IRWIN AND JOAN JACOBS
CENTER FOR COMMUNICATION AND INFORMATION TECHNOLOGIES

Routing Protocols for Wireless Sensor Networks

Alex Aronski and Adrian Segall

CCIT Report #741
August 2009

Electronics
Computers
Communications

DEPARTMENT OF ELECTRICAL ENGINEERING
TECHNION - ISRAEL INSTITUTE OF TECHNOLOGY, HAIFA 32000, ISRAEL

CCIT REPORT #741 August 2009

Routing Protocols for Wireless Sensor Networks

Alex Aronski and Adrian Segall

This Report is based on a MSc thesis of Alex Aronski, supervised by Prof. Adrian Segall

Table of Contents

List of Figures

Abstract

Up-to-date technology makes possible the production of low-cost micro-sensor devices, which can perform short-range wireless communications and relatively complicated calculations. An ad-hoc network consisting of a large number of such devices, which are randomly distributed in a specified area, can be used in a variety of commercial and military applications. Such micro-sensor devices with small production expenses are battery powered and hence have extremely short lifetime. Therefore, the use of wireless sensor networks is ineffective without proper attention to utilization of energy resources.

Providing reliable and yet energy efficient routing protocols is of an utmost importance in Sensor Networks. Wireless Sensor Networks imply multi-hop data forwarding over unreliable and moving nodes. The main challenge is to find the right equilibrium point between quality of data delivery and the energy invested. Insufficient quality of data delivery may fail the application deployed over the wireless sensor network, while an energy wasteful protocol may significantly shorten the lifetime of the network, thus making the deployment inefficient for its purpose.

A wide range of routing protocols were proposed in the recent years, but only few of them take into consideration movement of the sensor nodes, focusing mainly on their unreliability or channel errors. Two main types of approaches to routing in WSN's can be identified in the literature: data-centric and path-based. In this work we will show that by borrowing a few concepts from data-centric protocols, it is possible to vastly improve the efficiency of classic path-based approaches. Our simulations show that the suggested routing protocol, which we name Data Centric Braided Multipath (DCBM), is well suited to handle routing in Wireless Sensor Networks with moderate sensor node movement.

List of Abbreviations

AODV – Ad-hoc On-demand Distance Vector

DCBM - Data Centric Braided Multipath

DD – Directed Diffusion

DD/SIR – Directed Diffusion with Stepwise Interest Retransmission

DSDV – Destination-Sequenced Distance Vector

DSR - Dynamic Source Routing

GRAB – Gradient Broadcast

MAC – Media Access Control

MANET – Mobile Ad-Hoc Network

WSN – Wireless Sensor Network

RCDR – Reliable Cost-based Data-centric Routing

Chapter 1 - Introduction

1.1 Wireless Sensor Networks (WSN)

Developing technology has lead in the recent years to availability of small, cheap nodes with considerable computation, sensing and communication capabilities. Various applications relying on networks of such nodes were suggested in the literature. A Sensor Network can be quickly and easily deployed and thus is suitable and very attractive for many environmental, commercial and military applications. A general-purpose sensor network is commonly a dense network that consists of a large number of possibly mobile energy-constrained nodes and it is likely to be deployed in difficult access regions, while being remotely operated by only a few operators.

Energy constraints are one of the main considerations when deploying applications based on wireless sensor networks. The leading aspect of energy consumption is communication. The communication energy waste is mostly due to the following factors:

- Control overhead (example: route establishment and maintenance)
- Overhearing (receiving packets destined for other nodes)
- Packet collision and collision avoidance techniques (example: RTS, CTS in 802.11)
- Idle listening to the medium

There are many related areas of research, for instance:

- MAC layer optimization; examples of this research are [1], [2], [3], [4].
- Network coverage; not all nodes in the coverage area of the sensed event are required to monitor the event, some of them may be placed in sleeping mode until they are needed for reinforcing the density of the sensing network. An example of this area of research is [5].
- Clustering; in order to reduce the amount of messages forwarded to sinks, it is suggested to establish cluster heads, whose purpose is to perform data accumulation and correlation. The sensor network is divided into clusters of nodes and all data produced by nodes in a cluster is transmitted to the cluster head. Example of this area of research is [6].
- Route establishment and data forwarding; efficiency of the route establishment and of its utilization during data forwarding can significantly reduce both control and overhearing overhead. The present work focuses on this aspect.

Our research concentrates on data forwarding and route establishment for mobile WSN’s. Examples are WSN's that comprise units designed for animal monitoring and surveillance or sensor nodes included in personal identification cards and tags. The latter allow monitoring of the personal wellbeing at workplace and retrieving his/her location in case of disaster. Mobile WSN’s present additional challenges on top of the previously listed. Many of the proposed protocols rely on regional/geographic awareness, which, in mobile WSN's, is energy consuming and very hard to obtain and update. When the location of the units is changing, maintaining topology information leads to excessive energy depletion. We believe that in such networks, routes should be managed mainly as a function of the quality of data delivery.

1.2 WSN's versus classic mobile ad-hoc networks (MANETs)

In this section, we shall point out the unique characteristics of WSN’s and indicate the need for research of new special-purpose routing protocols. We shall also briefly analyze the main routing algorithms proposed for MANETs and their weaknesses when applied to WSN’s in general and to mobile WSN’s in particular.

The WSN concept suggests many small and cheap nodes sensing the environment and reporting back to the information aggregation point, referred to as the *sink* or *base station*. The role of the sink is to correlate the information and to create the most accurate presentation of the sensed environment. The main difference from general MANETs is that most of the traffic is destined to the sink. Because of this difference, leading WSN-focused protocols employ some sort of global data dissemination paradigm, thus avoiding route discovery for each node pair.

1.2.1 AODV

AODV [7], [8] is a reactive protocol. Route Request (RREQ) messages are used to initiate a route discovery to the destination. Route Request is identified by the {source_addr, dest_addr, broadcast_id} triple and only the first packet with the same identifier is forwarded. Intermediate nodes with a recent route to the destination or the destination itself reply with a Route Reply (RREP) message. RREP is a unicast message and is forwarded along the reverse path as established by the RREQ. Multiple RREP’s can be created in the network; each node forwards the RREP only if its sequence number is larger than the previously forwarded one or if it carries the same sequence number while the hop count to the destination is improved. The neighbors are maintained via hello messages and lack of such message indicates a change in the neighboring relationship. If such change affects an active route, a disruption message is issued and sent to the source, notifying the need for a new path discovery.

The drawbacks of AODV in our environment are:

- For each source node, a global broadcast will be initiated. Although an intermediate node with an updated path to the destination will not rebroadcast the packet, in scenarios of hundreds and thousands of nodes, the broadcast may prove very costly.
- Nodes forward only the first RREQ with a given sequence number, thus inefficient routes may be selected.
- Only one route is maintained. If a path is broken at any point, which may often be the case in networks with moving nodes, this results in a global RREQ broadcast. Multipath routing based on AODV, combined with local route repair, was suggested in [9], thus resulting in significant improvement in control overhead.
- Nodes maintain neighbor lists by employing hello messages, another energy consuming mechanism.

1.2.2 DSR

DSR [10] – Dynamic Source Routing has a path discovery mechanism somewhat similar to AODV. Whenever a route is required to the destination, a route request packet is sent. It contains source, destination and route record, the latter listing all nodes traversed by the request. If the intermediate node does not have a route to the destination, it adds itself to the route record and re-broadcasts the packet, else it issues a route reply with the cashed route. The source routes the packets on a path that is a concatenation of the route record and the cashed route. DSR has similar drawbacks to AODV, and in addition it creates large packet headers, thus heavily affecting packet transmission energy. As showed in [11], DSR it is not suited for mobile WSN scenarios. Article [11] suggests some techniques to improve the performance of DSR in a mobile WSN environment. Another attempt on improving DSR performance was made at [12], where the main target has been to decrease the size of the forwarded packets by storing information at the nodes. The suggested algorithm also provides multiple disjoint paths based on the DSR route discovery technique.

1.2.3 DSDV

DSDV [13] – Destination-Sequenced Distance Vector is a proactive algorithm. It builds a global routing table by employing a Distributed Bellman-Ford (DBF) algorithm and uses periodic updates to maintain the routes. The features that have been added to make the algorithm suitable to the ad-hoc environment are sequence numbers to avoid loops (instead of split horizon and poison-reverse techniques) and prevention of unstable link information forwarding.

The main drawbacks in our environment are:

- Periodic global updates
- Maintenance of unused routes, thus wasting the network energy on maintaining areas of the network not used for information forwarding, which can prove very costly in WSNs with or without node mobility.

Article [14] compares AODV and DSDV for both mobile and static environments. DSDV performs better, but still the authors perceive the need for a different type of solution for mobile WSN networks.

1.3 WSN-related previous works

As previously stated, the goal of many WSN dissemination protocols is to create paradigms that allow multiple data sources to communicate to a single destination. These protocols generally use a data-centric paradigm, meaning that the routing maintenance and the forwarding of data are mainly based on the data itself and on the quality of its delivery. Topology changes do not affect maintenance decisions if they do not influence data delivery. This type of protocols is also called *Query-responsive* [15]: first query messages initiated by the sink are used for distributing information about data requested by the sink and for route establishment and then data is disseminated via the established routes.

Two classes of approaches can be identified: the first class is characterized by the lack of communication hierarchy, the second employs hierarchy. The main advantage of communication hierarchy is the correlation of accumulated data. Hierarchical solutions also seem to scale better, but the maintenance of the hierarchy is a significant drawback in mobile networks. In the present work we shall focus on a non-hierarchic approach. Further work should be performed to indicate if those two approaches can be combined together. As shown in [6], even a simple multihop routing for intra and inter cluster communication improves performance of hierarchical solutions in static WSN environments.

When examining the non-hierarchical protocols, two types of protocols [15] emerge.

- *Reverse-path-based forwarding* - In this approach, data reports flow towards the sink, namely in the direction opposite to the query propagation. The sink sends out a query message that expresses its interest, normally using flooding. Whenever a node receives a query from a neighbor, it sets up a forwarding state in the form of a pointer from itself to the neighbor, i.e., indicating the reverse data path. Data reports generated by sources travel along the pointers from one node to the next, until they reach the sink.

- *Cost field-driven dissemination* - Cost-field based forwarding offers an alternative approach to data dissemination. In this approach, the forwarding states of the nodes consist only of the cost denoting the distance to the sink, measured in certain units, like hop count, expected energy consumption or physical distance. The cost value is directionless, but implies directions, in the sense that data from each node can flow only to neighbors with smaller cost.

In this section, we shall briefly review protocols of both types and their treatment of mobile WSN characteristics.

1.3.1 Directed Diffusion – DD

Directed Diffusion [16] is the first protocol to have employed a data-centric communication paradigm. The routing of the data packets is performed by data diffusion towards the sink, based on the data properties and end-to-end delivery service. Instead of creating a general-purpose routing scheme, DD provides several design choices that allow adapting the paradigm to task-specific applications.

The idea of the protocol comes from the observation of biological systems, like ant colonies. At this point, it is worth mentioning another biologically inspired protocol: Bee-Inspired Power Aware Routing Protocol for Wireless Sensor Networks (BeeSensor) [17].

1.3.1.1 Path Establishment and Maintenance

DD is based on *reverse-path-based forwarding*. The first part of the Query-responsive cycle is initiated by the sink. The sink is responsible to inform the sensors what is the *query-interest*. The latter can be for instance: “Monitor targets moving with a speed greater than 1 km/h”.

The initial query – *exploratory interest* is flooded. The *interest* is propagated in the network and at the same time a reverse pointer is created by the node, thus establishing a *gradient* towards the node the *interest* was received from. The latter specifies the direction for data forwarding. The interests are recorded with a timestamp (and a duration time to render them inactive in the future). The *interest* is rebroadcast by the node if no similar interest was recently broadcast. *Gradients* that are established by the propagation of the *exploratory interest* usually demand low rate notification and their purpose is to allow detection of the requested information sources. When a sensor senses an event of a requested data type, the response is diffused via the established local *gradients* towards the sink.

After the initial batch of data responses is collected at the sink, it initiates *reinforcement interests* to select the path with the best data forwarding quality according to some metric (examples: minimum delay, hop count, energy consumed). Positive *reinforcement interests* are sent by the sink and resent by the intermediate nodes only to the neighbor that has delivered the data with the best metric. Basically the *reinforcement interests* increase the data rate of the selected path.

The *gradients* established during the *exploratory* phase are used as alternative paths. The source and the intermediate nodes always send a small portion (a low data rate) of data via all gradients in order to maintain multiple paths. If the data forwarding quality of the main path deteriorates, the sink initiates negative *reinforcement interests* to lower its data rate and positive *reinforcement interests* to increase the data rate of an alternative path. In case there are no paths that provide a sufficient data forwarding quality, a new *exploratory interest* is flooded.

1.3.1.2 Data propagation

Each Sensor broadcasts data packets to all neighbors and the received data packets are cached in order to prevent loops. If the receiving node has neighbors with *gradients* for the provided data type, it forwards the data according to the *gradient* parameters (for example if the gradient parameter is data rate, and node A has a neighbor B with gradient 10 and a neighbor C with gradient 1, only one in ten packets will be forwarded via C).

1.3.1.3 Recent work and performance analysis

In the original work [16], DD was shown to perform better than another algorithm considered in the same work: Flooding and Omniscient Multicast. Since then a large amount of work has been invested in developing various improvements to the original directed diffusion, for example clustering and energy efficient/aware path reinforcement.

Mobility was ignored in [16] and in many of the later articles. For example in [18], the SEER protocol was shown to be more effective in terms of prolonging lifetime of the network. However, as the authors mention, SEER does not support mobile environments.

DD has the mechanism to cope with topology changes by resending interests. In [19], the authors present a version of DD called DD/SIR – Directed Diffusion with Stepwise Interest Retransmission and achieve better efficiency in terms of control packets overhead. DD/SIR assumes that a significant delay is allowed between phenomena detection and the report to the sink, a property that holds in applications such as livestock management. The allowed delay is divided into a number of intervals used by the sink to poll the data sources. The number of intervals is set according to the maximum hop count to the most distanced data source. Data sources that are distanced n hops from the source and only those are polled in the n-th interval. It is also assumed that the topology does not change within the polling interval, therefore path establishment is performed in a similar manner to regular DD, but path reinforcement and maintenance is not required. As shown in [19], in mobile WSN's, DD/SIR outperforms the conventional DD by successfully reducing the control overhead.

1.3.2 Gradient Broadcast – GRAB

GRAB [20] is a *cost field-driven dissemination* protocol. In GRAB, nodes record a cost at the time when they forward the interests, thus building a cost field. Nodes with smaller costs are "closer" to the sink in terms of gradient. In GRAB no next hop is defined for the packet, instead, based on its cost to the sink, each node decides if it is allowed to forward a packet. To cope with node failures and radio channel errors, GRAB allows multiple copies to be forwarded along an interleaving multipath mesh, whose width is controlled by the credit parameter. Data with no credit is delivered only along the minimum cost path.

GRAB also deals with controlling the sensor network density, a topic that is not within the scope of the present work.

1.3.2.1 Building and Maintaining the Cost Field

In GRAB, interest messages are called ADV (advertisement) packets. The sink sends the first ADV packet with cost 0. Upon receiving the ADV, each node updates its cost as the sum of the received cost and the cost to the neighbor the ADV was received from. It then waits for a duration proportional to the cost to the neighbor the ADV was received from. Other ADV messages received during this period are processed similarly and the minimum sum is stored and broadcast in an ADV at the end of the period. For an ideal network with no unpredictable delays (processing or channel), article [20] proves that each node will rebroadcast only a single ADV and the obtained route is optimal. For a non ideal network, the article shows by simulation that only a limited number of nodes broadcast ADV packets more than once.

Maintaining the cost field is the sink prerogative and the quality of delivery is monitored for the following parameters: success ratio, number of duplicated data packets, average energy used to deliver a data packet and the average number of hops of the delivered data packet. Simulation shows that, in addition to the event driven approach, a scheduled field refreshing is required.

1.3.2.2 Data Propagation

In the second part, after establishing the cost field, each node may start forwarding data. Each packet is provided with a credit, which is an extra amount over the minimum cost to the sink. The packet is broadcast to all neighbors with the following parameters {credit, consumed cost, cost of the broadcasting node}. The credit parameter is constant, while the consumed cost is updated with each rebroadcast. In order to rebroadcast a packet, a node must have a lower cost than the cost of the node the packet was received from. An algorithm to control the width of the mesh is also suggested and is used as a second condition for packet rebroadcast. The algorithm is based on the relative amount of credit used, compared to the remaining cost to the sink. For further details see [20] and [21].

1.3.2.3 Performance analysis

In [20] GRAB was shown to supersede DD in static topologies with considerable node failures and moderate radio channel error rate. Only static WSN topologies are simulated. Article [22] compares GRAB with DD and TTDD [23]. Several conclusions can be deducted from [22]:

- GRAB routing overhead is the lowest and the event driven field refresh controlled by the sink is a highly efficient way to cope with major network changes.
- A high and unpredictable amount of duplicated packets in GRAB leads to redundancy and to overall worst energy consumption.
- DD has the lowest energy consumption among the tested protocols (GRAB, DD and TTDD).

1.3.3 The Reliable Cost-based Data-centric Routing protocol – RCDR

RCDR [24] is inspired by the GRAB protocol and is also a *cost field-driven dissemination* protocol. The target of RCDR is to improve GRAB performance in mobile WSN's. Local algorithms are suggested in order to cope with node mobility, by continuously rebuilding the network cost field. In RCDR as in GRAB, data packets are sent over multiple routes. It is important to note that RCDR assumes the SMAC [2] layer-2 protocol to allow management of the local neighbors table.

1.3.3.1 Building and maintaining the cost field

RCDR uses a building technique of the cost field that is simpler and less effective than in GRAB. The Data Query rebroadcast (equivalent to interest in DD and ADV packet in GRAB) is delayed for a random time. Only Data Queries received in this interval are processed. To maintain the cost field, a sensor movement adjustment mechanism is deployed. Each node manages a neighbors table. The Lowest Cost Neighbor (LCN) is found at the time of cost field establishment and when the LCN is lost, a node queries its neighbors for their cost and elects a new LCN. If a new neighbor is detected, the node sends a Cost Update (CU) message to notify the new neighbor with the node cost. To avoid loops, lost neighbor events are treated before new neighbor detection events.

1.3.3.2 Data propagation

The propagation mechanism is also simpler than in GRAB. Premium cost (equivalent to credit in GRAB) is added to the cost in the source node. The packet is broadcast and all nodes with cost lower than the one in the packet rebroadcast it further, after decreasing the cost.

1.3.3.3 Performance analysis

In [24] RCDR is shown to provide better delivery ratio and lower total energy consumption than GRAB, when simulated for mobile sensor nodes.

1.4 Intuition

Several conclusions can be deducted from the previously mentioned articles.

- If the network consists of a single data sink and multiple data sources, providing a global data forwarding scheme toward the sink is more energy efficient than the autonomous discovery for each source-sink pair.
- The focus of the routing protocol should be the data itself and not the network topology. The latter is important only for supporting data forwarding. Therefore all topology changes should be managed and maintained according to the quality of the data delivery at the sink node.
- We argue that maintaining topology in areas not used for data forwarding is energy wasteful; an algorithm that limits control messages to relevant network areas should be more efficient.
- We claim that *reverse-path-based forwarding* is more energy effective than *cost field-driven dissemination*, provided it ensures acceptable data delivery quality. The reason is that duplicating packets cause unnecessary energy depletion, except in networks with constant, severe radio channel interferences. This is partially shown in [22] for static WSNs.
- The design choice of *reverse-path-based forwarding*, if used in combination with braided multiple path establishment, should provide robustness to cope with mobile WSN's. In

[25], the authors suggest multipath routing as part of the Directed Diffusion paradigm and also show that braided multipath is more energy efficient than disjoint multipath.

Chapter 2 - The Data Centric Braided Multipath (DCBM) routing protocol

In this section, we shall present a new routing protocol, referred to as the Data-Centric Braided Multipath (DCBM) protocol, which was designed to take into consideration the conclusions of Section 1.4.

2.1 The main idea

The primary goal of DCBM is to provide resilient and energy efficient multipath routing between sensor nodes and a sink, while minimizing control message overhead:

- The sink initiates a two phase mechanism for the purpose of path establishment. In the first phase, control messages MSG1 are used to broadcast data queries, to carry metric data and to trigger the selection of *designated neighbors* – the neighbor with the best known distance to the sink. Therefore the *designated neighbor* is the candidate to be the next hop in the reverse routing path. In the second phase, control messages MSG2 are used for activation of the reverse routing paths and as a mechanism to prevent routing loops. A node is allowed to forward data via any node it has received a MSG2 from, provided that the cost-to-sink advertised by that node is less than the cost of the node itself. All those nodes are included in the *eligible neighbors list*. The neighbor to which data is forwarded is chosen for each packet from the *eligible neighbors list* and is referred to as the *active next hop*. This mechanism allows the establishment of braided multi-paths for all sensor nodes in one two-phase instance of the protocol.
- The design of the protocol allows for local restoration of paths that have been disrupted by local topological changes, with no need for global flooding of control messages. The search for paths is performed only in the proximity of the intermediate nodes used by the source whose data flow has been disrupted.

DCBM's second primary goal is to cope with node movement and failures and to minimize energy depletion in a manner requiring minimum control overhead, while allowing acceptable data delivery ratio. To achieve that:

- Each intermediate node caches the last forwarded data packet. If the data packet is not overheard from the *active next hop* node, the latter is made invalid for forwarding purposes and is removed from the *eligible neighbors list* (retries can be configured). If the intermediate node has another node in the *eligible neighbors list*, the data packet is forwarded again. If no node is available, the packet is dropped and a prune control message is broadcast in order to prevent neighbors from using this intermediate node as next hop.

- The sink is aware of the currently active sensors and is constantly monitoring the data delivery parameters. The sink initiates a new instance of the path establishment mechanism if the data delivery parameters drop below some threshold.

The DCBM path establishment two-phase mechanism can be tuned to fit various requirements, depending on the applications deployed in the WSN. For example it can focus on establishing paths with minimum delay for time critical applications or preferring near-optimum reliable paths for data critical applications. Important mechanism properties are loop avoidance and convergence to optimal routes for static networks. Moreover, the mechanism can be localized towards specific sources in order to avoid global broadcasts.

As mentioned before, the two types of control messages that are used in path establishment are MSG1 and MGS2, both initiated by the sink. MSG1 carries the distance to the sink and is used by nodes to learn about their neighbors and their distances to the sink. Based on the distance information, the node elects the *designated neighbor*. The received message is rebroadcast by the node with an updated distance. MSG2 is rebroadcast only if received from the *designated neighbor*, thus ensuring that no loops are created. To provide alternative paths, nodes store other neighbors it receives MSG2 from, but only if the published distance of the node is less than the distance published by the node itself.

At this point it is useful to summarize the terms defined so far. The *designated neighbor* is the neighbor that has reported the best distance to the sink in the current cycle. MSG2 received from this neighbor (and only from this neighbor) triggers rebroadcast of MSG2. The *Eligible neighbors list* contains all neighbors that may be used to forward data. The *Active next hop* is the neighbor with the best distance to the sink among all nodes in the *eligible neighbors list*.

We have designed two main versions of the algorithm: a *Fast Propagation Algorithm* version and an *Delayed Propagation Algorithm* version. In the sequel, we shall give the informal and formal descriptions of the two versions and shall provide proofs of their correctness.

The motivation for *Fast Propagation Algorithm* stems from sensitive applications requiring continuous data delivery even in case of significant topology changes. Therefore the objective is to reestablish data delivery in the shortest time frame possible. Another incentive is urgent changes in parameters related to deployed application monitoring or reporting, that may require fast *query/interest* diffusion. To meet these requirements, the *Fast Propagation Algorithm* forwards MSG1 messages in the fastest possible way at the expense of path energy and reliability considerations.

The motivation for the *Delayed Propagation Algorithm* version is the attempt to achieve the longest possible lifetime of the wireless sensor network. This version attempts to minimize energy depletion across the WSN. For this purpose the rebroadcast of MSG1 is delayed in order to allow accumulation of neighbor information. The delay provides the opportunity to select a better *designated neighbor* in terms of energy consumption.

The first version is discussed in Sec 2.2 , the second in Sec 2.3.

2.2 The Fast Propagation Algorithm

2.2.1 Path Establishment

The sink initiates refresh cycles of routing paths by broadcasting control messages $MSG1$ and $MSG2$. These control messages are rebroadcast over the entire network and propagate information about the nodes distance to the sink. The node distance is defined as the minimum of the sum of the distances received from neighbors plus the distance to that neighbor. The minimum is calculated once per cycle, upon receiving the first $MSG1$ of that cycle and is taken over distances received from neighbors in the previous cycle. That minimum also serves for election of the *designated neighbor* of node *i* for that cycle *c*, denoted by $e_i[c]$. Upon receiving the *first* $MSG1$ of a cycle, a node also re-broadcasts the $MSG1$ with the newly calculated distance, thus providing the fastest propagation of the path refresh. Recall that $MSG1$ also carries the application parameters, referred to as the *interest*, and fast propagation of $MSG1$ also facilitates fast broadcast of the interest to the sources.

$MSG2$ is propagated in a different way. Only receipt of $MSG2$ from the *designated neighbor* triggers re-broadcast of $MSG2$. We shall show that this procedure minimizes data forwarding loops. Data can visit the same node more than once only in situations when data is being forwarded while refreshes are taking place. Unless refreshes are taking place continuously, data finally reaches the sink or is being dropped.

Each node maintains an *eligible neighbor list* denoted by $n_i(k)$. A node marks a neighbor as eligible for data forwarding if $MSG2$ is received from that neighbor in the latest cycle and the distance stated in the $MSG2$ is less than the distance stated in the $MSG2$ sent in this cycle by the node itself. If the node itself did not send $MSG2$ in the current cycle it may list any of the neighbors it had received $MSG2$ from as eligible for data forwarding.

Loops in terms of *designated* neighbors may occur due to non-updated information at the nodes. The occurrence of this loop not necessary halts the data forwarding of the affected nodes. Nodes may propagate data through other neighbors in the *eligible neighbor list.* It may seem that a node will participate in data forwarding only if it is a data *source* because no $MSG2$ is sent to the neighbors. This is not necessary the case because if neighbors did not receive any $MSG2$ the node may still be in the *eligible neighbor list.* Thus when using *Fast Propagation Algorithm* for forwarding urgent *queries* we will not destroy previous paths without creating at least one new path.

The first cycle of the algorithm is unique since there is no previous cycle to base the selection of the *designated neighbor* on. The sink sends only *MSG1* and nodes select the *active next hop* as the neighbor the first *MSG1* is received from. This neighbor will also be the only one in the *eligible neighbor list.*

2.2.2 Data Forwarding

Data packets are forwarded via the node with the lowest distance to the sink among the nodes in the *eligible neighbors list*. This node is referred to as the *active next hop.*

Since we assume that every transmission is overheard by all nodes within hearing distance, data must carry the identity of the *active next hop*. When a data packet is received at the *active next hop*, the node rebroadcasts the data (containing its own *active next hop)*. In addition, the node cashes the packet and waits to overhear its retransmission by the next hop. If no retransmission is heard within a given time period, the packet is retransmitted. We assume that all nodes have equal transmission power, thus all neighboring relations are symmetric. After a pre-assigned number of unsuccessful retransmissions, the current *active next hop* is removed from the *eligible neighbor list* and a new *active next hop* is selected.

A node *i* that receives a data packet and has no eligible neighbor, broadcasts a *Down(i)* message, announcing neighbors that node *i* is not eligible for data packet forwarding. A node that has *i* in the *eligible neighbor list* and receives a *Down(i)* message removes node *i* from the list.

Data packets contain the distance of the sending node to the sink. Thus an overhearing node is able to refresh its *eligible neighbors list* with the most updated data. If the distance to the sink of a node changes, this information will be detected by the neighbors overhearing the data packets sent by the node and subsequently the update will move upwards the data stream with each data packet broadcast. This allows on-line adaptation of the forwarding pattern to changes in path quality, because intermediate upstream nodes will be able to switch to alternative paths with a lower distance to the sink. The residual energy in the node is changing with the broadcast of each packet. As a result, if residual energy is used as the metric for routing decisions, alternative paths with similar metric will be used interchangeably due to constant change in the reported metric. Use of alternative paths is viewed as an advantage of the protocol, as it results in load balancing.

2.2.3 Route maintenance

Local route integrity is maintained by the overhearing mechanism as mentioned in the last but one paragraph of the previous section. To reiterate, each node manages an *eligible neighbor list.* A neighbor *k* is deleted from the list of node *i* if the data packet sent by *i* via *k* is not overheard as forwarded further by *k*. If the *eligible neighbors list* becomes empty, node *i* broadcasts a *Down(i)* message. The overhearing mechanism can be replaced by any other packet acknowledgement mechanism.

Global route integrity is maintained by the sink. In addition to scheduled path refreshes, the sink constantly monitors the quality of data delivery. In our implementation, the sink is always aware of all sources and of the expected data rate from each source. If the received data rate from a source drops below some threshold, the sink assumes that the topology has changed and no alternative path was found, resulting in the triggering of a new path establishment cycle. If more than single source data delivery was disrupted global refresh is performed. Moreover globalized refresh is performed periodically.

The election of the *designated neighbor* is based on information collected during the previous cycle. In networks with considerable mobility, this information may be outdated since the neighbors list may change significantly between cycles. This leads to the possibility that when the *designated neighbor* broadcasts its $MSG2$, it is already out of range. Since nodes broadcast $MSG2$ when they receive $MSG2$ from their *designated neighbor*, this phenomenon may limit the propagation of $MSG2$, thus decreasing the number of paths to the sink. In order to remediate this problem, we require the sink to initiate more than one, say M, consecutive contiguous refresh cycles The optimal number M of consecutive cycles has been explored via simulation, as discussed in Sec 3.2.1.

2.2.4 Localizing the refresh

In order to save energy, it is important to localize refreshes of the reverse paths if topological changes affect only the data flow from a single source. Refresh localization is achieved by limiting the participation in the refresh to nodes that are close to the disrupted path.

The limiting technique is implemented by the use of several lists and parameters, maintained at nodes and/or included in the control messages.

- *Active Source List*- each node *i* maintains a list of sources whose data packets it is forwarding in the current cycle
- Each control message *MSG1* contains an additional three parameters named *Source*, *ttl* and *TTL* respectively.
    - *Source* denotes the identity of the source whose data delivery has been disrupted.
    - *ttl* is the number of hops left before the message shall be discarded.
    - *TTL* is the maximum allowed *ttl*.

Global flooding is identified in MSG1 messages by setting the *Source* field to '-1' . To limit the flooding of the control messages, the *ttl* value is decremented if the rebroadcasting node has not been an intermediate node on the disrupted path between *Source* and the sink (does not have *Source* in its Active Source List). Otherwise, the ttl parameter is reset by the node as *TTL* . If the value of *ttl* in received MSG1 is 0, the message is discarded.

2.2.5 Algorithm pseudo-code

2.2.5.1 Symbols

$MSG1(k, c, d, source, ttl, TTL)$ - Message of type 1 of cycle *c* from neighbor *k*. Here d represents the estimated distance to the sink from node *k*, *source* identifies the source for which the refresh is intended ('-1' for global refresh), *ttl* – the number of remaining rebroadcasts, TTL - maximum remaining rebroadcasts.

$MSG2(k, c, d)$ - Message of type 2 of cycle *c* from neighbor *k*. Here d represents the estimated distance to sink from node *k*.

In the following, subscript *i* indicates parameters stored at node *i*

$c1_i$ – the largest *c* received in $MSG1$

$c2_i$ – the largest *c* received in $MSG2$

$e_i[c]$ - *designated neighbor* elected in cycle c

$a_i = 1$ if node *i* has sent $MSG2(e_i(c1_i), c1_i, \bullet)$ and $= -1$ if the message has not been received.

$n_i(k)$ = 1 for all neighbors k in the *eligible neighbor list* and = -1 for the other neighbors.

d_{ik} – distance from node i to node k

$D1_i(k)$ – distance of node k to the sink as stored

$D1_i$ - Best distance to sink as calculated at the end of cycle $c1_i$ by node *i*, based on $D1_i(k)$.

$D2_i(k)$ - distance of node *k* to the sink through *designated neighbors* path

$D2_i$ - Distance to sink via the *designated neighbor*.

r_i - *active next hop*, namely the node with lowest $D1_i(k)$ among nodes in the *eligible neighbor list* or '-1' if the *eligible neighbor list* is empty

$Down(i)$ – control message notifying neighbors that node *i* has no energy or that r_i equals -1

$DataMSG(source, sender, nexthop, d, data)$ - Data packet. The parameter *source* denotes the origin of the data, , *sender* denotes the previous hop, *nexthop* – the next hop of the packet, *d* – the distance from the *sender* to *sink*, *data* – payload of data forwarded to the sink

$Timer(DataMSG())$ - holds a timer for overhearing a packet sent to the next hop before assuming that the packet was lost.

$asl_i(k)$ =1 if source k is in the *Active Source List*, namely if node *i* is forwarding data of Source k in the last cycle, = -1 otherwise

2.2.5.2 Pseudo-code for *Fast Propagation Algorithm*

<u>Initiation:</u>

$c1_i = -1$

$c2_i = -1$

$a_i = -1$

$n_i(l) \leftarrow -1 \;\; \forall l$

<u>A</u>lgorithm at node *i (not sink)*:

Receive $MSG1(k, c, d, source, ttl, TTL)$

A If (c=0) //First cycle

A1 $D1_i(k) \leftarrow d$ // Save neighbor distance

A2 If ($c1_i = -1$)

A3 $c1_i \leftarrow 0$ // Save cycle of MSG1

A4 $c2_i \leftarrow 0$ // Save cycle of MSG2

A5 $D1_i \leftarrow D1_i(k) + d_{ik}$ // Calculate distance to neighbor

A6 send (neighborcast) $MSG1(k, c, d, 0, ttl, TTL)$ // rebroadcast MSG1

A7 $n_i(k) \leftarrow 1$ // Mark *k* as eligible for data forwarding

B Else // All cycles but first

B1 $D1_i(k) \leftarrow d$ // Save neighbor distance

C If ($c > c1_i$ and ($source = -1$ or $ttl \neq 0$)) //If first MSG1 of refresh cycle and this is // MSG1 of global refresh or *ttl* is the MSG1 //is valid node *i* can process the packet

C1 $c1_i \leftarrow c$ // update cycle

C2 $D1_i \leftarrow \min_l (D1_i(l) + d_{il})$ //Calculate minimum distance to sink

C3 $e_i[c1_i] \leftarrow \arg\min_l (D1_i(l) + d_{il})$ //Elect *designated neighbor* based on //minimal distance to sink

C4 If ($asl_i(source) = 1$ or $source \neq -1$) // If node *i* participated in data forwarding //of source in the previous cycle or this is // global refresh, rebroadcast MSG1 with // maximum *ttl* else decrease *ttl* value in // the rebroadcasted MSG1

C5 send (neighborcast) $MSG1(i, c, D1_i, source, TTL, TTL)$

C6 Else

C7 send (neighborcast) $MSG1(i, c, D1_i, source, ttl - 1, TTL)$

C8 $D1_i(l) \leftarrow \infty \;\; \forall l \neq k$ //Invalidate previous cycle distances to neighbors

C9 If ($c2_i = c1_i$ and $n_i(e_i[c1_i]) = 1$) //If MSG2 of $c1_i$ from *designated neighbor* // had been received

C10 $a_i \leftarrow 1$ //Note that MSG2 had been sent in this cycle

C11 $D2_i \leftarrow D2_i(e_i[c1_i]) + d_{ie_i(c1_i)}$ //Save distance via *designated neighbor*

C12 send (neighborcast) $MSG2(i, c1_i, D2_i)$ //Rebroadcast MSG2

C13 $n_i(l) \leftarrow -1 \quad \forall m$ s.t. $D2_i(l) \geq D2_i$ //Remove from *eligible neighbor list* all //neighbors with distance equal to or greater //than distance published in MSG2 by node *i*

C14 $e_i[c1_i] \leftarrow -1$ // Invalidate designated neighbor

C15 If ($source \neq -1$) // Localized refresh cycle

C16 $asl_i(source) = -1$

C17 Else // Global refresh cycle

C18 $asl_i(l) \leftarrow -1 \ \forall l$

Receive $MSG2(k, c, d)$

D If ($c > c2_i$) // If first MSG of the refresh cycle

D1 $c2_i \leftarrow c$

D2 $a_i \leftarrow -1$ // No MSG2 rebroadcasted in current cycle

D3 $n_i(l) \leftarrow -1 \ \forall l$ // Invalidate *eligible neighbor list*

D4 $D2_i(l) \leftarrow \infty \ \forall l$ // Invalidate previous cycle neighbor distances

E If ($c2_i = c$) // For any MSG2 in the current cycle

E1 $D1_i(k) \leftarrow d$ // Update distance

E2 $D2_i(k) \leftarrow d$ // Update *designated neighbor* path distance of node *k*

E3 If ($e_i[c2_i] = k$) // If MSG2 is from *designated neighbor*

E4 $a_i \leftarrow e[c2_i]$ // MSG2 had been sent in current cycle

E5 $D2_i \leftarrow d + d_{ik}$ // Calculate distance via *designated neighbor*

E6 send (neighborcast) $MSG2(i, c2_i, D2_i)$ //Rebroadcast MSG2

E7 $n_i(k) \leftarrow 1$ //Insert *k* into *eligible neighbor list*

E8 $n_i(l) \leftarrow -1 \ \forall l$ s.t. $D2_i(l) \geq D2_i$ //Remove from *eligible neighbor list* //all neighbors with distance equal to or //greater than distance published in //MSG2 by node *i*

E9 $e_i[c1_i] \leftarrow -1$ // Invalidate designated neighbor

E10 Else // If MSG2 is not from *designated neighbor*

E11 If ($a_i = -1$ Or ($a_i \neq -1$ And ($D2_i(k) < D2_i$))) //If no MSG2 was sent in current cycle or distance of the neighbor k is lower //than that published by node *i in MSG2*

E12 $n_i(k) \leftarrow 1$ //Insert *k* into *eligible neighbor list*

2.2.5.3 Pseudo-code for data-forwarding

Algorithm at node *i*:

Receive $DataMSG(source, sender, nexthop, d)$ or Timer Expired for $DataMSG(source, sender, nexthop, d)$

F $D2_i(sender) \leftarrow d$ // Update neighbor distance

G If (Timer Expired) // Remove the nexthop the *DataMSG* in the *Buffer* was sent // to from the *eligible neighbor list*

G1 $n_i(nexthop) = -1$

H If ($i =$ $nexthop$ or Timer Expired)

H1 $asl_i(source) = 1$ // Node forwarded *source* data in this cycle

H2 If ($n_i(l) = -1 \ \forall l$) // If no *eligible neighbor* exist

H3 $r_i \leftarrow$ -1 // Note that no *eligible neighbor* exist

H4 else

H5 $r_i \leftarrow \arg \min_{l \text{ s.t. } n_i(l)=1} (D2_i(l) + d_{il})$ //Choose as *active next hop* the *eligible neighbor* *// with the lowest distance*

H6 If (r_i = *-1* or node *i* has energy for only one message)

H7 Drop $DataMSG$ // Drop the message

H8 Send (neighborcast) $Down(i)$ // Alert neighbors to i from *eligible* *// neighbors list*

H9 Else , save sent data //packet in the *Buffer,* set *Timer* on the buffer

H10 Send $DataMSG(source, sender, r_i, \min_{m \text{ s.t. } n_i(m)=1} (D2_i(m) + d_{im}))$ //Send data packet to //the *active next hop*

H11 Save $DataMSG(source, sender, r_i, \min_{m \text{ s.t. } n_i(m)=1} (D2_i(m) + d_{im}))$ // Save message

H12 Set $Timer(DataMSG(source, sender, r_i, \min_{m \text{ s.t. } n_i(m)=1} (D2_i(m) + d_{im}))$ // Set timer

I Else

I1 If (Saved $DataMSG$ with r_i=*sender* exist) // Check if received packet is an overheared //rebroadcast

I2 Delete all $DataMSG$ with r_i=*sender*

I3 Delete Timer for all $DataMSG$ with r_i=*sender*

2.2.5.4 Pseudo-code for local route-maintenance

Algorithm at node i:

Receive $Down(k)$

J $n_i(k) \leftarrow 0$

K If ($n_i(l) = -1\ \forall l$ // If no *eligible neighbor* exist

K1 $r_i \leftarrow$ -1 // Note that no *eligible neighbor* exist

L If ($r_i = -1$ or node *i* has energy for only one message)

L1 Send (neighborcast) *Down(i)*

2.2.5.5 Pseudo-code for refresh cycle generation at sink

Additional symbols:

M - Number of consecutive contiguous refresh cycles to be performed

$last\_arrival_i(source)$ - time at which last data packet from *source* was received

t_i - Time at node *i*

data_freq – required *source* report rate (equal and preconfigured for all sources in the network), each *source* reports in this rate if it has relevant information.

λ - factor denotes estimated number of undelivered packets to initiate a refresh cycle

Algorithm at sink:

Receive DataMSG(source, sender, nexthop=sink, d)

M If new data packet

M1 $asl_i(source) = 1$ // Save that *source* is active

M2 $last\_arrival_i(source) = t_i$ // Save the last arrival of data packet from *source*

Receive event ‘Check data delivery’

N If ({ $t_i - last\_arrival_i(source) > \frac{1}{data\_freq} * \lambda$ } for single source)

// Check if some source did not
// deliver any data packet in the
//selected time interval decided by λ

N1 Perform the next section M times

N2 Send (neighborcast) $MSG1(\sin k, c, 0, source, TTL, TTL)$ // Issue a limited refresh cycle

N3 Send (neighborcast) $MSG2(\sin k, c, 0)$

O If({ $t_i - last\_arrival_i(source) > \frac{1}{data\_freq} * \lambda$ } for more than one source)

O1 Perform the next section M times

O2 Send (neighborcast) $MSG1(\sin k, c, 0, -1, TTL, TTL)$ // Issue a global refresh cycle

O3 Send (neighborcast) $MSG2(\sin k, c, 0)$

O4 Reschedule Global refresh event

Receive event 'Perform Global refresh'

O5 Send (neighborcast) $MSG1(\sin k, c, 0, -1, TTL, TTL)$ // Issue a global refresh cycle

O6 Send (neighborcast) $MSG2(\sin k, c, 0)$

O7 Reschedule Global refresh event

2.2.6 Properties of the Fast Propagation algorithm

Some of the proofs provided in this section are based on work performed in [26].

We first show that the number of control messages in each cycle is limited.

Theorem 2.1

In each cycle, every node i sends at most one control packet $MSG1$ and at most one $MSG2$.

Proof:

From <C>, <C1> follows that part C is executed by a node i only when it receives the first packet of each cycle. Moreover, <C5> and <C7> are the only lines where $MSG1$ is sent, therefore node i can send at most one $MSG1(...,c,...)$ for any cycle c. Assume now that *MSG2* is sent twice by some node. Let *t* be the first time when *MSG2* is sent for the second time by some node, i say. The designated neighbor $e_i[c]$ is elected only once by *i* for each cycle c, <C3>. Since *MSG2* is sent by *i* only upon or after receiving *MSG2* from $e_i[c]$, <C12> or <E6> . This means that $e_i(c)$ sent *MSG2* twice before time *t,* contradiction.

Energy depletion can be also caused by data forwarding loops. In Theorem 2.2 we show that the *active next hops* do not form a loop. In the sequel, we show that messages can still loop, but only for the brief moment of message propagation.

Theorem 2.2

Denote by $K(t)$ the graph formed by all nodes in the network i and their link to the *active next hop* r_i at time t For any given time t there is no loop in $K(t)$.

We first prove the following Lemma:

Lemma 2.3

If k=r_j at time t, then holds at time t:

a) $c2_j < c2_k$ or

b) $(c2_j = c2_k)$ and $D2_j(k) > D2_k(r_k)$.

Proof

From < D > and <D1> follows that at every node i , the counter $c2_i$ is non-decreasing.

We first prove that $c2_j \le c2_k$ .Suppose the opposite namely $c2_j > c2_k$. Statements <D> and <D1> show that $MSG2(k, c2_j, ...)$ was received by node j. But at the time node k was selected to be r_j in

<H5> ,it was true that $n_j(k)=1$. The variable $n_j(k)$ can be set to 1 only in <E7> or <E12>. According to lines <D>, <D3> ($n_j(k)$ is set to -1 when the first message MSG2 of cycle c is received) and line <E>, receiving $MSG2(k,c,...)$ is a prerequisite for statement n_j(k) =1 to hold while $c2_j = c$. Therefore k is not eligible to be r_j. This proves that indeed $c2_j \le c2_k$.

Now we prove that if $c2_j = c2_k$ then $D2_j(k) > D2_k(r_k)$. According to lines <H5>, the fact that k=r_j implies n_j(k) =1. As previously shown, for statement n_j(k) =1 to hold in cycle $c2_j$, node j must receive $MSG2(k,c2_j,D2_k)$. Therefore k has send $MSG2(k,c2_j,D2_k)$. Then according to lines <C13>, <E8> and <E11> the quantity $D2_k$ sent in this message must be strictly larger than $D2_k(m)$ for any m eligible to be r_k (n_k(m) =1).

Moreover we know:

$D2_j(k)=D2_k$ according to line <E2> , thus $D2_j(k) = D2_k > D2_k(r_k)$

<u>Proof of Theorem 2.2</u>

Since Lemma 2.3 shows that $c2_j$ must be non-decreasing around the loop in $K(t)$, all the $c2_j$ in the loop must be equal. But Lemma 2.3 b) shows that $D2_j$ must be strictly decreasing, contradiction.

Next we shall show that the algorithm converges to optimal routing in a final number of refresh cycles if no changes or packet loss occurs.

<u>Theorem 2.3</u>

Suppose that changes in the network topology cease before the time when cycle c' starts (nodes are stationary, link weights are constant, propagation time is constant). Then a finite number of path refresh cycles afterwards, the distance parameter $D1_i\left[c\right]$ held by each node does not change between refresh cycles and is identical to the optimal distance to the sink. In addition, the *designated neighbor* $e_i(c)$ is the next hop on the optimal path from i to sink.

<u>Denote:</u>

$D1_i^*$ = optimal distance of node i to the sink

$D1_i[c]$ = the distance $D1_i$ at cycle c

$D1_i(k)[c]$ =the distance $D1_i(k)$ at cycle c

e_i^* = next hop neighbor on optimal path of node i to the sink

Lemma 2.4

If starting refresh cycle c'', all distance parameters $D1_i\left[c\right]$ held by all nodes do not change between refresh cycles, then $\forall c > c''$ each node i holds $e_i[c] = e_i^*[c]$ and $D1_i[c] = D1_i^*$.

Proof

Assume there is at least one node i with $D1_i[c] < D1_i^*$ for some $c > c''$. Let K be the group of nodes with $D1_i[c] < D1_i^*$. Suppose $j \in K$ and j is the node with minimal $D1_i[c]$, namely $D1_j[c] \le D1_i[c] \quad \forall i \in K$.

For $c > c''$, denote $k = e_i\left[c\right]$. Due to statements <C2> and <C3>, holds $D1_j[c] = D1_k[c] + d_{kj}$. Therefore, since d_{kj} is strictly positive, $k \notin K$, holds $D1_k[c] \ge D1_k^*$. Since k and j are neighbors, $D1_k^* + d_{kj} \ge D1_j^*$, so that we finally get $D1_j[c] = D1_k[c] + d_{kj} \ge D1_k^* + d_{kj} \ge D1_j^*$, contradicting the fact that $j \in K$.

Assume now that there is at least one node i with $D1_i[c] > D1_i^*$ for some $c > c''$. Let K be the group of nodes i with $D1_i[c] > D1_i^*$. Suppose j is the node in group K with minimal $D1_i^*$, namely $D1_j^* \le D1_i^* \quad \forall i \in K$ and k is the next hop of node j in the optimal path to sink. Obviously holds $D1_k^* + d_{kj} = D1_j^*$. Therefore, since d_{kj} is strictly positive, $k \notin K$, holds $D1_k[c] \le D1_k^*$.

According to statement <C2>, the parameter $D1_j[c]$ is selected as the minimum of $D1_j(i)$ over all neighbors of j. Therefore $D1_k[c] + d_{kj} \ge D1_j[c]$, so we finally get $D1_j^* = D1_k^* + d_{kj} \ge D1_k[c] + d_{kj} \ge D1_j[c]$ contradicting the fact that $j \in K$.

We now prove that indeed the distances $D1_i\left[c\right]$ stop changing.

Lemma 2.5

Starting refresh cycle $c'+1$, for every node j and every finite number z, there is a finite number of events when j reduces its $D1_j$ to a value $\le z$.

Proof

This is shown by first proving that for every event in a node, there has been a corresponding event in one of its neighbors. According to statements <B1>,<C>, <C2>, node j may reduce $D1_j[c-1]$ in refresh cycle $c > c'+1$ to a value $D1_j[c]$ in three cases:

- Node j receives the first message of refresh cycle c, $MSG1(k, c, D1_k[c])$, that satisfies $D1_k[c] + d_{kj} < D1_j[c-1]$. Since we assume that after cycle c message propagation times do not change, node j has received the first message of the previous refresh cycle $c-1$ from the same neighbor k. Therefore, from line <C2> in the algorithm, follows $D1_j[c-1] \le D1_k[c-1] + d_{kj}$. Thus $D1_k[c] < D1_k[c-1]$ and also $D1_k[c] < D1_j[c-1]$.
- Node j receives the first message of refresh cycle c from neighbor l and it has previously received a message $MSG1(k, c-1, D1_k[c-1])$ that satisfies $D1_k[c-1] + d_{kj} < D1_j[c-1]$ from a different neighbor $k \ne l$. Since we assume that after cycle c message propagation times do not change, the order of message receipt in cycle c and in cycle $c-1$ is identical. Therefore, from line <C2> in the algorithm follows that $D1_j[c-1] \le D1_k[c-2] + d_{kj}$. Thus $D1_k[c-1] < D1_k[c-2]$ and also $D1_k[c-1] < D1_j[c-1]$.
- Node j found a new neighbor in cycle l but this cannot happen after cycle c, since no topological changes occur.

Denote by K the set of nodes j that reduce their $D1_j[c]$ an infinite number of times to values $D1_j[c] \le z$. For $j \in K$, denote $z_j = \liminf D1_j[c]$. Clearly, $z_j \le z$ and let $j*$ be the node that achieves $\min z_j$ over $j \in K$. As shown above, to every event $D1_{j*}[c] < D1_{j^*}[c-1]$ corresponds an event $D1_k[c] < D1_k[c-1]$ or $D1_k[c-1] < D1_k[c-2]$ at some neighbor k and also, correspondingly $D1_k[c] < D1_{j*}[c]$ or $D1_k[c-1] < D1_{j*}[c]$. Since $j*$ has only a finite number of neighbors, it must have a neighbor $k*$ that has an accumulation point of $D1_{k^*}[c]$ at $z_{j*} - d_{j*k*}$. Therefore $k* \in K$ and $z_{k*} < z_{j*} - d_{j*k*}$ contradicting the fact that z_{j*} is minimal.

Lemma 2.6

Starting refresh cycle $c'+1$ for every node j and every finite number z there is a finite number of events when j increases its $D1_j$ from a value $\le z$.

Proof

This is shown by first proving that for every event in a node, there has been a corresponding event in one of its neighbors. According to statements <B1>,<C>, <C2> , node j may increase $D1_j[c]$ in refresh cycle $c > c'+1$ from a value $D1_j[c-1]$ in three cases:

- Node j receives the first message of refresh cycle c , $MSG1(e_j[c-1], c, D1_{e_j[c-1]}[c])$ that satisfies $D1_{e_j[c-1]}[c] + d_{e_j[c-1]j} > D1_j[c-1]$ from its *designated neighbor* $e_j[c-1]$. Since we assume that after cycle c message propagation times do not change, node j has received the first message of the previous refresh cycle $c-1$ from the same neighbor $e_j[c-1]$. Therefore, from lines <C2> and <C3> in the algorithm follows that $D1_j[c-1] = D1_{e_j[c-1]}[c-1] + d_{e_j[c-1]j}$. Thus $D1_{e_j[c-1]}[c] > D1_{e_j[c-1]}[c-1]$ and also $D1_{e_j[c-1]}[c-1] < D1_j[c-1]$.
- Node j receives the first message of refresh cycle c from neighbor l and it has previously received a message $MSG1(e_j[c-1], c-1, D1_{e_j[c-1]}[c-1])$ that satisfies $D1_{e_j[c-1]}[c-1] + d_{e_j[c-1]j} > D1_j[c-1]$ from its *designated neighbor* $e_j[c-1] \neq l$. Since we assume that after cycle c message propagation times do not change, the order of message receipt in cycle c and in cycle $c-1$ is identical. Therefore, from lines <C2> and <C3> in the algorithm, follows $D1_j[c-1] = D1_{e_j[c-1]}[c-2] + d_{e_j[c-1]j}$. Thus $D1_{e_j[c-1]}[c-1] > D1_{e_j[c-1]}[c-2]$ and also $D1_{e_j[c-1]}[c-2] < D1_j[c-1]$.
- Node j loses its *designated neighbor* during refresh cycle $c-1$, but this cannot happen because no topology changes occur.

Denote by K the set of nodes that increase their $D1_i[c]$ an infinite number of times from values $D1_j[c] \leq z$. For $j \in K$, denote $z_j = \liminf D1_j[c]$. Clearly $z_j \leq z$ and let j^* be the node that achieves $\min z_j$ over $j \in K$. As shown above, to every event $D1_{j^*}[c] > D1_{j^*}[c-1]$ corresponds an event $D1_k[c] > D1_k[c-1]$ or $D1_k[c-1] > D1_k[c-2]$ at some neighbor k of j^*. We have also shown that $D1_k[c-1] < D1_{j^*}[c-1]$ or $D1_k[c-2] < D1_{j^*}[c-1]$. Since j^* has only a finite number of neighbors, it must have a neighbor k^* that has an accumulation point of $D1_{k^*}[c]$ at $z_{j^*} - d_{j^*k^*}$. Therefore $k^* \in K$ and $z_{k^*} < z_{j^*} - d_{j^*k^*}$ contradicting the fact that z_{j^*} is minimal.

Proof of the Theorem 2.3

Since every new value of $D1_i[c]$ is either after an increase or after a decrease, Lemmas 2.5 and 2.6 show that there is only a finite number of new values of $D1_i[c] \le z$ for every finite z and therefore a finite number of changes in $D1_i[c]$. Thus the conditions of Lemma 2.4 hold and thus there final values are optimal.

Next we give some indication as of the number of broadcasts that a data message can experience at any given node.

Lemma 2.7

If a node j broadcasts a *DataMSG* and subsequently broadcasts the same *DataMSG* again, then the cycle counter $c2_j$ must have been increased between the two events.

Proof

Denote by $t1$ and $t2$ the time of the two events respectively. Let $\{j, r_1, r_2, r_3 ..., r_l, j\}$ represent the path of *DataMSG*. According to Lemma 2.3 holds $c2_j(t1) \le c2_{r_1} \le ... \le c2_{r_l} \le c2_j(t2)$ at the time of the broadcast. Therefore, if $c2_j(t1) = c2_j(t2)$, holds $c2(t1) = c2_{r_1} = ... = c2_{r_l} = c2(t2)$ at the time of broadcast. According to Lemma 2.3, the equality in counter numbers above implies $D2_j(r_1)(t1) > D2_{r_1}(r_2) > ... > D2_{r_l}(j) > D2_j(k)(t2)$ at the time of the broadcast, where k is the *active next hop* of node j. We get $D2_j(r_1)(t1) > D2_j(k)(t2)$, and since $c2_j(t1) = c2_j(t2)$ this leads to a contradiction to statement <H5> that implies: $D2_j(r_1)(t1) \le D2_j(k)(t2)$.

Theorem 2.4

Let N be the number of nodes in the network and $t^{\max}$ the maximum propagation time between each two neighbors. If the time between two refresh cycles is larger than $3 * N * t^{\max}$, then each *DataMSG* can be broadcast by a given node at most twice.

Proof

We prove the Theorem by contradiction. Suppose *DataMSG*'s can be broadcast by nodes more than twice and let j be the *first* node that broadcasts a *DataMSG* for the third time. Let

$t1$ - time of first broadcast of *DataMSG* at node j

$c2' = c2_j(t1)$, cycle counter at node j at $t1$

$t2$ - time of second broadcast of *DataMSG* at node j

$c2'' = c2_j(t2)$, cycle counter at node j at $t2_i$

$T(c2')$ – time the first message of refresh cycle $c2'$ was broadcast by the *sink*

$T(c2'')$ – time the first message of refresh cycle $c2''$ was broadcast by the *sink*

N - number of nodes in the network

t^{max} - maximum propagation of a single hop

t_p^{min} - minimum propagation of a single hop

$t3$ - time of third broadcast of *DataMSG* at node j

$c2'''$ - $c2_j(t3)$, cycle counter at node j at $t3$

$T(c2''')$ – time the first message of refresh cycle $c2'''$ was broadcast by the *sink*

According to Lemma 2.5, the value of $c2_j$ is increased each time the packet is broadcast by node j.

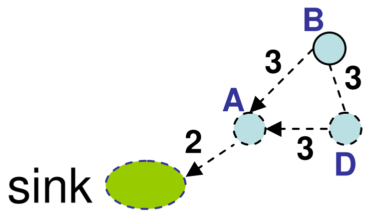
We know that $t1 < T(c2'') + N * t_p^{max}$, since cycle $c2''$ starts at time $T(c2'')$ and therefore by time $T(c2'') + (N-1) * t^{max}{}_p$ all nodes in the network have $c2_i \geq c2''$. We also know that $t3 \geq T(c2''') + t^{min}{}_p$, since only after $T(c2''') + t^{min}{}_p$ the first node can change its $c2_i$ to $c2_i = c2'''$. Therefore we can conclude that: $t3 - t1 \geq T(c2''') + t_p^{min} - (T(c2'') + N * t_p^{max}) > 3 * N * t_p^{max} - N * t_p^{max}$

Thus, *DataMSG* has passed at least one node in the network more than twice before arriving at node j, contradicting the fact that j is the first node to have broadcast *DataMSG* 3 times.

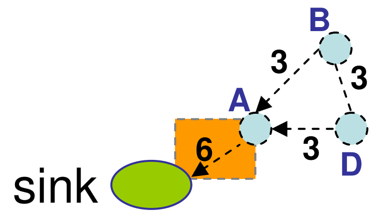
2.2.7 Designated neighbor loops example

In the previous section we stated that loops in terms of *designated neighbors* $e_i(c_i)$ may occur due to non-updated information at the nodes. Remember that data is forwarded according to the *active next hop* and not to the *designated neighbor*. Therefore designated neighbor loops only affect receipt of $MSG2$.

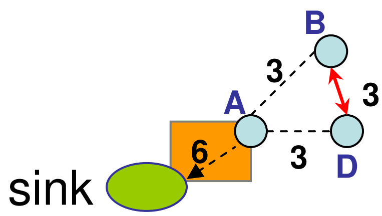
The purpose of the example is to show how loops in terms of $e_i(c_i)$ may occur.



a). Cycle 'c-1' topology



b). Changes before cycle 'c'



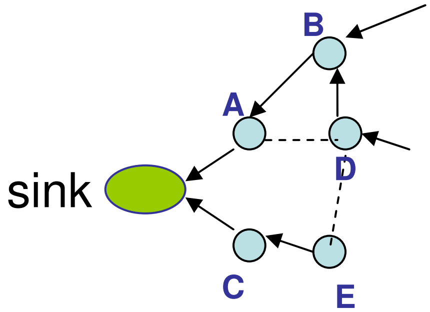
c). Topology at cycle 'c'

Figure 2.1 – Two node loop in terms of $e_i(c)$

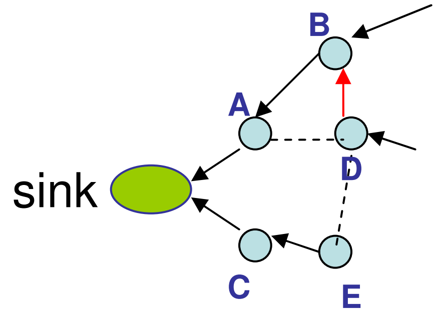
Figure 2.1.a) presents a network topology where the link weights are as marked. The *designated neighbors* are marked by arrows. Nodes B and D select node A as their *designated neighbor*. Node B holds $D1_B(A)=2, D1_B(D)=5$ and node D holds $D1_D(A)=2, D1_D(B)=5$ The weight of link {A,sink} changes between cycle $c-1$ and cycle c from 2 to 6, as shown in Figure 2.1.b). Assume that $MSG1(A,c,6)$ from node A reaches nodes B and D at approximately the same time. Thus now node B holds $D1_B(A)=6, D1_B(D)=5$ and node D holds $D1_D(A)=6, D1_D(B)=5$. According to statement <C3> of the algorithm, node B selects node D as its *designated neighbor* and node D selects node B as its *designated neighbor*. The loop in terms of $e_i[c]$ is shown in Figure 2.1.c) by red arrows.

2.2.8 Temporary loop example

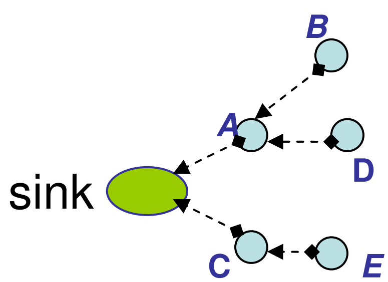
As mentioned in Sec 2.2.5, temporary data loops may occur while a refresh cycle is performed. Figure 2.2 shows an example of how this can happen.



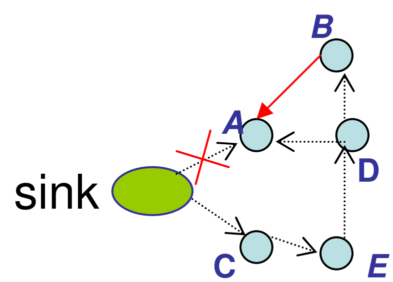
a). *Active next hop* topology at cycle 1



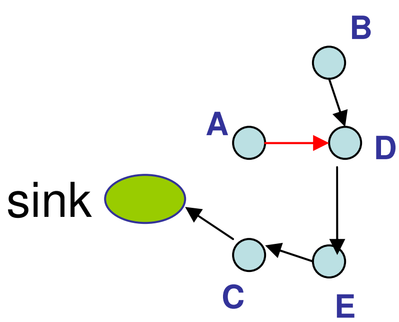
b). *Active next hop* topology at cycle 1 while red arrow notes current data message position



c). *designated neighbors* topology



d). $MSG2$ propagation at cycle 2 , $MSG2$ from *sink* does not reach node A



e). *Active next hop* topology at cycle 2 while red arrow notes current data message position

Figure 2.2 – Temporary loop of data forwarding

Figure 2.2.a) shows a topology in terms of *active next hop* while the cycle number $c2_i$ is 1 at all nodes. In Figure 2.2.b) we see that $DataMSG$ is forwarded from node D to node B based on *active next hop* topology. Figure 2.2.c) shows *designated neighbor* topology. Figure 2.2.d) shows the propagation of $MSG2$ based on the *designated neighbor* topology with $MSG2$ sent from *sink* fails to reach node A. Figure 2.2.d) also shows that $DataMSG$ is forwarded from node B to node A and a control message $MSG2$ of cycle c reaches node A from node D. Assuming that $MSG2$ arrives at A just before $DataMSG$, the latter will be forwarded from A to D as shown in Figure 2.2.e). Figure 2.2

provides a simple example of a temporary data forwarding loop created when a refresh cycle propagates while data is being forwarded. As we can observe from Figure 2.2.e), the next *DataMSG* arriving at node D will be forwarded directly to node E.

2.3 The Delayed Propagation algorithm

The original idea of the Delayed Propagation Algorithm stems from [20] and [21]. The idea is to delay the propagation of control messages by an amount proportional to the distance between neighbors. We apply this idea to our $MSG1$, thus allowing nodes to collect more information about the distance from neighbors to the sink in the current cycle. This alleviates the problem of the *Fast Propagation Algorithm,* by reducing the possibility that the elected *designated neighbor* moves out of range, prior to its sending the MSG2. As a result the *Delayed Propagation Algorithm* needs no consecutive contiguous refresh cycles.

Here is a more detailed description of the algorithm. When node i receives the first $MSG1$ from node k, it calculates the estimated distance to the *sink* through k. Then the node postpones the transmission of the $MSG1$ for an interval proportional to the distance to node k, namely $d_{ik} * \gamma$, where γ is some proportionality factor. If during this interval no $MSG1$ that results in a better estimated distance to the *sink* is received, node k is selected as the *designated neighbor* and a $MSG1$ with the distance through k is broadcast. Otherwise, the procedure is repeated for every $MSG1$ that improves the estimated distance to the sink, from neighbor m say. The new interval is calculated as $d_{im} * \gamma$ from the time MSG1 is received from m. In an environment with no propagation or processing delay, the algorithm renders optimal reverse paths. However, in a real environment, delays are accumulated, messages receive a back-off because the channel is busy and thus optimal reverse path is not always achieved..

The proportionality factor γ should be carefully selected. The product $d_{ik} * \gamma$ should be large enough to overcome propagation and processing delays, but small enough to provide rapid topology updates. In our implementation, since d_{ik} is selected as the residual energy (metric solution inspired by [27]), we make γ selected by the sink, depending on the gathered information about the residual energy at the nodes.

The properties of the *Delayed Propagation algorithm* are similar to the *Fast Propagation algorithm* version and are proved in Chapter 6 - Appendix A.

The only change in the *Delayed Propagation Algorithm* version is in the *reverse path* establishment phase. Therefore we present only this part of the pseudo-code.

2.3.1 Algorithm pseudo-code

2.3.1.1 Symbols

Additional/Changed Symbols

$MSG1(k,c,d,source,TTL,ttl,\gamma)$ - the additional parameter γ is the factor of expression $d_{ik}*\gamma$ to calculate the time window in which *MSG1* 's are accepted.

t_{int} - time interval to allow receipt of additional *MSG1* packets from neighbors

$e\_temp_i(c1_i)$ - Holds the last candidate to be *designated neighbor* in refresh cycle $c1_i$

$e_i(c1_i)$ - Holds the *designated neighbor* in refresh cycle $c1_i$

2.3.1.2 Pseudo-code for *Delayed Propagation Algorithm*

Initiation:

$c1_i = -1$

$c2_i = -1$

$a_i = -1$

$n_i(l) \leftarrow -1 \;\; \forall l$

Algorithm at node *i (not sink)*:

Receive $MSG1(k,c,d,source,ttl,TTL)$

P If (c=0) //First cycle
  P1 $D1_i(k) \leftarrow d$ // Save neighbor distance
  P2 If ($c1_i = -1$)
  P3 $c1_i \leftarrow 0$ // Save cycle of MSG1
  P4 $c2_i \leftarrow 0$ // Save cycle of MSG2
  P5 $D1_i \leftarrow D1_i(k) + d_{ik}$ // Calculate distance to neighbor
  P6 send (neighborcast) $MSG1(k,c,d,0,ttl,TTL)$ // rebroadcast MSG1
  P7 $n_i(k) \leftarrow 1$ // Mark *k* as eligible for data forwarding
Q Else // All cycles but first
  Q1 $D1_i(k) \leftarrow d$ // Save neighbor distance
  R If ($c > c1_i$ and ($source = -1$ or $ttl \neq 0$)) //If first MSG1 of refresh cycle and this is
  // MSG1 of global refresh or *ttl* is the MSG1
  //is valid node *i* can process the packet
  R1 $c1_i \leftarrow c$ // update cycle
  R2 $D1_i \leftarrow \infty$ // Reset the distance via *designated neighbor*
  R3 $t_{\text{int}} \leftarrow d_{ik} * \gamma$ // Calculate and set time window for
  // *MSG1* with better distance to arrive

S If ($c1_i = c$ and $t_{\text{int}} \neq \infty$ and $D1_i > d + d_{ik}$)
S1 $t_{\text{int}} \leftarrow d_{ik} * \gamma$ // Calculate and reset time window for
// *MSG1* with better distance to arrive
S2 $D1_i \leftarrow d + d_{ik}$ // Calculate distance via *designated neighbor*
S3 $e\_temp_i(c1_i) \leftarrow k$ // save *k* as new candidate for *designated*
// *neighbor*
T t_{int} expired
T1 $t_{\text{int}} \leftarrow \infty$ // No additional time window will allowed
// in current cycle
T2 $e_i(c1_i) \leftarrow e\_temp_i(c1_i)$ // Choose *designated neighbor*
T3 If ($asl_i(source) = 1$ or $source \neq -1$) // If node i participated in data forwarding
//of source in the previous cycle or this is
// global refresh, rebroadcast MSG1 with
// maximum *ttl* else decrease *ttl* value in
// the rebroadcasted MSG1
T4 send (neighborcast) $MSG1(i, c, D1_i, source, TTL, TTL)$
T5 Else
T6 send (neighborcast) $MSG1(i, c, D1_i, source, ttl - 1, TTL)$
T7 If ($c2_i = c1_i$ and $n_i(e_i(c1_i)) = 1$) //If MSG2 of $c1_i$ from *designated neighbor*
// had been received
T8 $a_i \leftarrow 1$ //Note that MSG2 had been sent in this cycle
T9 $D2_i \leftarrow D2_i(e_i(c1_i)) + d_{ie_i(c1_i)}$ //Save distance via *designated neighbor*
T10 send (neighborcast) $MSG2(i, c1_i, D2_i)$ //Rebroadcast MSG2
T11 $n_i(l) \leftarrow -1 \quad \forall m$ s.t. $D2_i(l) \geq D2_i$ //Remove from *eligible neighbor list* all
//neighbors with distance equal or greater
//than distance published in MSG2 by node *i*
T12 $e_i(c1_i) \leftarrow -1$ // Invalidate designated neighbor
T13 If ($source \neq -1$) // If MSG1 of localized refresh note that no data from
//*source* was forwarded in current cycle
T14 $asl_i(source) = -1$
T15 Else // If global refresh note than date of any source was
//forwarded
T16 $asl_i(l) \leftarrow -1 \ \forall l$

Receive $MSG2(k, c, d)$ - This part is not changed compared to the *Fast Propagation Algorithm* version

Chapter 3 - Performance Evaluation

3.1 Simulation environment

The DCBM, RCDR, GRAB, DDSIR and AODV algorithms were simulated in the ns-2 simulation environment. In this section, we shall list the simulation parameters and characteristics.

Ns-2 is a discrete event simulator targeted at networking research. Ns-2 provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks. NS was built in C++ and provides a simulation interface through OTcl, an object-oriented dialect of Tcl. The user describes a network topology by writing OTcl scripts, and then the main ns program simulates that topology with specified parameters. Ns-2 is now developed in collaboration between a number of different researchers and institutions, including SAMAN (supported by DARPA), CONSER (Collaborative Simulation for Education and Research)(through the NSF), and ICIR (formerly ACIRI). It is currently maintained by volunteers. Long-running contributions have also come from Sun Microsystems and the UCB Daedelus and Carnegie Mellon Monarch projects, cited by the ns homepage for wireless code additions [28].

3.1.1 Nodes

Many sensor networks implementations require cheap and simple node design, therefore the following properties were assumed.

- Nodes' transmission power is constant.
- A message is sent by a node only if the node has sufficient energy to send it.

3.1.2 Movement

Node movement is a major characteristic of our simulation environment. The Waypoint algorithm (part of ns-2) has been used to create simulation scenarios with node movement. The scenario generation algorithm sets random initial positions for all nodes in the network (uniform distribution). Then each node receives a randomly generated next interim location (uniform distribution) and movement speed (uniformly distributed between 0 and *maximum speed*). Upon arrival to the interim location, the node briefly stays there and then a new pair is generated. We have varied the *maximum speed* parameter between 1m/sec and 5m/sec.

3.1.3 Radio channel and MAC layer

The physical layer model used in our simulation is the two-ray ground signal propagation model. The two-ray ground reflection model considers both the direct path and a ground reflection path.
The MAC layer model used in our simulation is 802.11.

3.1.4 Node density

The simulation field size is [1000mX1000m]. The number of sensors distributed in the field change between 70 and 130. The transmission radius is 175m, which results in an average number of nodes in the transmission radius between ~3.4 and ~6.25.

3.1.5 Data sources

Each data source sources (node that are sensing the phenomena to be reported to the *sink*) has constant data rate that it needs to report to the sink.

3.1.6 Sink and data sources positioning

The location of the sink and of the data is constant throughout the simulation scenarios unless otherwise stated. Each of those nodes is distanced 200 m from each border:

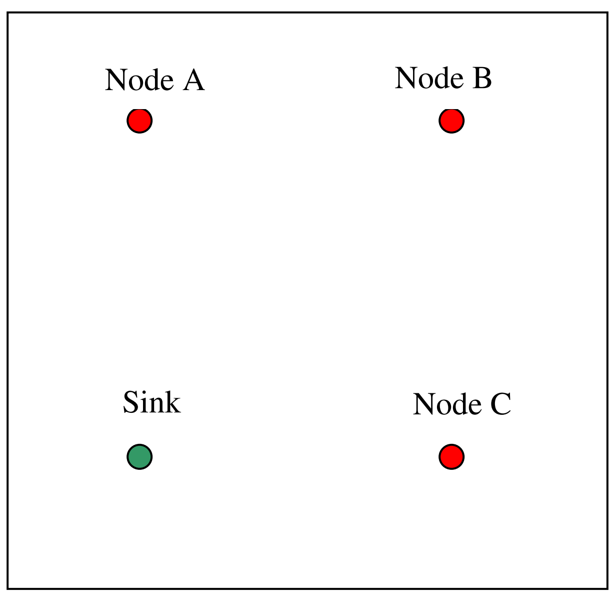


Figure 3.1 – Sink and active nodes positioning

3.2 DCBM simulation results

In the simulations we measure two parameters to evaluate the performance of the algorithms:

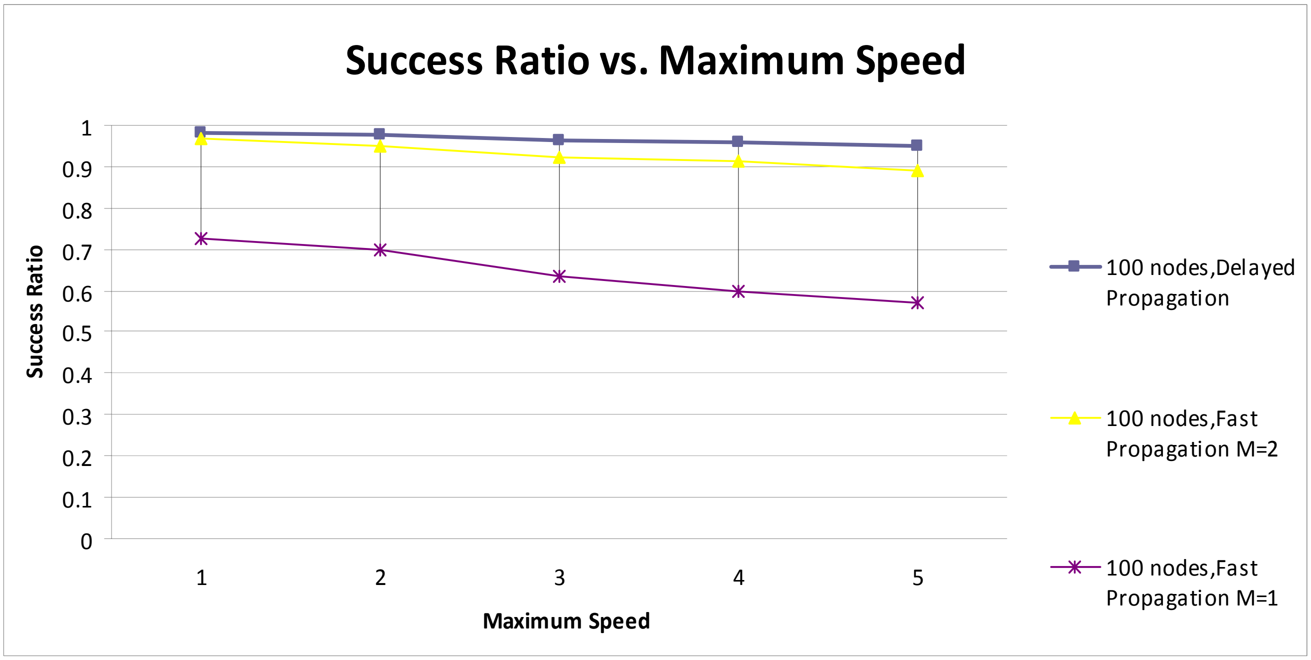
- Success ratio – percentage of data packets successfully delivered to the sink.
- Overhead – percentage of packets sent in excess of packets used to deliver data, i.e. number of duplicated data packets plus number of control packets divided by the total number of sent packets.

3.2.1 Version comparison

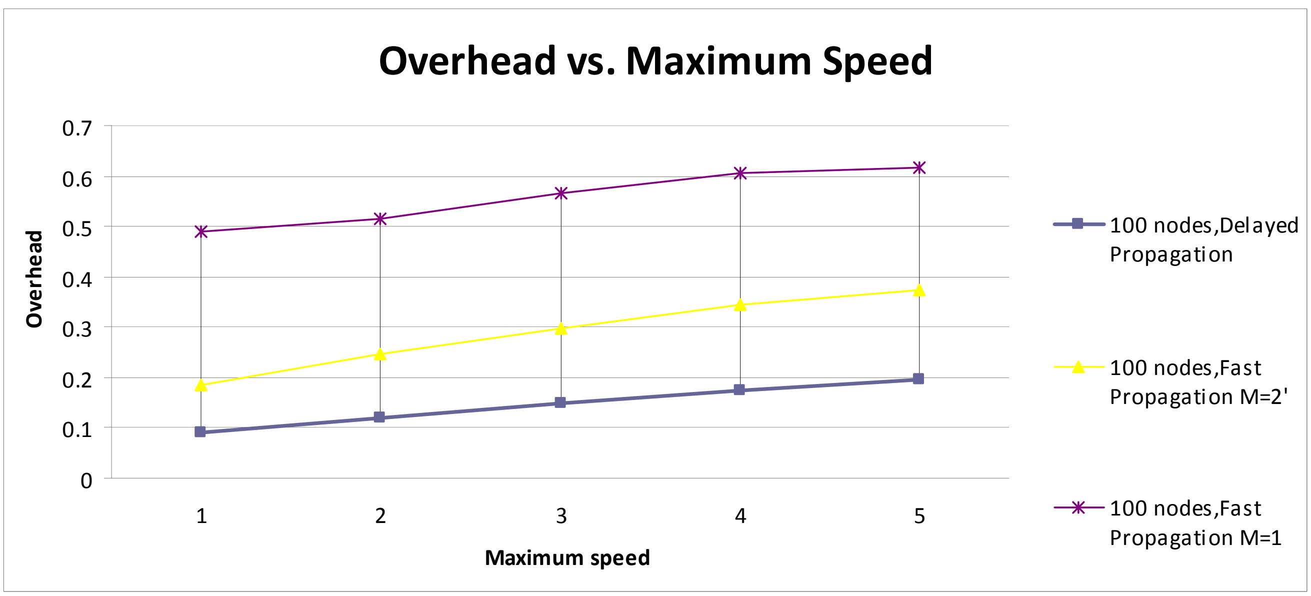
First we compare the performance of the two versions of the algorithm. The *Fast Propagation algorithm* was simulated with the consecutive contiguous refresh cycles parameter $M = 1,2,3,4$. Simulation showed no significant gain in terms of success ratio for $M > 2$. Single additional consecutive refresh cycle is enough to almost completely reduce selection of neighbor that moved out of transmission as *designated neighbor*.

As shown in Figure 3.2.a). and 3.2.b) , the *Fast Propagation algorithm* version provides smaller success ratio and larger control overhead than the *Delayed Propagation algorithm* version. Figure 3.2 also shows that the performance gap in both parameters increases with node mobility.

There are two main reasons for the performance gap, the first being the number of refresh cycles generated upon detection of a network topology change, while the second is the frequency of *designated neighbor* loops. Each loop can significantly limit the propagation of $MSG2$. A limited propagation of $MSG2$ decreases forwarding path redundancy, thus creating more instances of packet loss and repeated refresh cycles. Another reason behind the performance gap is the fact *that designated neighbor*s are selected using fast propagated information. Again the non optimal election of the *designated neighbor* may reduce the number of nodes in the *eligible neighbor list* and decrease forwarding path redundancy.



a). Success Ratio vs. Maximum speed



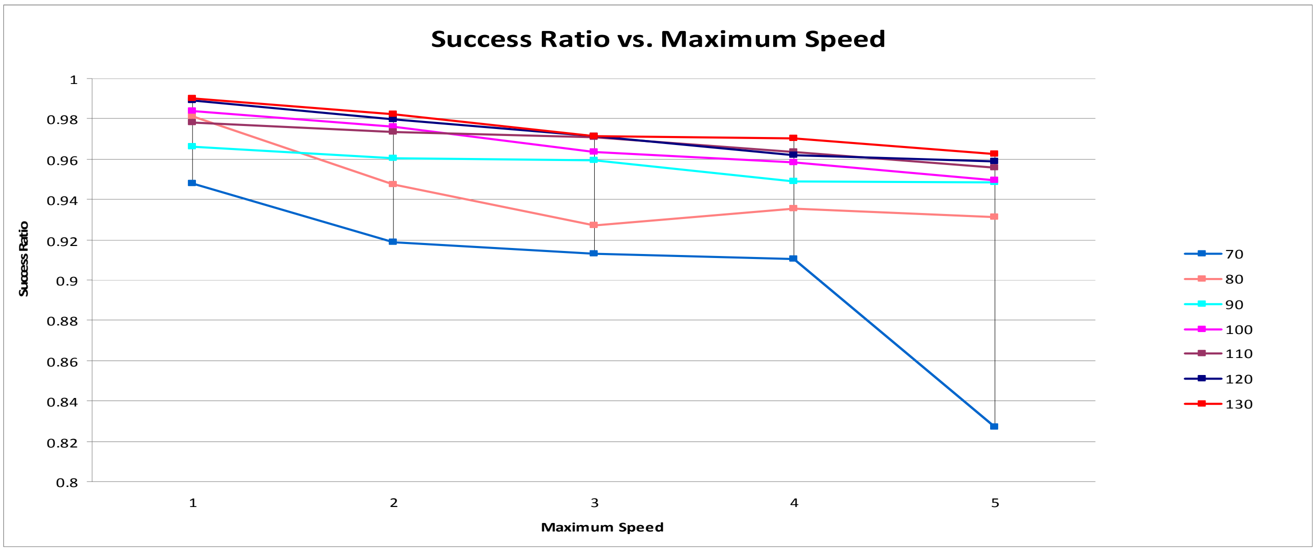
b). Overhead vs. Maximum Speed

Figure 3.2 – Fast Propagation and Delayed Propagation Algorithm versions comparison

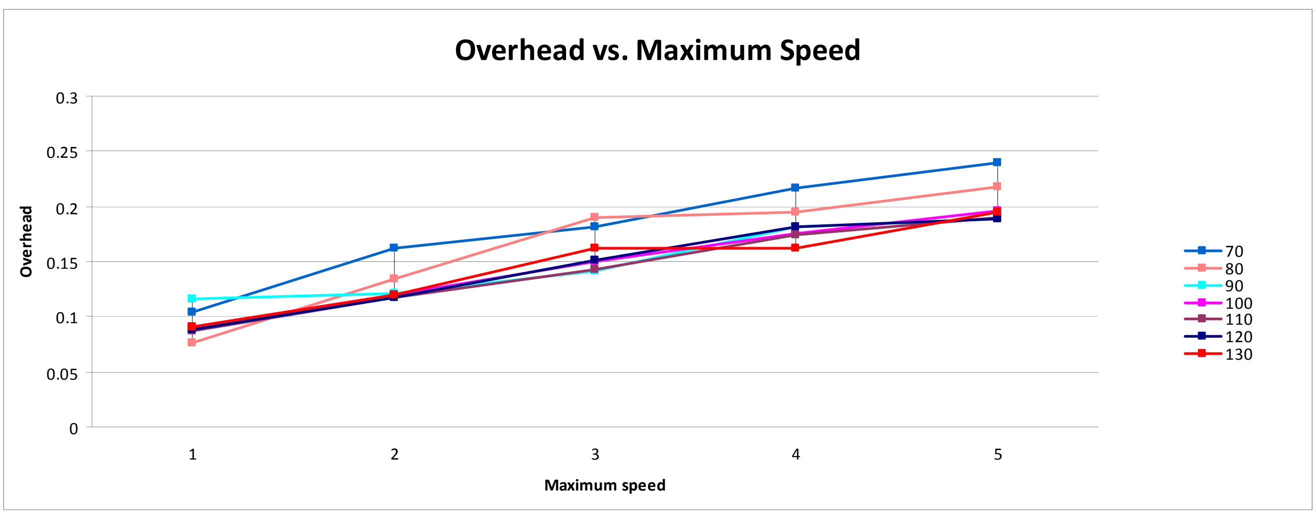
In the following sections we shall focus on the performance of the *Delayed Propagation Algorithm*.

3.2.2 Behavior of the Delayed Propagation Algorithm

The algorithms results for various values of maximum node speed and node density are shown in Fig. 3.3.



a). Success Ratio vs. Maximum speed



b). Control Overhead vs. Maximum Speed

Figure 3.3 – Delayed Propagation Algorithm behavior

As expected, we see in Figure 3.3.a) that the success ratio drops drastically in networks with low node density and high node mobility. Low node density results in a very small number of alternative paths and packets are frequently dropped. The sink detects decrease in the quality of delivered data and therefore new refresh cycles are generated often, which in turn increases the amount of control overhead. This is shown in Figure 3.3.b).

We also can see that the increase of node density is handled well by the algorithm in terms of overhead. High node density allows more redundant paths, therefore decreasing the number of required refresh cycles..

3.2.3 Effects of limited refresh

The purpose of the *Limited Refresh* enhancement is to limit the control overhead of the refresh cycles. We will explore both the advantages of *Limited Refresh* and the influence of the TTL parameter. In order to emphasize the effects, we change the simulation environment as follows:

The simulation field size is [1500mX1500m].

- Number of nodes - 260
- Sink and Data sources positioning is changed as follows:

The enhancement dictates that if the sink detects deterioration of data rate from one source, it performs *Limited Refresh*, but upon observing such deteriorations from more than one source, it performs Global Refresh.

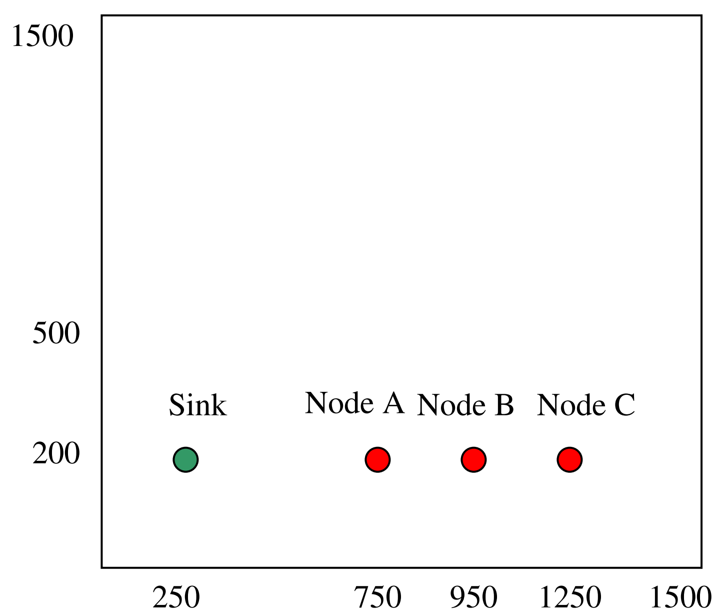
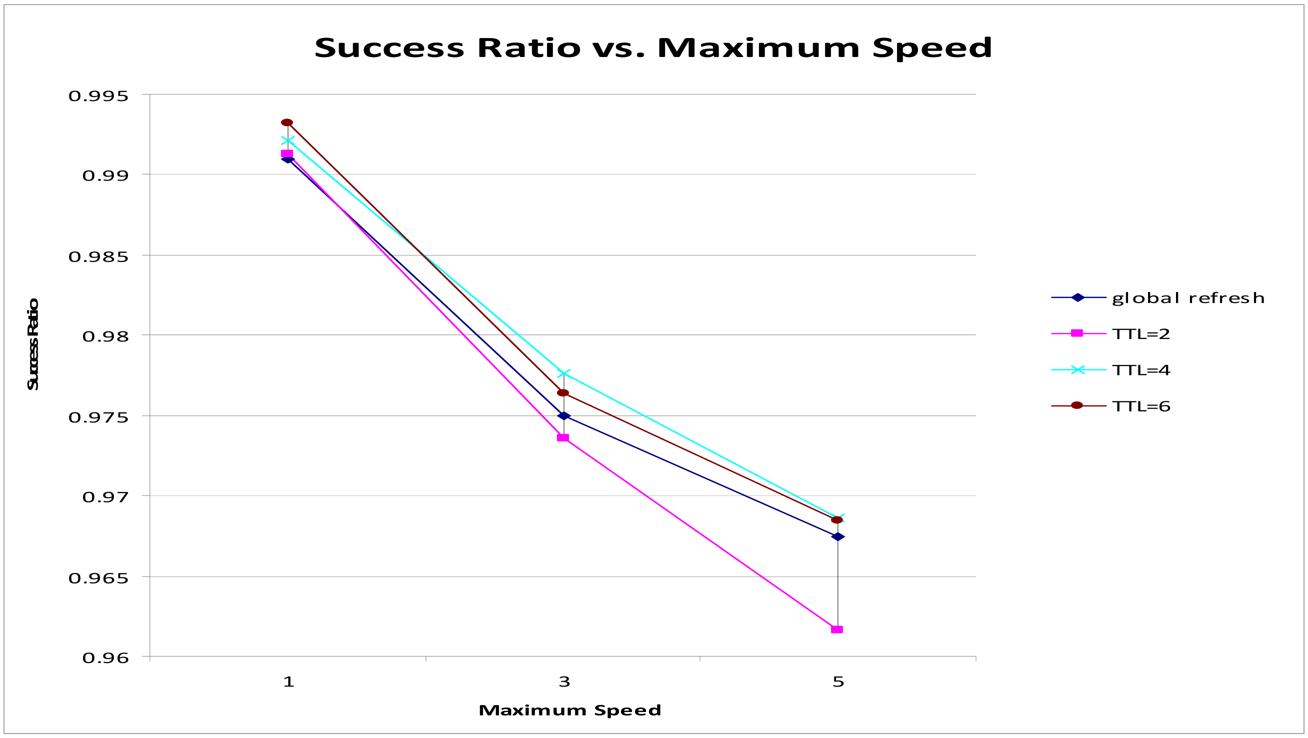
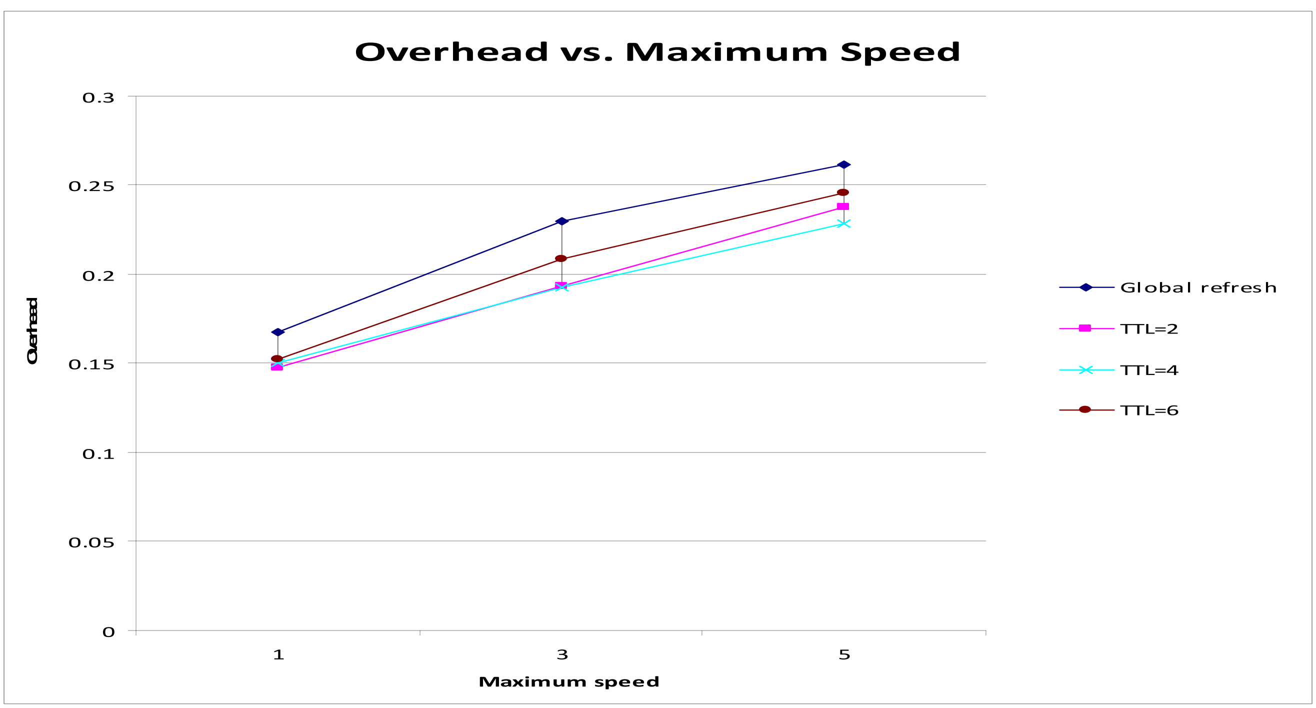


Figure 3.4 – Sink and active nodes positioning

The purpose of the scenario is to show a case when all sources are concentrated in one direction. This is the case when Limited Refresh is most efficient, since it saves a large amount of overhead, while the success ratio is almost unaffected, as seen from Fig. 3.5. We also investigate how the TTL parameter affects the performance. TTL = 2 saves in overhead per cycle, but results in more cycles due to smaller amount of alternative paths. TTL = 6 has opposite effects and behaves almost as global refresh. Seems that TTL = 4 is a good choice.



a). Success Ratio vs. Maximum speed



b). Overhead vs. Maximum speed

Figure 3.5 – Limited Refresh performance

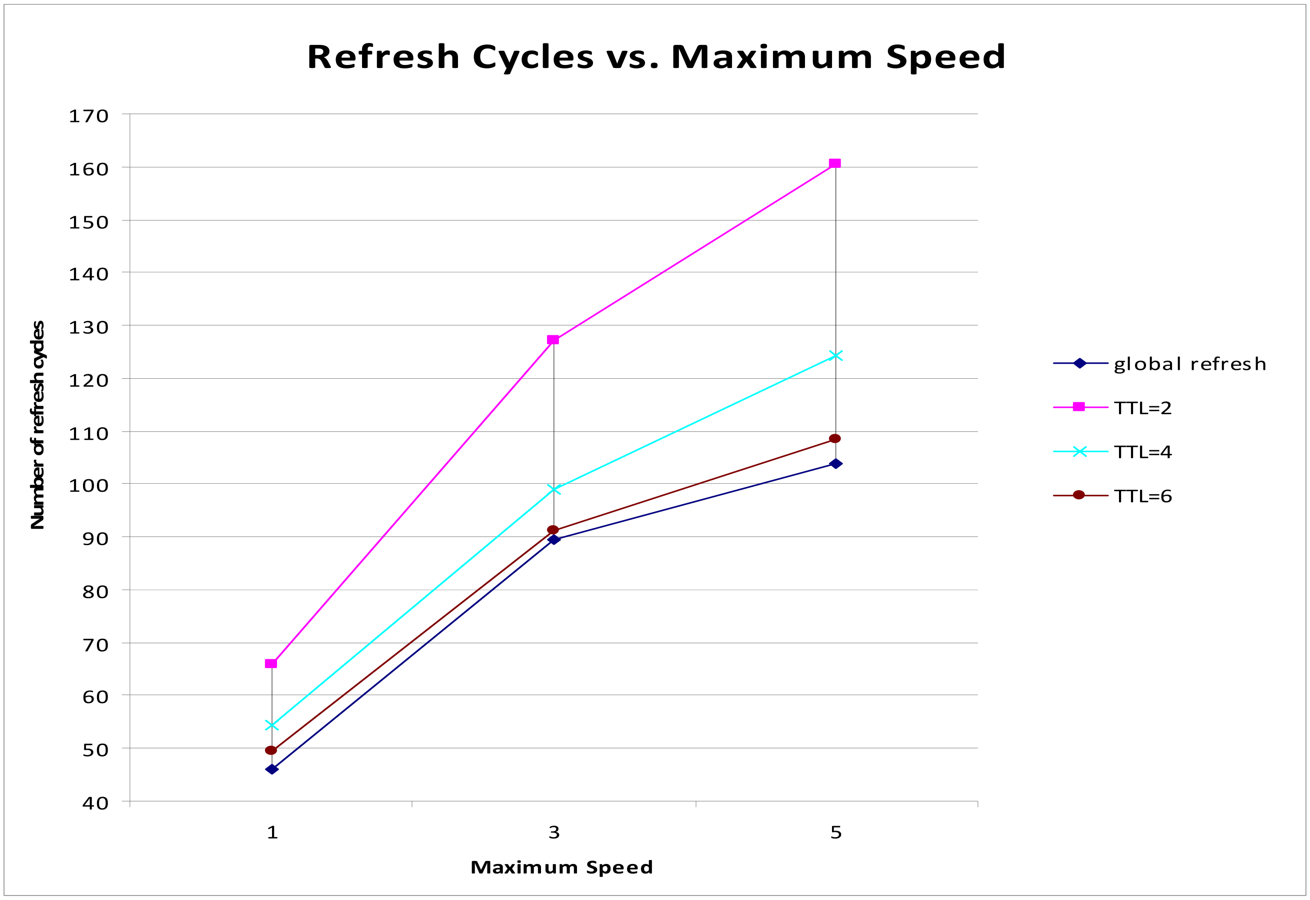


Figure 3.6 – Limited refresh cycles

3.3 Comparison of algorithm performance

In this section we compare the performance of our algorithm with that of previously proposed algorithms that have been described in Sec. 1.3 and 1.2.1.

3.3.1 GRAB

The GRAB algorithm does not address explicitly node mobility, but it can cope with it by creating duplicated data packets which are forwarded on multiple paths. Topology changes are monitored by collecting data delivery parameters at the sink. Therefore the parameter that can affect performance is the width of the forwarding mesh. However, the mesh width is much harder to control in a mobile environment than in a static one, because the movement of the nodes may create extra credit.

3.3.2 RCDR

The RCDR algorithm also uses duplicated data packets, but as opposed to GRAB, is designed to deal with mobility. RCDR monitors neighbors and performs management operations when changes occur.

The algorithm assumes that the MAC layer contains an enhancement that detects topological changes, like appearance and/or disappearance of a neighbor.

3.3.3 DD/SIR

In this algorithm, the sink is not aware of the current hop distance to data sources. Therefore, in our implementation, the sink is configured to poll the network with hop count between 1 and a predefined maximum hop count.

3.3.4 AODV

In the simulations we used the AODV implementation that exists in ns-2. AODV is a reactive protocol and therefore the paths are established from each data source to each data sink.

3.3.5 Summary of the results

Several conclusions related to the success ration can be obtained based on results presented in Fig. 3.7.

- Low density scenarios favor *Reverse-path-based forwarding* algorithms over *Cost field-driven dissemination* algorithms because the forwarding mesh of the latter required for data delivery redundancy is very limited.
    - DD/SIR performs better than DCBM because it does not require path redundancy. The data is transferred immediately upon receipt of the polling control message, before changes in topology may occur.
    - RCDR performs better than GRAB because it has mechanisms to cope with node mobility and adapts the cost field by employing local neighbors' interactions.
- High density scenarios slightly favor *Cost field-driven dissemination* because a high level of redundancy is achieved via duplicated packets.

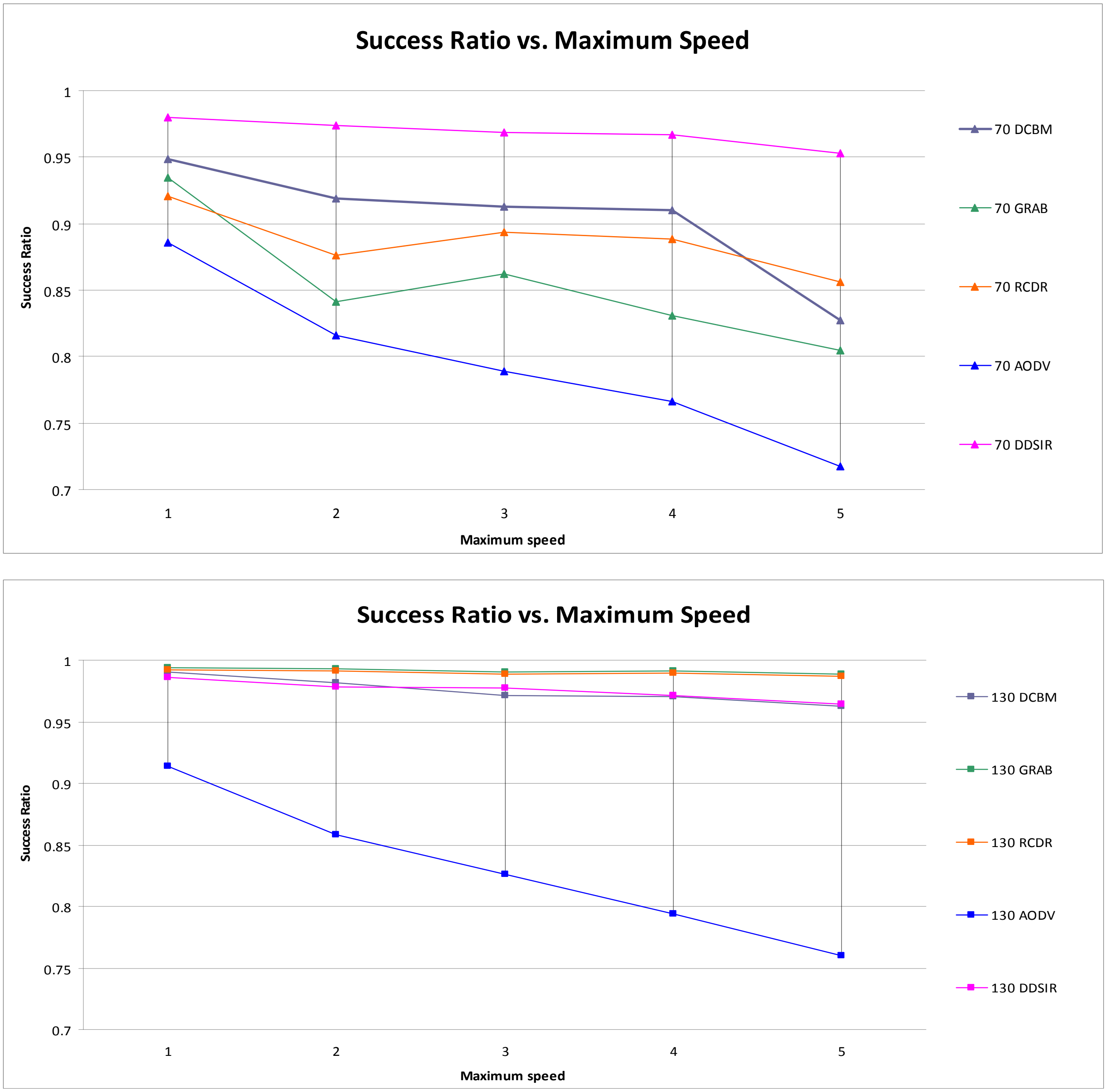


Figure 3.7 – Algorithms comparison, Success Ratio vs. Maximum speed

The results presented in Fig. 3.8 provide several conclusions in terms of overhead:

- The overhead of *Reverse-path-based forwarding* algorithms is lower than the overhead of the *Cost field-driven dissemination* algorithms due to the fact that the latter employs duplicated packets.
    - RCDR has more overhead than the GRAB algorithm because of the local cost field adaptation mechanism, which requires neighbor negotiation upon detecting a change in the neighbor status.

- DCBM has less overhead than the DD/SIR algorithm because the later creates multiple polling cycles instead of a single one as used in DCBM.

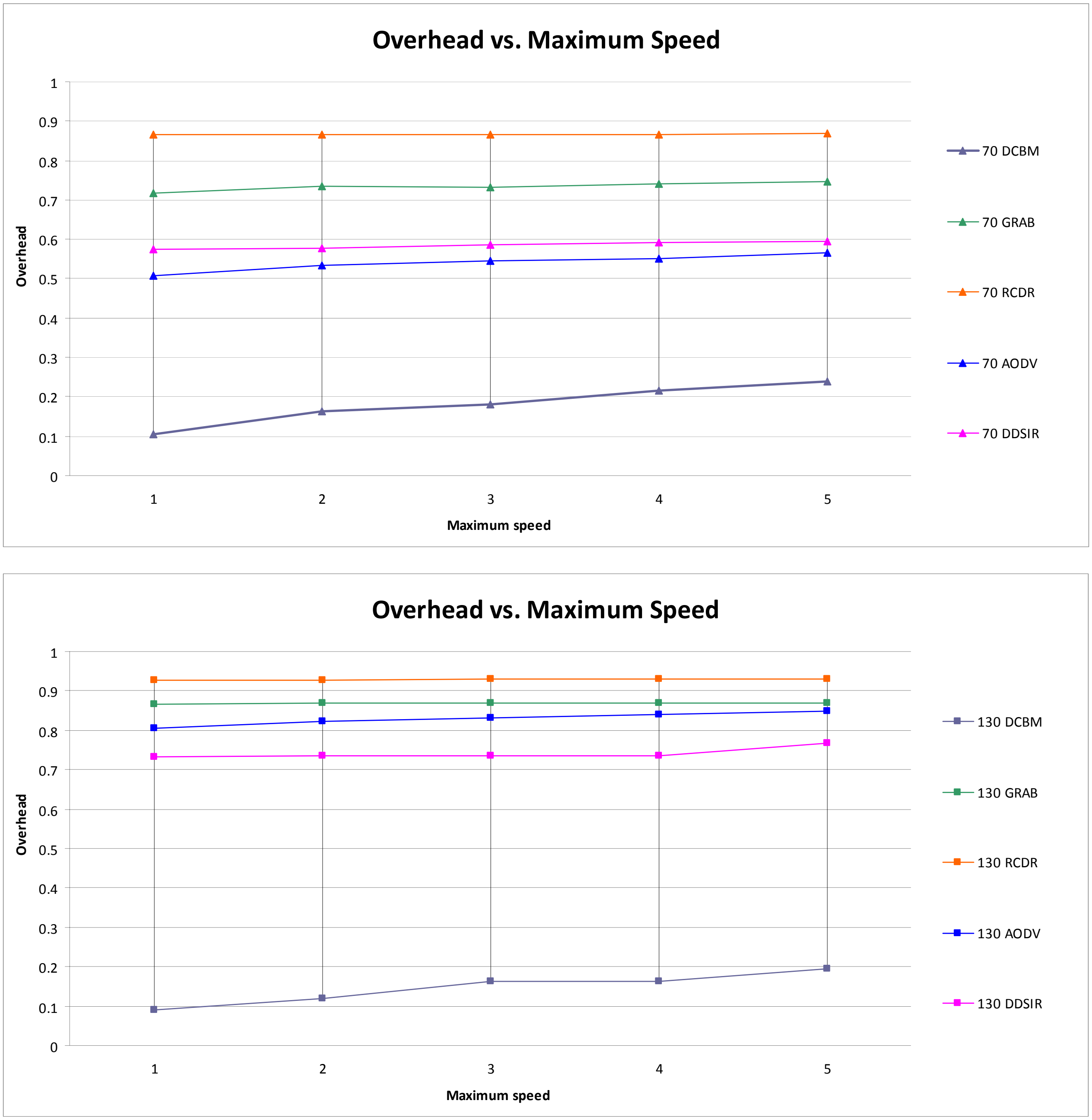


Figure 3.8 – Algorithms comparison, Overhead vs. Maximum Speed

Chapter 4 - Summary

We have shown that our *Reverse-path-based forwarding* algorithm, DCBM, is well suited to cope with mobile WSN environments. The main limitation of the WSN environment is the energy of the deployed sensors. Our DCBM algorithm creates and maintains a braided multipath forwarding scheme, whose maintenance requires a relatively small amount of overhead. Furthermore, the redundancy of the braided multipath and the local maintenance mechanism allow a high level of data delivery success ratio. We proved the properties of the algorithm such as convergence to optimal path and loop avoidance. These properties are important when considering the deployment of the algorithm in real environments.

Our simulations suggest that the algorithm may be used for applications requiring a constant data rate from data sources like sensors that detect certain phenomena and deployed in environments with sensor mobility. Examples of such applications can be the gathering of health information from tags deployed in livestock management systems, micro sensors deployed into patient blood streams and environmental monitoring, such as oceans stream monitoring.

Chapter 5 - References

[1] A.G. Ruzzelli, R. Tynan, G.M.P. Hare, “An energy-efficient and low-latency routing protocol for wireless sensor networks” Systems Communications, 2005. Proceedings Volume , Issue , 14-17 Aug. 2005 Page(s): 449 – 454.

[2] W. Ye, J. Heidemann, and D. Estrin. “Medium access control with coordinated adaptive sleeping for wireless sensor networks”. Twenty-First Annual Joint conference of the IEEE Computer and Communication Societies (INFOCOM), 3:1567–1576, June 2002

[3] L. van Hoesel and P. Havinga. “A lightweight medium access protocol (LMAC) for wireless sensor networks”. In 1st Int. Workshop on Networked Sensing Systems (INSS 2004), June 2004.

[4] R. Kalidindi, L. Ray, Ra. Kannan1, Si. Iyengar, "Distributed Energy Aware MAC Layer Protocol For Wireless Sensor Networks", International Conference on Wireless Networks , Las Vegas, Nevada, June 2003.

[5] P. Kok Keong Loh, “A Scalable, Efficient and Reliable Routing Protocol for Wireless Sensor Networks”, J. Ma et al. (Eds.): UIC 2006, LNCS 4159, pp. 409 – 418, 2006. © Springer-Verlag Berlin Heidelberg 2006

[6] J. Zhao, A.T. Erdogan, “A Novel Self-Organizing Hybrid Network Protocol for Wireless Sensor Networks”. Adaptive Hardware and Systems, 2006. AHS 2006. First NASA/ESA Conference on Volume , Issue , 15-18 June 2006 Page(s): 412 – 419

[7] C. E. Perkins and E. M. Royer. “Ad-hoc On-Demand Distance Vector Routing”. Proceedings of the 2-nd IEEE Workshop on Mobile Computing Systems and Applications, pages 90–100, New Orleans, LA, February 1999.

[8] C. Perkins, E. Royer, S. Das, “Ad Hoc On Demand Distance Vector (AODV) Routing”, IETF Internet draft, Feb 2003.

[9] K. Kim, W. Jung, J. Park, H. Seo, S. Jo, C. Shin, S. Park, and H. Kim, “A Resilient Multipath Routing Protocol for Wireless Sensor Networks”, P. Lorenz and P. Dini (Eds.): ICN 2005, LNCS 3421, pp. 1122–1129, 20. Springer-Verlag Berlin Heidelberg 2005

[10] D. Johnson, Y. Hu and D. Maltz. “The dynamic Source Routing Protocol for Mobile Ad Hoc Networks”, IETF Internet draft, Apr 2003.

[11] R. V. Boppana and A. Mathur, “Analysis of the Dynamic Source Routing Protocol for Ad Hoc Networks,” IEEE Workshop on Next Generation Wireless Networks (WoNGeN), December 2005.

[12] S. Dulman, J. Wu, P. Havinga, "An energy efficient multipath routing algorithm for wireless sensor networks", IEEE International Symposium on Autonomous Decentralized Systems (ISADS 2003), Pisa, Italy.

[13] C. Perkins, "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers" , ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications.

[14] T. Camilo, J. Sá Silva, F. Boavida, "Assessing the use of ad-hoc routing protocols in Mobile Wireless Sensor Networks" in Proc. of Conference on Mobile and Ubiquitous Systems, CMUS2006, Braga June 2006.

[15] F. Ye, H. Luo, J. Cheng, S. Lu, and L. Zhang, "Dissemination protocols for large sensor networks. Wireless sensor networks book contents", Pages: 109 - 128 Year of Publication: 2004. ISBN:1-4020-7883-8

[16] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. "Directed Diffusion for Wireless Sensor Networking" *ACM/IEEE Transactions on Networking*, *11* (1), pp. 2-16, February, 2002.

[17] M. Saleem, M. Farooq, "A framework for empirical evaluation of nature inspired routing protocols for wireless sensor networks" ; Evolutionary Computation, 2007. CEC 2007. IEEE Congress Publication Date: 25-28 Sept. 2007, On page(s): 751-758. Singapore.

[18] N. Nasser, Y. Chen, "Secure Multipath Routing Protocol for Wireless Sensor Networks", 27th International Conference on Distributed Computing Systems Workshops (ICDCSW'07).

[19] M. Bandai, T. Mioki, and T. Watanabe, "A Routing Protocol with Stepwise Interest Retransmission for Wireless Sensor Networks", IEICE TRANS. COMMUN., VOL.E91–B, NO.5 MAY 2008.

[20] F. Ye, G. Zhong, S. Lu, and L. Zhang, "GRAB: A Robust Data Delivery Protocol for Large Scale Sensor Networks," presented at IEEE IPSN, 2003.

[21] F. Ye, S. Lu, and L. Zhang. "GRAdient Broadcast: A Robust, Long-lived Large Sensor Network". http://irl.cs.ucla.edu/papers/grab-tech-report.ps, 2001.

[22] T. Bokareva, N. Bulusu, S. Jha, "A performance comparison of data dissemination protocols for wireless sensor networks" Global Telecommunications Conference Workshops, 2004. GlobeCom Workshops 2004. IEEE Volume , Issue , 29 Nov.-3 Dec. 2004 Page(s): 85 – 89.

[23] F. Ye, H. Luo, J. Cheng, S. Lu, and L. Zhang, "A Two-Tier Data Dissemination Model for Large-scale Wireless Sensor Networks", ACM International Conference on Mobile Computing and Networking (MOBICOM'02), 2002.

[24] J. Wu, P. Havinga, "Reliable Cost-based Data-centric Routing Protocol forWireless Sensor Networks" Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2006. SNPD 2006. Seventh ACIS International Conference on Volume , Issue , 19-20 June 2006 Page(s): 267 – 272.

[25] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin."Highly-resilient, energy-efficient multipath routing in wireless sensor networks". ACM SIGMOBILE Mobile Computing and Communications Review, 5(4):11–25, 2001

[26] A. Segall, "Distributed Network Protocols Lecture Notes" CC PUB #294, October 1999.

[27] Y. Ledvitch and A. Segall, "Threshold-Related Throughput – a New Criterion for Evaluation of Sensor-Networks Performance", Ad-Hoc Networks Journal, Special Issue on Recent Research Directions in Wireless Ad Hoc Networking, Vol. 5, Issue 8, pp. 1329- 1348, Nov. 2007.

[28] http://en.wikipedia.org/wiki/Ns-2

Appendix A. - Properties of the Delayed Propagation algorithm

We first show that the number of control messages in each cycle is limited.

Theorem 2.1
In each cycle, every node i sends at most one control packet $MSG1$ and at most one $MSG2$.

Proof:
From <R>, <R1>,<S>,<T1> follows that part T is executed by a node i only once in each cycle, since if t_{int} expired it can be defined again only when first packet of new cycle is received. Moreover, <T4> and <T6> are the only lines where $MSG1$ is sent, therefore node i can send at most one $MSG1(\bullet, c, \bullet)$ for any cycle c. Assume now that *MSG2* is sent twice by some node. Let *t* be the first time when *MSG2* is sent for the second time by some node, i say. The designated neighbor $e_i[c]$ is elected only once by i for each cycle c, <T10>. Since *MSG2* is sent by i only upon or after receiving *MSG2* from $e_i[c]$, <T7> and <T10> or <V3> and <V6> . This means that $e_i(c)$ sent *MSG2* twice before time *t,* contradiction.

Energy depletion can be also caused by data forwarding loops. In Theorem 2.2 we show that the *active next hops* do not form a loop. In the sequel, we show that messages can still loop, but only for the brief moment of message propagation.

Theorem 2.2
No loop in terms of r_i (*active next hops*) in any snapshot of the network

We first prove the following Lemma:
Lemma 2.3
If k=r_j then holds at any given time either:
a) $c2_j < c2_k$ or
b) $(c2_j = c2_k)$ and $D2_j(k) > D2_k(r_k)$

Proof

From < D > and <D1> follows that at every node i , the counter $c2_i$ is non-decreasing.

We first prove that $c2_j \le c2_k$ .Suppose the opposite namely $c2_j > c2_k$. Statement <D> and <D1>show that $MSG2(k, c2_j, \bullet)$ was received by node j. But at the time node k was selected to be

r_j in <H5> ,it was true that $n_j(k)=1$. The variable $n_j(k)$ can be set to 1 only in <E7> or <E12>. According to lines <D>, <D3> ($n_j(k)$ is set to -1 when the first message MSG2 of cycle c is received) and line <E>, receiving $MSG2(k,c,...)$ is a prerequisite for statement n_j(k) =1 to hold while $c2_j = c$. Therefore k is not eligible to be r_j. This proves that indeed $c2_j \le c2_k$.

Now we prove that if $c2_j = c2_k$ then $D2_j(k) > D2_k(r_k)$. According to lines <H5>, the fact that k=r_j implies n_j(k) =1. As previously shown, for statement n_j(k) =1 to hold in cycle $c2_j$, node *j* must receive $MSG2(k,c2_j,D2_k)$. Therefore k has send $MSG2(k,c2_j,D2_k)$. Then according to lines <C13>, <E8> and <E11> the quantity $D2_k$ sent in this message must be strictly larger than $D2_k(m)$ for any m eligible to be r_k (n_k(m) =1).

Moreover we know:

$D2_j(k) = D2_k$ according to line <E2> , thus $D2_j(k) = D2_k > D2_k(r_k)$

Proof of Theorem 2.2

Since Lemma 2.3 shows that $c2_j$ must be nondecreasing around the loop, all the $c2_j$ in the loop must be equal. But Lemma 2.3 b) shows that $D2_j$ must be strictly decreasing, contradiction.

Next we shall show that the algorithm converges to optimal routing in final number of refresh cycles if no changes or packet loss occurs.

Theorem 2.3

Suppose that changes in the network topology cease before the time when cycle c' starts (nodes are stationary, link weights are constant, propagation time is constant). Then a finite number of path refresh cycles afterwards, the distance parameter $D1_i\left[c\right]$ held by each node does not change between refresh cycles and is identical to the optimal distance to the sink. In addition, the *designated neighbor* $e_i(c)$ is the next hop on the optimal path from *i* to sink.

Denote:

$D1_i^*$ = optimal distance of node i to the sink

$D1_i[c]$ = the distance $D1_i$ at cycle c

$D1_i(k)[c]$ =the distance $D1_i(k)$ at cycle c

e_i^* = next hop neighbor on optimal path of node i to the sink

<u>Lemma 2.4</u>

If starting refresh cycle c'', all distance parameters $D1_i\left[c\right]$ held by all nodes do not change between refresh cycles, then $\forall c > c''$ each node i holds $e_i[c] = e_i^*[c]$ and $D1_i[c] = D1_i^*$.

<u>Proof</u>

Assume there is at least one node i with $D1_i[c] < D1_i^*$ for some $c > c''$. Let K be the group of nodes with $D1_i[c] < D1_i^*$. Suppose $j \in K$ and j is the node with minimal $D1_i[c]$, namely $D1_j[c] \le D1_i[c] \quad \forall i \in K$.

For $c > c''$, denote $k = e_i\left[c\right]$. Due to statements <S2>, <S3> and <T2>, holds $D1_j[c] = D1_k[c] + d_{kj}$. Therefore, since d_{kj} is strictly positive, $k \notin K$, holds $D1_k[c] \ge D1_k^*$. Since k and j are neighbors, $D1_k^* + d_{kj} \ge D1_j^*$, so that we finally get $D1_j[c] = D1_k[c] + d_{kj} \ge D1_k^* + d_{kj} \ge D1_j^*$, contradicting the fact that $j \in K$.

Assume now that there is at least one node i with $D1_i[c] > D1_i^*$ for some $c > c''$. Let K be the group of nodes i with $D1_i[c] > D1_i^*$. Suppose j is the node in group K with minimal $D1_i^*$, namely $D1_j^* \le D1_i^* \quad \forall i \in K$ and k is the next hop of node j in the optimal path to sink. Obviously holds $D1_k^* + d_{kj} = D1_j^*$. Therefore, since d_{kj} is strictly positive, $k \notin K$, holds $D1_k[c] \le D1_k^*$.

According to statements <S> and <S2>, the parameter $D1_j[c]$ is selected as the minimum of $D1_j(i)$ over all neighbors of j. Therefore $D1_k[c] + d_{kj} \ge D1_j[c]$, so we finally get $D1_j^* = D1_k^* + d_{kj} \ge D1_k[c] + d_{kj} \ge D1_j[c]$ contradicting the fact that $j \in K$.

We now prove that indeed the distances $D1_i\left[c\right]$ stop changing.

<u>Lemma 2.5</u>

Starting refresh cycle $c'+1$, for every node j and every finite number z, there is a finite number of events when j reduces its $D1_j$ to a value $\le z$.

Proof

This is shown by first proving that for every event in a node, there has been a corresponding event in one of its neighbors. According to statements <S> and <S2>, node j may reduce $D1_j[c-1]$ in refresh cycle $c > c'+1$ to a value $D1_j[c]$ in three cases:

- Node j receives the message of refresh cycle c, $MSG1(k,c,D1_k[c])$, that satisfies $D1_k[c]+d_{kj} < D1_j[c-1]$. Where node k was one of the neighbors that its $MSG1(k,c-1,D1_k[c-1])$ has arrived <u>before</u> $e_j\left[c-1\right]$ was set. Therefore, from statements <S> and <S2> in the algorithm, follows $D1_j[c-1] \le D1_k[c-1]+d_{kj}$. Thus $D1_k[c] < D1_k[c-1]$ and also $D1_k[c] < D1_j[c-1]$.
- Node j receives the message of refresh cycle c, $MSG1(k,c,D1_k[c])$, that satisfies $D1_k[c]+d_{kj} < D1_j[c-1]$. Where node k was one of the neighbors that had its $MSG1(k,c-1,D1_k[c-1])$ arrive <u>after</u> $e_j\left[c-1\right]$ was set. We assume that after cycle c' message propagation times do not change. Thus the order and the timing of message receipt in cycle c and in cycle $c-1$ is identical unless distance of node k to sink decreased (decreasing the delay experienced by $MSG1$ on its paths from sink to node k). Thus $D1_k[c] < D1_k[c-1]$ and also $D1_k[c] < D1_j[c-1]$.
- Node j found a new neighbor in cycle l but this cannot happen after cycle c, since no topological changes occur.

Denote by K the set of nodes j that reduce their $D1_j[c]$ an infinite number of times to values $D1_j[c] \le z$. For $j \in K$, denote $z_j = \liminf D1_j[c]$. Clearly, $z_j \le z$ and let j^* be the node that achieves $\min z_j$ over $j \in K$. As shown above, to every event $D1_{j^*}[c] < D1_{j^*}[c-1]$ corresponds an event $D1_k[c] < D1_k[c-1]$ at some neighbor k and also, correspondingly $D1_k[c] < D1_{j^*}[c]$. Since j^* has only a finite number of neighbors, it must have a neighbor k^* that has an accumulation point of $D1_{k^*}[c]$ at $z_{j^*} - d_{j^*k^*}$. Therefore $k^* \in K$ and $z_{k^*} < z_{j^*} - d_{j^*k^*}$ contradicting the fact that z_{j^*} is minimal.

Lemma 2.6

Starting refresh cycle $c'+1$ for every node j and every finite number z there is a finite number of events when j increases its $D1_j$ from a value $\le z$.

Proof

This is shown by first proving that for every event in a node, there has been a corresponding event in one of its neighbors. According to statements <S> and <S2>, , node j may increase $D1_j[c]$ in refresh cycle $c > c'+1$ from a value $D1_j[c-1]$ in three cases:

- Node j receives the message of refresh cycle c , $MSG1(e_j[c-1], c, D1_{e_j[c-1]}[c])$ that satisfies $D1_{e_j[c-1]}[c] + d_{e_j[c-1]j} > D1_j[c-1]$ from its *designated neighbor* $e_j[c-1]$. And the message arrives <u>before</u> $e_j[c]$ was set. Therefore, from lines <S2>,<S3> and <T2> in the algorithm follows that $D1_j[c-1] = D1_{e_j[c-1]}[c-1] + d_{e_j[c-1]j}$. Thus $D1_{e_j[c-1]}[c] > D1_{e_j[c-1]}[c-1]$ and also $D1_{e_j[c-1]}[c-1] < D1_j[c-1]$.
- Node j receives the message of refresh cycle c from neighbor $e_j[c-1]$ <u>after</u> $e_j[c]$ was set. . We assume that after cycle c' message propagation times do not change. Thus the order and the timing of message receipt in cycle c and in cycle $c-1$ is identical unless distance of node k to sink increased (increasing the delay experienced by $MSG1$ on its paths from sink to node k). Therefore, from lines <S2>,<S3> and <T2> in the algorithm, follows that $D1_j[c-1] = D1_{e_j[c-1]}[c-1] + d_{e_j[c-1]j}$. Thus $D1_{e_j[c-1]}[c] > D1_{e_j[c-1]}[c-1]$ and also $D1_{e_j[c-1]}[c-1] < D1_j[c-1]$.
- Node j loses its *designated neighbor* during refresh cycle $c-1$, but this cannot happen because no topology changes occur.

Denote by K the set of nodes that increase their $D1_i[c]$ an infinite number of times from values $D1_j[c] \le z$. For $j \in K$, denote $z_j = \liminf D1_j[c]$. Clearly $z_j \le z$ and let j^* be the node that achieves $\min z_j$ over $j \in K$. As shown above, to every event $D1_{j^*}[c] > D1_{j^*}[c-1]$ corresponds an event $D1_k[c] > D1_k[c-1]$ or $D1_k[c-1] > D1_k[c-2]$ at some neighbor k of j^*. We have also shown that $D1_k[c-1] < D1_{j^*}[c-1]$ or $D1_k[c-2] < D1_{j^*}[c-1]$. Since j^* has only a finite number of neighbors, it must have a neighbor k^* that has an accumulation point of $D1_{k^*}[c]$ at $z_{j^*} - d_{j^*k^*}$. Therefore $k^* \in K$ and $z_{k^*} < z_{j^*} - d_{j^*k^*}$ contradicting the fact that z_{j^*} is minimal.

Proof of the Theorem 2.3

Since every new value of $D1_i[c]$ is either after an increase or after a decrease, Lemmas 2.5 and 2.6 show that there is only a finite number of new values of $D1_i[c] \leq z$ for every finite z and therefore a finite number of changes in $D1_i[c]$. Thus the conditions of Lemma 2.4 hold and thus there final values are optimal.

Next we give some indication as of the number of broadcasts that a data message can experience at any given node.

Lemma 2.7

If a node j broadcasts a *DataMSG* and subsequently broadcasts the same *DataMSG* again, then the cycle counter $c2_j$ must have been increased between the two events.

Proof

Denote by $t1$ and $t2$ the time of the two events respectively. Let $\{j, r_1, r_2, r_3 ..., r_l, j\}$ represent the path of *DataMSG*. According to Lemma 2.3 holds $c2_j(t1) \leq c2_{r_1} \leq ... \leq c2_{r_l} \leq c2_j(t2)$ at the time of the broadcast. Therefore, if $c2_j(t1) = c2_j(t2)$, holds $c2(t1) = c2_{r_1} = ... = c2_{r_l} = c2(t2)$ at the time of broadcast. According to Lemma 2.3, the equality in counter numbers above implies $D2_j(r_1)(t1) > D2_{r_1}(r_2) > ... > D2_{r_l}(j) > D2_j(k)(t2)$ at the time of the broadcast, where k is the *active next hop* of node j. We get $D2_j(r_1)(t1) > D2_j(k)(t2)$, and since $c2_j(t1) = c2_j(t2)$ this leads to a contradiction to statement <H5> that implies: $D2_j(r_1)(t1) \leq D2_j(k)(t2)$.

Theorem 2.4

Let N be the number of nodes in the network and $t^{\max}$ the maximum propagation time between each two neighbors. If the time between two refresh cycles is larger than $3 * N * t^{\max}$, then each *DataMSG* can be broadcast by a given node at most twice.

Proof

We prove the Theorem by contradiction. Suppose *DataMSG*'s can be broadcast by nodes more than twice and let j be the *first* node that broadcasts a *DataMSG* for the third time. Let

$t1$ - time of first broadcast of *DataMSG* at node j

$c2' = c2_j(t1)$, cycle counter at node j at $t1$

$t2$ - time of second broadcast of *DataMSG* at node j

$c2'' = c2_j(t2)$, cycle counter at node j at $t2_i$

$T(c2')$ – time the first message of refresh cycle $c2'$ was broadcast by the *sink*

$T(c2'')$ – time the first message of refresh cycle $c2''$ was broadcast by the *sink*

N - number of nodes in the network

t^{max} - maximum propagation of a single hop

t_p^{min} - minimum propagation of a single hop

$t3$ - time of third broadcast of *DataMSG* at node j

$c2''' - c2_j(t3)$, cycle counter at node j at $t3$

$T(c2''')$ – time the first message of refresh cycle $c2'''$ was broadcast by the *sink*

According to Lemma 2.5, the value of $c2_j$ is increased each time the packet is broadcast by node j. We know that $t1 < T(c2'') + N * t_p^{max}$, since cycle $c2''$ starts at time $T(c2'')$ and therefore by time $T(c2'') + (N-1) * t^{max}{}_p$ all nodes in the network have $c2_i \geq c2''$. We also know that $t3 \geq T(c2''') + t^{min}{}_p$, since only after $T(c2''') + t^{min}{}_p$ the first node can change its $c2_i$ to $c2_i = c2'''$. Therefore we can conclude that: $t3 - t1 \geq T(c2''') + t_p^{min} - (T(c2'') + N * t_p^{max}) > 3 * N * t_p^{max} - N * t_p^{max}$

Thus, *DataMSG* has passed at least one node in the network more than twice before arriving at node j, contradicting the fact that j is the first node to have broadcast *DataMSG* 3 times.