PERFORMANCE OPTIMIZATION OF AD-HOC NETWORK USING BACK UP PATH TECHNIQUE

BY

Md. Rayhan Arafat
Id: 062-19-462

This Report Presented in Partial Fulfillment of the Requirements for the Degree of Bachelor of Science in Electronics and Telecommunication Engineering

Supervised By

Md. Whaiduzzaman
Senior lecturer

Department of ETE

Daffodil International University

DAFFODIL INTERNATIONAL UNIVERSITY

DHAKA, BANGLADESH

FEBRUARY 2010

APPROVAL

This Project titled "**Performance Optimization of Ad-hoc wireless network using back up technique**" by Md. Rayhan Arafat has been submitted to the Department of Electronics and Telecommunication Engineering of Daffodil International University in partial fulfillment of the requirements for the Degree of Bachelor of Science in Electronics and Telecommunication Engineering. This Project has been accepted as satisfactory by the following Honorable member of the Board of Examiners after its presentation that was held on February 26, 2010.

Board of Examiners

_____

Dr. Md. Golam Mowla Choudhury **Chairman**
Professor and Head
Department of Electronics & Telecommunication Engineering
Daffodil International University

_____

A.K.M. Fozlul Haque **Internal Examiner**
Assistant Professor
Department of Electronics & Telecommunication Engineering
Daffodil International University

_____ **Internal Examiner**

MD. Mirza Golam Rashed
Assistant Professor
Department of Electronics & Telecommunication Engineering
Daffodil International University

_____

Dr. Subrata Kumar Aditya **External Examiner**
Professor & Chairman
Department of Applied Physics, Electronics & Communication Engineering
University of Dhaka

DECLARATION

We do declare that the work presented in this Project is done by us under the supervision of **Md. Whaiduzzaman, Senior Lecturer, Department of ETE, DIU** We also declare that neither this report nor any part thereof has been submitted elsewhere for the award of any degree or diploma.

Supervised by:

_____

Md. Whaiduzzaman
Senior Lecturer
Department of ETE
DIU

Submitted by:

_____

Md. Rayhan Arafat
ID: 062-19-462
(Candidate)

ACKNOWLEDGMENT

I would like to express our deep felicitation to our supervisor, **Md. Whaiduzzaman** for introducing me the interesting and widespread area of Ad-hoc wireless network and teaches me how to perform research works. My project works couldn't be accomplished without his invaluable cooperation and guidance during the course of project. I am highly indebted to him for constantly encouraging me by giving his critics on my work. I am grateful to him for having given me the support and confidence. I would also like to express my warm greetings and thanks to all of our teachers for their encouragement and motivation. Furthermore, I take my elder brother named Mr. HariChandan Roy's kind help to write the JAVA CODE. I also gratitude to him.

Md. Rayhan Arafat
Dept. of ETE
Daffodil International University

ABSTRACT

Minimizing energy consumption and ensuring fault tolerance are two important issues in ad-hoc wireless networks. In this paper, we describe a distributed topology control algorithm which minimizes the amount of power needed to maintain bi-connectivity. The algorithm selects optimum power level at each node based on local information only. The resultant topology has two properties: (1) it preserves the minimum energy path between any pair of nodes and (2) it ensures fault tolerance by maintaining bi-connectivity. By presenting experimental results, we show the effectiveness of our proposed algorithm.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Ad-hoc wireless networks are getting widespread with the recent development of wire-less communication system. Since the basic components of multi-hop wireless networks are mostly battery-operated devices, power conservation is one of the key issues of such networks. It is not energy client to use the communication networks where each node transmits with its maximum power. So, power control is needed which deals with the problem of choosing the minimum power level by each node to minimize the energy consumption for the whole network. To ensure minimum power level, the topology has to preserve minimum-energy paths between the nodes. Besides preserving minimum-energy path, the topology control automatically maintains some properties such as reduced average node degree, smaller average transmission power etc. The network built in this way has profound effect on the performance of routing layer. Power control also results in extending battery life of the nodes.

On the other hand, by reducing the number of links in the network, topology control algorithms actually decrease the degree of routing redundancy. As a result, the topology thus derived is more susceptible to node failures/departures. Besides this, failure of nodes is a common phenomena in ad-hoc wireless network. This problem can be mitigated if an adequate level of fault tolerance can be properly ported into topology control. Fault tolerance increases the robustness of the network maintaining connectivity in case of any breakdown or an increase in load at any vicinity of the network.

Here, we propose a distributed topology construction algorithm based on local information only. By the term local information we mean that a node only has information about the position of one hop or two hops neighbors. Requiring more than two hops neighbor information a much overhead is incurred which will subdue the benefit of the topology control. Finally, the proposed algorithm preserves all minimum-energy paths between every pair of nodes and ensures fault tolerance by maintaining global bi-connectivity.

1.1 Ad-hoc Wireless Network

An ad-hoc network is a (possibly mobile) collection of communication devices (nodes) that wish to communicate, but have no fixed infrastructure available and have no pre-determined organization of available links. A wireless ad-hoc network is a decentralized wireless network. The network is ad-hoc because each no de is willing to forward data for other no des, and so the determination of which nodes should be forwarded data is made dynamically based on the network connectivity. This is in contrast to wired networks in which routers perform the task of routing. It is also in contrast to managed (infrastructure) wireless networks, in which a special node known as an access point manages communication among other nodes. The earliest wireless ad-ho c networks were the "packet radio" networks (PRNETs)from the 1970s, sponsored by DARPA after the ALOHA net project.

1.1.1 Ad-hoc network characteristics

Some of the characteristics those are common in ad-hoc network listed bellow:

- Ad-ho c network has no fixed infrastructure.
- It is a Dynamic Topology (Mobility).
- In case of Multi-hopping, there may be obstacles and there may be needed spectrum reuse and energy conservation in ad-hoc network.
- Ad-ho c network has the property of Self-organization such as in addressing, routing, clustering, location, power control.
- Energy conservation is a crucial characteristic of ad-hoc wireless network.
- Ad-ho c network has limited security.
- Bandwidth is constrained in ad-hoc network where congestion is a norm.

1.1.2 Application of Ad-hoc network

The decentralized nature of wireless ad-hoc networks makes them suitable for a variety of applications where central nodes can't be relied on, and may improve the scalability of wireless

ad-hoc networks compared to wireless managed networks, though theoretical and practical limits to the overall capacity of such networks have been identified. Minimal configuration and quick deployment make ad-hoc networks suitable for emergency situations like natural disasters or military conflicts. The presence of a dynamic and adaptive routing protocol will enable ad-ho c networks to be formed quickly. Wireless ad-hoc networks can be further classified by their application:

- mobile ad-hoc networks (MANETs)
- wireless mesh networks
- wireless sensor networks.

1.2 Issues in Ad-hoc Networks

- Medium Access is distributed, there is no time sync and there is use of directional antennas.
- Routing in ad-hoc network has route acquisition delay and has quick reconfigure property. It is also loop free.
- Multicasting is common in ad-hoc network such as in case of emergency/military, group-oriented computing etc.
- Transport Layer has frequent path breaks.
- Support for Quality of Services (QoS) is integral to the design of ad-hoc networks.
- Uses Self-organization in case of neighbor discovery, report link failures etc.
- Security is a multi-layer issue in ad-hoc network where issues like Denial of Services (DoS), jamming, energy depletion can arise.
- Energy management is an important issue in ad-hoc network which considers transmission power, battery monitoring, processor power etc.
- Addressing and Service Discovery is global.

1.3 Research Challenges

1. Performance limits: Various kinds of performance limits such as throughput, delay, energy tradeoffs etc. remain as challenges in ad-hoc networks. Among these maintaining energy efficiency remains as an unavoidable challenge in the research field of ad-hoc networks.

Reducing energy consumption is an important goal in ad-hoc wireless networks by which network capacity and longevity can be increased. Much achievement has been gained in this respect but it still remains as a major field of study.

2. Security: Security is one of the most important and unexplored challenges in ad- hoc networks. In ad-hoc networks the communicating no des do not necessarily rely on a fixed infrastructure, which sets new challenges for the necessary security architecture they apply. Nodes are susceptible to denial of services (DoS) attacks that are harder to track down than in wired networks. Besides, as topology of ad-hoc networks vary depending on situation, level of security will also change as well.

3. Quality of Services (QoS): Support for QoS is integral to the design of ad-hoc networks. QoS is still a problem in wired networks while it becomes more challenging in ad-hoc networks. The most commonly studied QoS metrics are band width, delay and jitter. Limited bandwidth, dynamic topology, and limited processing and storage capability of the nodes make the issue of QoS harder. For ad-hoc networks, it is difficult to provide hard QoS guarantees due to fluctuations in the wireless channel and interference from non-neighboring nodes. But with the rapid development of ad-hoc networks, support for QoS has become an unavoidable task and itis providing newer challenges for researches in the field of ad-hoc wireless networks.

1.4 Problems in Ad-hoc Networks

There are quite a number of open problems in ad-hoc wireless networks. Scalability is one of them. Scalability in ad-hoc networks can be defined as whether the network is able to provide an acceptable level of service to packets even in the presence of a large number of nodes in the network. Another crucial problem is that of minimum energy conservation. As individual nodes have to rely on portable, limited power sources like battery, the idea of energy efficiency therefore becomes an important problem in ad-hoc networks. Limited network information capacity is another vital problem in ad-hoc network. No global information is available in ad-hoc network. Maintaining an adequate degree of fault tolerance is another problem which has great impact on ad-hoc networks. We concentrate on the problem of minimizing energy consumption of the network at the same time ensuring fault tolerance by maintaining bi-connectivity.

1.5 Topology Control Problem

1.5.1 Definition

Topology control is the art of coordinating nodes' decisions regarding their transmitting ranges, in order to generate a network with the desired properties. From system-level perspective, Topology control optimizes the choice of the nodes' transmit power levels to achieve a certain global property. From wireless channel perspective, Power control, optimize the choice of the transmit power level for a single wireless transmission, possibly along several hops.

1.5.2 Impact on energy conservation

Ad-ho c wireless network differs from traditional wired networks. and thus enlarge the durability of the nodes of the network .Topology control also reduces the average degree of the nodes. As a result, routing redundancy, in case of any neighbor discovery procedure such as flooding, decreases. In a word, topology control reduces the contention in a node's neighborhood as well as in the network.

1.5.3 Necessity of topology control

Both energy and capacity are limited resources in ad-ho c networks. In case of wireless ad-hoc networks, energy consumption is especially critical. The ad-hoc network designer should strive for reducing no de energy consumption and providing sufficient network capacity. The answer to the above problem is Topology Control (TC) which maintains a topology with certain properties (e.g., connectivity) while reducing energy consumption and/or increasing network capacity.

1.6 Motivation

One of the most vibrant and active new fields today is ad-hoc networks. Within the past few years, the field has seen a rapid expansion in work and research due to the extensive use of wireless devices and the interest in mobile computing. Although significant re-search has been going on ad-hoc network, there are still quite a number of problems that are open.

Since ad-hoc networks do not assume the availability of fixed infrastructure, it follows that individual nodes have to rely on portable, limited power sources. The idea of energy efficiency therefore becomes an important problem in ad-hoc network. Works have been done on this issue since the last decade of the previous century. Power control algorithms have been proposed to reduce energy consumption. But achieving energy efficient topology is still a challenging task because there is no central authority in ad-hoc wireless networks. So, a vital problem is that only local information is available in ad-hoc networks for constructing energy-efficient topology. If the problem is approached assuming global information is available, then effective solutions can be reached easily. But availability of no global information only makes this problem challenging. And this challenge prompts us to propose a distributed topology control algorithm based on local information only.

Due to the dynamic nature, ad-hoc wireless networks are more susceptible to no de failures/departures. A node failure can affect the network badly. For this reason, fault tolerance, to a given degree, should be ensured in the ad-ho c network. Fault tolerance will ensure that even in the event of a node failure or breakdown, the network can maintain connectivity and continue its operation.

1.7 Contribution

We use the same model as [7] which considers, a n -node, multi-hop, ad-ho c wireless network deployed on a two-dimensional plane. Suppose that each no de is capable of adjusting its transmission power up to a maximum denoted by P . Such a network can be mo deled as a graph G = (V, E), with the vertex set V representing the no des, and the edge set E defined as follows:

$$E = \{(x, y) | (x, y) \quad V \times V \quad d(x, y) \leq R_{max} \quad \} \qquad (1.1)$$

where d(x, y) is the distance between nodes x and y and R is the maximum distance reachable by a transmission at the maximum power P . The graph G defined this way is called the maximum powered network.

Note that the graph constructed this way is a visual representation of the inherent topology of the network. That is why we use the term topology and graph interchangeably throughout this paper. Formally, the aim of this thesis is to construct a graph $G' \leq G$ in a distributed fashion based on local information, where for any node pair u and v the minimum-energy path between u and v in G is also preserved in G' and moreover it provides fault tolerance as G has at least two vertex-disjoint paths between any two nodes. Controlling topology in this way has a benefit to maintain connectivity through another backup path and hence make the topology more resilient to any node failures or departures.

1.8 Organization of the Report

1. Chapter 2 describes the related works and Preliminaries of our thesis work.

2. Chapter 3 describes the algorithms to construct various types of topologies.

3. Chapter 4 describes the performance analysis among different topologies from the simulation results.

4. Chapter 5 discusses future work that is to be done.

Chapter 2

Background

2.1 Related Works

A significant amount of research has been directed at power control algorithms for wireless mobile networks but a very few consider the problem of minimizing energy consumption and providing fault tolerance simultaneously. Ramanathan et al. [8] considered the problem of adjusting the transmission powers of nodes and presented two centralized algorithms CONNECT and BICONN-AUGMENT.

They introduced two heuristics to deal with the dynamics of the mobile environment. But neither heuristic absolutely preserves connectivity, even if it is achievable in principle. Cone-Based Topology Control (CBTC), proposed by Li et al. [5], generates a graph structure. A serious drawback of the algorithm is the need to decide on the suitable initial power level and the increment at each step. Bahramgiri et al. [1] augmented the CBTC algorithm [5] to provide fault tolerance. However there is no guarantee that the proposed modification preserves minimum-energy paths.

Rodoplu and Meng [9] addressed a work targeting significant reductions in energy consumption. They introduced the notion of relay region based on a specific power model. Their work provides a distributed position-based network protocol optimized for minimum-energy consumption in mobile wireless networks. However the algorithm does not consider fault tolerance. Li [4] modified the algorithm of Rodoplu and Meng [9]. The sub-network constructed by their algorithm is provably smaller than that constructed by Rodoplu and Meng [9].But the algorithm has the problem of partially-enclosed nodes. If a node is partially-enclosed, it has to use its maximum transmission power, which will so on drain out its battery power. Also this algorithm does not consider fault tolerance. Shen et al. [10] proposed a distributed topology control algorithm, which preserves minimum-energy property and bi-connectivity. But the algorithm uses another algorithm presented in [3] to identify cut vertices which can not be done using local information. Acquiring global topology information is expensive and wastes more energy.

Hajiaghayi et al. [2] addressed minimizing power while maintaining k -fault tolerance.They present three approximation algorithms. The first algorithm, Global k-UPVCS(G(V,E)) where k refers that the graph remains connected when any set of at most $k - 1$ vertices is removed and k-UPVCS represents an Undirected Minimum Power k -Vertex Connected Sub graph. This algorithm gives an $O(ka)$ approximation where a is the best approximation factor for the related problem. Then, they present another more complicated algorithm which achieves $O(k)$ for general graphs. Finally, they give a distributed algorithm for the cases of 2 -connectivity and 3 -connectivity. However their distributed algorithm uses Minimum Spanning Tree (MST) for which no locally computable algorithm is available. So, their distributed algorithm is not a local algorithm. Ning Li and Jennifer C. Hou [6] considered k -connectivity of wireless network to meet the requirement of fault tolerance. They presented a centralized greedy algorithm, called Fault-tolerant Global Spanning Subgraph(FGSS) and based on F GSS , proposed a localized algorithm, called Fault-tolerant Local Spanning Subgraph (FLSS). But their work, not necessarily, contains minimum-energy paths between any two node.

2.2 Preliminaries

2.2.1 Power Model

We use the same power model as [7]. Here, we assume the well known, generic, two-way, channel path loss model where the minimum transmission power is a function of distance. To send a packet from node x to node y , separated by distance $d(x, y)$, the minimum necessary transmission power is approximated by

$$P_{trans}(x, y) = t \times d^{\alpha}(x, y) \qquad (2.1)$$

Where $\alpha \geq 2$ is the path loss factor and t is a constant. Signal reception is assumed to cost a fixed amount of power denoted by r . Thus, the total power required for one-hop transmission between x and y becomes

$$P_{trans}(x, y) = t \times d\alpha(x, y) + r \qquad (2.2)$$

The model assumes that each no de is aware of its own position with a reasonable accuracy, e.g., via a GPS device.

2.2.2 Smallest minimum-energy path-preserving subgraph of G

We say that a graph $G' \leq G$ is a minimum-energy path-preserving graph or, alternatively, that it has the minimum-energy property, if for any pair of nodes (u, v) that are connected in G, at least one of the (possibly multiple) minimum-energy paths between u and v in G also belongs to G. Minimum-energy path-preserving graphs were first defined in [?]. Typically, many minimum-energy path-preserving graphs can be formed from the original graph G. It has been shown that the smallest of such subgraphs of G is the graph $G_{min} = (V, E_{min})$, where (u, v) €Ei if there is no path of length greater than 1 from u to v that costs less energy than the energy required for a direct transmission between u and v. Let $G = (V, Ei)$ be a subgraph of $G = (V, E)$ such that (u, v) Ei if $(u, v) E$ and there is no path of length i that requires less energy than the direct one-hop transmission between u and v . Then G can be formally defined as follows:

$$G_{min} = G2 \cap G3 \cap G4 \cap ..... \cap Gn\text{-}1 \qquad (2.3)$$

2.2.3 Cover region and Cover set

Consider a pair of nodes (s, f), such that f lies within the transmission range of s , i.e., is reachable by s at Pmax . Consider the set of all points that can possibly act as relays max between s and f , such that it would be more power efficient for s to use an intermediate node located at one of those points instead of sending directly to f . We will call it the cover region of s and f and denote by $C(s, f)$. The collection of all nodes falling into the cover region of s and f will be called the cover set of s and f . Formally the cover region and cover set, are described by the following definition. The Cover Region $C(s, f)$ (Figure 2.1) of a pair of nodes (s, f) in G , where f is reachable from s is defined as:

$$C(s, f) = \{< x, y > | td\alpha(s, < x, y >) + td\alpha(< x, y >, f) + r = td\alpha(s, f)\} \quad (2.4)$$

where $\alpha = 2$. In the above equation, d(s, < x, y >) denotes the distance between node s and a hypothetical no de located at < x, y > and r is the fixed receiving power. The Cover Set of the same pair (s, f) in G is defined as:

$$\text{Є}G(s, f) = \{v | v \text{Є} \quad V \quad Loc(v) \quad C(s, f)\} \qquad (2.5)$$

where Loc(v) is the location of the node positioned at < x, y > in the network.

2.2.4 Articulation point

A vertex v in a connected graph G is an articulation point if the deletion of vertex v together with all edges incident to v disconnects the graph into two or more nonempty components. A graph G is bi-connected if it contains no articulation points.

2.2.5 Minimum-energy path-preserving bi-connected graph

A graph G` is a minimum-energy path-preserving bi-connected graph if it contains no articulation points and it keeps pairwise minimum-energy paths. A graph G` = (V , E` ) is a minimum-energy path-preserving bi-connected subgraph of a bi-connected graph G = (V, E ) if E` ЄE and G` is bi-connected and for each pair of (u, v) Є V , minimum-energy paths between u and v in G are also in G.

A variety of minimum-energy path-preserving graphs can be created if one or two hops' neighbor information is available. For example, let us consider a graph G1 = (V, E1) which is a subgraph of G = (V , E) such that (u, v) E1 if (u, v) E and there is no path 1 2 of length two that requires less energy than the direct path between u and v . Shortly,(u, v) ЄE if ЄG(u, v) is empty. Note that such graph can be locally constructed if only one hop neighbors' position information is available to a node because a path of length two between u and v can only be created by using a neighbor z of u which is also a neighbor of v . Another minimum-energy path-preserving graph is denoted as G2 = (V , E2), which (u, v) ЄE2 if (u, v) E and there are no two or more vertex-disjoint paths of length three built with the neighbors of either u or v that require less energy than the direct path between u and v . However, to construct such a graph a node must know all its one hop and two hop neighbors exact location.

Chapter 3

Topology Construction Algorithm

From the previous chapter it is known that the basic components of multi-hop wireless Networks are mostly battery-operated devices, so power conservation is one of the key Issues of such networks. Battery life depends on the transmission power of a node. More power used to transmit results less battery life. It means that it is not energy efficient to use the communication networks where each node transmits with its maximum power. We have to ensure minimum power level for each node so that battery life can be extended. Power control is thus obviously needed, which deals with the problem of choosing the minimum power level by each node to minimize the energy consumption for the whole network. To ensure minimum power level, the topology has to preserve minimum energy paths between the nodes. Besides preserving minimum-energy paths, the topology control automatically maintains some properties such as reduced average no de degree, smaller average transmission power etc.

The network built in this way (such as ensuring minimum power level) has profound effect on the performance of routing layer. In most wireless ad-hoc networks the nodes compete to access the shared wireless medium, often resulting in collisions. Performance degrades when too many packets (in-formation in packet form delivered by nodes) are present in the subnet. This situation is called congestion. When the number of packets dumped into the subnet by the hosts is within its carrying capacity, they are all delivered (except for a few that are afflicted with transmission errors) and the number delivered is proportional to the number sent. However, as traffic increases too far, the routers are no longer able to cope and they begin losing packets. This tends to make matters worse. So, having lower average node degree there is less possibility to occur such congestion. Now it is seemingly useful to reduce the number of degree in each node as much as possible preserving minimum-energy paths. But fault tolerance is an important issue because the occurrence of faults which is much common in Ad-hoc wireless networks.

In Ad-hoc wireless networks it is not energy efficient to use centralized algorithms and also it is hardly possible for ad-hoc wireless networks. Distributed algorithm is thus our attention. In this chapter we describe distributed algorithm for constructing G1, G2 and G3 topologies for a given topology G . The notations used in this chapter indicate the same meaning as in definition section. As we are considering distributed algorithm, the following algorithms run at each node as per necessity. It is obvious that ad-hoc wireless networks are much more different from wired networks. In the ad-ho c networks all the usual rules about fixed topologies, fixed and known neighbors, fixed relationship between IP address and location, and more are suddenly tossed out the window. With an ad-ho c network, the topology may be changing all the time, so desirability and even validity of paths can change spontaneously, without warning. Needless to say, these circumstances make routing in ad-hoc networks quite different from routing in their fixed counterparts. At first a node broadcasts a single neighbor discovery message (NDM) at the maximum power Pmax . All nodes receiving the NDM from that node send back a reply. And after receiving each reply we update the information of that no de and its neighbors. For example, Consider a node s is constructing either of G1, G2 and G3 . At first broadcasts a single NDM at the maximum power Pmax . All nodes receiving the NDM from s send back a reply. While s collects the replies of its neighbors, it learns their identities and locations. It also constructs the cover sets with those neighbors. Initially, all those sets are empty. The set NG(s) , which also starts with empty set, keeps track of all the no des discovered in G the neighborhood of s in G . Whenever s receives a reply to its NDM from a node v , it executes the algorithm update C over Region(s, v) described in Algorithm 1. After running

Algorithm 1 update Cover Region(s, v):

1: for each w Є $N_G(s)$ do
2: if Loc(v) Є C(s, w) then
3: $\xi G(s, w) = \xi G(s, w) \cup \{v\}$
4: else if Loc(w) C(s, v) then
5: $\xi G(s, v) = \xi G(s, v) \cup \{w\}$
6: end if
7: $N_G(s) = NG(s) \cup \{v\}$
8: end for

3.1 G1 Construction

Assume the algorithm (Algorithm 2) is running on node s. For each neighbor v of s in G this algorithm checks whether the cover set between s and v is empty or not. If it is empty then v in included into the neighbor set of s in G1 since it indicates that there is path between s and v. If it is not empty, it indicates that there is lower energy path between s and v of length 2 than the direct path between them. So the node v is not included into the neighbor set of s in G1.

Algorithm 2 G1 Topology Construction:

1: for each $v \in N_G(s)$ do
2: if $\xi G(s, v)$ is empty then
3: $N_{G1}(s) = N_{G1}(s) \cup \{v\}$
4: end if
5: end for

3.2 G2 Construction

Assume the algorithm (Algorithm 3) is running on node s . For each neighbor v of s in G this algorithm checks whether number of nodes in the cover set is less than 2 or not. If the number of nodes in the cover set is less than 2 then v is included into the neighbor set of s in G2 since it indicates that there are no two or more nodes that can be used as relay to transmit message using lower energy than the direct path between s and v . If the number of nodes in the cover set is greater than or equal 2 then it is not included into the neighbor set of s in G since it indicates there are two or more nodes that can be used as relay to transmit message in lower energy cost than the direct path between s and v .

Algorithm 3 G2 Topology Construction:

1: for each $v \in N_G(s)$ do
2: if $|\xi G(s, v)| < 2$ then
3: $N_{G2}(s) = N_{G2}(s) \cup v$
4: end if
5: end for

3.3 G3 Construction

Assume the algorithm (Algorithm 4) is running on node s. Initially it is assumed that the all links in G are also in G3. Then for each neighbor v of s in G this algorithm checks whether number of nodes in the cover set is greater than or equal to 2 or not. If the number of nodes in the cover set $\xi G(s, v)$ is greater than or equal to 2 then the vertex v is removed from NG3 (s) as it indicates that there are two or more lower energy paths than the direct path between s and v. If the cover set consists of only one node z, then the algorithm checks the presence of any path of length 3 built with neighbors of s excluding z. This path of length 3 is denoted by P-Z in the algorithm where -z indicates that z is excluded and P-Z is Boolean. If P-Z is true then there exists such path of length 3 otherwise not. So, if P-Z is true then v is removed from N G3 (s). If the cover set is empty, v is removed from N G3 (s) if there exists at least two vertex-disjoint paths of length 3. These two vertex-disjoint paths must be built with neighbors of s.

Algorithm 4 G3 Topology Construction:

1: $N_{G3}(s) = N_G(s)$
2: for each $v \in N_G(s)$ do
3: if $|\xi G(s, v)| \geq 2$ then
4: $N_{G3}(s) = N_{G3}(s) - \{v\}$
5: else if $\xi G(s, v)$ is $\{z\}$ and P-z is true then
6: $N_{G3}(s) = N_{G3}(s) - \{v\}$
7: else if $\xi G(s, v)$ is empty and at least two vertex-disjoint paths of length 3 consisting of the neighbors of s exist then
8: $N_{G3}(s) = N_{G3}(s) - \{v\}$
9: end if
10: end for

Chapter 4

Simulation

To evaluate the performance of our algorithm we have created some samples of ad-hoc network where nodes are deployed under uniform distribution. Initially we deployed 50 - 100 nodes over a flat square area. We have used a flat area of 650m × 650m , 1000m × 1000m and 1200 × 1200 for 50 , 75 and 100 nodes respectively. Some samples for each 50 , 75 and 100 nodes are taken as input to build desired topologies. Here the following figures are shown only one sample's desired topologies for each 50 , 75 and 100 nodes.

4.1 Simulation Outputs

Figure 4.1(a) shows a typical deployment of 50 nodes, each having a maximum communication range of 250m . This is the starting graph G for our algorithm. The later one in Figure 4.1(b) is our reference graph GREF which is created for comparison purpose with G1 , G2 and G3. GREF is constructed in the following way.

First, we find the minimum-energy path between a pair of vertices such as u , v by Dijkstra's algorithm.

Then we find the second minimum-energy path between u , v by Dijkstra's algorithm which is exclusive of the vertices of the former one. This process continues for all pairs of the vertices.

Finally we remove all the edges not included in the paths chosen by this process.The graph found by G1 topology construction algorithm from graph G is in Figure 4.1(c) which reduces a great number of edges than original graph G. The next graph at Figure 4.1(d) is found by G2 topology construction algorithm from graph G , this also reduces a large number of edges with respect to original graph G . But the amount of reduction in G1 is greater than G2. The next graph at Figure 4.1(e) is found by G3 topology construction algorithm from graph G. Again G3 topology also reduces the number of edges than original graph G , here also, the amount of reduction in G1 is greater than

G3 but from graphical view we can't ahead in comparison between G2 and G3. We will clear about that in later from comparison table. Similarly, we have run simulation for 75 nodes. Figure 4.2(a) shows the original topology G for 75 nodes. GREF built from G is shown in Figure 4.2(b). It keeps 100% backup path with minimum-energy path of G. The topology built by G1 topology construction algorithm is shown in Figure 4.2(c). Figure 4.2(d) and Figure 4.2(e) show the topologies built by G2 topology construction algorithm and G3 topology construction algorithm respectively. From Figure 4.2 we can easily say G1 topology reduces a great number of edges than original graph G. G2 topology also reduces the number of edges than original graph G but the amount of reduction in G1 is greater than G2. Again G3 topology also reduces the number of edges than original graph G, here also, the amount of reduction in G1 is greater than G3 but from graphical view we can't comparison between G2 and G3. It will be clear in later from comparison table.

For 100 nodes the topologies are shown in Figure 4.3. Figure 4.3(a) shows the original Topology G for 75 nodes. GREF built from G is shown in Figure 4.3(b). It keeps 100% backup path with minimum-energy path of G. The topology built by G1 topology construction algorithm is shown in Figure 4.3(c). Figure 4.3(d) and Figure 4.3(e) show the topologies built by G2 topology construction algorithm and G3 topology construction algorithm respectively. From Figure 4.3 we can easily say G1 topology reduces a great number of edges than original graph G . G2 topology also reduces the number of edges than original graph G but the amount of reduction in G1 is greater than G2. Again G3 topology also reduces the number of edges than original graph G, here also, the amount of reduction in G1 is greater than G3 but comparison between G2 and G3 will be clear in later from comparison table.

4.2 Comparison

We have seen graphical representation of various topologies in previous section. Now we compare the topologies with respect to average number of edges and fault tolerance.

4.2.1 Comparison among different topologies with respect to average number of edges

The result is presented in Table 4.1, 4.2, and 4.3. From Table 4.1 shows the average number of edges in different topologies of 50, 75, and 100 no des. The values shown in the table are taken as the nearest integer. From this table we see that the edge reduction in G1 is the most. And G3 less reduction than G1 and G2 has less reduction than G3. This holds for each of 50, 75, and 100 nodes. Also G1 has more reduction than our assumed GREF. This also holds for each each of 50, 75, and 100 nodes. But unfortunately it may not ensure bi-connectivity. On the

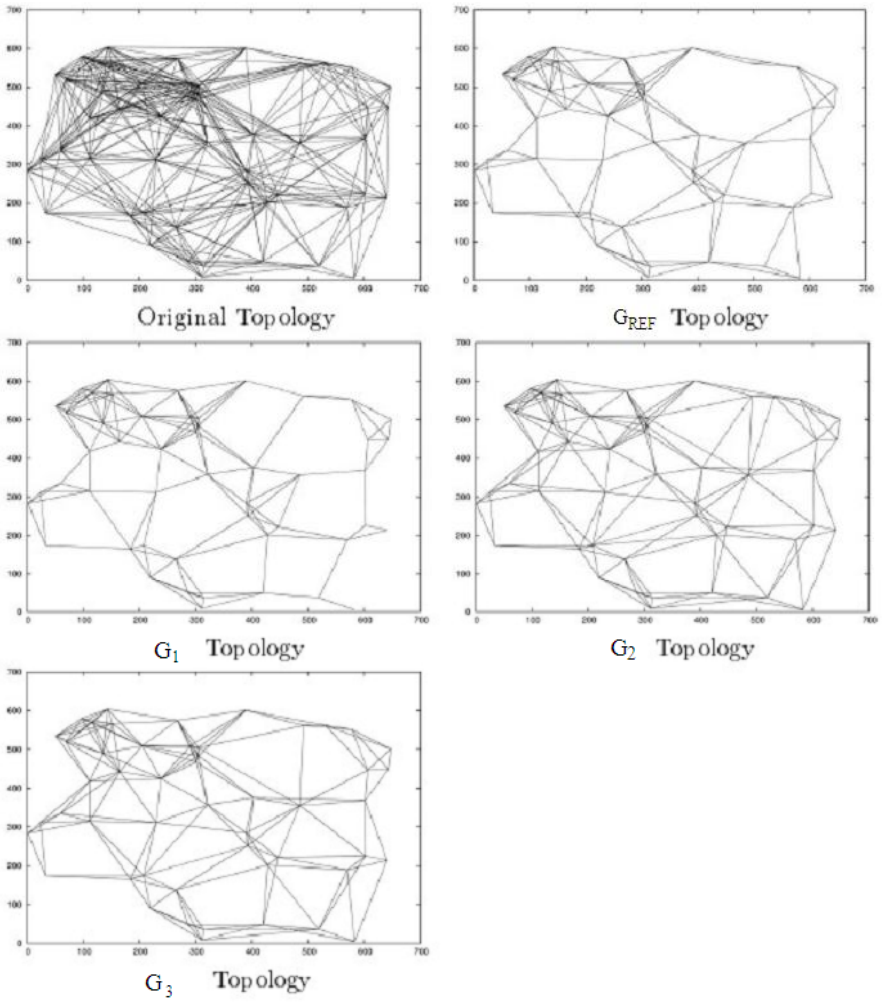
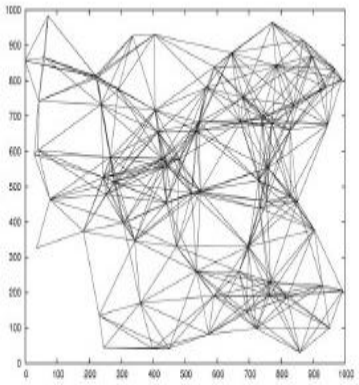
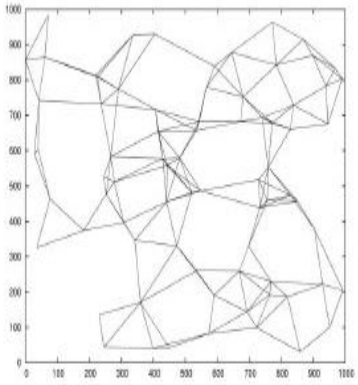


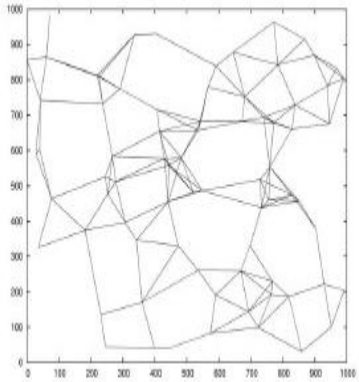
Figure 4.1: Topologies for 50 nodes



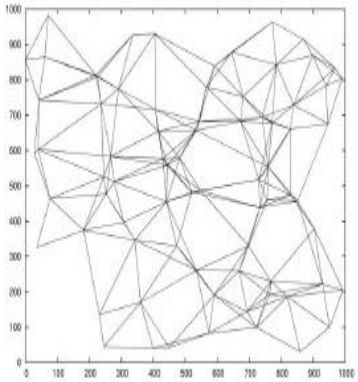
a. Original Topology



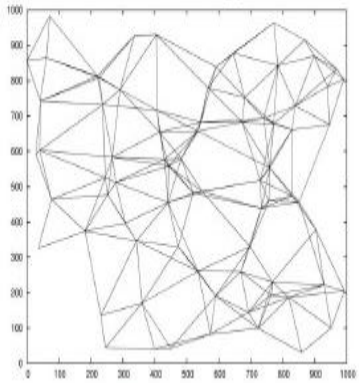
b. G_{REF} Topology



c. G_1 Topology

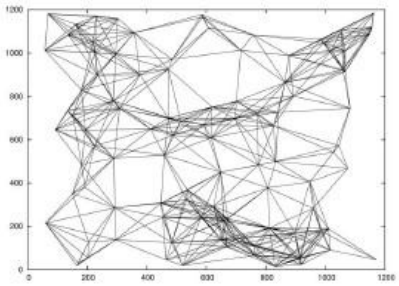


d. G_2 Topology

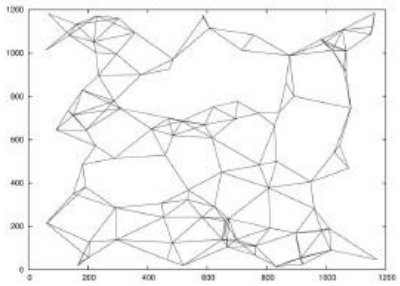


e. G_3 Topology

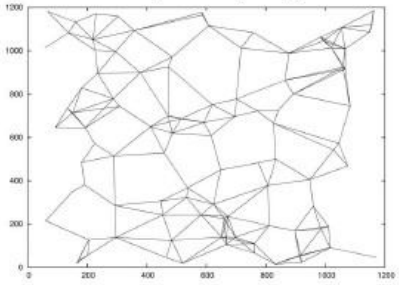
Figure 4.2: Topologies for 75 nodes



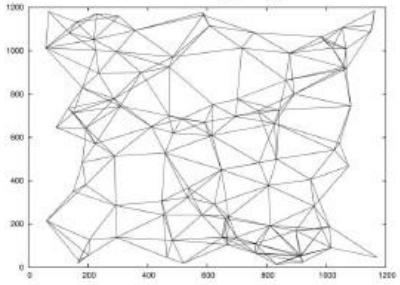
a. Original Topology



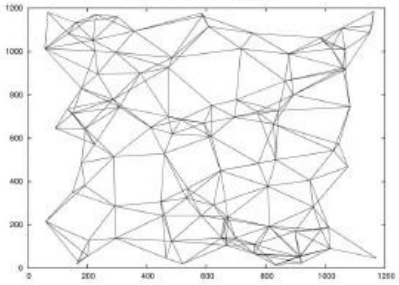
b. G_{REF} Topology



c. G_1 Topology



d. G_2 Topology



e. G_3 Topology

Figure 4.3: Topologies for 100 nodes

other sides GREF , G3 , and G2 all ensure bi-connectivity. For each each of 50, 75, and 100 nodes GREF has more reduction than both G3, and G2 . For example, for 50 nodes original topology has 391 edges, G1 topology has 140 edges on average indicates great reduction of edges in G1. GREF topology has 146 edges on average which is greater than the number of G1. Similarly, G3 topology has 184 edges on average which is greater than the number of GREF and G2 topology has 187 edges on average which is greater than the number of G3. So with respect to average number of edges G3 is more useful to use considering bi-connectivity. Table 4.2 is a result from Table 4.1 after having some calculations. Its First column consists of negative numbers that indicate G1 topology has more reduction than our assumed reference topology GREF again. And G2 , G3 topologies has more edges in percentage than GREF. For example, for 50 nodes G1 topology has 4.23% less edges than GREF topology. But G2 and G2 topologies have 27.73%, 25.68% more edges than GREF topology respectively.

4.2.2 Comparison among different topologies with respect to faultTolerance

We now compare the issue of fault tolerance among various topology construction algorithms. Fault tolerance means maintaining network connectivity in case of any breakdown of the network. In case of bi-connected graphs, there are two disjoint paths between any two nodes. When one path gets destroyed, another path can still sustain the connectivity. So if a topology is bi-connected then it can ascertain fault tolerance. We have shown that G1topology may not ensure bi-connectivity. So, it will not always ensure fault tolerance. But G2 and G3 are both bi-connected. So, they ensure fault tolerance. In Table 4.3 we show what percentage of node pairs have a vertex-disjoint backup path besides the minimum-energy path. Among 50 nodes, G1 has 93.75% no despairs with backup paths, G2 has 99.30% and G3 has 98.30% no de pairs with backup paths

Table 4.1: Average number of edges in different topologies

Number of nodes	Original	G_{REF}	G_1	G_2	G_3
50	391	146	140	187	184
75	435	184	173	238	237
100	556	245	234	315	314

Table 4.2: Percentage of edges more than the

Number of nodes	G_1	G_2	G_3
50	-4.23	27.73	25.68
75	-6.09	29.57	28.59
100	-4.59	28.47	27.96

Table 4.3: Percentage of node pairs having vertex disjoint backup paths

Number of nodes	Original	G_{REF}	G_1	G_2	G_3
50	100.0	100.0	93.75	99.30	98.30
75	94.68	94.68	87.75	93.92	93.92
100	100.0	100.0	92.79	97.99	97.98

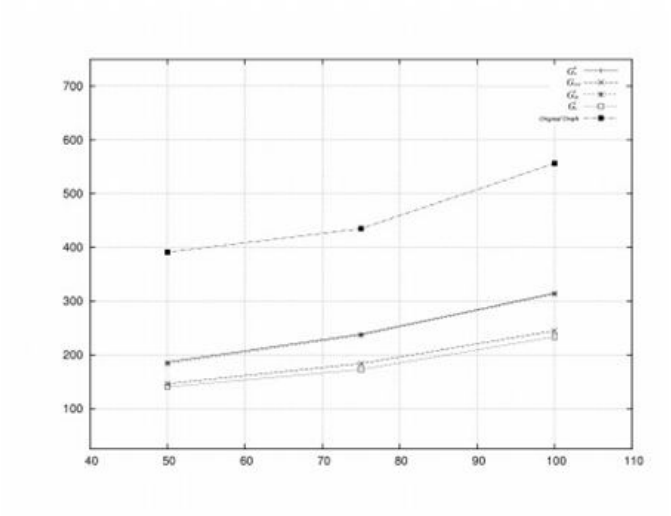


Figure 4.4: Comparison among different topologies with respect to average number of edges

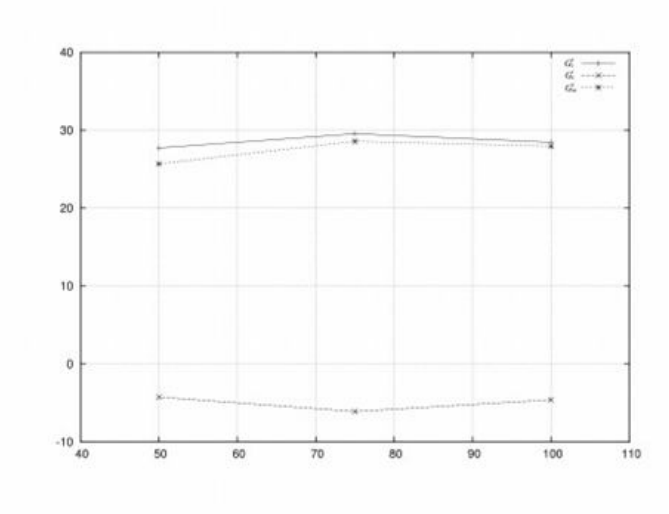


Figure 4.5: Percentage of edges more than the G_{REF}

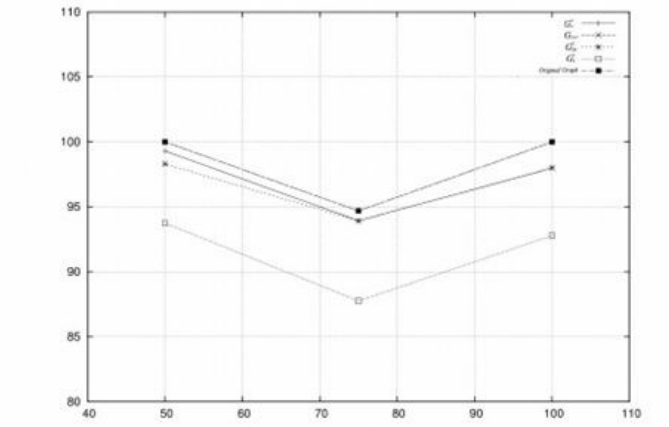


Figure 4.6: Percentage of node pairs having vertex disjoint backup paths

Chapter 5

Conclusion

The algorithm we have presented to build minimum-energy path-preserving bi-connected graph is a distributed algorithm that maintains minimum-energy paths as well as provides fault tolerance in ad-hoc wireless networks. We have proposed G1, G2 and G3 topology construction algorithms of which G1 seems less robust but G2 and G3 have higher fault tolerance and robustness. Simulation results show that G3 has less number of edges than G2 on the average. However G2 preserves vertex-disjoint backup paths in addition to minimum-energy paths among more node pairs.

Chapter 6

Future Work

We discussed two algorithms (G2 and G3) which preserve minimum-energy path between any two nodes and ensure bi-connectivity for whole network. The future work will involve the following:

1. We preserved bi-connectivity between any two nodes that is, we ensured two vertex disjoint paths between any two nodes but it is not ensured that one is minimum energy path and other is vertex-disjoint with that. More attention is to be given on keeping one is minimum-energy path and other is vertex-disjoint with that.

2. Assuming three or more hops information is available for each node, fault-tolerant minimum-energy path preserving algorithms are to be exercised.

Bibliography

[1] M. Bahramgiri, M. Ha jiaghayi, and V. S. Mirrokni. "Fault-tolerant and 3-dimensional distributed topology control algorithms in wireless multi-hop networks." In Proc. Eleventh International Conference on Computer Communications and Networks(ICCCN), pages 392 - 397, Oct. 2002.

[2] M. Ha jiaghayi, N. Immorlica, and V. S. Mirrokni. "Power optimization in fault-tolerant topology control algorithms for wireless multi-hop networks." IEEE/ACM Trans. Netw.,15(6):1345 - 1358, 2007.

[3] A. Kazmierczak and S. Radhakrishnan. "An optimal distributed ear decomposition algorithm with applications of biconnectivity and outerplanarity testing." IEEE Trans. Parallel Distrib. Syst., 11(2):110 - 118, 2000.

[4] L. Li and J. HalpernRahman. "Minimum energy mobile wireless networks revisited." In Proc. IEEE International Conference on Communications (ICC), june 2001.

[5] L. Li, J. Y. Hlapern, P. Bahl, Y. Wang, and Watenhofer R. "Analysis of a con-based topology control algorithm for wireless multi-hop networks." In ACM Symposium on Principle of Distributed Computing (PODC), 2001.

[6] Ning Li and Jennifer C. Hou. "FlSS: A fault tolerant topology control algorithm for wireless networks." In Pro c. of the 10th annual international conference on Mobile computing and networking, pages 275 - 286, 2004.

[7] A. Rahman and P. Gburzynski. "MAC-assisted topology control for ad-hoc wireless network." International Journal of communication Systems, 19(9):995 - 976, 2006.

[8] R. Ramanathan and R. Rosales-Hain. "Topology control of multihop wireless networks using transmit power adjustment." In Pro c. of IEEE INFOCOM 2000, pages 404 - 413, Tel Aviv, Israel, March 2000.

[9] V. Rodoplu and T. Meng. "Minimum energy mobile wireless networks." IEEE Journal of Selected Areas in Communications, 17:1333 - 1344, 1999.

[10] Z. Shen, Y. Chang, C. Cui, and X. Zhang. "A fault-tolerant and minimum-energy path-preserving topology control algorithm for wireless multi-hop networks." In Proc. of the International Conference on Computational Intelligence and Security- I (CIS), pages 864 - 869, 2005.

Source code:

```
package thesis;

import java.io.ByteArrayInputStream;
import java.io.DataInputStream;
import java.util.Calendar;
import java.util.Date;
import java.util.Vector;
import javax.microedition.lcdui.Font;
import org.json.me.JSONArray;
import org.json.me.JSONObject;

/**
 *This class contains the utility functions of String
 * @author Administrator
 */
public class Thesis
{
    private static Vector vTemp = new Vector(0, 1);
    private static String[] sTemp;
    private static char[] chars =
    {
        '|'
    };

    public static String replace(String _text, String _searchStr, String _replacementStr)
    {
        // String buffer to store str
        StringBuffer sb = new StringBuffer();

        // Search for search
        int searchStringPos = _text.indexOf(_searchStr);
        int startPos = 0;
        int searchStringLength = _searchStr.length();

        // Iterate to add string
        while(searchStringPos != -1)
        {
            sb.append(_text.substring(startPos, searchStringPos)).append(_replacementStr);
            startPos = searchStringPos + searchStringLength;
            searchStringPos = _text.indexOf(_searchStr, startPos);
        }

        // Create string
        sb.append(_text.substring(startPos, _text.length()));

        return sb.toString();
    }

    public static Vector mobileMapsResponseParser(JSONObject response)
    {
        Vector searchResults = null;
        if(response == null)
        {
```

```
            return null;
        }
        try
        {
            JSONArray records = response.getJSONObject("Search").getJSONArray("Record");
            if (records.length() == 0)
                return null;
            searchResults = new Vector();
            for(int i = 0; i < records.length(); i++)
            {
                LuceneResponse luceneResponse = new LuceneResponse();
                JSONObject record = (JSONObject)records.get(i);
                String address = record.getString("Address");
                String longitude = record.getString("Longitude");
                String latitude = record.getString("Latitude");

                int index = address.indexOf(",");

                luceneResponse.setAddress1(address.substring(0, index));
                luceneResponse.setAddress2(address.substring(index + 1));
                luceneResponse.setLon(Double.parseDouble(longitude));
                luceneResponse.setLat(Double.parseDouble(latitude));
                searchResults.addElement(luceneResponse);
            }

        }
        catch(Exception exp)
        {
            exp.printStackTrace();
            return null;
        }
        return searchResults;
    }

    public static String getFormattedDate()
    {
        Calendar cal = Calendar.getInstance();
        int day = cal.get(Calendar.DATE);
        int month = cal.get(Calendar.MONTH) + 1;
        int year = cal.get(Calendar.YEAR);
        return year + "-" + month + "-" + day;
    }

    public static String convertToDisplayMSISDN(String msisdn)
    {

        if(msisdn.startsWith("+27"))
        {
            return "0" + msisdn.substring(3);
        }

        if(msisdn.startsWith("27"))
        {
            return "0" + msisdn.substring(2);
        }
```

```
        return msisdn;
    }


    /**
     * this function splits a string according to the split string, this function is used
     * here to mainly split responses with | as separator
     * @param sStr the string to be splitted
     * @param match the string to match with
     * @return an array of string containing the splitted strings
     */
    public static String[] split(String sStr, String match)
    {

        int ind;
        int indStart = 0;
        vTemp.removeAllElements();
        try
        {
            while((ind = sStr.indexOf(match, indStart)) != -1)
            {
                vTemp.addElement(sStr.substring(indStart, ind));
                indStart = ind + match.length();
            }
            vTemp.addElement(sStr.substring(indStart));
        }
        catch(Exception e)
        {
            //#if EnableDebug=="true"
//#             e.printStackTrace();
            //#endif
        }
        sTemp = new String[vTemp.size()];
        vTemp.copyInto(sTemp);
        return sTemp;
    }

/****************************** G{2}{1} Topology Construction Algorithm ******************/

    public static void G21Topology(String string, int width, Font font, Vector retVec)
    {
        retVec.removeAllElements();


        if(string == null)
        {
            return;
        }

        if(string.equals(""))
        {
            retVec.addElement("");
            return;
        }

        String[] retString = split(string, " ");
```

```
int j = 0;
for(int i = 0; i < retString.length; ++i)
{
   if(!retString[i].equals(""))
   {
      retString[j++] = retString[i];
   }
}

int len = j;
int spaceWidth = font.stringWidth(" ");
if(len > 0)
{
   int x = 0;
   String thisLineString = "";
   for(int i = 0; i < len; ++i)
   {
      int thisWidth = font.stringWidth(retString[i]);
      int thisWidthWSpace = thisWidth + spaceWidth;

      if(x == 0)
      {
         if(thisWidth <= width)
         {
            thisLineString = retString[i];
            x += thisWidth;
         }
         else if(thisWidth > width)
         {
            int si = 0, ci;
            while(si < retString[i].length())
            {
               String curtStr = "";
               for(ci = si; ci < retString[i].length(); ++ci)
               {
                  curtStr += retString[i].charAt(ci);
                  int twidth = font.stringWidth(curtStr);
                  if(twidth >= width)
                  {
                     break;
                  }
               }

               thisLineString += retString[i].substring(si, ci);

               if(ci == retString[i].length())
               {
                  x = font.stringWidth(thisLineString);
               }
               else
               {
                  retVec.addElement(thisLineString);
                  thisLineString = "";
                  x = 0;
               }
```

```
                si = ci;
            }
        }
    }
    else if(x > 0)
    {
        if(x + thisWidthWSpace <= width)
        {
            thisLineString += " " + retString[i];
            x += thisWidthWSpace;
        }
        else if(x + thisWidthWSpace > width)
        {
            retVec.addElement(thisLineString);
            thisLineString = "";
            x = 0;

            int si = 0, ci;
            while(si < retString[i].length())
            {
                String curtStr = "";
                for(ci = si; ci < retString[i].length(); ++ci)
                {
                    curtStr += retString[i].charAt(ci);
                    int twidth = font.stringWidth(curtStr);
                    if(twidth >= width)
                    {
                        break;
                    }
                }

                thisLineString += retString[i].substring(si, ci);

                if(ci == retString[i].length())
                {
                    x = font.stringWidth(thisLineString);
                }
                else
                {
                    retVec.addElement(thisLineString);
                    thisLineString = "";
                    x = 0;
                }
                si = ci;
            }
        }
    }
  }

  if(!thisLineString.equals(""))
  {
    retVec.addElement(thisLineString);
  }
}
else
{
```

```
            retVec.addElement("");
        }
    }

/****************************** G{2}{2} Topology Construction Algorithm *****************/

    public static Vector G22Topology(String string, int width, int height, Font font, Vector retVec)
    {
        retVec.removeAllElements();
        int noOfLines = height / font.getHeight() + 1;

        if(noOfLines <= 0)
        {
            return retVec;
        }

        if(string == null)
        {
            return retVec;
        }

        if(string.equals(""))
        {
            retVec.addElement("");
            return retVec;
        }

        String[] retString = split(string, " ");

        int j = 0;
        for(int i = 0; i < retString.length; ++i)
        {
            if(!retString[i].equals(""))
            {
                retString[j++] = retString[i];
            }
        }

        int len = j;
        int spaceWidth = font.stringWidth(" ");
        if(len > 0)
        {
            int x = 0;
            String thisLineString = "";
            for(int i = 0; i < len; ++i)
            {
                int thisWidth = font.stringWidth(retString[i]);
                int thisWidthWSpace = thisWidth + spaceWidth;

                if(x == 0)
                {
                    if(thisWidth <= width)
                    {
                        thisLineString = retString[i];
                        x += thisWidth;
                    }
```

```
    else if(thisWidth > width)
    {
        int si = 0, ci;
        while(si < retString[i].length())
        {
            String curtStr = "";
            for(ci = si; ci < retString[i].length(); ++ci)
            {
                curtStr += retString[i].charAt(ci);
                int twidth = font.stringWidth(curtStr);
                if(twidth >= width)
                {
                    break;
                }
            }

            thisLineString += retString[i].substring(si, ci);

            if(ci == retString[i].length())
            {
                x = font.stringWidth(thisLineString);
            }
            else
            {
                retVec.addElement(thisLineString);
                if(retVec.size() >= noOfLines)
                {
                    return retVec;
                }
                thisLineString = "";
                x = 0;
            }
            si = ci;
        }
    }
}
else if(x > 0)
{
    if(x + thisWidthWSpace <= width)
    {
        thisLineString += " " + retString[i];
        x += thisWidthWSpace;
    }
    else if(x + thisWidthWSpace > width)
    {
        retVec.addElement(thisLineString);
        if(retVec.size() >= noOfLines)
        {
            return retVec;
        }
        thisLineString = "";
        x = 0;

        int si = 0, ci;
        while(si < retString[i].length())
        {
```

```
                String curtStr = "";
                for(ci = si; ci < retString[i].length(); ++ci)
                {
                   curtStr += retString[i].charAt(ci);
                   int twidth = font.stringWidth(curtStr);
                   if(twidth >= width)
                   {
                      break;
                   }
                }

                thisLineString += retString[i].substring(si, ci);

                if(ci == retString[i].length())
                {
                   x = font.stringWidth(thisLineString);
                }
                else
                {
                   retVec.addElement(thisLineString);

                   if(retVec.size() >= noOfLines)
                   {
                      return retVec;
                   }

                   thisLineString = "";
                   x = 0;
                }
                si = ci;
             }
          }
       }

       if(!thisLineString.equals(""))
       {
          retVec.addElement(thisLineString);
       }
    }
    else
    {
       retVec.addElement("");
    }

    return retVec;
 }

/****************************** G{2|3}{2} Topology Construction Algorithm ******************/

 public static Vector G223Topology(String string, int width, Font font)
 {
    Vector retVec = new Vector(1);

    if(string == null)
    {
```

```
    return retVec;
}

if(string.equals(""))
{
    retVec.addElement("");
    return retVec;
}

String[] retString = split(string, " ");

int j = 0;
for(int i = 0; i < retString.length; ++i)
{
    if(!retString[i].equals(""))
    {
        retString[j++] = retString[i];
    }
}

int len = j;
int spaceWidth = font.stringWidth(" ");
if(len > 0)
{
    int x = 0;
    String thisLineString = "";
    for(int i = 0; i < len; ++i)
    {
        int thisWidth = font.stringWidth(retString[i]);
        int thisWidthWSpace = thisWidth + spaceWidth;

        if(x == 0)
        {
            if(thisWidth <= width)
            {
                thisLineString = retString[i];
                x += thisWidth;
            }
            else if(thisWidth > width)
            {
                int si = 0, ci;
                while(si < retString[i].length())
                {
                    String curtStr = "";
                    for(ci = si; ci < retString[i].length(); ++ci)
                    {
                        curtStr += retString[i].charAt(ci);
                        int twidth = font.stringWidth(curtStr);
                        if(twidth >= width)
                        {
                            break;
                        }
                    }

                    thisLineString += retString[i].substring(si, ci);
```

```
            if(ci == retString[i].length())
            {
               x = font.stringWidth(thisLineString);
            }
            else
            {
               retVec.addElement(thisLineString);
               thisLineString = "";
               x = 0;
            }
            si = ci;
         }
      }
   }
   else if(x > 0)
   {
      if(x + thisWidthWSpace <= width)
      {
         thisLineString += " " + retString[i];
         x += thisWidthWSpace;
      }
      else if(x + thisWidthWSpace > width)
      {
         retVec.addElement(thisLineString);
         thisLineString = "";
         x = 0;

         int si = 0, ci;
         while(si < retString[i].length())
         {
            String curtStr = "";
            for(ci = si; ci < retString[i].length(); ++ci)
            {
               curtStr += retString[i].charAt(ci);
               int twidth = font.stringWidth(curtStr);
               if(twidth >= width)
               {
                  break;
               }
            }

            thisLineString += retString[i].substring(si, ci);

            if(ci == retString[i].length())
            {
               x = font.stringWidth(thisLineString);
            }
            else
            {
               retVec.addElement(thisLineString);
               thisLineString = "";
               x = 0;
            }
            si = ci;
         }
      }
```

```
            }
        }

        if(!thisLineString.equals(""))
        {
            retVec.addElement(thisLineString);
        }
    }
    else
    {
        retVec.addElement("");
    }

    return retVec;
}

/**********************************************************************************************
/

public static String singleLine(String string, int width, Font font)
{
    if(font.stringWidth(string) <= width)
    {
        return string;
    }
    else
    {
        StringBuffer stringBuffer = new StringBuffer();
        int temp = font.stringWidth("...");

        for(int i = 0; i < string.length(); i++)
        {
            stringBuffer.append(string.charAt(i));
            if(font.stringWidth(stringBuffer.toString()) > width - temp)
            {
                stringBuffer.deleteCharAt(stringBuffer.length() - 1);
                stringBuffer.append("...");
                break;
            }
        }
        return stringBuffer.toString();
    }
}

public static Vector wrapString(String string, int width, Font font, int n, Vector retVec)
{
    retVec.removeAllElements();
    int index = 0;

    /*
    String curString = string.substring(index, index + 30);
    int stringWidth = font.stringWidth(curString);
    int count = 1;
    if (stringWidth > width) {
    while (stringWidth > width) {
    stringWidth -= font.charWidth(curString.charAt(index + 30 - count));
```

```
        count++;
        }
        count--;



        } else if (stringWidth < width) {
        while (stringWidth < width) {
        stringWidth += font.charWidth(curString.charAt(index + 30 + count));
        count++;
        }
        }
         */

        //for (int k = 0; k < string.length(); k++) {
        //}

        String[] retString = split(string, " ");

        int j = 0;
        for(int i = 0; i < retString.length; ++i)
        {
           if(!retString[i].equals(""))
           {
              retString[j++] = retString[i];
           }
        }

        int charWidth = font.charWidth(' ');
        int spaceWidth = font.stringWidth(" ");
        int totalWidth = 0;
        StringBuffer buffer = new StringBuffer();

        for(int k = 0; k < retString.length; k++)
        {
           totalWidth += (spaceWidth + font.stringWidth(retString[k]));
           if(totalWidth <= width)
           {
              buffer.append(" " + retString[k]);
           }
           else
           {
              retVec.addElement(buffer.toString());
              totalWidth = 0;
              buffer.delete(0, buffer.length());
              k--;
           }
        }
        return retVec;

     }
}
```