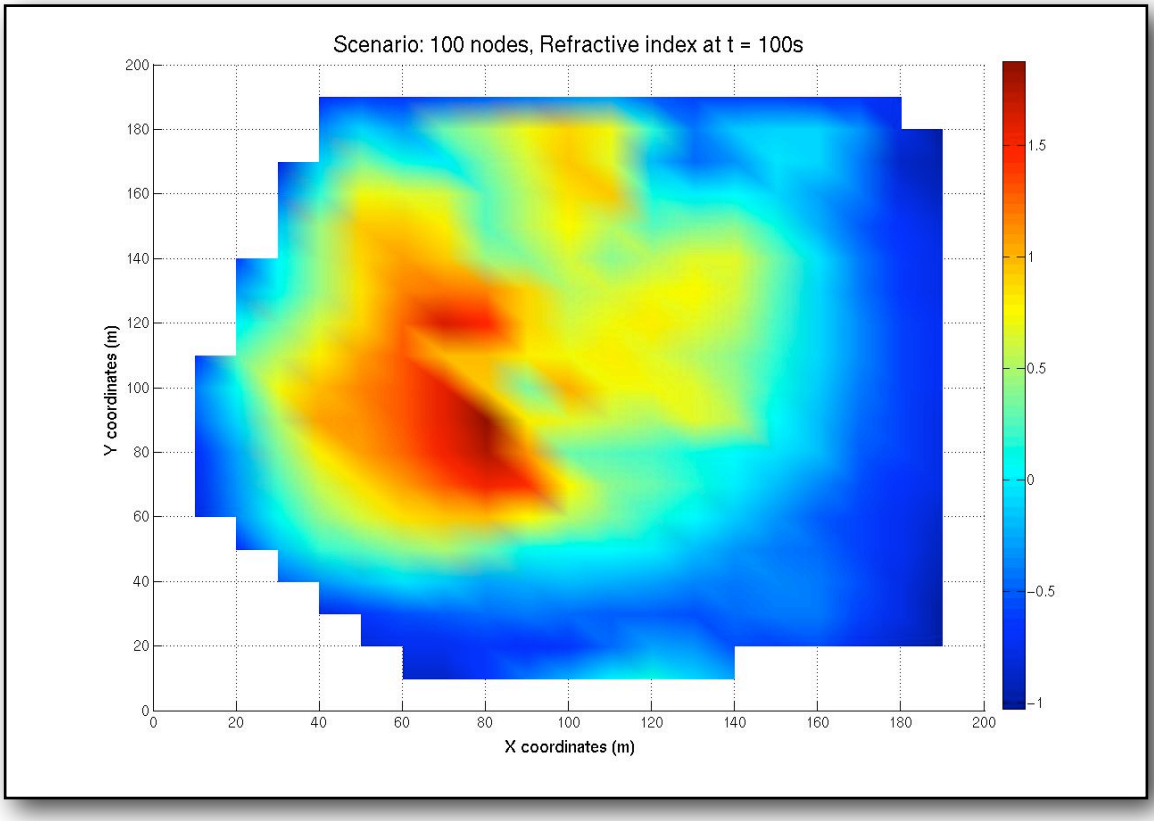


Figure C.40: Refractive index distribution at 100 simulation seconds - 100 nodes

C.5 140-node scenario

C.5.1 Energy distribution

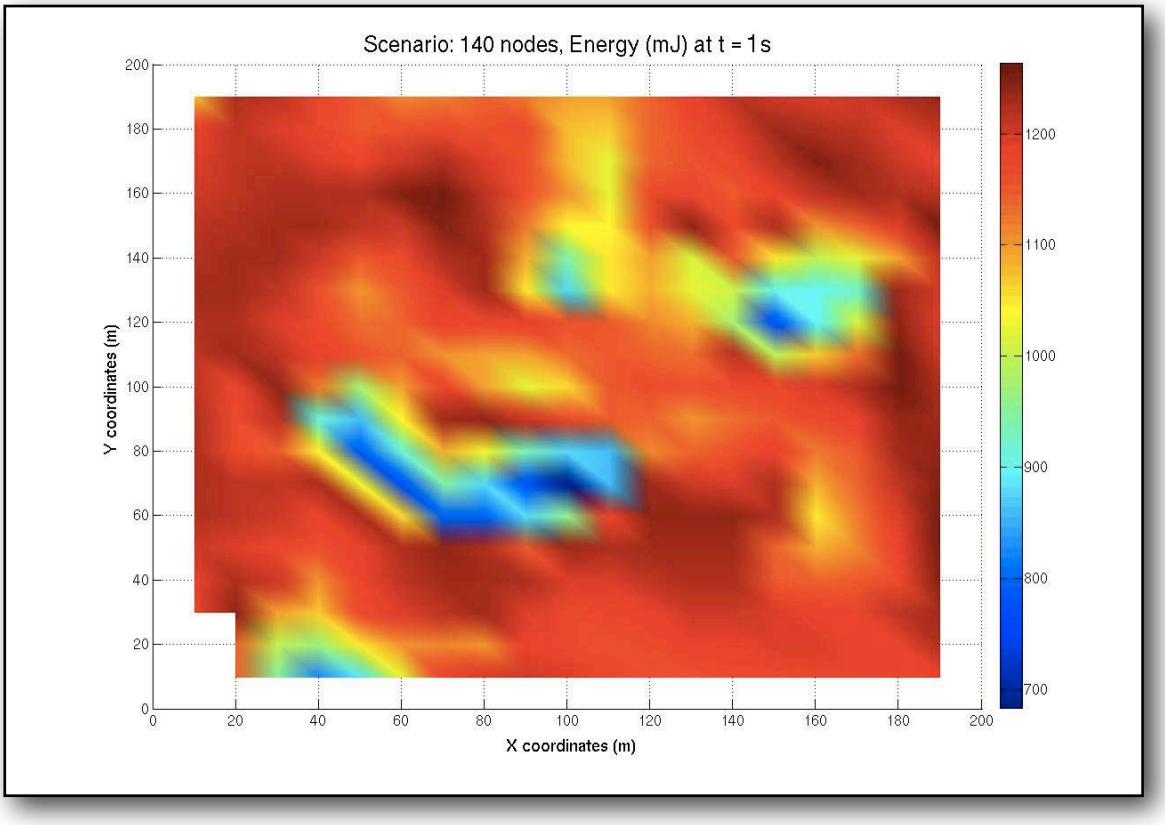


Figure C.41: Energy distribution at 1 simulation seconds - 140 nodes

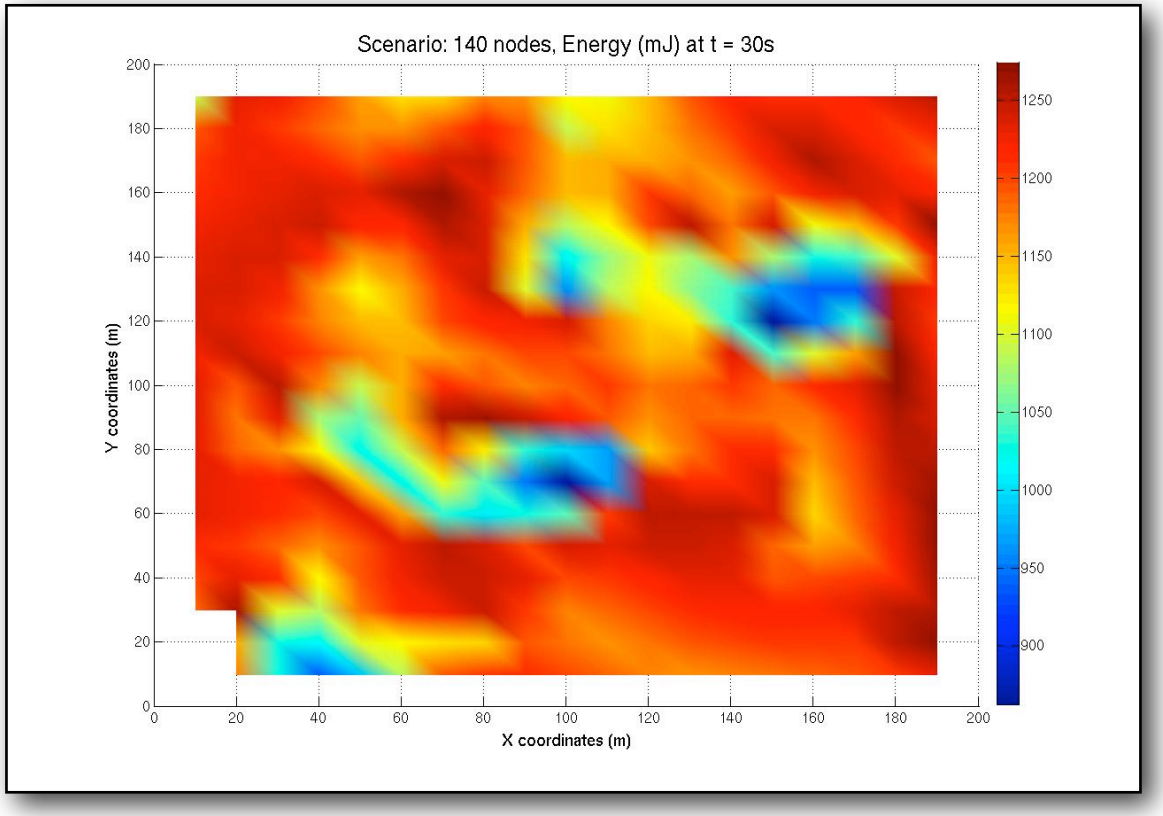


Figure C.42: Energy distribution at 30 simulation seconds - 140 nodes

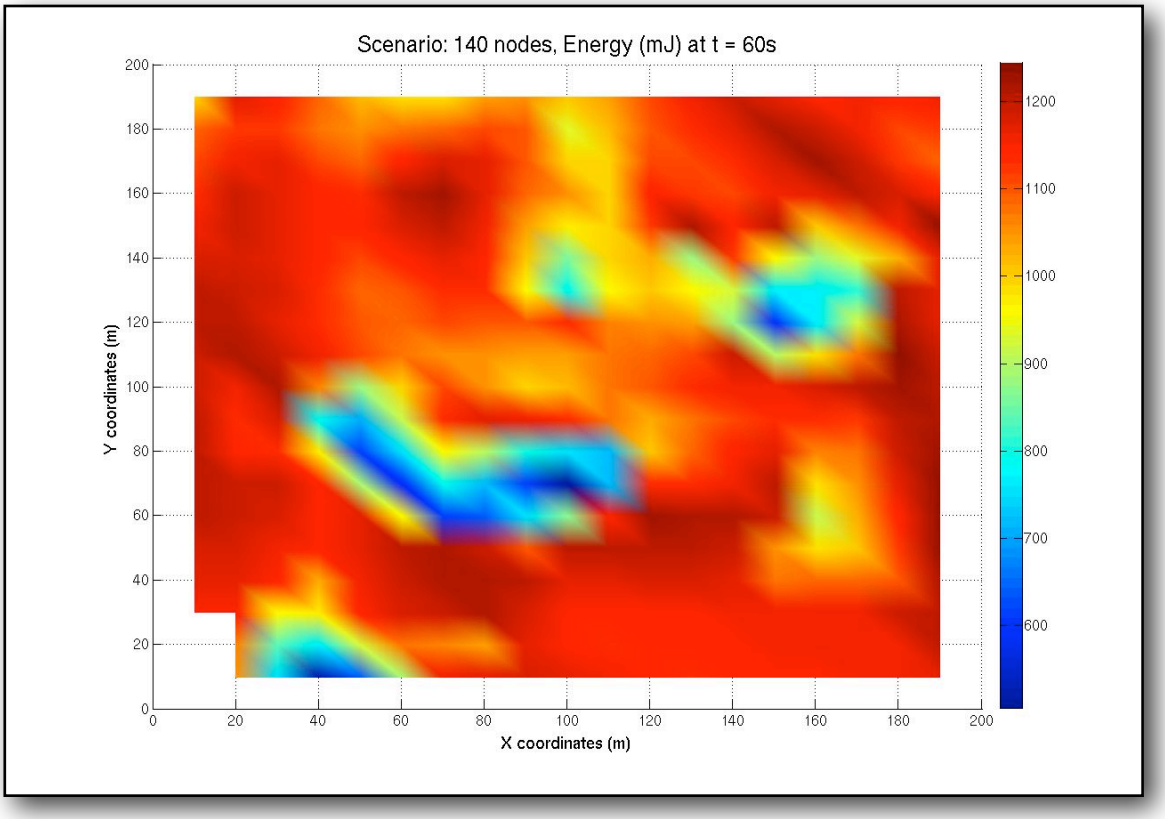


Figure C.43: Energy distribution at 60 simulation seconds - 140 nodes

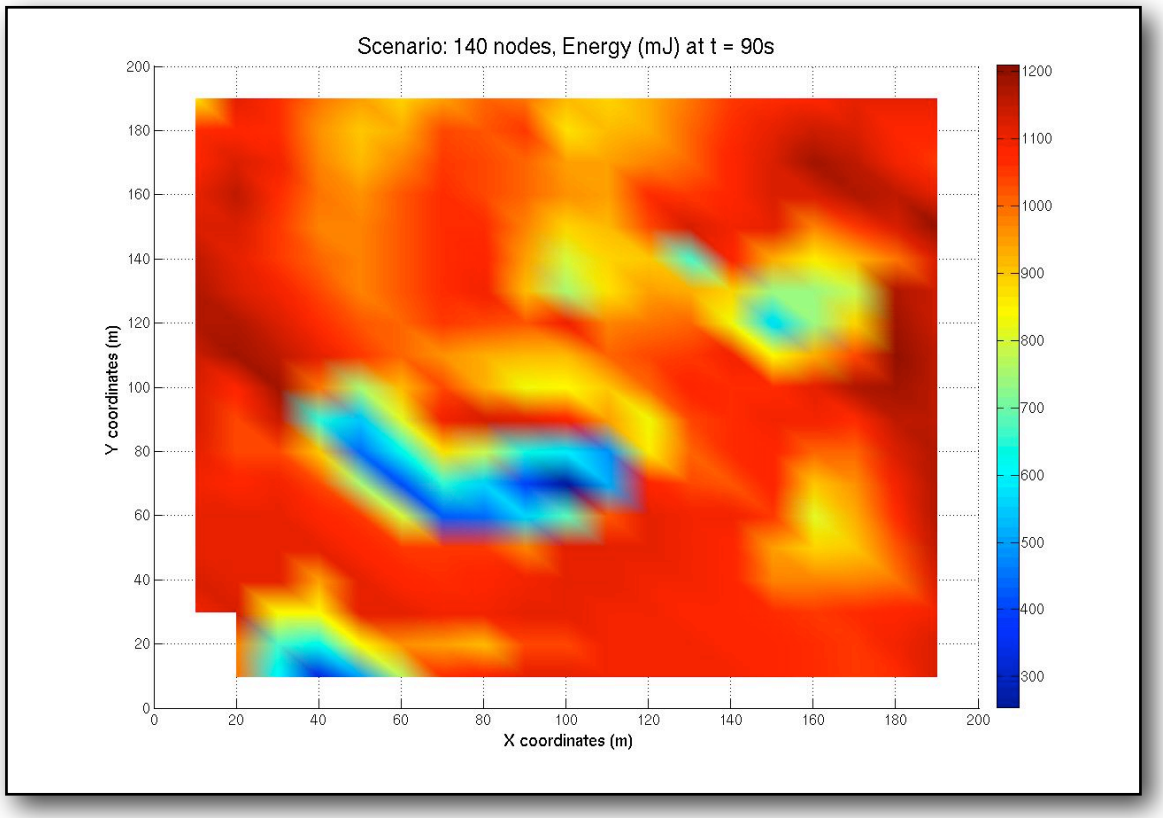


Figure C.44: Energy distribution at 90 simulation seconds - 140 nodes

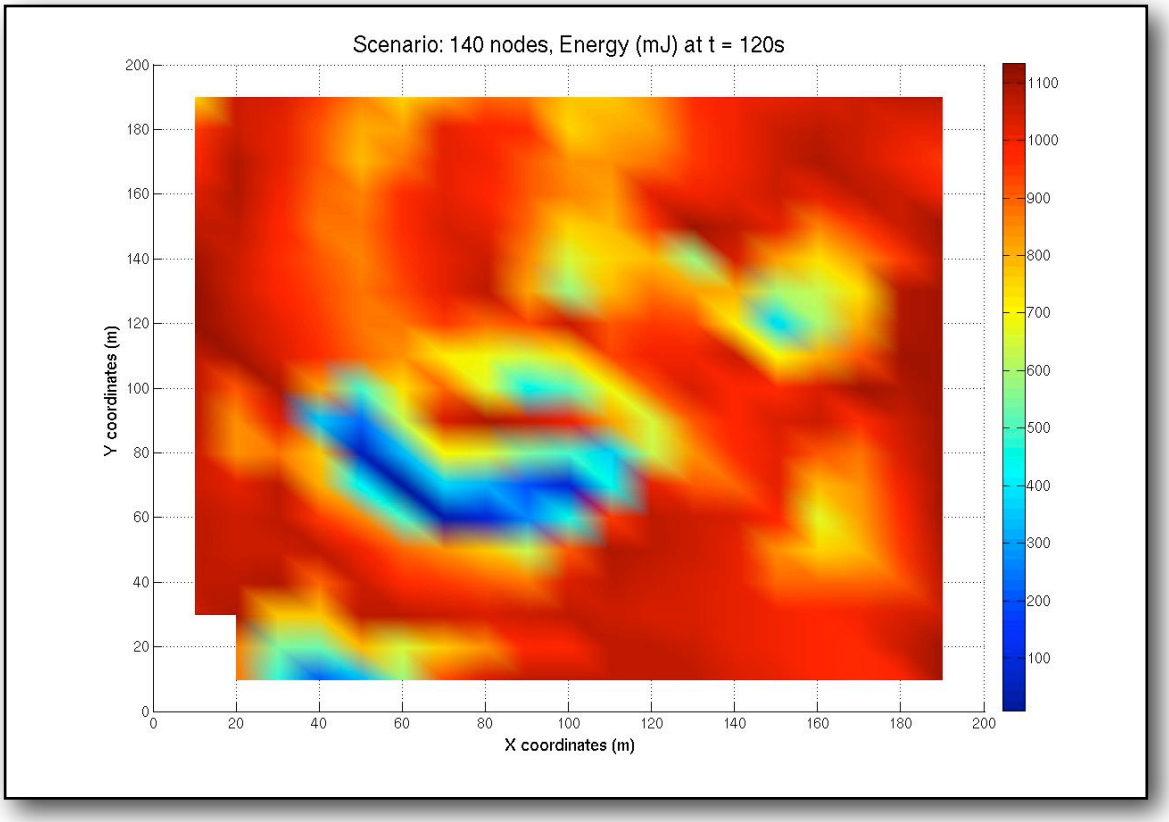


Figure C.45: Energy distribution at 120 simulation seconds - 140 nodes

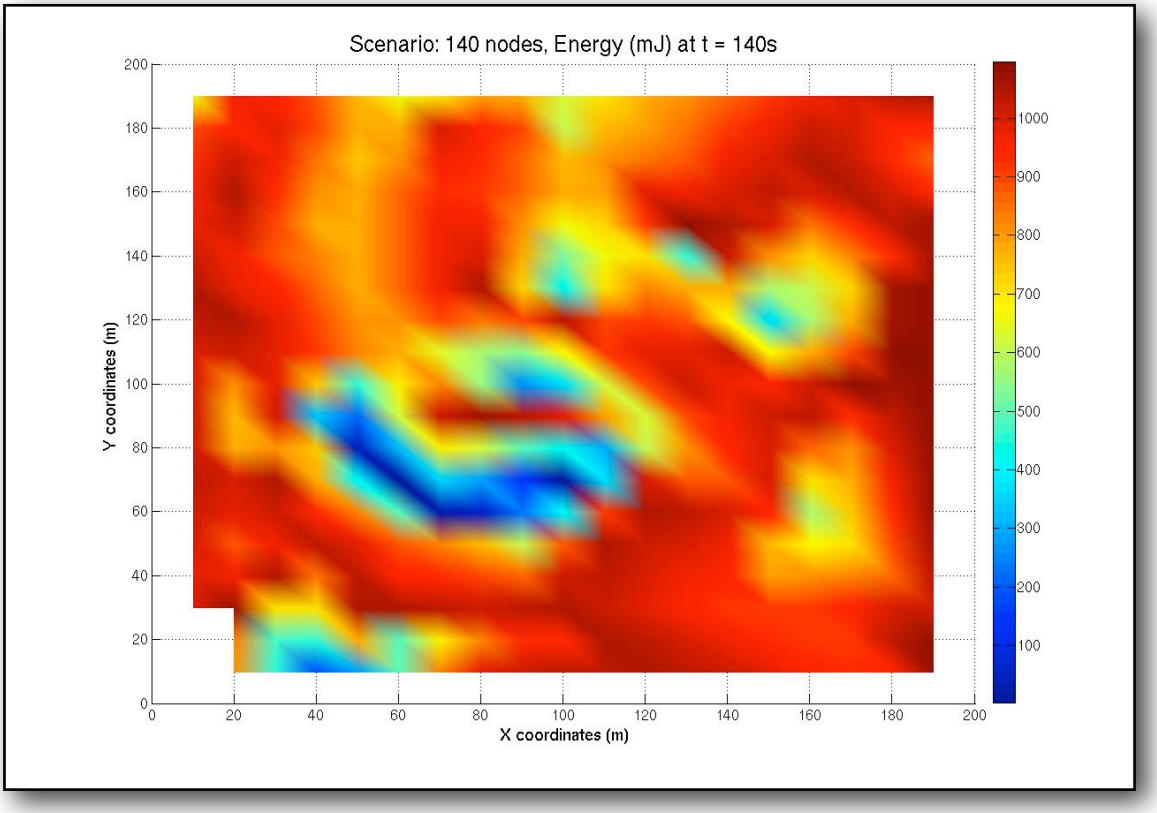


Figure C.46: Energy distribution at 140 simulation seconds - 140 nodes

C.5.2 Refractive index distribution

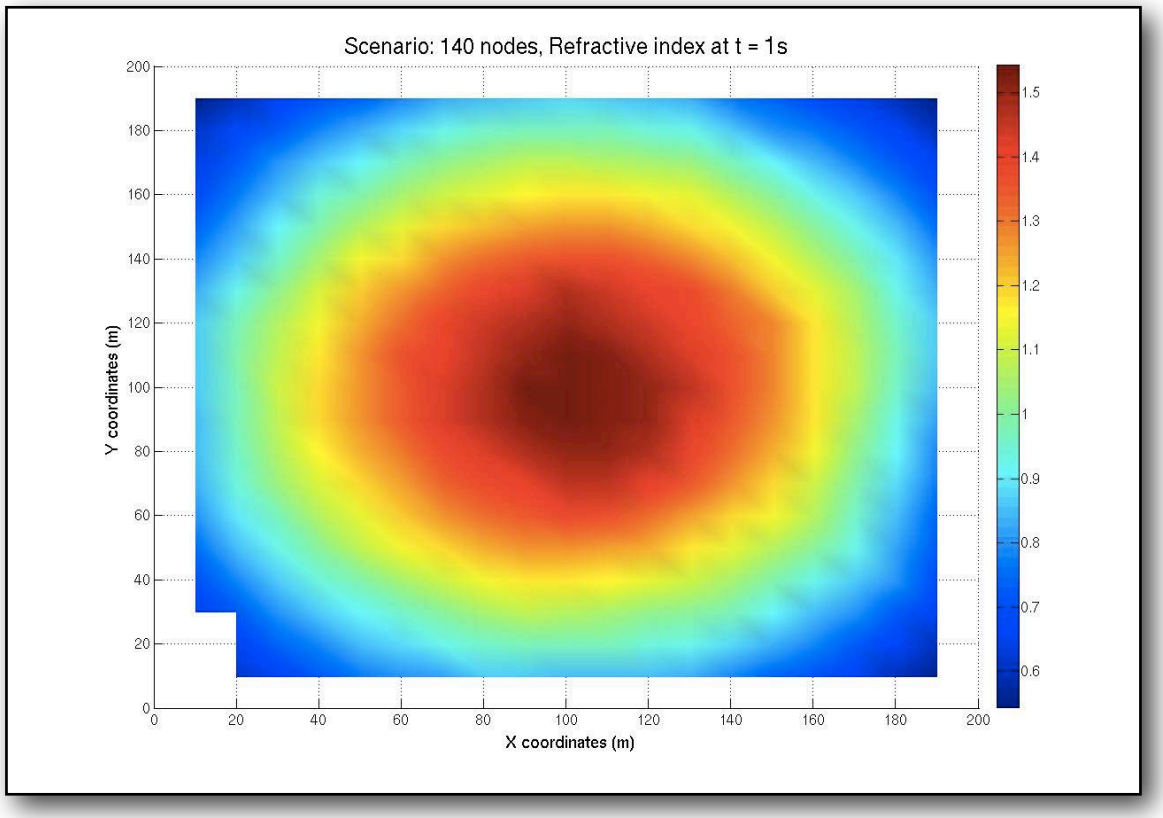


Figure C.47: Refractive index distribution at 1 simulation seconds - 140 nodes

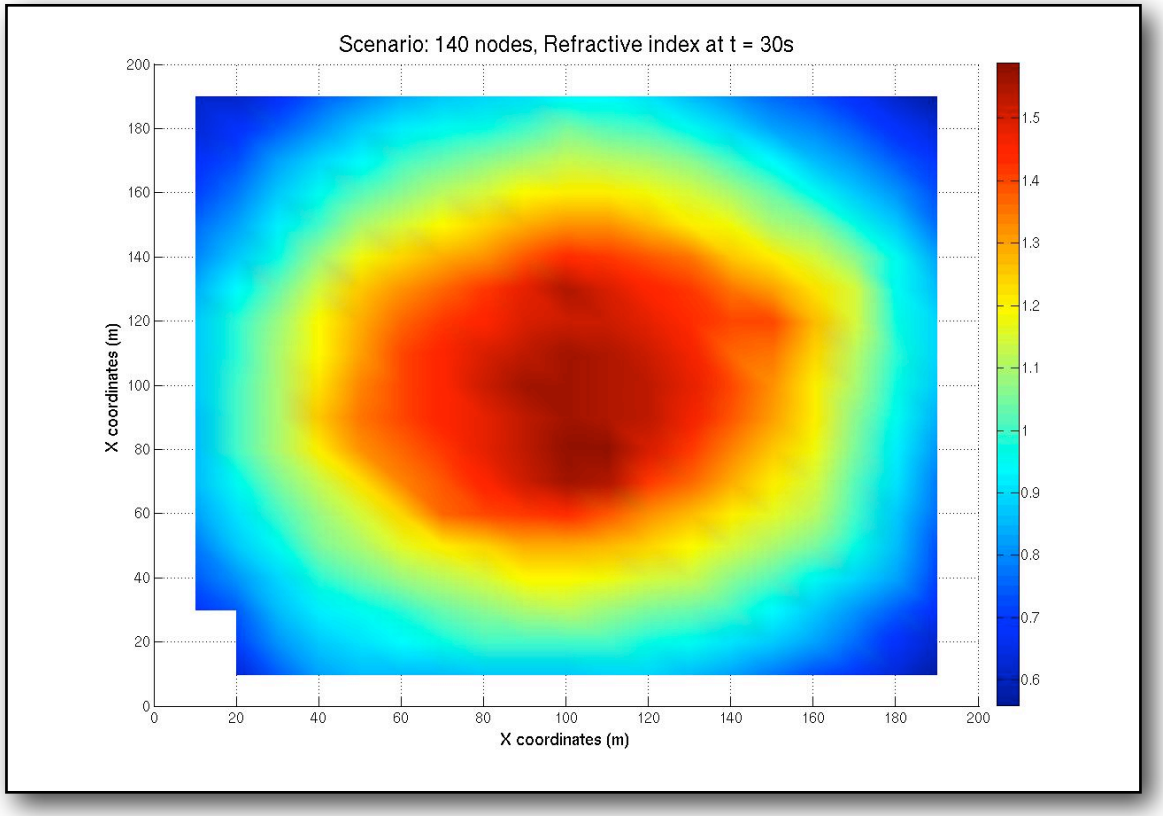


Figure C.48: Refractive index distribution at 30 simulation seconds - 140 nodes

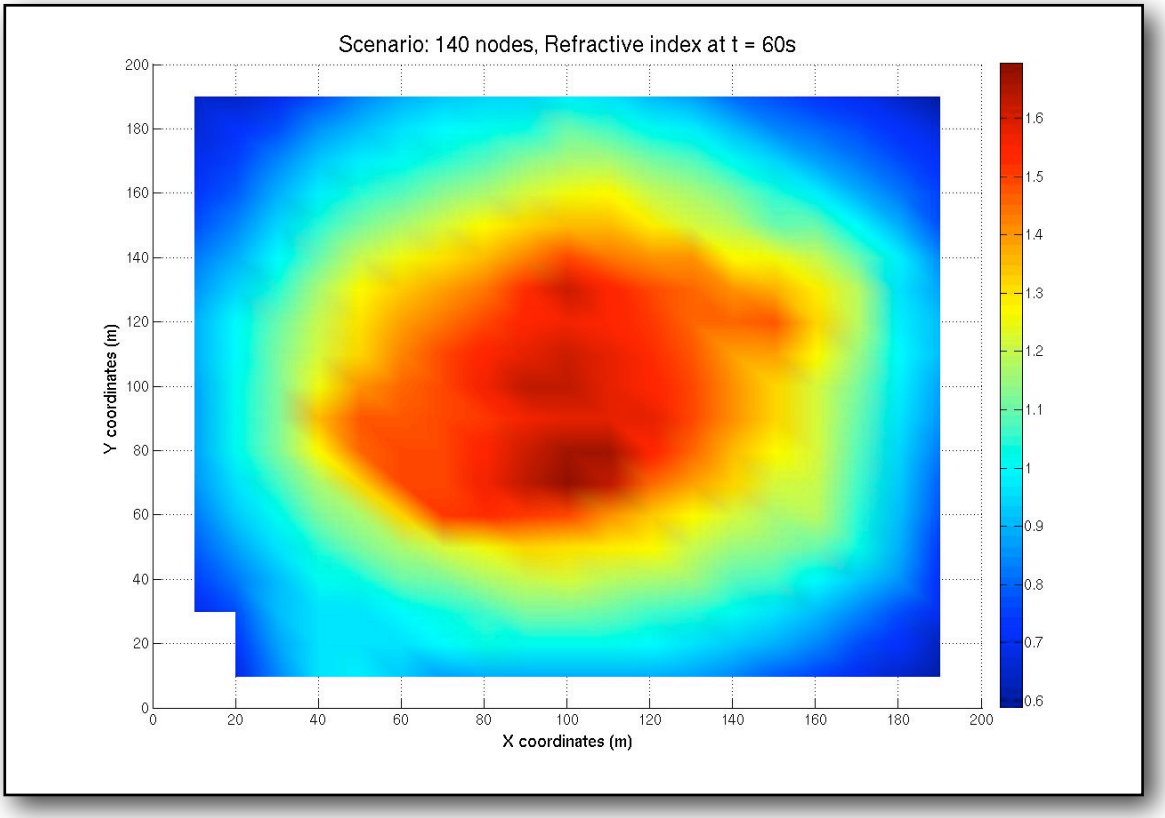


Figure C.49: Refractive index distribution at 60 simulation seconds - 140 nodes

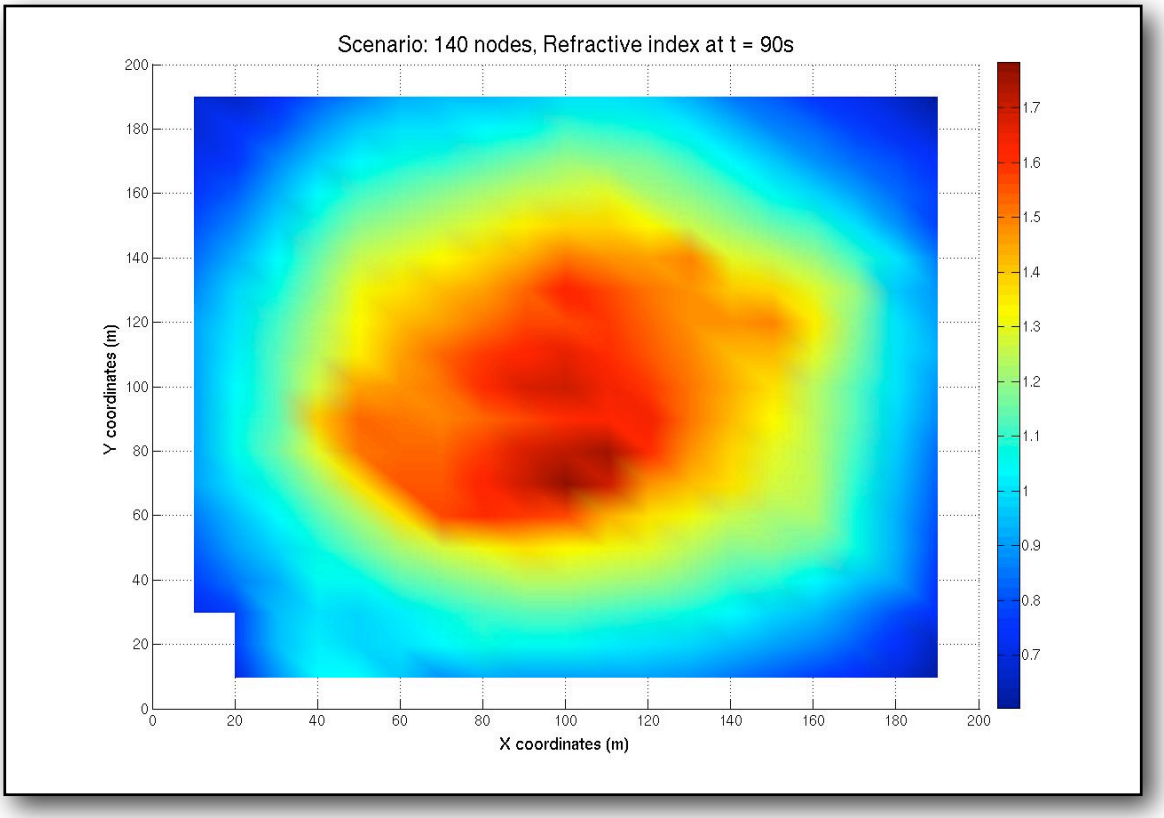


Figure C.50: Refractive index distribution at 90 simulation seconds - 140 nodes

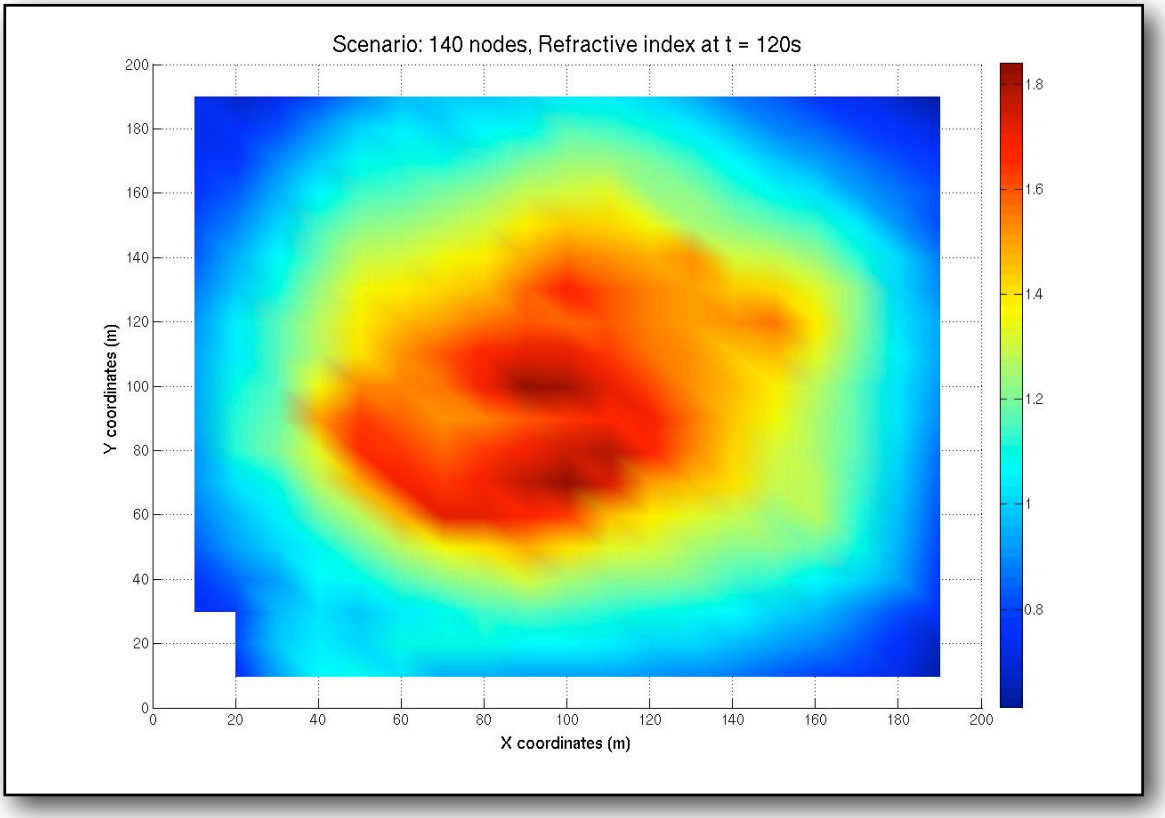


Figure C.51: Refractive index distribution at 120 simulation seconds - 140 nodes

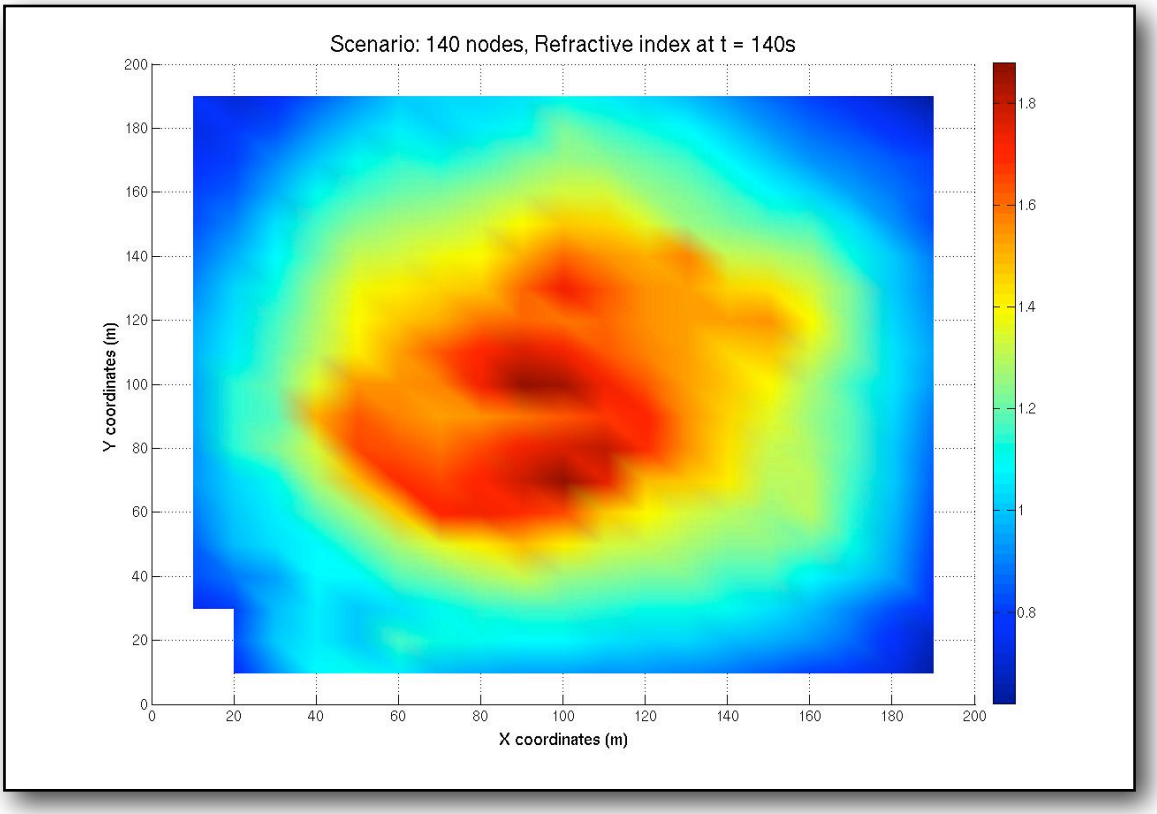


Figure C.52: Refractive index distribution at 140 simulation seconds - 140 nodes

C.6 50-node scenario (s-d pairs)

C.6.1 Energy distribution

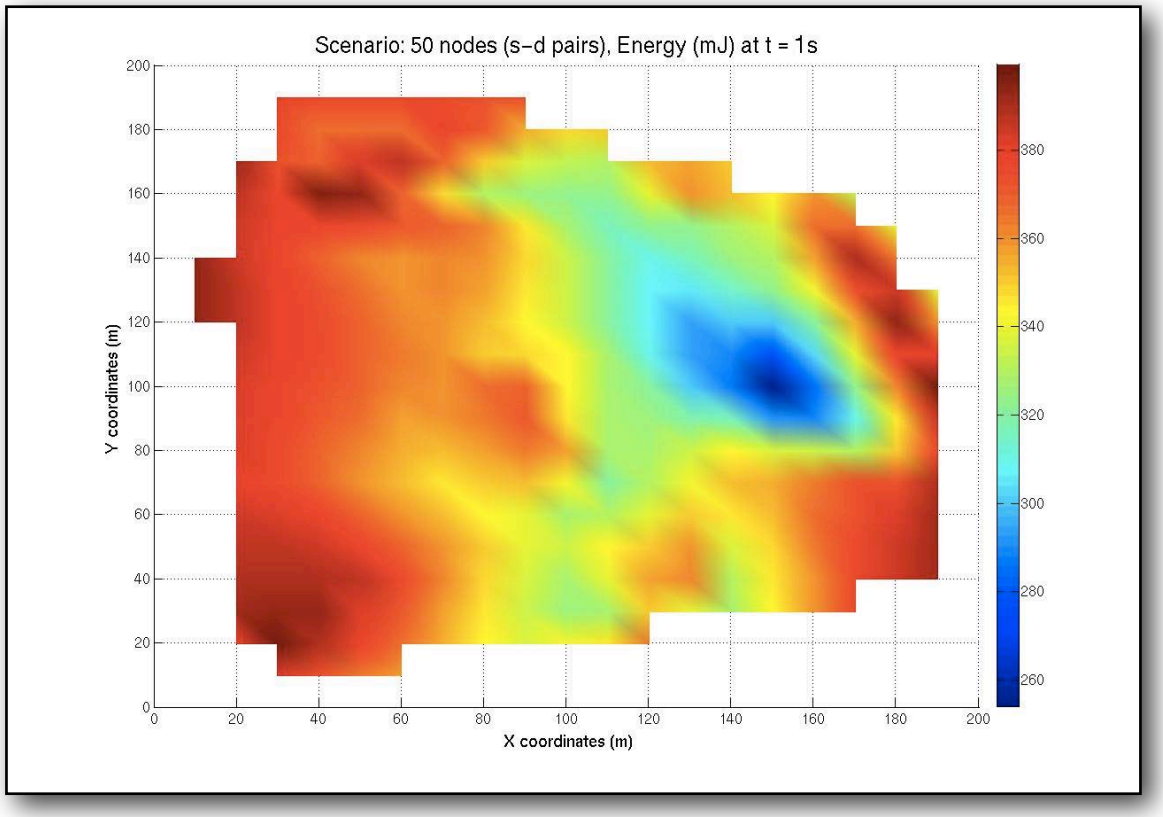


Figure C.53: Energy distribution at 1 simulation seconds - 50 nodes (s-d pairs)

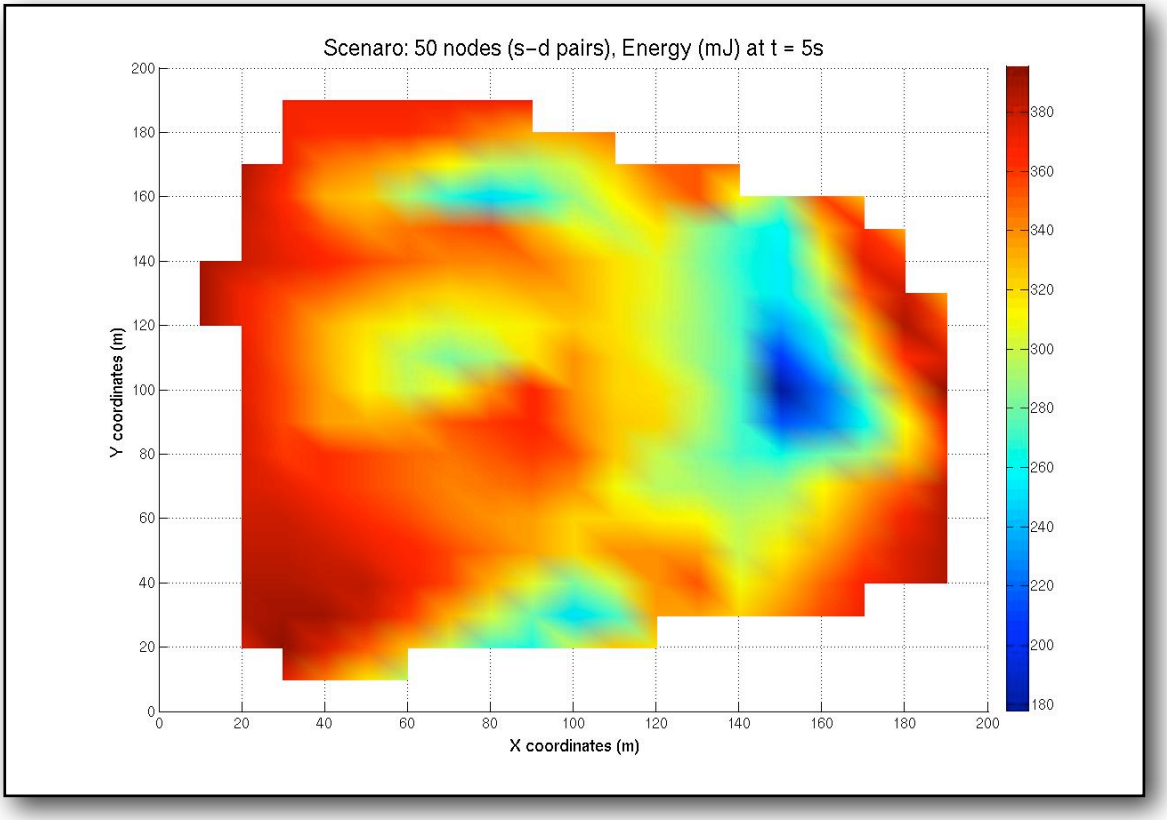


Figure C.54: Energy distribution at 5 simulation seconds - 50 nodes (s-d pairs)

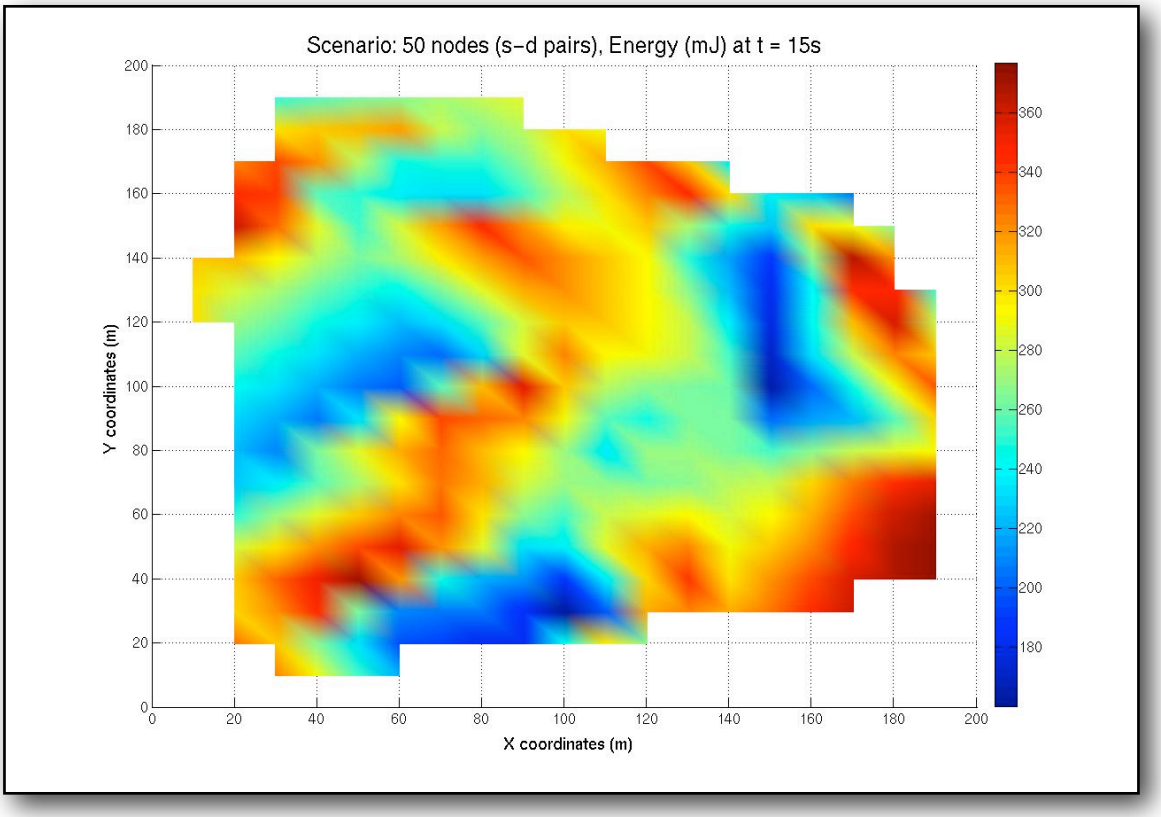


Figure C.55: Energy distribution at 15 simulation seconds - 50 nodes (s-d pairs)

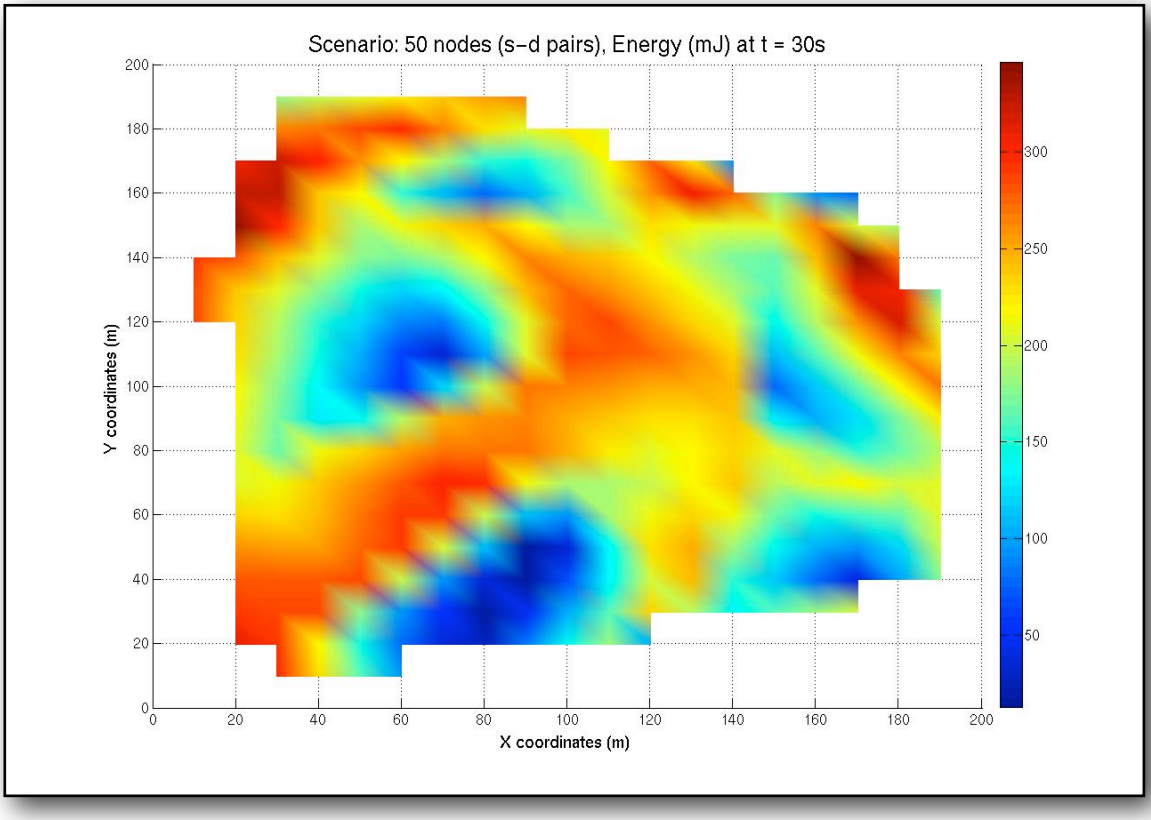


Figure C.56: Energy distribution at 30 simulation seconds - 50 nodes (s-d pairs)

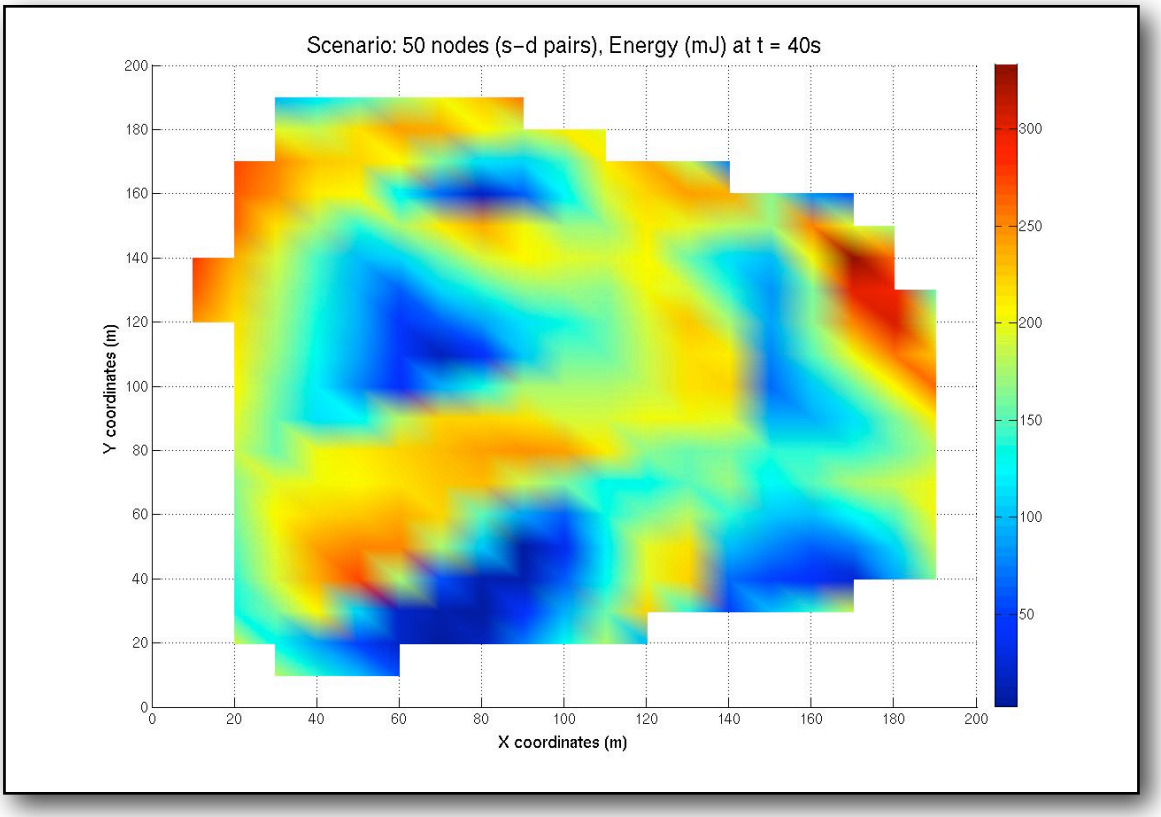


Figure C.57: Energy distribution at 40 simulation seconds - 50 nodes (s-d pairs)

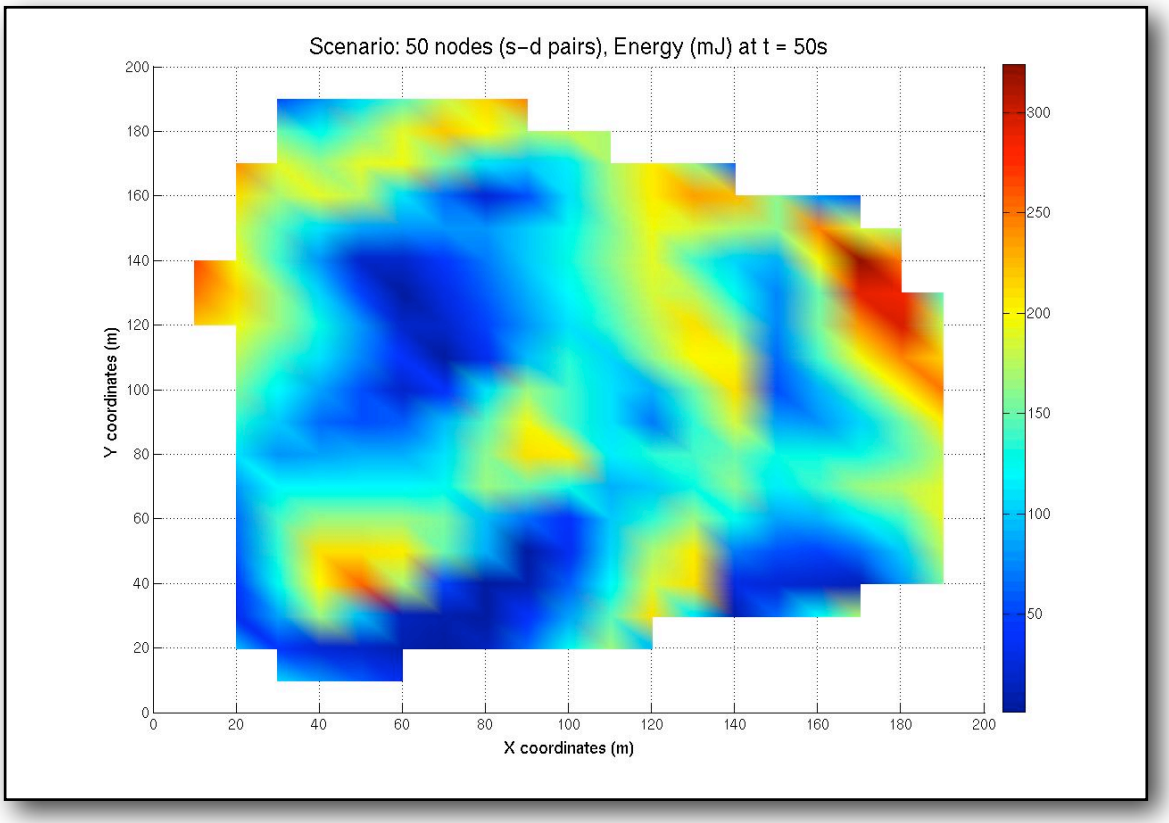


Figure C.58: Energy distribution at 50 simulation seconds - 50 nodes (s-d pairs)

C.6.2 Refractive index distribution

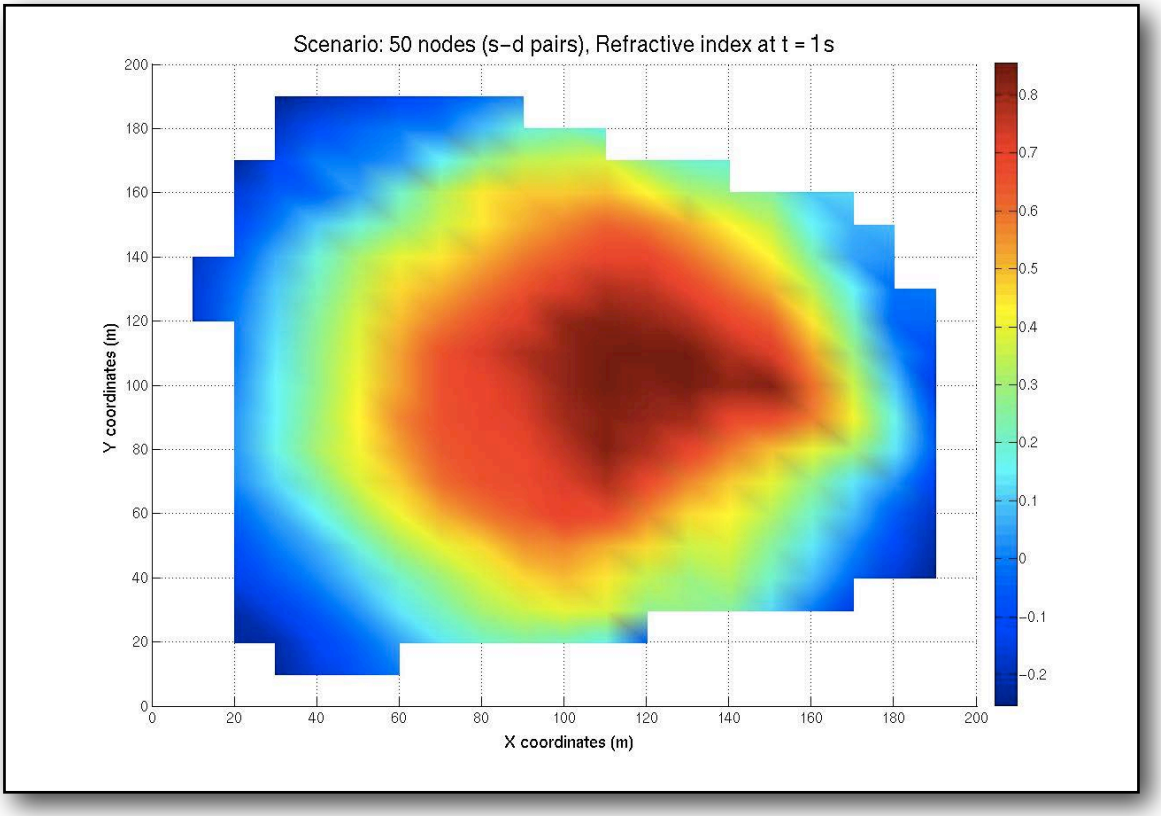


Figure C.59: Refractive index distribution at 1 simulation seconds - 50 nodes (s-d pairs)

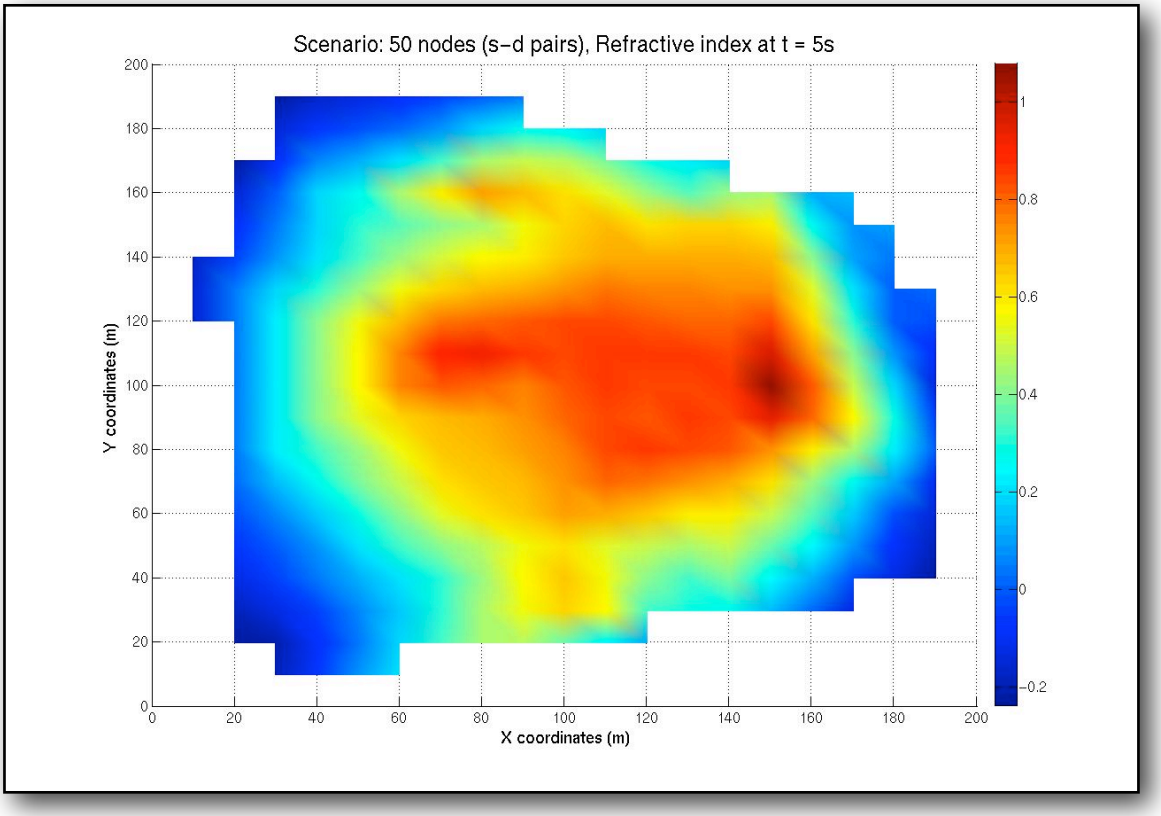


Figure C.60: Refractive index distribution at 5 simulation seconds - 50 nodes (s-d pairs)

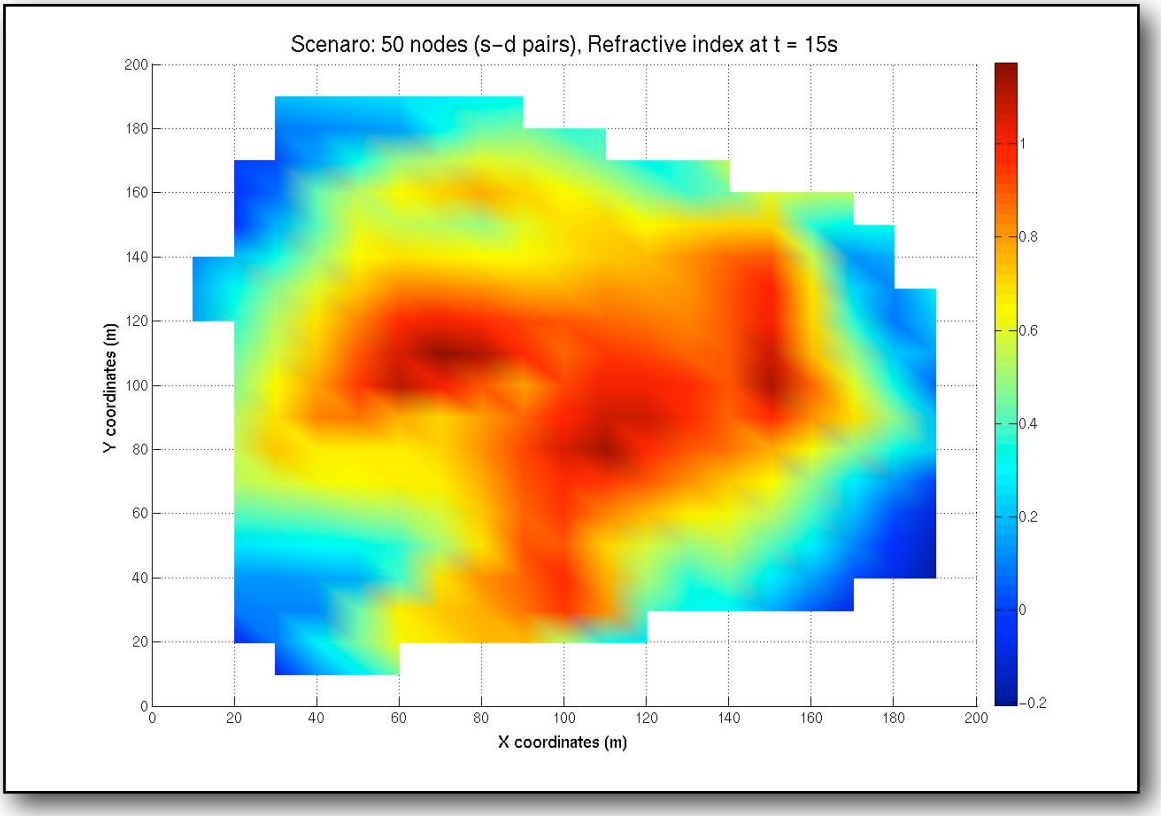


Figure C.61: Refractive index distribution at 15 simulation seconds - 50 nodes (s-d pairs)

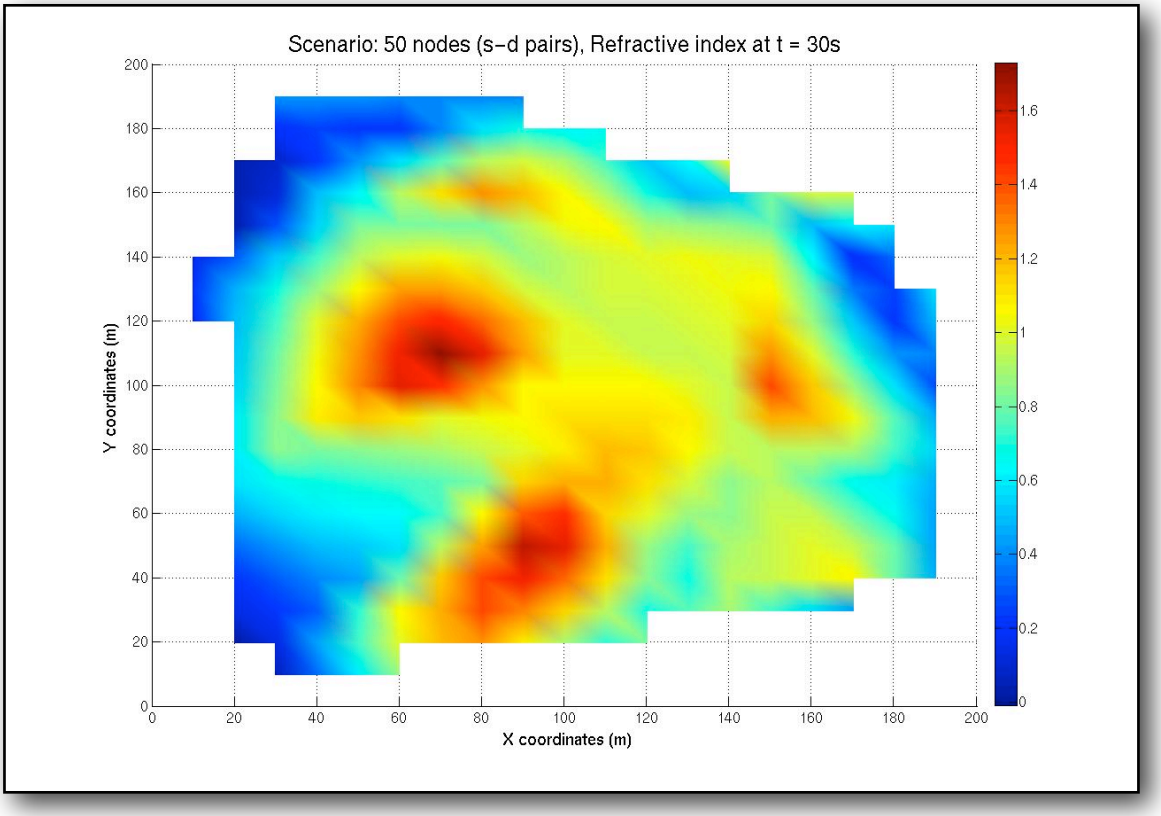


Figure C.62: Refractive index distribution at 30 simulation seconds - 50 nodes (s-d pairs)

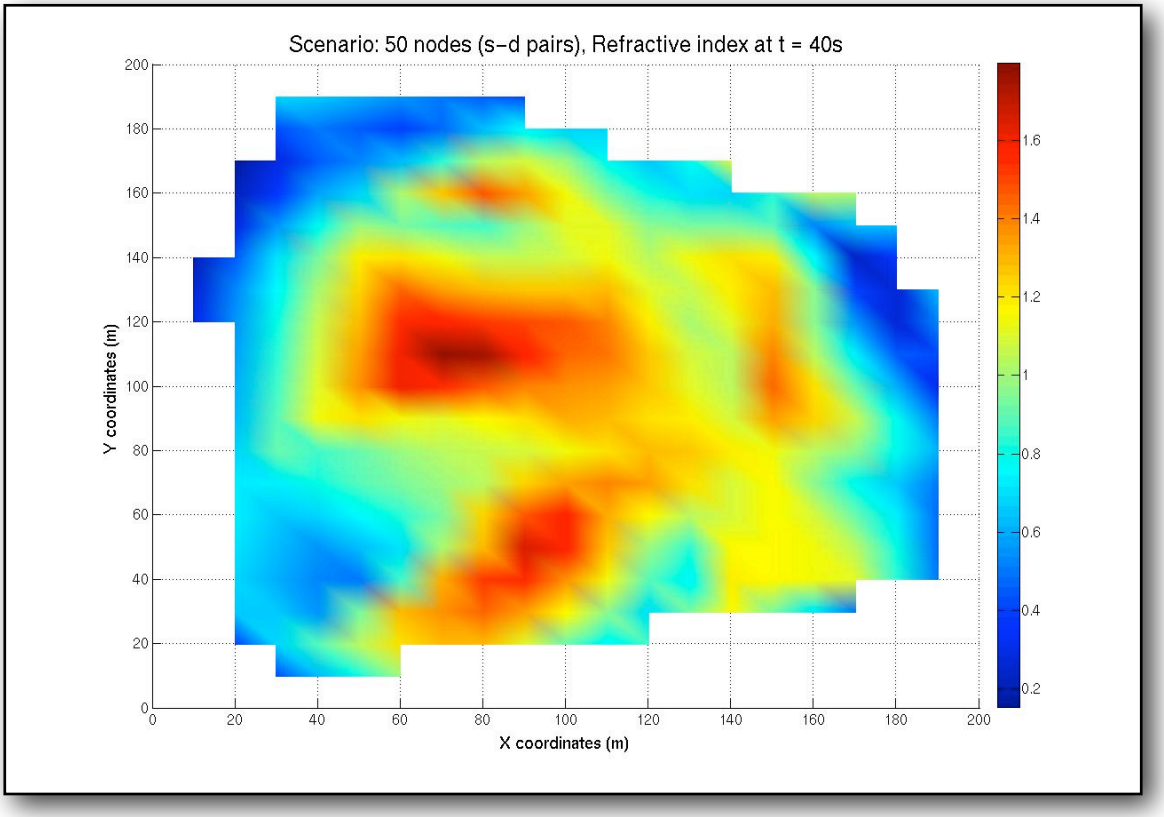


Figure C.63: Refractive index distribution at 40 simulation seconds - 50 nodes (s-d pairs)

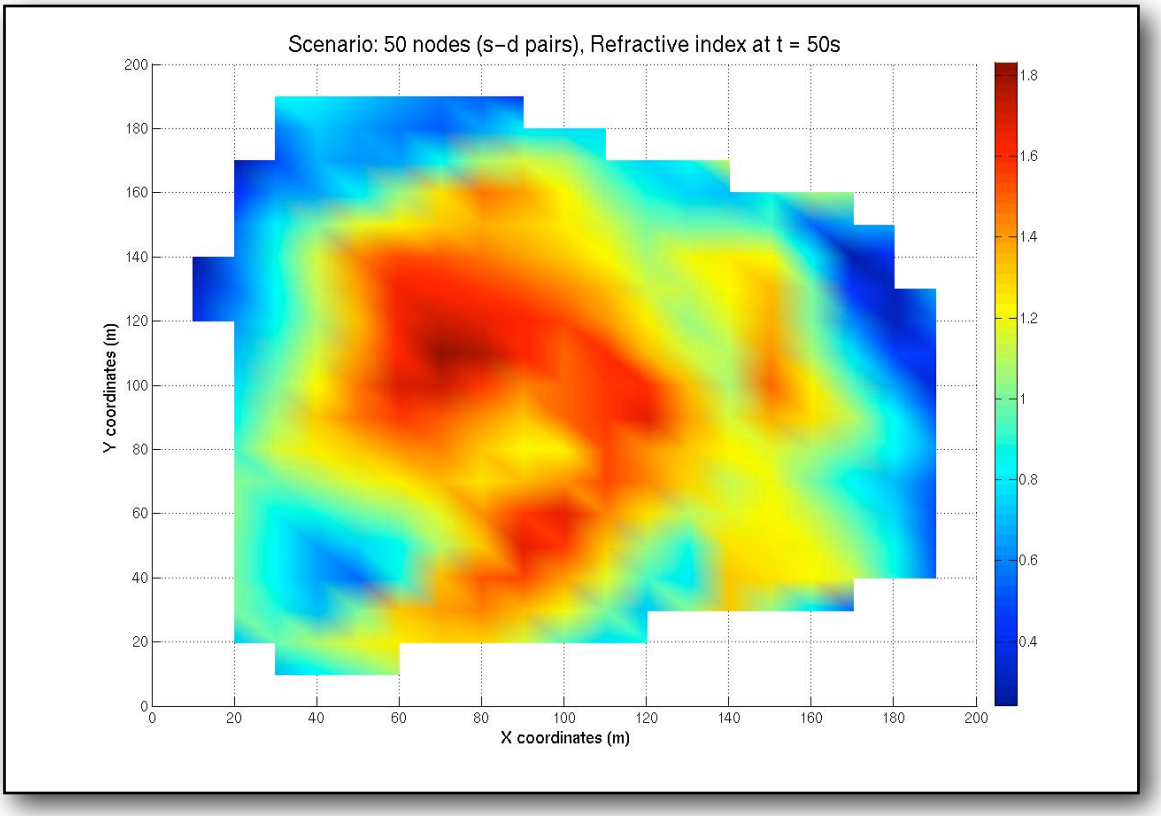


Figure C.64: Refractive index distribution at 50 simulation seconds - 50 nodes (s-d pairs)

APPENDIX D

Publications

Position paper:

Tsanaka, A. & Constantinou, C. (2008). Extending Network Lifetime in Wireless Sensor Networks. *Proceedings of PhD-Now Workshop on Ad-hoc and Wireless Networks*, Sophia-Antipolis, France.

References

Akkaya, K. & Younis, M. (2005). Energy and QoS Aware Routing in Wireless Sensor Networks. *Journal of Cluster Computing 2005*, *8*(2-3).

Born, M. & Wolf, E., (1970). *Principles of Optics*. 4th ed. Oxford: Pergamon Press.

Bose, P., Morin, P., Stojmenovic, I. & Urrutia, J. (1999). Routing with Guaranteed Delivery in Ad Hoc Wireless Networks. *Proceedings of the 3rd ACM International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, DIAL M,* 48-55.

Buratti, C., Conti, A., Dardari, D. & Verdone, R. (2009). An Overview on Wireless Sensor Networks Technology and Evolution. *Sensors 2009, 9*(9), 6869-6896.

Catanuto, R., Morabito, G. & Toumpis, S. (2006). Optical Routing in Massively Dense Networks: Practical Issues and Dynamic Programming Interpretation. *Proceedings of IEEE ISWCS,* 83-87.

Catanuto, R., Toumpis, S. & Morabito, G. (2007). Opti{c,m}al: Optical/Optimal Routing in Massively Dense Wireless Networks. *Proceedings of the 26th IEEE International Conference on Computer Communications*, 1010-1018.

Clark, B.N., Colbourn, C.J. & Johnson, D.S. (1990). Unit Disc Graphs. *Discrete Mathematics, 86*(1-3), 165-177.

Cormen, T.H., Leiserson, C.E., Rivest, R.L. & Stein, C. (2009). *Introduction to Algorithms*. 3rd ed. Cambridge: MIT Press & McGraw-Hill.

Diestel, R. (2005). *Graph Theory*. 3rd ed. Heidelberg: Springer.

Dijkstra, E. W. (1959). A Note on Two Problems in Connection with Graphs. In: *Numerische Math.1*, 269-271.

Duarte-Melo, E.J. & Liu, M. (2002). Analysis of Energy Consumption and Lifetime of Heterogeneous Wireless Sensor Networks. *Proceedings of the IEEE Global Telecommunications Conference GLOBECOM*, 21-25.

Frey, H. & Stojmenovic, I. (2006). On Delivery Guarantees of Face and Combined Greedy-Face Routing in Ad-hoc and Sensor Networks. *Proceedings of the 12th ACM International Conference on Mobile Computing and Networking, MobiCom*, 390-401.

Gupta, P. & Kumar, P.R. (1998). Critical Power for Asymptotic Connectivity. *Proceedings of the 37th IEEE Conference on Decision and Control*, 1106-1110.

Jacquet, P. (2004). Geometry of Information Propagation in Massively Dense Ad Hoc Networks. *Proceedings of the 5th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 157-162.

Kim, J., Lin, X., Shroff, N.B. & Sinha, P. (2010). Minimizing Delay and Maximizing Lifetime for Wireless Sensor Networks with Anycast. *IEEE/ACM Transactions on Networking*, *18*(2), 515-528.

Langkemper, S. (2006). Scalability of Routing Protocols in Wireless Ad-hoc Networks. *LinuxOnly online blog 2006*.

Le, H.P., Johns, M. & Pister, K. (2009). Energy-Aware Routing in Wireless Sensor Networks with Adaptive Energy-Slope Control. *Technical report submitted by Electrical Engineering & Computer Science Department, University of California, Berkeley*.

Leonhardt, U. (2009). Perfect Imaging without Negative Refraction. *New Journal of Physics 2009, 11*(1).

Marchard, E.W. (1978). *Gradient Index Optics*. New York: Academic Press.

Matlab. (2009). *Software version R2009a*. Available at: <http://www.mathworks.com>

Min, X., Wei-ren, S., Chang-jiang, J. & Ying, Z. (2010). Energy Efficient Clustering Algorithm for Maximizing Lifetime of Wireless Sensor Networks. *AEU - International Journal of Electronics and Communications, 64*(4), 289-298.

Mollerach, S. & Roulet, E. (2002). *Gravitational Lensing and Microlensing*. London: World Scientific Publishing Co. Pte. Ltd.

OPNET. (2007). *Software version Modeler 14.0*. Available at: <http://www.opnet.com/solutions/network_rd/modeler.html>

Paris, D.T. & Hurd, F.K. (1969). Basic Electromagnetic Theory. New York: McGraw-Hill.

Perkins, C. (2001). *Ad Hoc Networking*. New York: Addison-Wesley.

Popa, L. et al. (2007). Balancing Traffic Load in Wireless Networks with Curveball Routing. *Proceedings of the 8th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 170-179.

Press, W.H. (1986). *Numerical Recipes : the art of scientific computing*. Cambridge: Cambridge University Press.

Rabaey, J.M. & Shah, R.C. (2002). Energy Aware Routing for Low Energy Ad Hoc Sensor Networks. *Proceedings of the IEEE conference on Wireless Communication and Networking, WCNC, 1*(1), 350-355.

Schurgers, C. & Srivastava, M.B. (2001). Energy Efficient Routing in Wireless Sensor Networks. *Proceedings of the Military Communications Conference MILCOM*, 357-361.

Stojmenovic, I. (2002). Position-Based Routing in Ad Hoc Networks. *IEEE Communications Magazine, 40*(7), 128-134.

Talzi, I., Hasler, A., Gruber, S. & Tschudin, C. (2007). PermaSense: Investigating Permafrost with a WSN in the Swiss Alps. *Proceedings of the 4th ACM Workshop on Embedded Networked Sensors*, 8-12.

Toumpis, S. & Gitzenis, S. (2009). Load Balancing in Wireless Sensor Networks using Kirchhoff's Volatage Law. *Proceedings of the IEEE International Conference on Computer Communications INFOCOM* 1656-1664.

Vidhyapriya, R. & Vanathi, P.T. (2007). Energy Aware Routing for Wireless Sensor Networks. *Proceedings of the IEEE International Conference on Signal Processing, Communications and Networking ICSCN*, 545-550.

Xiaokang, Y. et al. (2011). Spherical Representation and Polyhedron Routing for Load Balancing in Wireless Sensor Networks. *Proceedings of the 30th IEEE International Conference on Computer Communications*, 621-625.

Yang, D., Li, X., Sawhney, R. & Wang, X. (2009). Geographic and Energy-Aware Routing in Wireless Sensor Networks. *International Journal of Ad Hoc and Ubiquitous Computing 2009, 4*(2), 61-70.