Topology control and Interference aware Resource allocation algorithms using Directional antenna for QoS (802.16e) performance in multi-hop Wireless Networks

By Vishwanath Annavarapu

Chapter 1

Introduction

1.1 Introduction to IEEE 802.16 'WiMAX'

WiMAX, Worldwide Interoperability for Microwave Access is a telecommunications technology that provides wireless transmission of data using a variety of transmission mode, from stationary point-to-multipoint links to portable and fully mobile internet access. The technology is IEEE 802.16 and also called Broadband Wireless Access. The 802.16 family is officially called WirelessMAN in IEEE; it has been commercialized under the name 'WiMAX' by the industry alliance called the WiMAX Forum [1]. The mission of the Forum is to promote and certify compatibility and interoperability of broadband wireless products based on the IEEE 802.16 standards.

IEEE 802.16 - Air Interface for 10 to 66 GHz [2], was approved in December 2001. It delivered a standard for point to multipoint broadband wireless transmission in the 10-66 GHz band, with only a line-of-sight (LOS) capability. It uses a single carrier (SC) physical (PHY) standard.

IEEE 802.16a was an amendment to 802.16 and delivered a point-to-multipoint capability in the 2-11 GHz band [3], which also offers a non-line-of-sight (NLOS) capability and the PHY standard was therefore extended to include Orthogonal Frequency Division Multiplex (OFDM) and Orthogonal Frequency Division Multiple Access (OFDMA).

An amendment to 802.16-2004, IEEE 802.16e-2005, addressing mobility, was completed in 2005. This added a number of enhancements to the previous standard, including better support for Quality of Service (QoS) and the use of Scalable OFDMA, and is sometimes called "Mobile WiMAX" [4].

The 802.16 standard essentially standardizes 2 aspects of the air interface – the physical layer (PHY) and the Media Access Control (MAC) layer.

QoS in 802.16e is supported by allocating each connection between the Subscriber Station (SS) and the Base Station (BS) called a *service flow* in 802.16 terminology) to a specific QoS class. The QoS classes are: UGS, unsolicited grant service; rtPS, real time polling service; nrtPS, non-real time polling service; ertPS, extended time polling service; BE, Best effort.

The WiMAX models and their functionalities can be vendor specific as there is room for customization in 802.16 standards, unlike other popular wireless standards such as 802.11/WLAN (WiFi).

A rough comparison between WiMAX and WiFi is drawn in the figure below (Fig. 1.1), the values are estimated.

	WiFi/802.11 WLAN	**802.16e/WiMAX**
QoS management	Best Effort	Five classes of QoS
Multiple access	CSMA/CA (MAC Layer common to 802.11, 802.11a, 802.11b and 802.11g); TDD	TDMA: TDD and FDD. Sophisticated bandwidth reservation mechanisms
Range	Order of magnitude: 100m	20 km
Frequency bands	Unlicensed	Unlicensed and licensed
Typical use	WLAN	Fixed wireless access, portability, mobility, etc.
Control	Distributed (unless using optional PCF)	Centralized

Fig. 1.1 - Comparison between WiFi and WiMAX.

In WiMAX communication typically a Base Station (BS) is connected to backhaul links or servers and Subscriber Stations (SSs), enabled with WiMAX capability, communicate with the BS for services. The BS controls the communication and service issues.

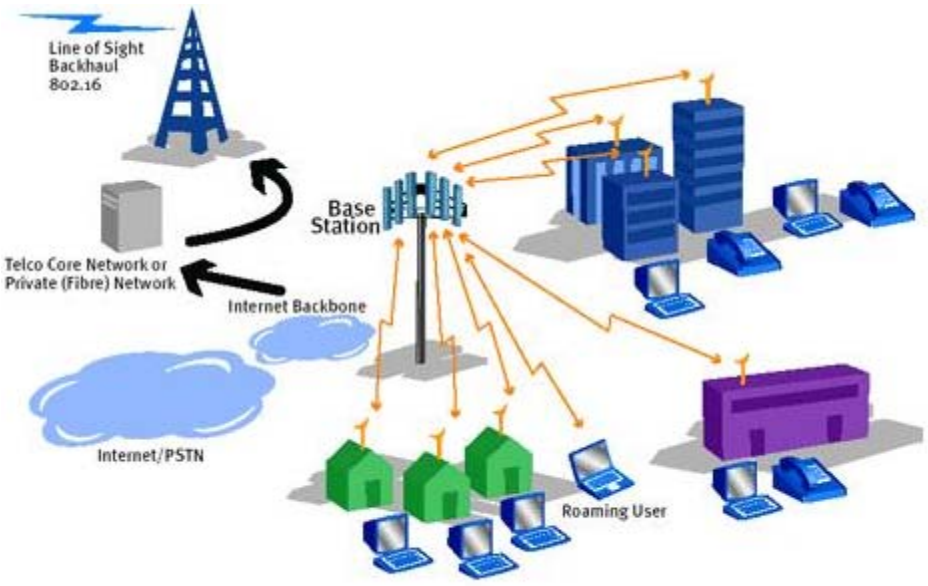
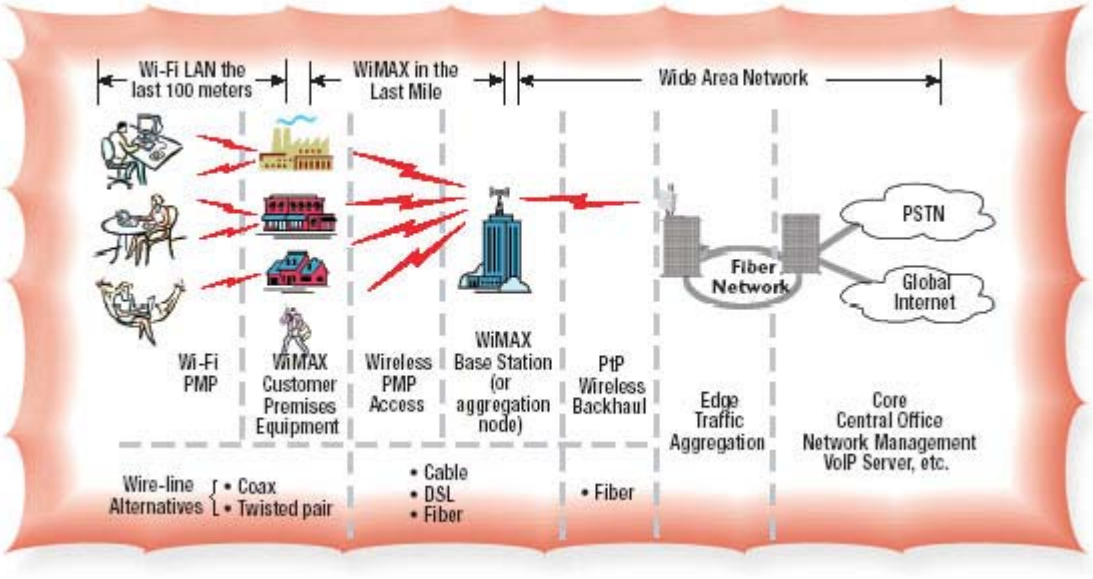


Fig. 1.2 - The figure below shows a typical WiMAX network and its application.

The WiMAX technology fills in as a bridge between short distance communications and long distance communications and works with other technologies and protocols to extend the 'last mile' support in data communication.



.

Fig. 1.3 - The figure shows WiMAX with other wireless technologies.

1.2 About the research work and its goals

In a WiMAX network, wireless communication happens between a Subscriber Station (SS) and a Base Station (BS). A BS is typically a service provider which has backhaul links or servers connected to it and an SS subscribes to the BS for the service. A BS and an SS exchange control messages and negotiate the connection parameters before setting up the communication link. These parameters can be changed during the communication depending on the requirements and availability of resources between the two entities.

An SS can listen to and communicate with a BS which is operating in same frequency channel as it's. Hence, each SS is assigned a frequency channel on which it can communicate with a BS from a set of potential serving BSs which are operating on the same frequency channel. Generally, all SSs and BSs in a cell (or a certain designated area) operate on same frequency channel and hence an SS associates itself to a BS which is nearest to it and when the SS gets communication links from their BS, data transfer happens. Any subsequent issues which arise during the communication session are solved by following the designated protocols of the layers where the issues belong. These issues include traffic congestion, interference, signal quality, Quality of Service, and mobility issues like handover.

The approach of the research is towards improving the WiMAX communication (from 802.16e standard) by developing new functionalities, designs and using methodologies to solve most of these vital issues or to avoid the occurrence of the above issues and suggesting novel, more useful and efficient system models and exploiting the scope for user specific implementations.

The overall goal of this research is to work towards improving the wireless communication and solving some existing issues in a WiMAX network. In the course of research, various issues have been addressed by providing feasible to optimal solutions; new designs and methodologies of the WiMAX nodes are incorporated to produce useful functionalities; communication models, antennas and other devices are technically enhanced. And using these ideas and products WiMAX communication is brought to an advanced level, where multi-hop scenarios were successfully simulated and studied. The issues of a multi-hop WiMAX scenario that would possibly occur in the future were identified and solutions are suggested and tested. Large numbers of WiMAX scenarios of different sizes with different type of applications, models, devices, resources and methodologies have been simulated and various aspects like mobility, QoS, interference control, traffic congestion and network balance have been studied intensively.

OPNET Modeler [5], generally, version 14.5 is used for simulating all the WiMAX scenarios and all the models used in this research are based on features available or added to OPNET Modeler.

The performance of a WiMAX communication is enhanced by applying new methods for resource scheduling, where the throughput and link capacity of the network is improved.

For improving the network throughput and link capacity few Topology Control algorithms are suggested and implemented which provides a productive set of communication links between the nodes.

For reducing the interference among the communication links channel assignment algorithms are also proposed.

For reducing traffic congestion and maintaining network balance a simple joint topology control and load balancing algorithm is suggested and implemented.

For improving the signal quality, reducing interference (mainlobe and sidelobe) and achieving high gain and throughput, directional antennas are incorporated in the node models, and successfully tested.

The feature of mobility is extended in multi-hop scenarios, where handover could be implemented at every hop of the communication using a 'Relay' Station (RS). The RS a customized communication model designed and implemented to work as a WiMAX 'relay' [6]. The 802.16j relay task group [7] has been working on the idea and functionality of multihop relay. But the implementation of a multihop relay model is not reported so far.

A new approach to handover is implemented, which is Network Initiated Handover or 'Forced Handover' [8] to centralize the control over handover. This implementation is a customized feature for a mobile network and the 802.16e standard doesn't by default provide this functionality. This functionality gives the network the control over communication nodes and their links for improving and maintaining overall network balance and efficiency.

The Quality of Service in a multi-hop WiMAX network is simulated and studied for various network sizes and should help significantly in planning and modeling applications and scenarios with QoS requirement.

Along with the development of new models, designs, approaches and methodologies which improved the network efficiency in several vital aspects and introduced several communication advancements, the intensive experiments and study provide a great help in understanding the behavior and limitations of existing WiMAX systems and in planning and designing future WiMAX systems.

1.3 Results of the research work

- A new WiMAX relay node model for providing multi-hop communication is successfully designed, developed and implemented in OPNET Modeler.
- Directional antennas were successfully designed and implemented and helped in reducing interference.

- Algorithms for providing a resource allocation schedule and antenna pattern selection were tested and showed significant improvement in throughput and reduction in interference.
- The network performance is enhanced by improving throughput and other vital network parameters using these algorithms and methods.
- Mobility support is implemented in Relay Stations in a multi-hop WiMAX network.
- The simulations of QoS based traffic scenarios brought out the behavior and priority details in the WiMAX multi-hop network.
- 'Forced Handover' is successfully implemented and various criteria and tested for providing network the control over handover.

1.4 Contribution made through this work

- Worked on a developing and implementing a successful customized 'off the shelf product', WiMAX 'relay' node in OPNET Modeler. Deployed this model in various WiMAX network scenarios and constantly worked on enhancing and complementing numerous versions.
- Successfully implemented Directional antennas at PHY layer of OPNET Modeler. Developed dynamic and adaptive features for the directional antennas.
- Proposed and worked on designing various algorithms addressing problems of resource allocation schedule, obtaining a network topology (with different objective functions), balancing the network load and reducing the interference traffic in a WiMAX network.
- Implemented and tested these algorithms using software tools like OPNET Modeler, VC++ and thoroughly studied the simulation results.
- The proposal and implementation of our algorithms using the customized models provide the solutions for end-user application without the need of any new network model or product to use or implement the required features.
- Implemented mobility support in Relay Stations in a multi-hop WiMAX network.
- Explored and studied the behavior of QoS service flows in a WiMAX multi-hop network and successfully performed simulation to study their priority and efficiency.
- Successfully implemented 'Forced Handover' in OPNET Modeler and customized various such functionalities to specific various criteria and conditions.

1.5 Organization of the research work into chapters

In subsequent chapters, the inspiration and importance of the thesis work is discussed after which the research work and its contribution is detailed. The next chapter includes the idea and details of the work related to the some of my thesis work. In chapter 3, the technical enhancement of WiMAX models and design and implementation of 'relay' node is discussed. This WiMAX

‘relay’ node model is a customized WiMAX node model which can be used in a multi-hop network and is designed specific to the requirements and objectives of the organization, by exploiting the vendor specific implementations. This model provides an off the shelf product for using in a multi-hop network scenario, satisfying the role of Relay Station (802.16j), which is still under the development process. The creation and implementation of directional antennas and special pencil beams is detailed. This chapter also has other physical layer related implementations.

In chapter 4, there are two proposed topology control algorithms for establishing a SS-RS assignment with the objective of maximizing the minimum link capacity of the network, which are explained and implemented. These two algorithms are used in simulating several WiMAX scenarios and their efficiencies are studied and explained with the help of results.

In chapter 5, quality of service issues in a multi-hop scenario are tested and studied. The algorithms from Chapter 4 are used in these scenarios. The QoS parameters are comprehensively tested and simulated for different traffic conditions and network sizes. The priority and behavior of different service flows like UGS, rtPS, BE is studied using the simulation results.

A new approach is suggested in chapter 6, for overall communication setup in a WiMAX multi-hop scenario, which consists of algorithms for SS-RS assignment directed towards achieving network connectivity and load balance. This approach also has a channel assignment algorithm for reducing co-channel interference [9] and determines appropriate directional antenna patterns and orientation for increasing the gain and hence improving the throughput.

Chapter 7 discusses the idea of network initiated handover and its implementation details in OPNET 14.0. This feature is tested and used in various mobile scenarios. In this chapter, the design and details of a mobile RS are given. The mobility and handover in a WiMAX multi-hop scenario is implemented and studied.

Chapter 8 of this thesis summarizes the contribution and scope of the thesis and discusses the future work this thesis can possibly inspire and help in.

Chapter 9 lists the references used to cite the work that is included and inspired in the thesis.

The Appendices section at the end has all the program code used for the work and other related details.

Chapter 2

Related work

In recent years, there has been a lot of research work done on wireless networking using directional antennas [10], [11], [12]. Yap et al [10] proposed the use of a simple directional antenna, which has a fixed number of beams and fixed respective beamwidths, with the requirement that the direction of the beams is changeable during runtime. Huang et al. [11] studied toplology control by reducing the power intensity directionally instead of omni-directionally. Kumar et al. [12] proposed to use multiple directional antennas on each node and orient them appropriately to create low interference topologies while maintaining network connectivity. In this paper it is assumed that the antenna beamwidth is sufficiently small such that a node can communicate with only one neighbor node using one directional antenna.

But none of approaches described above provide a solution for communication in a mobile scenario where the nodes move either in a specified trajectory or randomly. In the research work presented, directional antennas are successfully used in the mobile case and the beamwidth of the antenna can be changed during the runtime. Also, it is a costly and inefficient approach to communicate to each neighbor node with a separate radio and antenna. In this research work optimal antenna beamwidth and orientation is determined to maintain connectivity and maximize the efficiency of the data communication.

Another issue related to this research work is channel assignment in multi-channel multi-radio networks [9], [13]. A typical example of channel assignment is reported by Tang and Xue in [9]. They first defined co-channel interference and presented a channel assignment algorithm that minimizes the co-channel interference while keeping the resulting topology *K*-connected. They further presented flow allocation algorithms for a QoS request in the network topology. Subramanian et al. proposed a centralized and distributed channel assignment algorithm in [13] for minimum interference on Tabu search technique. These articles and other related work on channel assignment use omni-directional antennas.

Despite of vast amount of effort that has been published that exploits directional antennas and multiple radios, there is little work that considers them together. Recently, architecture of multi-radio directional antennas called DMesh was proposed [14]. The authors assume that one radio of a node can communicate with only one other radio, this limits the communication with multiple nodes where there may be a need to communicate and cover several nodes in practice.

Algorithms for maximizing the minimum link capacity by controlling network topology shall be a very useful and efficient approach. Also, a method to balance the network load and reduce traffic congestion might contribute to research in wireless communication. The combined

approach of communication set by topology control, channel assignment and adaptive use of directional antenna beams and orientation is new in WiMAX network research.

There hasn't been much work reported on Quality of Service in WiMAX networks. There has been some work related such as in [15], where the IEEE 802.16 QoS was simulated but mostly on Best Effort services with limited scope and scenarios. Our research includes simulation and detailed analysis of all the five service classes in varied conditions and scenarios.

To the best of my knowledge implementation of a WiMAX multi-hop scenario with a Relay station has not yet been reported. Similarly, the implementation of a Mobile Relay station and multi-hop handover has not been previously reported. Though the IEEE task group is coming up with a new IEEE 802.16j standard for a Mutlihop relay [7], the project was not completed during the time when our research was being carried out.

The IEEE 802.16j relay task proposes two types of 'relay' functionalities; transparent relay and non-transparent relay [16]. The non-transparent RS works as a BS sector and sends its own preamble and control headers. Also, each non-transparent RS can control and manages its data frame for communication. For a transparent RS it does not have its own preamble, FCH and MAP. It looks transparent to SSs and relays the SSs' data. The transparent RSs have to be centralized controlled by the Base station of the multihop network.

The design and implementation of the WiMAX 'relay' node model developed by our team behaves and works similar to the transparent RS. The approach and technique for developing the WiMAX 'relay' node can be related to the idea of the transparent RS as defined by IEEE 802.16j relay task group.

We believe that the implementation of all the above methods to provide solutions for various wireless network issues in a multi-hop scenario is the first of its kind and leads the way to advanced research in the field of 802.16 (WiMAX) networks.

The implementation of network initiated handover reported here is also believed to be unique in wireless networking. This should be a useful contribution to WiMAX communication for customized applications and future advancement in 802.16 researches.

Chapter 3

Design and implementation of WiMAX 'relay' nodes and directional antennas in OPNET

This chapter details the design and implementation of a multi-interface node model customized in OPNET Modeler for simulating multi-hop WiMAX network scenarios. According to IEEE 802.16e, the latest specification of wireless broadband standard upon which this work was based, the WiMAX communication models, typically Base Stations (BSs) and Subscriber Stations (SSs) could be used only in a single-hop wireless scenario [2]. There was no functional feature available for implementing or simulating a multi-hop scenario for WiMAX communication.

The chapter also discusses directional antennas and their modeling in OPNET. A directional antenna could be used to track a moving SS. Few special features relating to directional antennas were implemented in the OPNET library; the beamwidth of a directional antenna could be changed dynamically and adaptively during the data communication, especially in a mobile scenario.

3.1 WiMAX relay station with multiple interfaces

A SS has to directly associate with a BS for data communication, where the BS would have a backhaul link or server for providing service to its associated SSs. Each SS would be associated to a BS such that it can send and receive data on upstream and downstream channels respectively. The standard WiMAX models could only communicate either as client or server machines and hence there could be only one hop possible.

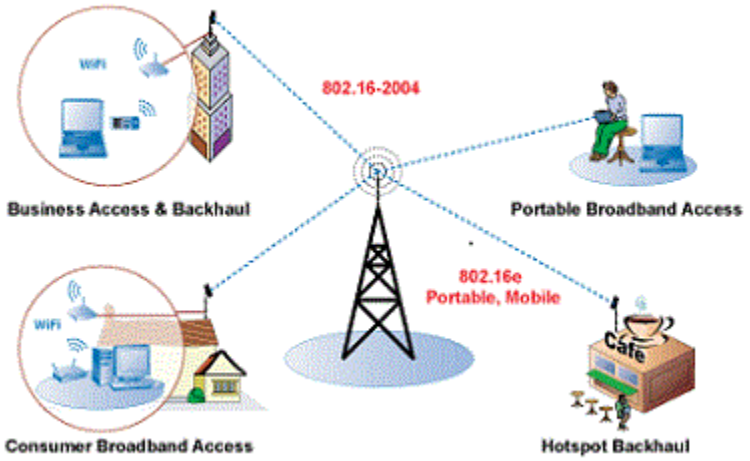


Fig. 3.1 - The figure shows a Single-hop WiMAX network, Base Station and Subscriber Stations.

To study several aspects and functionalities of WiMAX data communication over multi-hops, there has to be an intermediate node in the network which can carry the upstream or downstream data and passes it over the link to the destination or to another intermediate node. The introduction of such a node would not only extend the coverage and lower required signal power, but also provide better control and improvement in aspects like resource planning, traffic management, Quality of Service, and data processing.

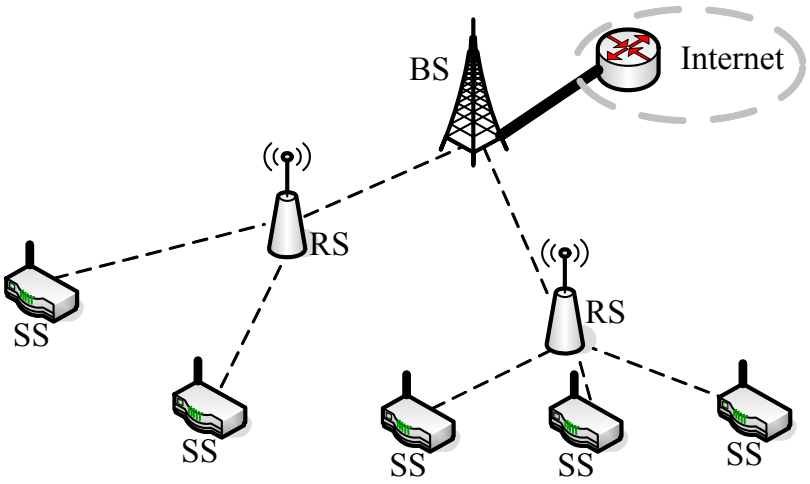


Fig 3.2 - The figure of a wireless multi-hop (2 hop) network with ‘Relay' Stations.

An intermediate node in the WiMAX scenario must to able to receive data from a source node and then transmit it to destination node (or to another intermediate node) over WiMAX links. According to the WiMAX (802.16 e) protocol, an SS radio can communicate with only a BS radio. In order to transmit data to an SS, this intermediate should posses the WiMAX BS functionality. Hence, the design of this intermediate needs to accommodate the functionalities of both SS and BS.

The design of the Relay Station (RS) node had two WiMAX interfaces, with an SS radio for one interface and a BS radio for the other. The interface with a SS radio could be used to communicate with the BS radio of other node and the BS radio of the relay with SS radios. Both these interfaces should be operating on non-interfering frequencies [6], hence the BS-RS link and RS-SS link should be operating on different frequencies.

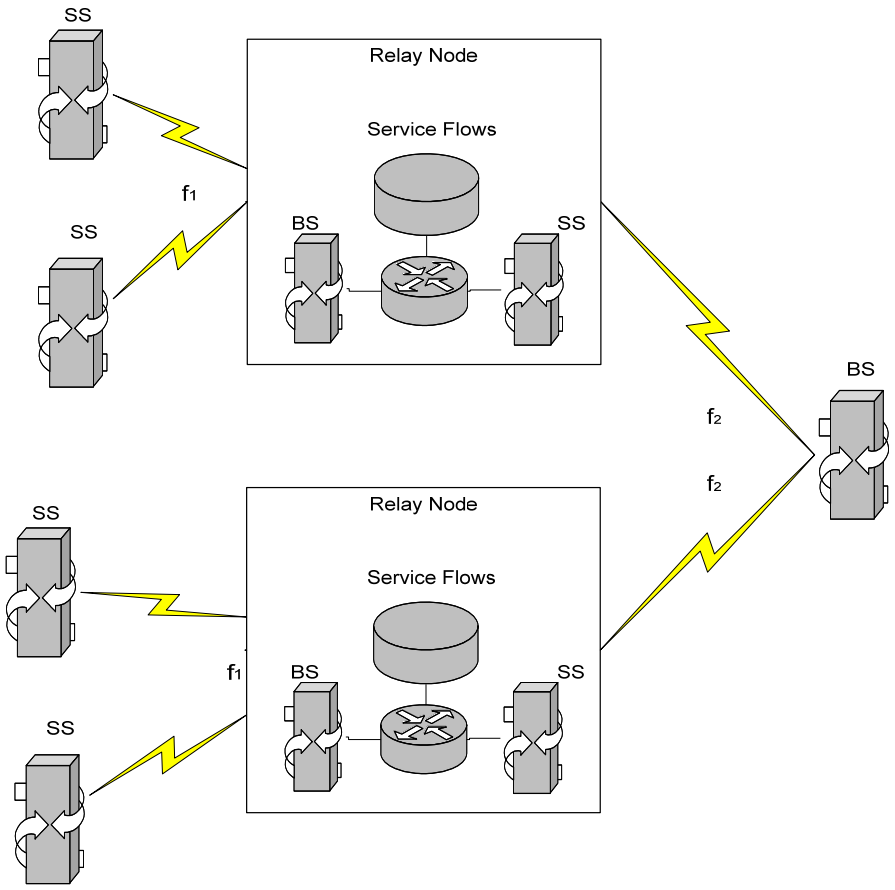


Fig. 3.3 - The figure has a BS, 2 RSs and 4 SSs. The design of the relay node consists of BS and SS interfaces.

These two interfaces of the Relay Station have separate MAC and ARP processes, integrated at the IP layer. At the MAC layer, one interface would assume the role of an SS and other of a BS. These two interfaces would be operated on different frequencies to avoid any co-channel interference. The two radios are independent of each other and could work as two separate WiMAX links. Hence, both can use different antennas and other PHY profiles. The data received by either of the interfaces of an RS can also be stored and managed in a separate server at the relay through an Ethernet interface.

In WiMAX, the relay node designed using OPNET Modeler [5], each interface has its own set of WiMAX parameters and IP address which can be set differently according to requirements of various applications and network tasks.

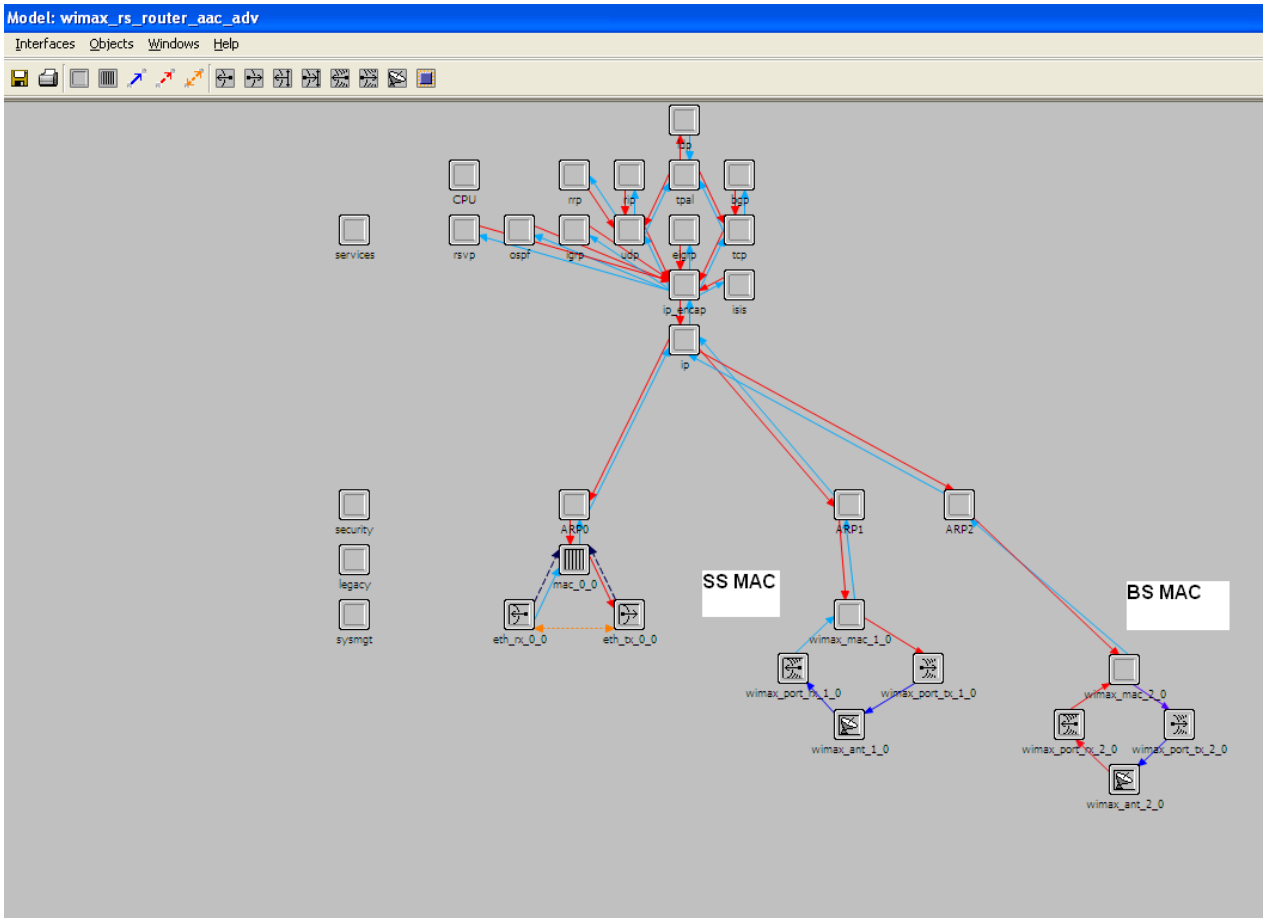


Fig. 3.4 - The figure shows the design of WiMAX relay node in OPNET Modeler.

3.2 WiMAX mobile relay station

The availability of the mobility parameters in a relay station is required to configure the RSs for mobile cases and to handle mobility issues such as handover. A new mobile relay station was designed to enable mobility in an RS and to implement a complete WiMAX multi-hop mobile scenario in OPNET. Using this new model, the mobile parameters such as home agent, foreign agent etc. could be configured and the RSs could be mobile and handle the handovers. This design introduces the functionality of multi-hop handover in a wireless network.

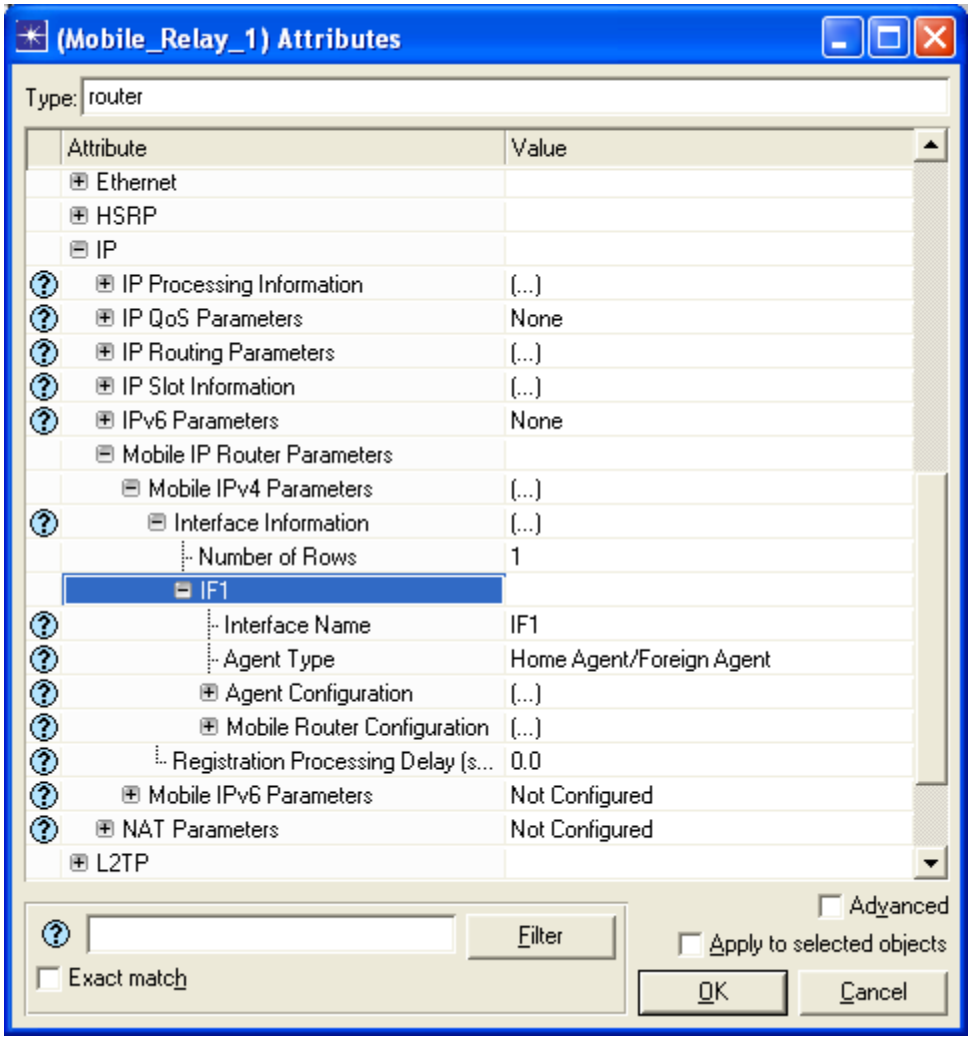


Fig. 3.5 - The figure shows mobility parameters of a WiMAX mobile relay in OPNET.

3.3 Directional antenna pattern in OPNET Modeler

Directional antennas in OPNET are created using the antenna pattern editor. In the antenna pattern editor, the gain of an antenna can be controlled and altered to provide directionality. This also helps in creating an antenna with high gain only in a small directional sector and the remaining directions with low gain such that the antenna shows significant resistance to mainlobes and sidelobes in the directions other than the small high gain sector.

The gain of an antenna is concentrated in a certain direction for a specific angle and beamwidth. The antenna can then be pointed towards its target nodes; this utilizes the gain of the antenna in the direction of communication. For a fixed power level, directional antennas provide more power in the direction of communication than an omni-directional antenna. [17]. Power utilization is improved by focusing gain in required direction only. This forms a relatively small

directional sector with high gain where the data communication takes place and the remaining directions with low gain, where no intended communication takes place.

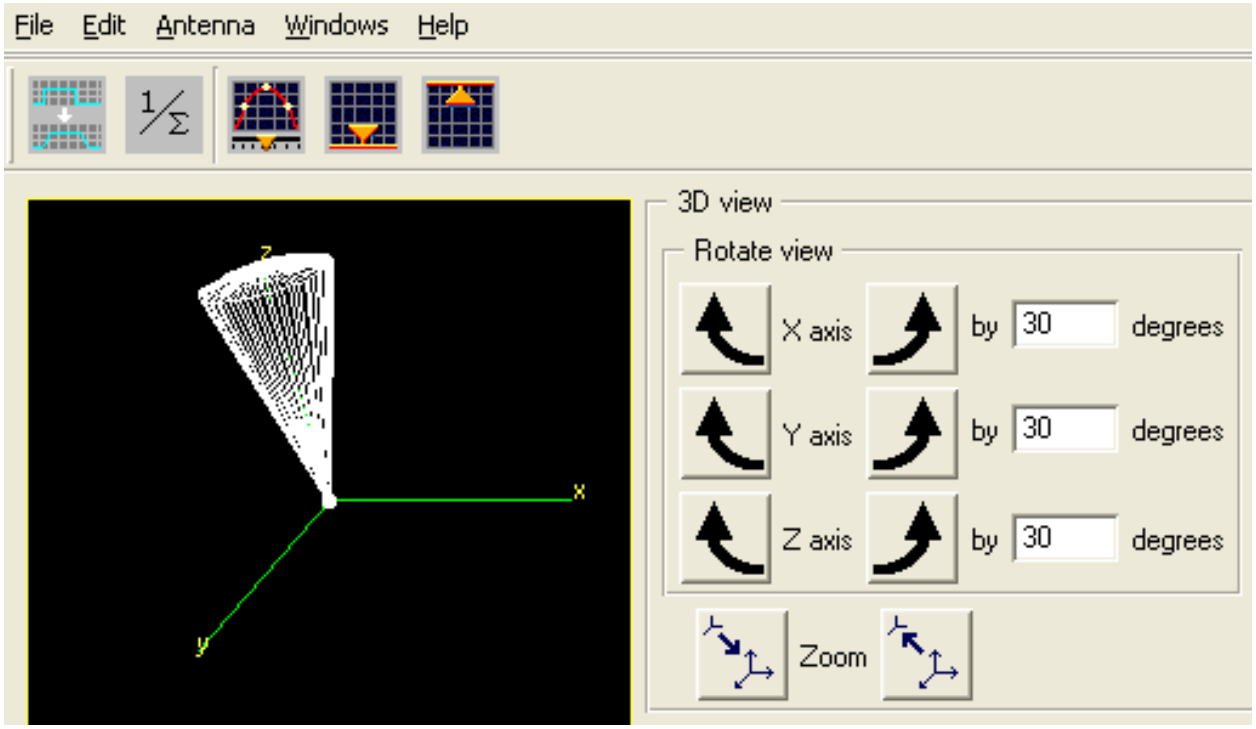


Fig. 3.6 - The figure show a directional antenna pattern created with the OPNET antenna pattern editor [17].

These antennas also improve overall network efficiency and performance as any interference of data communication is avoided by not transmitting the signal in space all around when not required. This introduces the spatial degree of freedom in wireless communication.

Pencil beam antenna pattern or 'fan type' pattern:

The directional antenna patterns can further exploit the concentration of gain to provide pencil beam or fan type beam. Unlike a directional sector, in a pencil beam antenna pattern the location of each of the target nodes is determined and the gain is focused accurately to each of them. This gives an antenna beam with a low gain values in between the peaks of high gain values, which are pointed to the targets. These narrow high gain beams are used for communication and the utilization of the power is further increased.

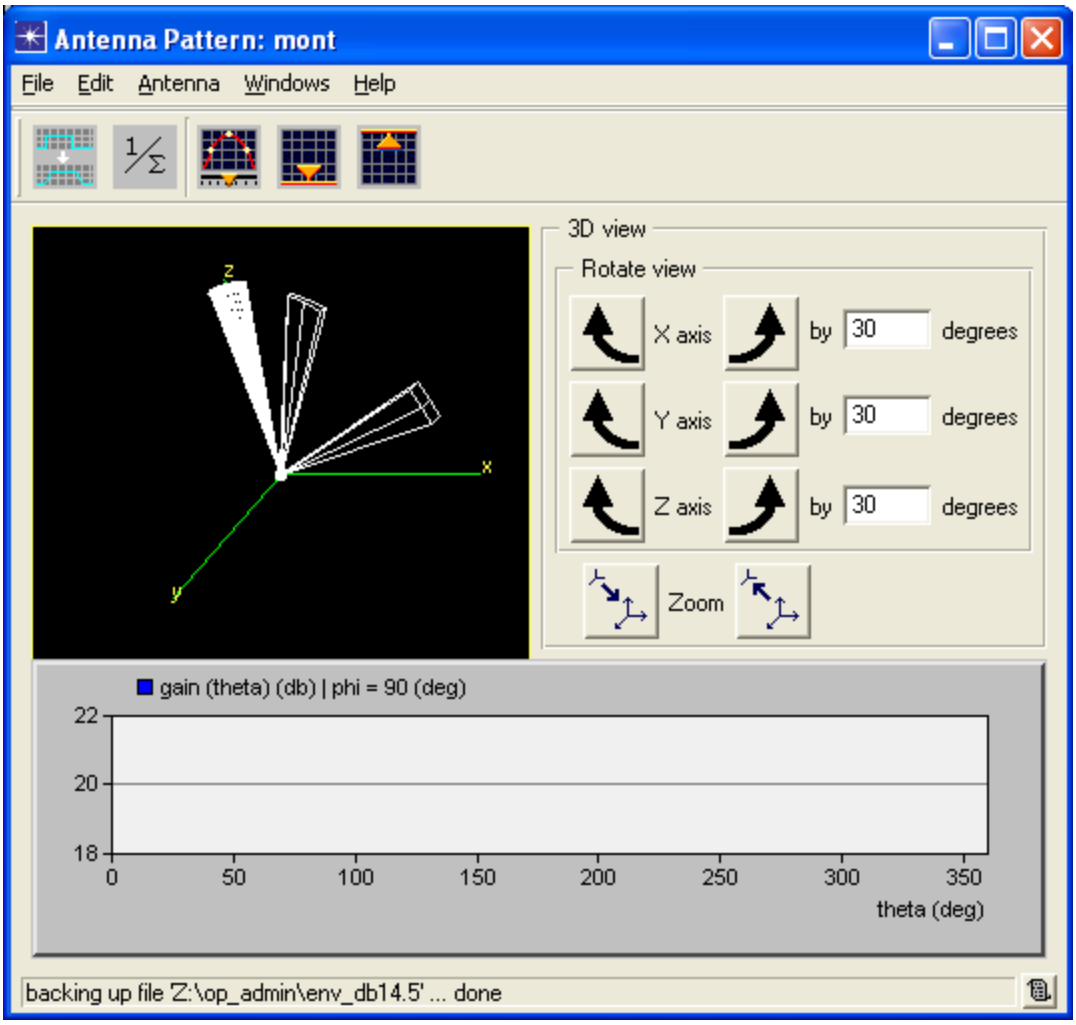


Fig. 3.7 - The figure shows a pencil beam antenna pattern [17].

However, the beam pointing has to be very accurate and is limited to very small relative movement of the set of the target nodes. Thus, this shall only be useful in a scenario where the movement of the Subscriber Stations is infrequent or slow. For any movement of nodes, where the directional antenna could no longer cover the target nodes with the pencil beam antenna, re-calculation of location and re-creation of the beam needs to be done.

3.4 Implementation of directional antennas by PHY layer coding in OPNET

Pointing the directional antenna towards the target nodes required control at physical layer. In OPNET Modeler, there are certain files which have the physical layer implementations. One such file is ‘wimax_phy_support.ex.c’, where we can control transmitter and packet sending implementations. The intended point of direction of the antenna beam can be specified here.

There is an OPNET Modeler kernel function called ‘op_ima_obj_attr_get()’, which can determine location of any entity in the network scenario, in latitude, longitude and altitude parameters. Hence, the locations of target nodes can be obtained using this function and then the appropriate target location angles can be calculated.

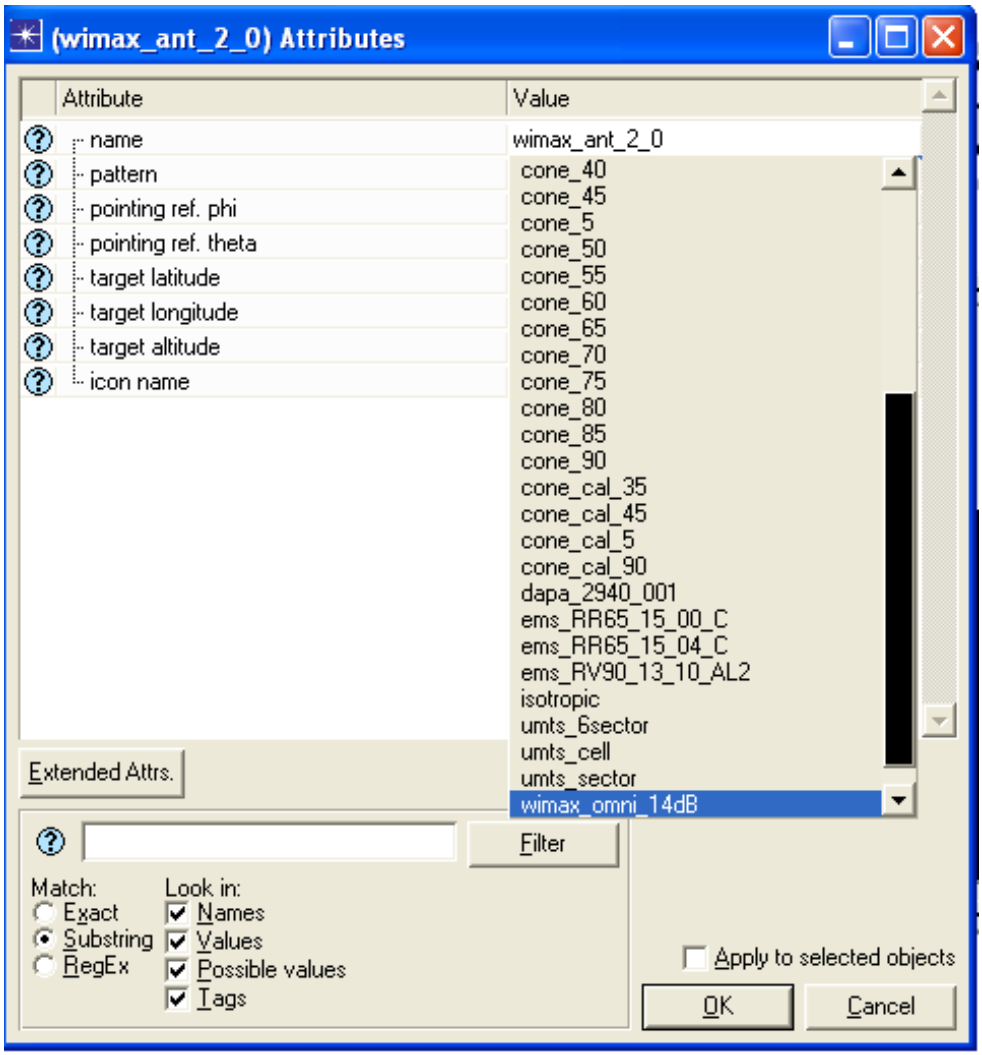


Fig. 3.8 - The figure shows the set of directional antennas which a transmitter of a node can choose from.

So, using the location parameters the target location for the antenna beam is set in a function called ‘wimax_phy_mcarrier_pk_send()’, where the transmitter is handled just before it sends

out the packets with the help of function called ‘op_ima_obj_attr_set()’ which is used to set several attributes.

Now, this target location can be changed over period of time or for every packet that is sent through the transmitter. The dynamic selection of the beamwidth in use can be made using the ‘op_ima_obj_attr_set()’ function where at any point in time the beamwidth in use can be changed by setting the transmitter with an antenna with a different beamwidth. Hence, if during a simulation of data communication a WiMAX relay station needs to accommodate few more new SS nodes or relinquish service to some of the existing ones, then it extends the coverage angle accordingly and chooses a different antenna beam. This is also used to study the difference in the behavior and results when different antenna beams work in a given condition.

The directional antenna could be made adaptive in a scenario where the antenna beam and its target location both change frequently. In a mobile scenario, where the SS nodes move either in specified trajectories or randomly, it is desirable that the antenna (of RSs and SSs) be adjusted according to the movement of these mobile SSs. Since, the trajectory of each node is not predetermined this frequent change in antenna beam and targeting needs to be adaptive.

```
rx_node_id11 = op_id_from_name (subnet_id, OPC_OBJTYPE_NODE_FIX, "BS");
op_ima_obj_pos_get (rx_node_id11, &latitude, &longitude,&altitude, &x_pos, &y_pos, &z_pos);
BS_lat=latitude;
BS_log=longitude;
BS_alt=altitude;
```

Fig. 3.9 - The figure shows a piece of code implemented in a PHY layer code file in OPNET.

The current location of each node in the network is determined and then the proper antenna beamwidth required by the RSs to cover the SSs is calculated and the appropriate target location is obtained. Then using the OPNET kernel functions [5] these parameters of transmitter are set accordingly every time a packet is sent. There can be a relaxed periodic implementation in this procedure if the mobility of the scenario is low and changes in antenna beam are required less frequently.

Chapter 4

Topology control algorithms

This chapter introduces the need and idea of the topology control algorithms and then discusses their details. In the end it shows their effectiveness and compares various results.

4.1 Need and importance of the algorithms

In a standard WiMAX network, an SS associates to a BS depending on the distance between its SS radio and the target BS radio [2], [5]. It chooses the BS whose radio is closest to its SS radio. Typically, the closer a BS is to an SS, more appropriate and probable is their association.

Similar is the association rule in a multi-hop scenario, where there are customized relay stations which have two WiMAX radios, a BS radio and an SS radio. Here, the SS radio of an RS (Relay Station) associates to nearest BS and the BS radio of the RS associates to the SSs for whom it is the closest BS radio.

Now, in the above standard scenarios all the nodes use omni-directional antennas. But in certain scenarios, directional antennas are used to improve the gain and thus in many ways the efficiency of the network.

In such scenarios, where there are directional antennas, it is not always certain that all the SSs would be able to communicate with their closest RSs and moreover, the association and thus communication in the network is dependent and limited to the beamwidth of the directional antenna.

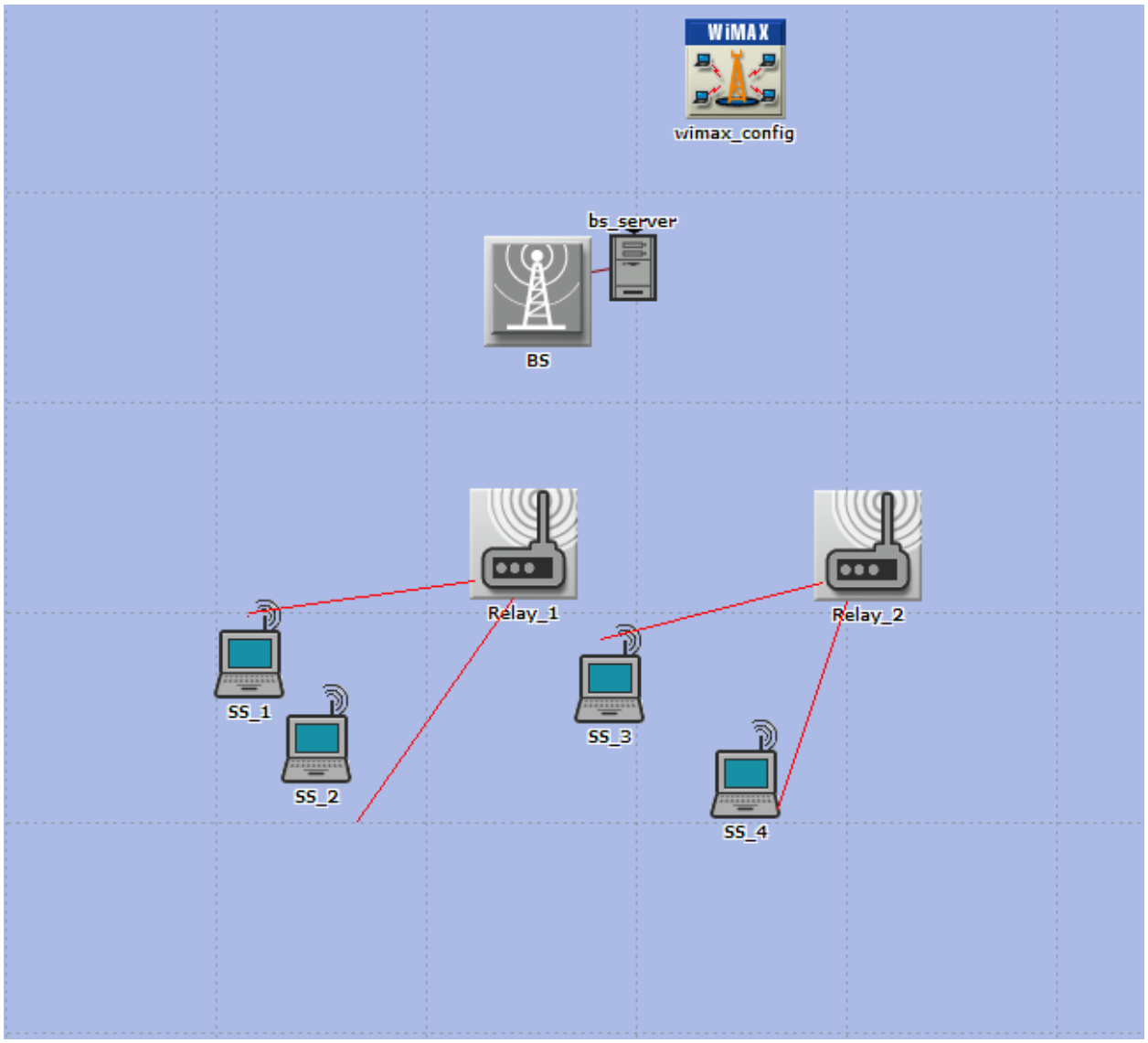


Fig. 4.1 - In the above figure *SS_3* cannot be associated to *Relay_1* even though it is closer to it due to limited beamwidth of the antenna.

Now, there is a need for a new method following which the coverage and association of every SS can be ensured. Thus, a method should be able to assert the topology of the network in a scenario with directional antennas.

While this can be done in several ways, the two topology control algorithms consider an effective way to achieve this where the goal is not just to associate the SSs and RSs but also to associate them in best possible way such that the overall network efficiency can be maximized.

4.2 Basic idea of the algorithms

In a scenario with directional antennas there is a deviation from the standard way of associating an SS and an RS, which was based on the distance between the nodes. A new method need not consider just the distance as the criterion and it must be different from being distance based as it may leave SSs not being able to communicate to the nearest RS with a directional antenna.

Since, the new method is not bound to a distance based criterion, there is opportunity for considering a new criteria set whose objective is to improve the overall network efficiency like increasing the end-to-end throughput, reducing the network delay or minimizing the interference traffic, while obtaining the SS-RS assignment which is feasible.

Here, the objective of these two topology control algorithms is to try to maximize the minimum data rate in a given network while making sure all the SSs can be covered and can communicated with one of the RSs.

These two algorithms are appropriate to use when all the SSs have constant data rate and have equal opportunities and needs. For example, in a network with randomly placed wireless sensor nodes (SSs) which send the primary information to their superior nodes such as intermediate data storage nodes (RSs) which can collect and collate information and submit data to the main data storage node (BS), all the wireless sensor nodes (SSs) are homogenous and have same job responsibility, have equal priority and requirements and need equal opportunity. Hence, achieving fairness and maximizing the consistency in their data rates is warranted and required.

Formulation of link capacity for this algorithm is as follows [18]:

Link capacity = k/ (POW (d, n) * θ);

- where, n= 2, 4.

**Since, there are directional antennas; the data rate of a link depends on the beamwidth of the antenna in use and the distance between the nodes. If the beamwidth is flexible and variable, each RS can offer a different data rate.*

4.3 Topology control algorithm-1

The assignment problem of obtaining an SS-RS assignment with minimum data rate of the network being maximized where each SS has more than one RS to choose with an RS beamwidth limitation and any possible beamwidth orientation such that every SS node is covered by an RS is a NP problem.

This is a linear greedy algorithm, where it assigns one by one each SS to an RS. The constraints for this algorithm are the fixed beamwidth of the directional antenna and maximum length

allowed for a communication link. Satisfying these constraints the algorithm assigns each SS one of the RSs it can possibly communicate to, where together the final beamwidth orientations of the all RSs would cover all the SSs, such that the minimum data rate of the given network can be maximized.

Following are the inputs for this algorithm:

- Location of each SS and RS node. The whole scenario and placement of nodes can be taken into a co-ordinate system.
- Maximum allowed length of a communication link, say 'd_max'. This is a constraint which limits the minimum data rate that the network can have.
- The beamwidth angle of the directional antenna the RSs use, say 'θ'.
- The values for calculation constants.

This algorithm works in steps, where in each step an SS that has to be associated to an RS is chosen determined by the its corresponding link capacity. It is the link capacity which determines the data rate of that link; hence by ensuring certain link capacity over a link for a node we provide the node with a value of achievable data rate.

First, a candidate RS set is generated for each SS in the network scenario using the input value of maximum allowed length of communication, d_max. In each SS's candidate set, an RS is included if the distance between the SS and that RS is less than or equal to the value of d_max.

Now, if any SS has only one candidate RS, then it is associated to that RS, as there is no other RS available for that SS. If there is no candidate RS, then the assignment problem cannot be solved for all those SSs and they have to be ignored for the algorithm. Ideally, this situation may be avoided by providing valid inputs and considering scenarios with at least one feasible assignment solution.

Then, at each step, for every SS node which is unassigned so far an RS selected and link capacity with the corresponding RS is determined. The RS which offers maximum link capacity to the SS among the other candidate RSs is selected. Here, a link capacity is valid only if the RS can cover the SS with its directional antenna with beamwidth of 'θ'. When an RS has at least one SS associated to it, its orientation is limited to a value of 'θ' or less. Hence, an SS cannot consider an RS if none of its current possible orientations can accommodate it.

When the current maximum possible link capacity for each unassigned SS is determined, the SS with least value of link capacity is selected since it is the 'bottleneck' node and it is associated with the corresponding RS. The association of this 'bottleneck' node ensures that this node gets its maximum possible link capacity, where other SSs can still possibly have better link capacities than this 'bottleneck' link capacity. Here, the constraint of coverage is satisfied as only valid link capacities are considered.

This procedure is used repeatedly, where at each step a ‘bottleneck’ node is selected and assigned to an RS; till there are no more unassigned SS are left.

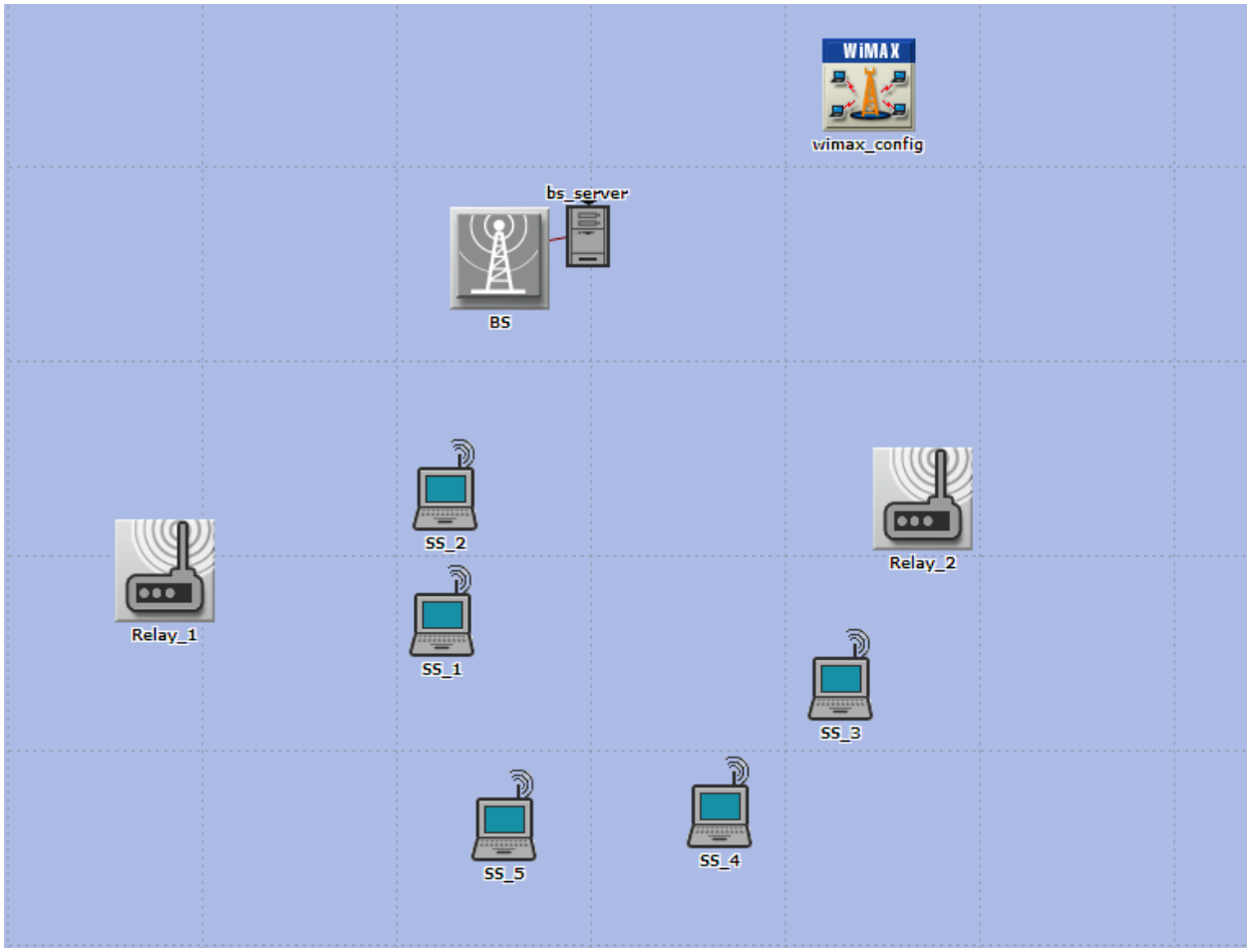


Fig. 4.2 - In this figure, there are 2 RSs and 5 SSs and beamwidth of each RS antenna is set to 40 degrees. Here, *SS_1* and *SS_5* both cannot be together associated to *Relay_1* though it is the nearest ‘relay’ due to the antenna beamwidth constraint.

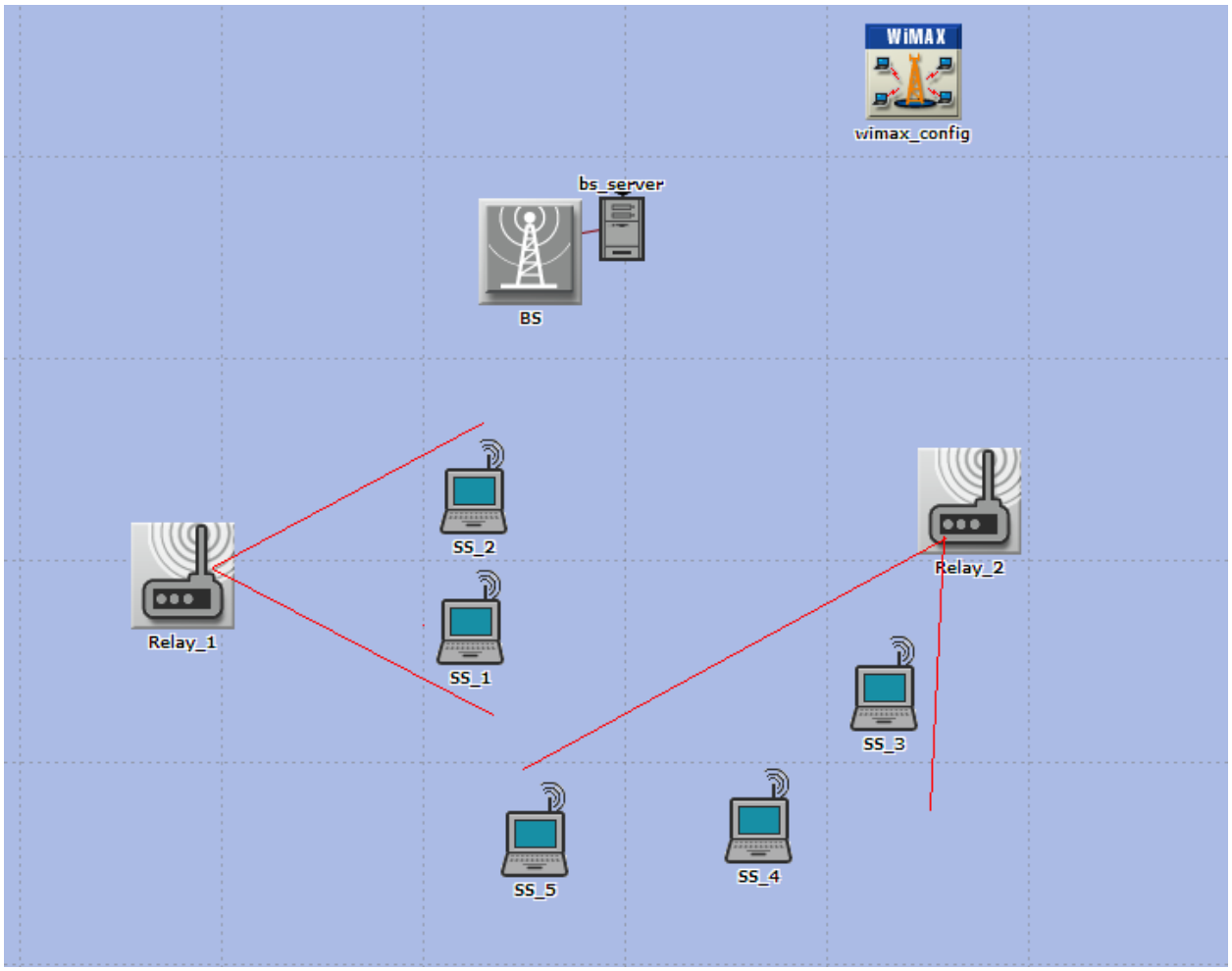


Fig. 4.3 - This figure shows the assigning of the SSs to their respective RSs after the assignment solution obtained by applying algorithm1; this assignment not only covers all the SSs but also maximizes the minimum link capacity of the given network.

4.4 Topology control algorithm-2

This algorithm focuses on improving the efficiency of the network by altering the current SS-RS assignment of a scenario. The method uses a local search based algorithm which always starts with the given feasible SS-RS assignment for the whole network and tries to improve it by re-assigning some SSs. The current assignment solution could have been given by any algorithm. For example, the greedy algorithm in the previous section can be executed to find out an initial solution at the very beginning.

Input for this algorithm is a feasible SS-RS assignment. The input values of parameters that were used for finding the initial assignment are retained for this algorithm as well. Since, this algorithm just alters the existing assignment; all the parameter values and constraints of the network scenario should be consistent throughout.

In each step, the algorithm selects the SS with current minimum link capacity and tries to assign it to one of the other candidate RSs such that the minimum link capacity in the whole network can be improved. It considers the next candidate RS which the SS hasn't tried to associate yet and checks if associating with this RS improves the minimum link capacity of the network. If none of the remaining candidate RSs could improve the minimum link capacity, then the algorithm stops, concluding that no better assignment was found.

If any of the candidate RSs yield improvement, then, the association with this RS is considered as a part of potential new solution. Then, it is checked if this new assignment still allows the new orientation of the directional antenna of the RS to cover all of the previously associated SSs. If yes again, the algorithm has found a better feasible solution. If no, we have to find a proper RS assignment for each of the SSs which would no more be covered by this RS, before validating that new assignment. While assigning a new RS to all of these SSs which would be left uncovered by the alteration of RS's antenna orientation, the improvement of minimum link capacity of the network is still a constraint. So, the RS assignment task for each of these SSs is the same as the task of assigning the SS with minimum link capacity, when the algorithm started.

If any of the SSs which were needed to get assigned to an alternate RS to validate all the preceding potential assignments run out of its candidate RSs and still hasn't found a valid assignment that improves the minimum link capacity of the network, the algorithm stops, having failed to give any better feasible solution for the given scenario.

If all the SSs which were needed to get assigned to an alternate RS to validate all the preceding potential assignments get assigned to an RS improving the current minimum link capacity, a new better feasible solution is found. This new SS-RS assignment is applied to the network and again, the SS with current minimum link capacity is selected and a new assignment is tried to obtain a new minimum link capacity of the network that is better. This process is repeated until there is no more significant improvement or the algorithm stops, failing to give a better assignment.

4.5 Comparison between algorithm-1 and algorithm-2

The two topology control algorithms try to maximize the minimum link capacity of the network while approaching in different ways. The first algorithm tries to find an SS-RS scheduling assignment in a linear greedy way, while the second algorithm needs an existing SS-RS assignment schedule to start with for altering the assignment and improving the existing minimum link capacity.

The first algorithm works in a linear way and is a polynomial time algorithm in the order of ‘n’, the number of SSs present in the network. The second algorithm is based on the idea of an exhaustive search technique and tries to optimize the solution exponentially, but the algorithm is implemented using an appropriate *constant* value to limit the search to the product of number of RSs and number of SSs in the network.

4.6 Example network scenario where both topology control algorithms were applied

An example scenario is shown in figure below, where algorithm-1 is used and after that algorithm-2 is used. This example shows how using algorithm-1 provides a feasible solution and then application of algorithm-2 further improves the objective of maximizing the minimum link capacity.

Fig. 4.4 - an example scenario with 8 SSs and 3 RSs.

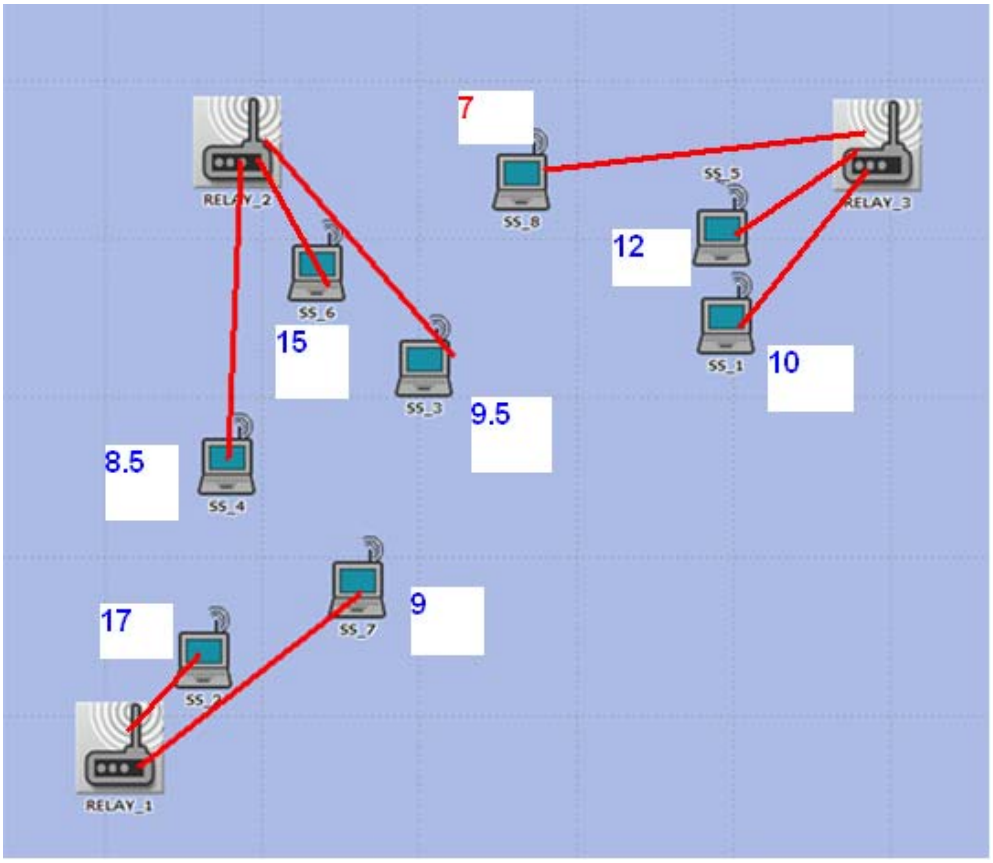


Fig. 4.5 - shows the SS-RS assignment schedule obtained by implementing algorithm-1.

The minimum link capacity for this network topology is 7Mbps, using a directional antenna beamwidth of 40 degrees.

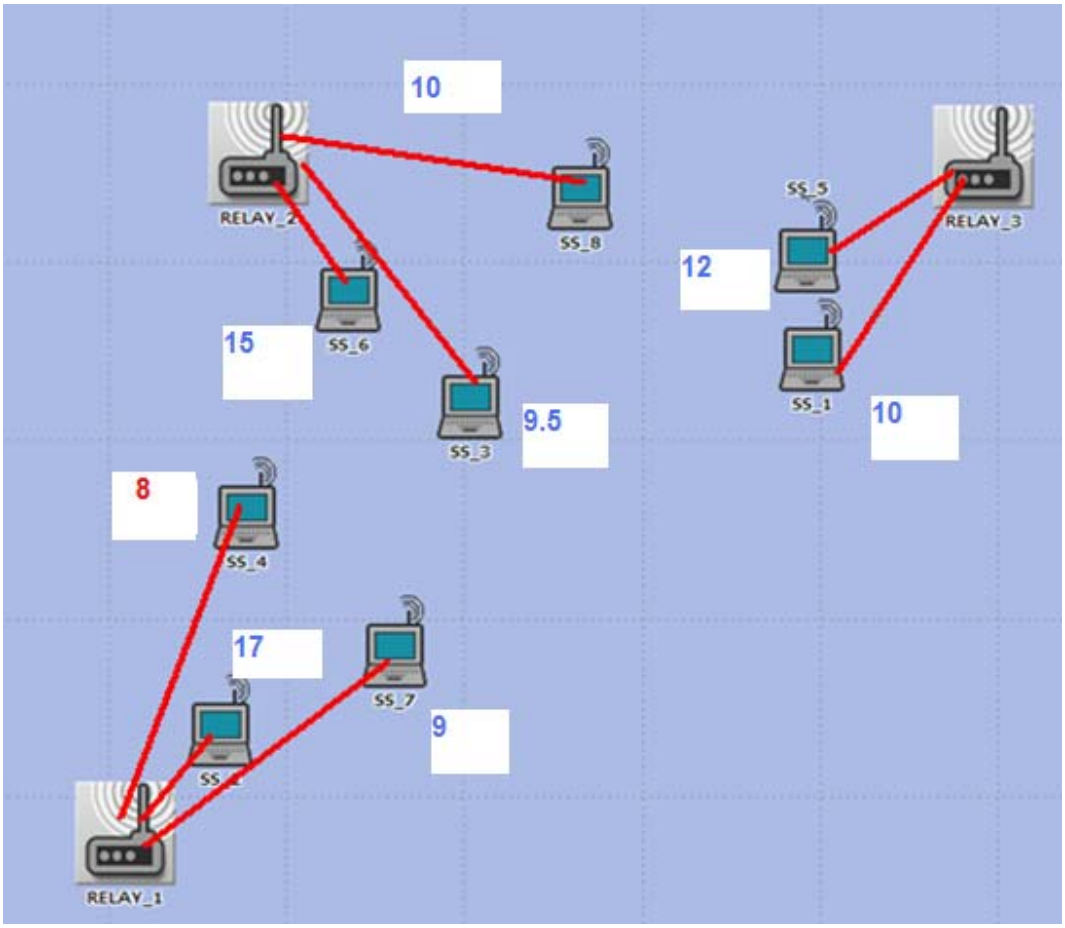


Fig. 4.6 - shows the SS-RS assignment schedule obtained after implementing algorithm-2, when algorithm-1 is already used for providing an initial schedule as shown in previous figure.

This figure shows that this SS-RS assignment schedule has improved minimum link capacity due to application of algorithm-2. The minimum link capacity for this network topology is 8Mbps, using a directional antenna beamwidth of 40 degrees.

Chapter 5

QoS in WiMAX multi-hop networks

The Quality of Service features in WIMAX stand out relative to existing wireless communication technologies. QoS in IEEE 802.16e is supported by allocating each connection between the SS and the BS (called a service flow) to a specific class of QoS. There are 5 QoS classes to which a service flow can be categorized.

Service	Abbrev	Definition	Typical Applications
Unsolicited Grant Service	UGS	Real-time data streams comprising fixed-size data packets issued at periodic intervals	T1/E1 transport
Extended Real-time Polling Service	ertPS	Real-time service flows that generate variable-sized data packets on a periodic basis	VoIP
Real-time Polling Service	rtPS	Real-time data streams comprising variable-sized data packets that are issued at periodic intervals	MPEG Video
Non-real-time Polling Service	nrtPS	Delay-tolerant data streams comprising variable-sized data packets for which a minimum data rate is required	FTP with guaranteed minimum throughput[citation needed]
Best Effort	BE	Data streams for which no minimum service level is required and therefore may be handled on a space-available basis	HTTP

Fig. 5.1 - The figure above briefly explains different 802.16e-2005 QoS classes.

In this section, these different QoS classes are tested and analyzed in WiMAX multi-hop scenarios using OPNET. This work has not been reported elsewhere to the bet of our knowledge. Hence, the detailed study and analysis of the QoS classes and their behavior, implementations shall be useful for planning to achieve QoS in various applications and networks.

The issues and parameters like network size, number of hops, network resources, gain and directivity of antennas and devices being used, traffic demand, channel bandwidth, frame division and coding techniques will have significant effect on the overall system performance. A comprehensive simulation study is conducted by setting these parameters appropriately and to different values to examine the effects for typical network scenarios.

General settings for the simulations are given in the table below.

Region Size	5 Km X 5 Km
Frequency	5.73 GHz to 5.87 GHz
Channel bandwidth	10MHz
Antenna beamwidth	40 degree
Antenna gain	15dBi
Simulation duration	10min

Table 5.1 – General settings for simulations.

The simulation is carried out on two scenarios of different sizes, a small scenario with 1 BS, 1 or 2 RSs and 3 SSs and a larger scenario with 1 BS, 6 RSs and 30 SSs.

For each scenario, the WiMAX data frame is divided in two ways:

- 75% downlink and 25% uplink
- 25% downlink and 75% uplink

These two division options of splitting a WiMAX data frame into downlink and uplink are considered as they create and represent two different network situations and applications. By considering these two division ratios we can bring in the influence of resource domination by upstream and downstream separately and thus can study their effects.

In a scenario, all the links follows the same setting and configuration for making the performance analysis simple yet comprehensive. However, for in practical applications, we could have different setting and configuration for each link and node in order to maximize the efficiency of the system. But while simulating a network in providing and considering various conditions as different and extreme as possible would make it appropriate and more comprehensive to study and research.

In these scenarios, the behavior of QoS classes is observed by comparing the results where specified QoS classes were used with the scenario where all the traffic was managed on Best Effort basis. For studying how each of the QoS classes behave, a large number of simulations were conducted, by using each QoS class separately and with others, including all possible combinations and network conditions.

The basic general behavior is shown in the scenarios that are presented in this section. In some of the scenarios, three different types of QoS classes were used together in a network for equal numbers of nodes, to study their prioritized behavior as these QoS classes co-exist and compete with one another.

Here, in scenarios where there are more than one RS, each RS would be operating in a frequency channel different to one another while communicating with SSs nodes. This configuration is considered to provide enough resources for the RSs and to eliminate any co-channel interference among them.

Before running these simulations, the network connection set up as was obtained by using topology control algorithm-1 and topology control algorithm-2. The SS-RS assignment was obtained by using these algorithms, in order to better arrange the SSs for achieving network with good link capacity for each of the SSs. Also, directional antenna beamwidth is 40 degrees and orientation is properly set towards the target nodes.

5.1 Simulations with small network size.

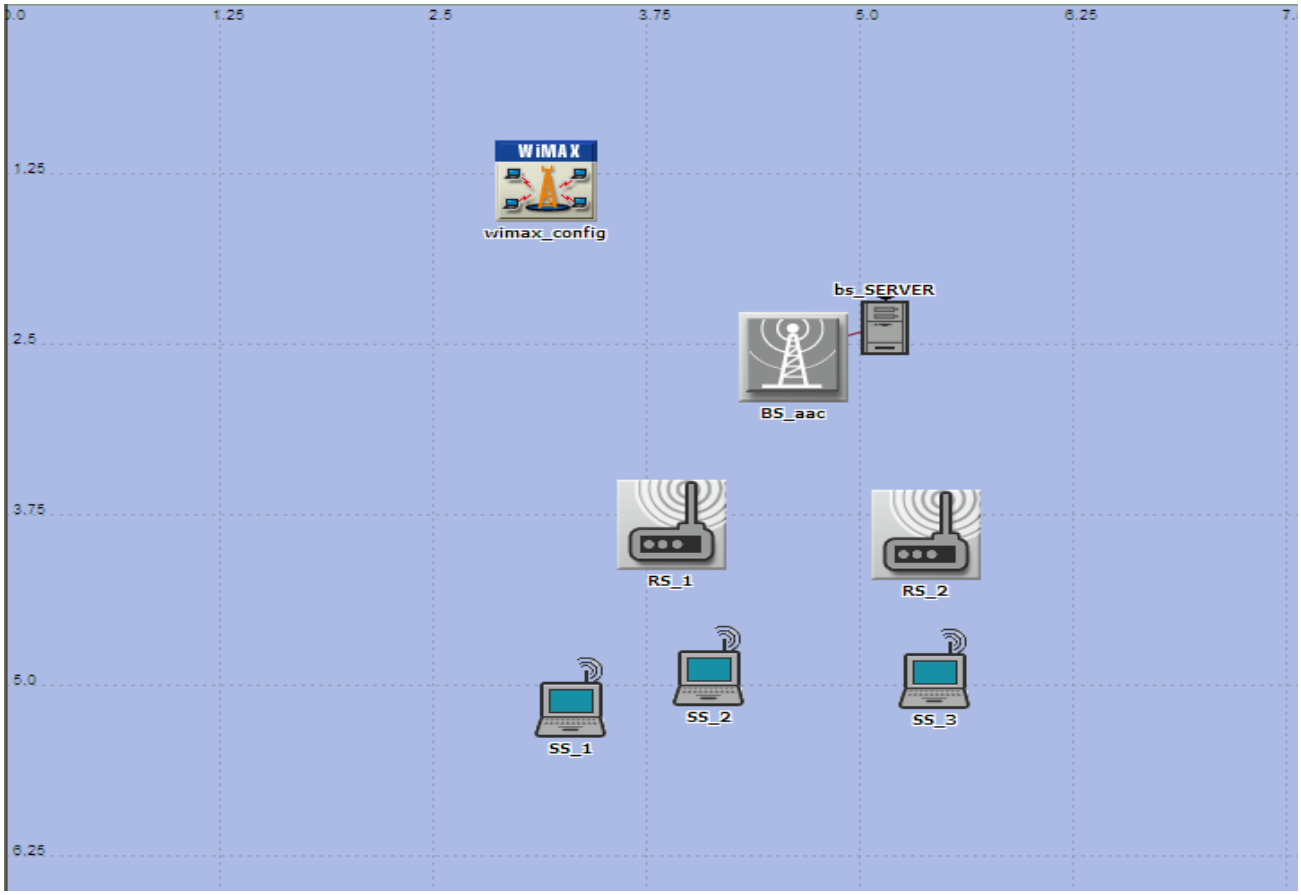


Fig. 5.2 - show the small scenario used for the simulation with 1 BS, 2 RSs and 3 SSs.

Scenario 1: All service flows with default type (Best effort)

Network size	1 BS, 2RSs and 3SSs
Service type	Best effort
Total Downlink traffic demand	28Mbps
Total Uplink traffic demand	3.5 Mbps
Frame division	75% downlink, 25% uplink

Table 5.2 – Scenario-1 settings.

Downlink:

Traffic sent by **BS** Total traffic sent by **all RSs** Total traffic received by **all SSs**

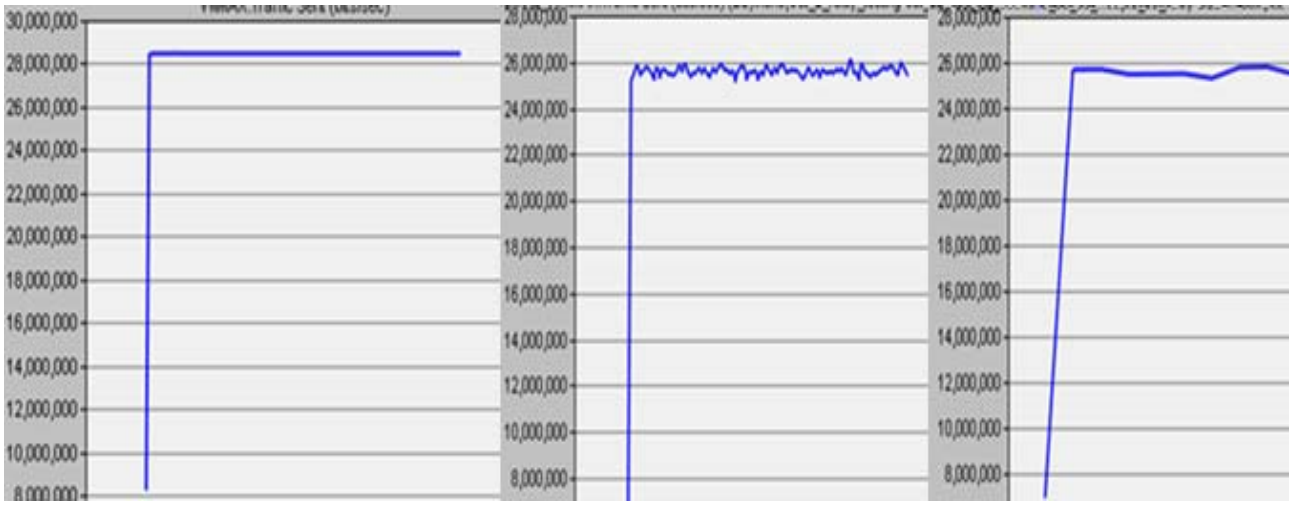


Fig. 5.3 – Downlink graphs for scenario-1

Uplink:

Total traffic sent by **all SSs** Total traffic received by **all RSs** Traffic received by **BS**

Fig. 5.4 – Uplink graphs for scenario-1

Scenario 2: All service flows with default type (Best effort)

Network size	1 BS, 2RSs and 3SSs
Service type	Best effort
Total Downlink traffic demand	3.5Mbps
Total Uplink traffic demand	28Mbps
Frame division	25% downlink, 75% uplink

Table 5.3 – Scenario-2 settings.

Downlink:

Traffic sent by **BS** Total traffic sent by **all RSs** Total traffic received by **all SSs**

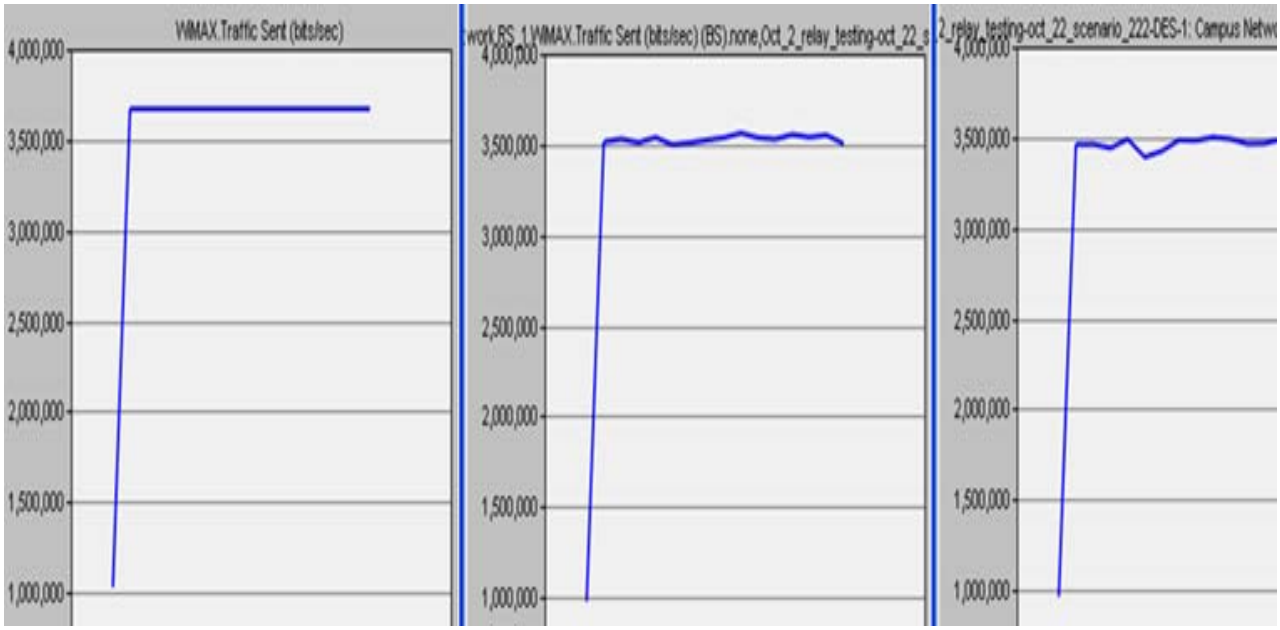


Fig. 5.5 - Downlink graphs scenario-2

Uplink:

Total traffic sent by **all SSs** Total traffic received by **all RSs** Traffic received by **BS**

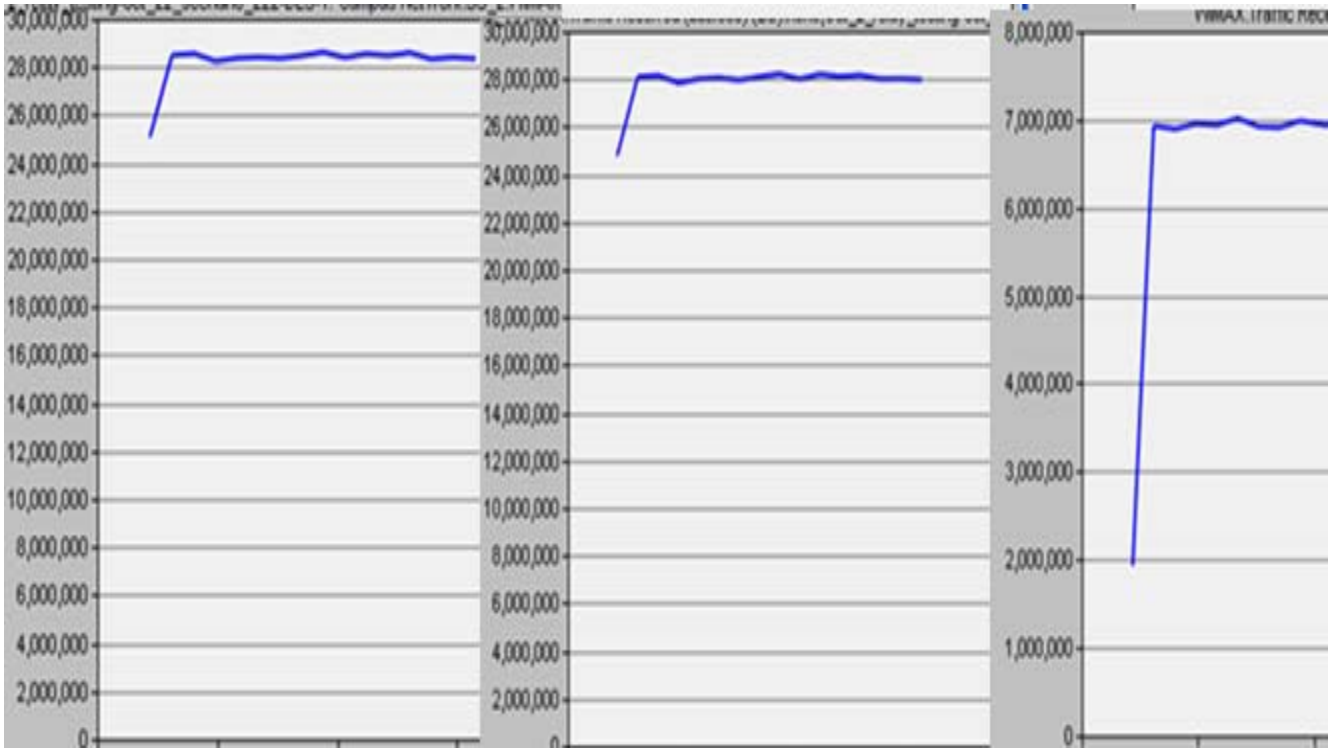


Fig. 5.6 - Uplink graphs scenario-2

Scenario 3: Service flows with QoS types: UGS, rtPS, BE (Each SS has a different service flow)

Network size	1 BS, 1RS and 3SSs
Service type	UGS/rtPS/Best effort
Uplink traffic demand	9.33/9.33/9.33Mbps
Frame division	25% downlink, 75% uplink

Table 5.4 – Scenario-3 settings.

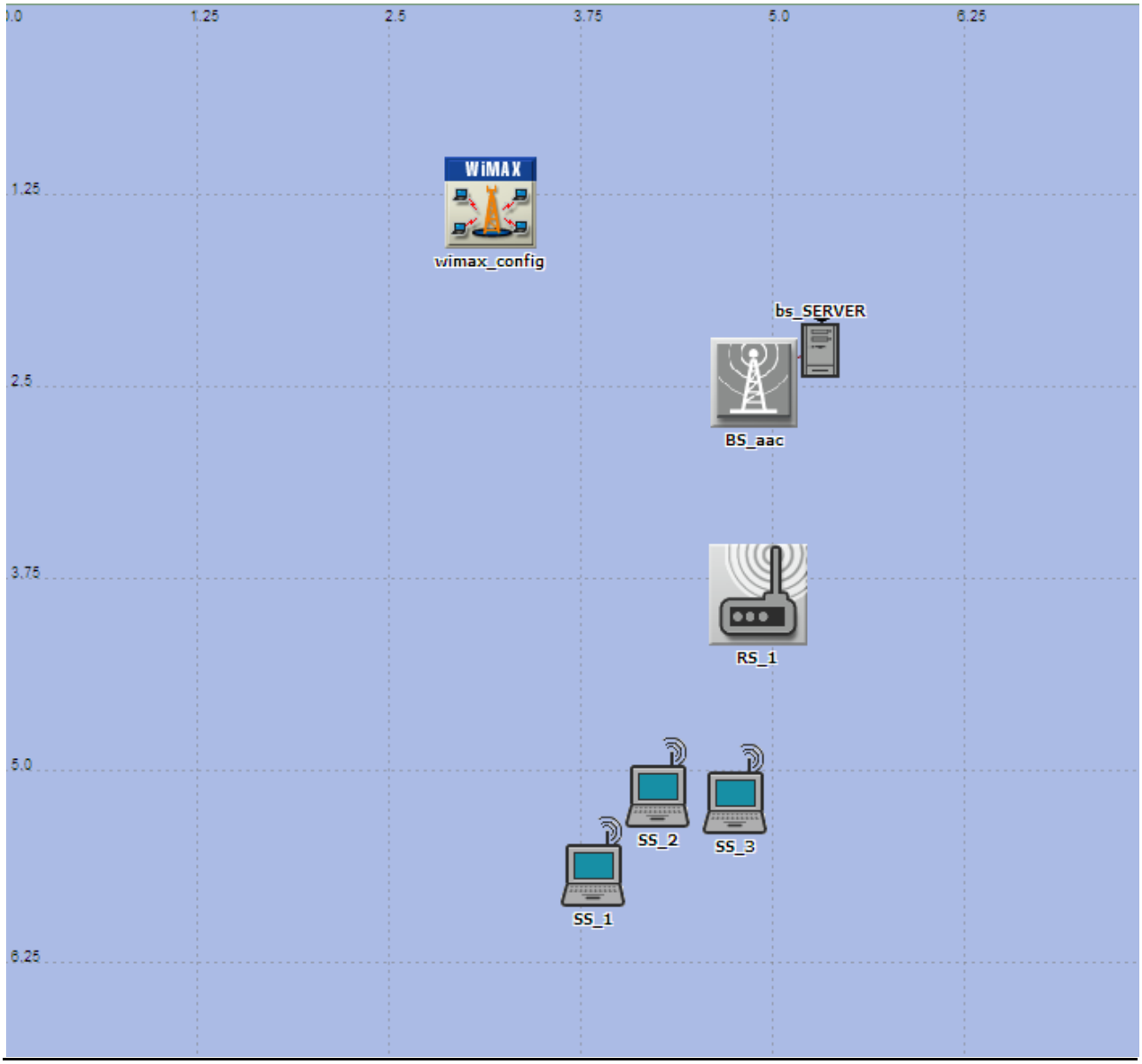


Fig. 5.7 - The figure show the small scenario used for the simulation with 1 RS and 3 SSs.

Bandwidth for UGS, rtPS, BE traffic

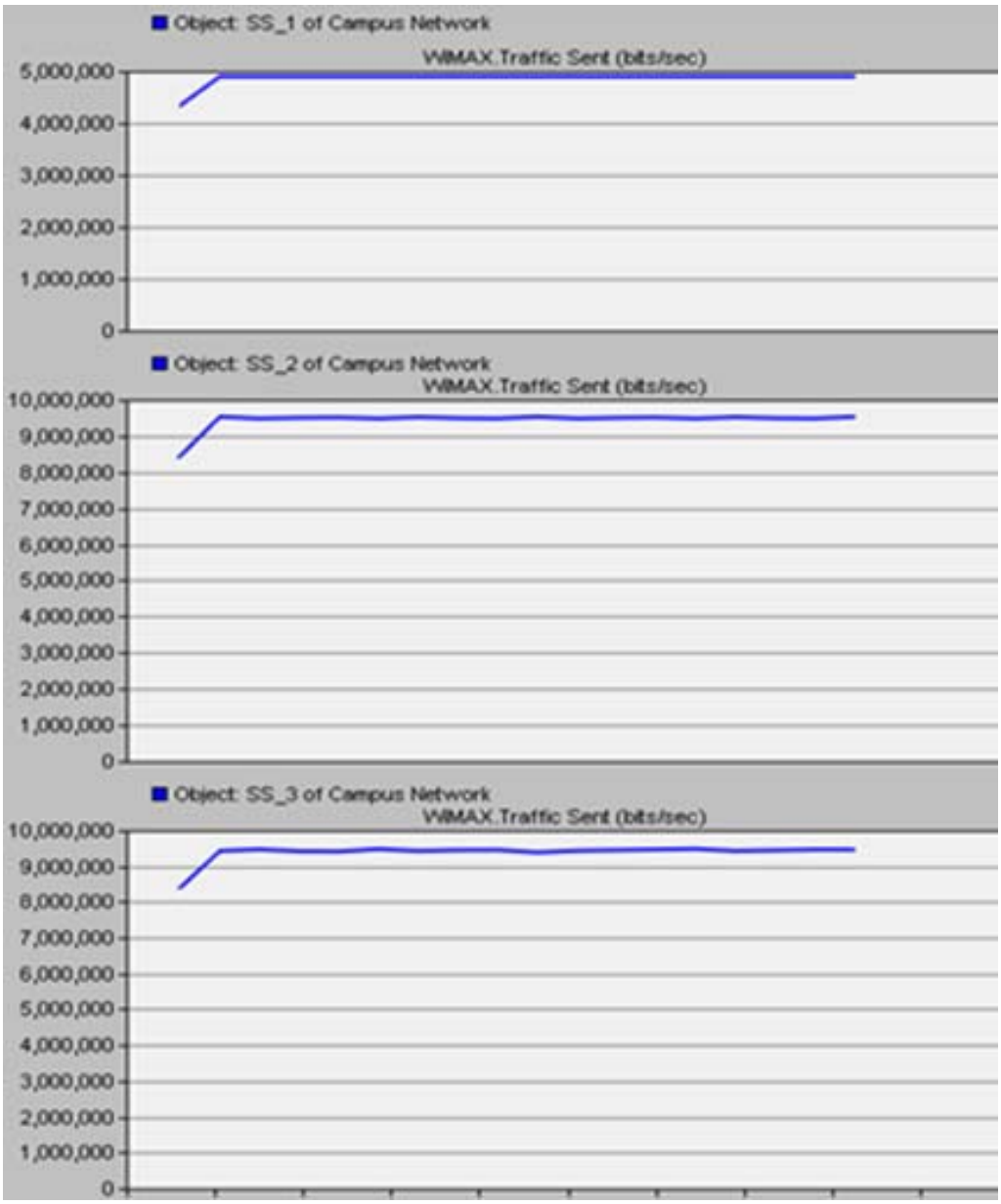


Fig. 5.8 - QoS graphs scenario-3

Total traffic received by **RS**

Fig. 5.9 – Relay receive graph scenario-3

Scenario 4: Service flows with QoS types: UGS, rtPS, BE (Each SS has a different service flow)

Network size	1 BS, 1RS and 3SSs
Service type	UGS/rtPS/BE
Uplink traffic demand	5/5/5 Mbps
Frame division	75% downlink, 25% uplink

Bandwidth for UGS, rtPS, BE traffic respectively

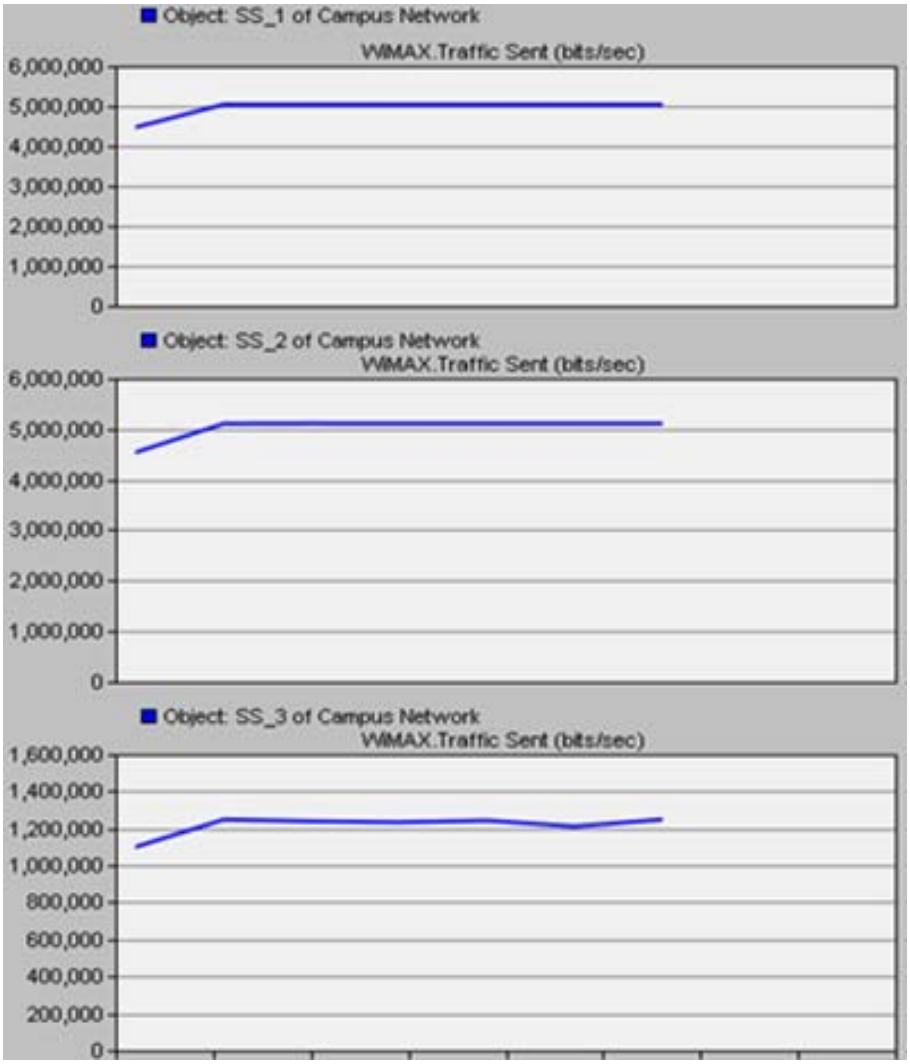


Fig. 5.9 - QoS graphs scenario-4

Total traffic received by **RS**

Fig. 5.10 – Relay receive graph scenario-4

5.2 Simulation with large network size.

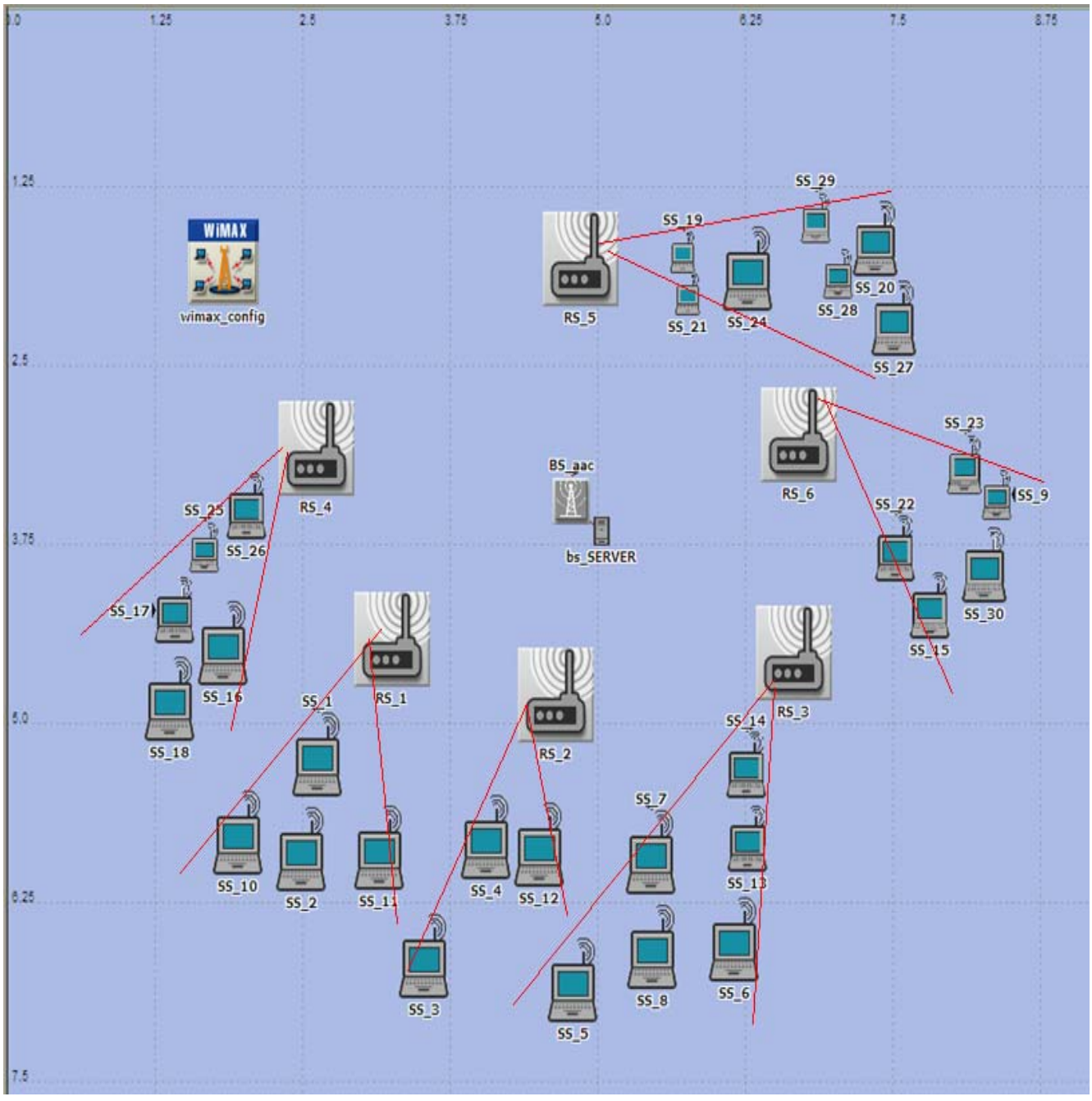


Fig. 5.11 - shows a larger network with 1 BS, 6 RSs and 30 SSs and their association when 40 degree directional antenna beams are used.

Scenario 5: All Service flows with default type (Best effort)

Network size	1 BS, 6RSs and 30SSs
Service type	Best effort
Downlink traffic demand	28Mbps
Uplink traffic demand	3.5 Mbps
Frame division	75% downlink, 25% uplink

Downlink:

Traffic sent by **BS** Total traffic sent by **all RSs** Total traffic received by **all SSs**

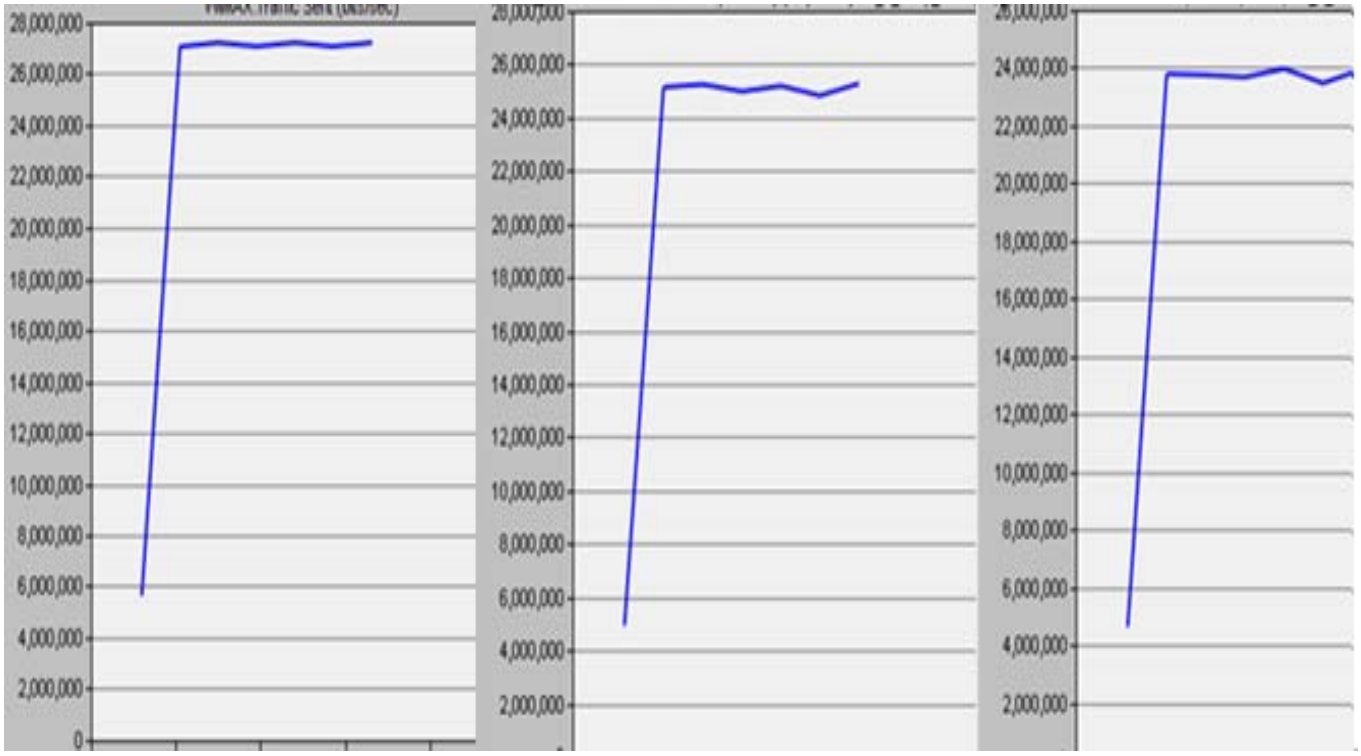


Fig. 5.12 – Downlink graph scenario-5

Uplink:

Total traffic sent by **all SSs** Total traffic received by **all RSs** Traffic received by **BS**

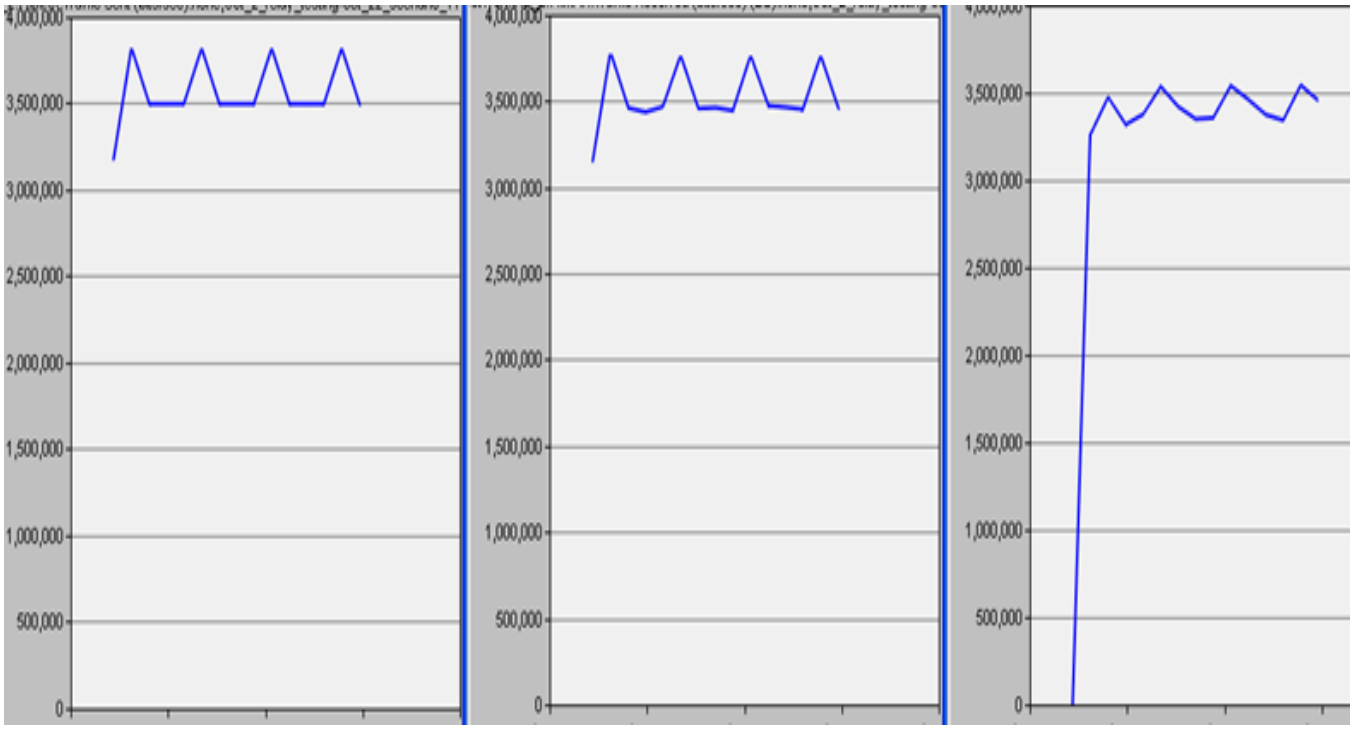


Fig. 5.13 – Uplink graph scenario-5

Scenario 6: All Service flows with default type (Best effort)

Network size	1 BS, 6RSs and 30SSs
Service type	BE
Downlink traffic demand	3.5Mbps
Uplink traffic demand	28Mbps
Frame division	25% downlink, 75% uplink

Downlink:

Traffic sent by **BS** Total traffic sent by **all RSs** Total traffic received by **all SSs**

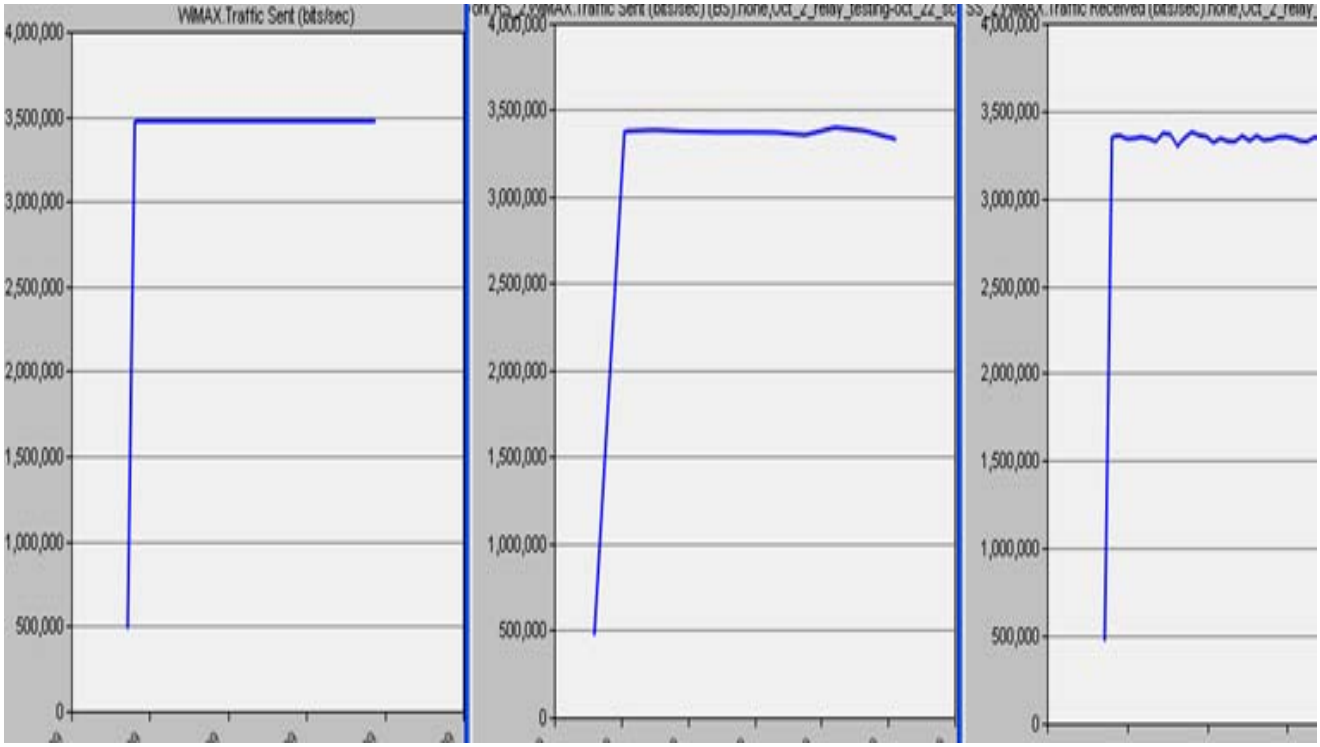


Fig. 5.14 – Downlink graph scenario-6

Uplink:

Total traffic sent by **all SSs** Total traffic received by **all RSs** Traffic received by **BS**

Fig. 5.15 – Downlink graph scenario-6

Scenario 7: Service flows with QoS types: UGS, rtPS, BE (Every 10 SSs have a similar service flow)

Network size	1 BS, 6RSs and 30SSs
Service type	UGS/rtPS/BE
Uplink traffic demand	9.33/9.33/9.33Mbps
Frame division	25% downlink, 75% uplink

Bandwidth for **UGS** traffic Bandwidth for **rtPS** traffic Bandwidth for **Best Effort** traffic

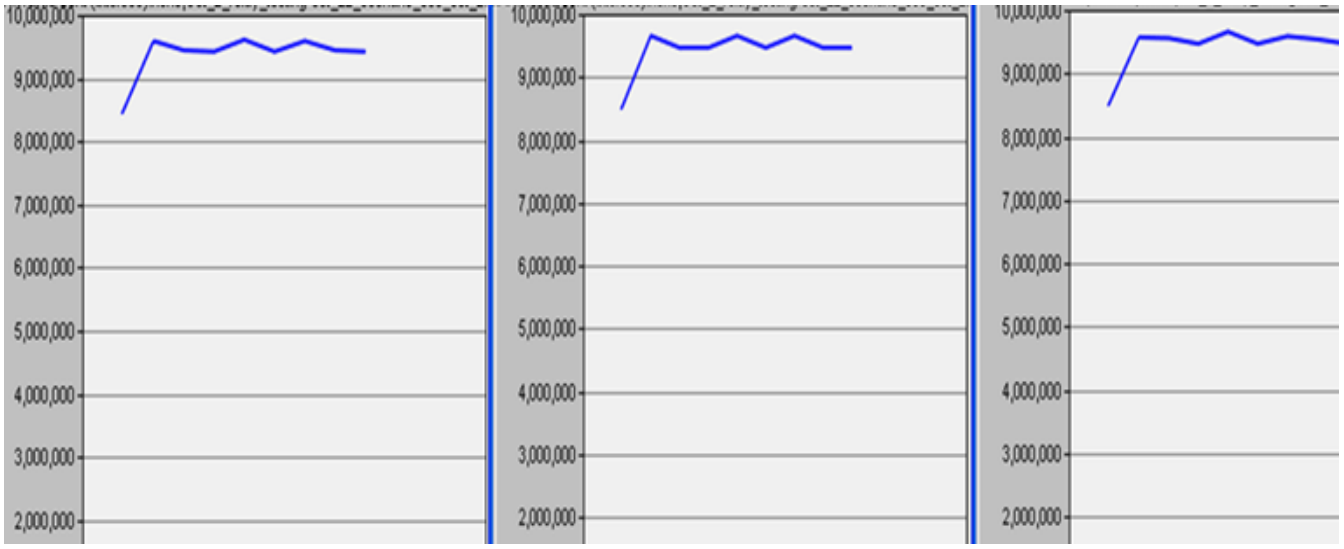


Fig. 5.16 – QoS graphs scenario-7

Total traffic sent by **all SSs** (UGS, rtPS, BE) Total traffic received by **all RSs**

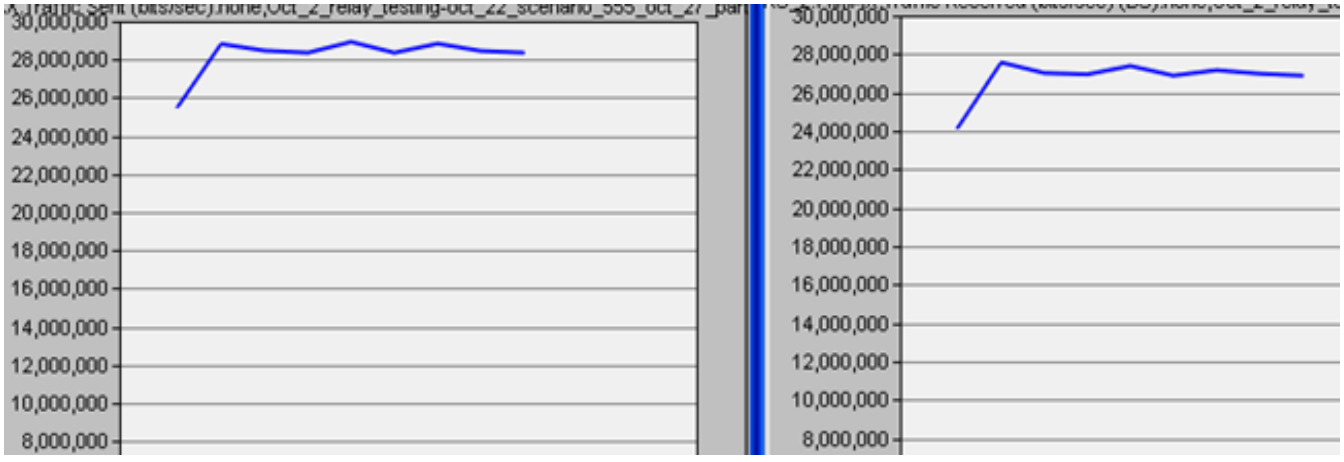


Fig. 5.17 – Traffic sent by SS vs traffic received by RS, scenario-7

Scenario 8: Service flows with QoS types: UGS, rtPS, BE (Every 10 SSs have a similar service flow)

Network size	1 BS, 6RSs and 30SSs
Service type	UGS/rtPS/BE
Uplink traffic demand	5/5/5Mbps
Frame division	75% downlink, 25% uplink

Bandwidth for **UGS** traffic Bandwidth for **rtPS** traffic Bandwidth for **BE** traffic

Fig. 5.18 – QoS traffic graphs scenario-8

Total traffic sent by **all SSs** (UGS, rtPS, BE) Total traffic received by **all RSs**

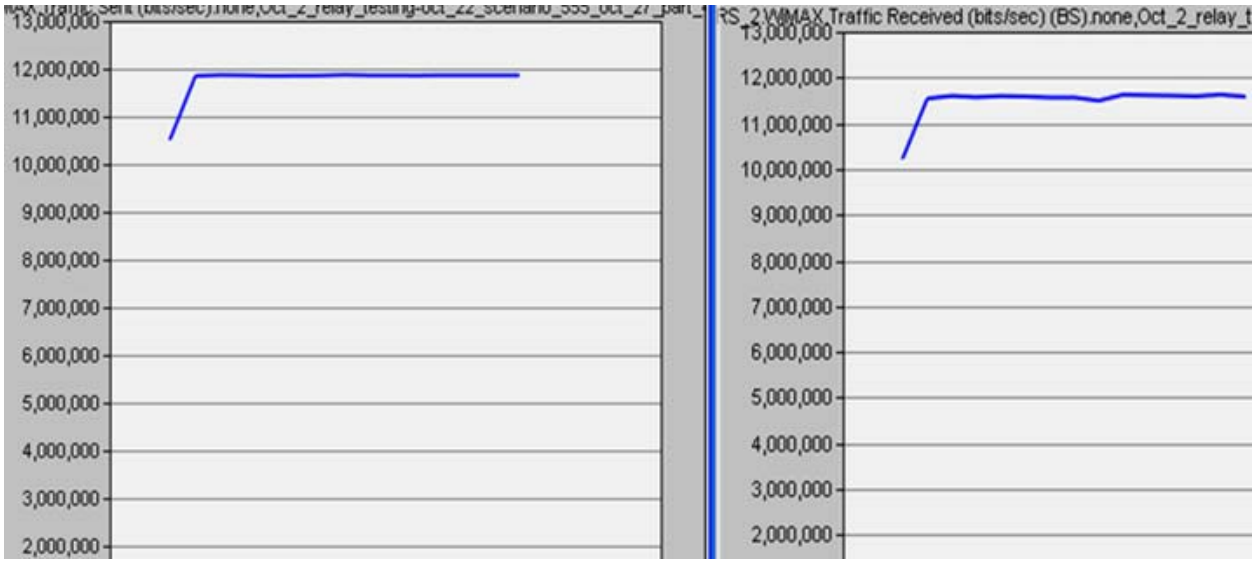


Fig. 5.19 – Traffic sent by SS vs traffic received by RS, scenario-8

5.3 Observation from the simulation results

The following observations were made from the simulation results:

1) In small network scenarios with only best effort traffic (Scenarios 1-2), for downlink traffic, we found the total traffic received at all the SSs is almost same as the traffic sent by the BS, throughput is about 93% (value obtained from graphs) over a 2-hop wireless network. Therefore, we conclude that using our algorithm and the smart antennas system, the WiMAX network can efficiently deliver all the packets without significant packet loss, overall packet loss is about 7% (value obtained from graphs) over a 2-hop network. For the uplink traffic also we can see similar behavior in the network.

   Note that we set the total uplink traffic demand to 28Mbps since we found this is the maximum achievable link data rate of the BS-RS link (Be advised that all the end-to-end connections need to share the wireless link in the first hop). In addition, the uplink traffic demand is only 1/8 of downlink traffic demand. According to statistics, in the current Internet, the total uploading traffic load is 1/8 of the downloading traffic load; hence by employing this ratio of uplink and downlink traffic load in our scenarios we can closely associate the observed behavior to real world Internet traffic.

In large network scenarios with only best effort traffic (Scenarios 5-6), similar observations can be made. In the corresponding network topology figures, the red lines show the RS assignment. Note that in these scenarios, we have 6 RSs and 30 SSs; however, the total traffic load remains the same as in scenarios 1-2. This is because the first hop forms the bottleneck and the total downlink throughput cannot go beyond 28Mbps no matter how many nodes you have in the network.

2) We also evaluated the WiMAX MAC protocol and our topology control algorithms in terms of QoS provisioning in the simulation. For small network scenarios (scenarios 3-4), for fair comparison, we only include one RS and three SSs, each of which generate traffic of a specific type. So, we have one SS using UGS flows, one SS using rtPS flows and remaining SS using Best effort service flow. We found that if there are enough resources (scenario 3), i.e., 75% of frame time is allocated for uplink, each SS obtains enough bandwidth to deliver all its packets no matter which type of traffic it generates. It is noticed that both rtPS and BE traffic obtain a bandwidth of 9.3Mbps that is equal to their respective demands, however, the bandwidth given for the traffic with highest priority, UGS, is only about 5Mbps. This is because the OPNET Modeler traffic scheduler sets an upper bound for UGS bandwidth per node to 5Mbps.

   However, in a scenario where there are not sufficient resources (scenario 4), i.e., only 25% of frame time is allocated for uplink, traffic with higher priorities (UGS, rtPS) obtains much higher bandwidth than lower priority traffic (BE), specifically, 5Mbps versus 1.2Mbps. The results match our expectation. In the corresponding large network scenarios (scenarios 7-8), we randomly divide 30 SSs into three groups, each of which generated a common type of traffic. Therefore, 10 SSs generated UGS traffic, 10 SSs generated rtPS traffic and all the others generated BE traffic. In terms of bandwidth allocation, we can make similar observations. However, we notice that in scenario 7, even UGS traffic obtains a bandwidth of 9.33Mbps (>5Mbps) because multiple RSs were used for packet forwarding and each of them was given a different frequency channel for operation.

3) We tested both topology control algorithms in large network scenarios; we found that they give slightly different RS assignments. However, in terms of network throughput, both of them lead to very similar results as they provide almost same assignments. The results presented are of a network topology with an RS assignment which was obtained after applying the algorithm-2.

Chapter 6

A Tree topology based Interference aware channel assignment algorithm using directional antenna for balancing the network load

The efficiency of every network is not determined only by parameters such as throughput or link capacity, but is dependent on the requirement and purpose of the network. In the wireless scenarios discussed in previous chapters the emphasis is on increasing the throughput of every node by maximizing the link capacity of each node. The SS-RS assignment is purely based on maximizing the minimum link capacity of the network, for a given beamwidth of the RS antenna.

This approach however doesn't take into consideration issues like interference, network load balance, traffic congestion and network stability. There may be certain wireless networks which are deployed to address the above issues seriously. Taking care of these issues would improve overall network performance like improving throughput and reducing the packet delay.

When the behavior and demands of all the nodes is known and similar to each other, then there needs to be an optimal and fair approach, while trying to minimize the effect of obvious problems like interference, resource allocation and network load balance and reduce traffic congestion.

6.1 System model

- WiMAX multi-hop relay network scenario.
- Network has low mobility and the relative positions of the nodes in the network remains constant (almost) or changes infrequently.
- The SS nodes in the scenario are generally distributed in a dense and uniform way, where most SS radios can choose to associate to more than one BS radio. This scenario typifies sensor nodes floating in a sea with many ships within its communication range.
- BS selects and grants the SS nodes association for communication.
- The traffic demand pattern is constant and all the SS nodes have almost equal demands.
- The uplink to downlink demand ratio is 3:1

- BS and SS are end-to-end entities (end-points of a service link)
    - Root node of the tree structure would be the BS which should be accessible to its client SSs directly or via relay nodes.
- A relay node can be accessed by an SS
- Each relay node is assumed to have two directional radio:
    - 1 BS radio
    - 1 SS radio
- Two BS radios and two SS radios can't communicate with each other.
- Each node uses the best possible antenna orientation pointing towards its parent and set of children.
- The antenna beamwidth of all radios may differ.
- The nodes should use the same channel for communication as assigned by its parent.
- If the network topology changes, the entire procedure can be re-run for the newly configured network.

6.2 Problem definition

- Given a wireless network(WiMAX) scenario,
    - A static tree topology is constructed such that every node in the network has a path to the root (BS) directly or via relay nodes.
    - Traffic load is balanced among the nodes in the network for given traffic demands, to control the traffic congestion at each hop.
    - Channels are assigned to each node for communication to minimize the co-channel interference and the antenna orientation and beamwidth is set for each node such that the interference traffic is minimized while achieving the maximum end-to-end throughput.

6.3 Algorithms/protocols

BFS based tree construction algorithm:

- Base station is inserted in a queue and is checked.
- It becomes the PARENT of a neighbor node, if that neighbor node is within the 'fixed allowed range' and is not yet marked (has no PARENT yet).
- If the CHILD node is a SS, just mark it.
- All relay nodes are pushed into the queue and are further popped out for finding its neighbors.
- We do this until the queue is empty. (That is, all the nodes reachable by distance of 'range' are marked)
- If any node is left unmarked, associate with the nearest BS radio. (This might be out of the 'fixed range' from all the BS_radios).

Time complexity of this is O (|E| + |V|).

Bottom-Up Hop-wise Node Balancing method (BHNB):

This method is used to make the final assignment of parent to child nodes. Each node is assumed to have similar traffic demands, so equally distributing the nodes to each parent would ensure the balancing of traffic load.

For a child node, we consider the switching of its parent within the same hop-level, such that it doesn't select a parent from different hop-level in the tree. This is to ensure the consistency in the number of hops for each node.

At each hop,

- We find out the number of CHILD_NODES and number of PARENT_NODES.
- We maintain variables called CP_ceil and CP_floor.
- CP_ceil = ceil(number of CHILD_NODES / number of PARENT_NODES)
- CP_floor = floor(number of CHILD_NODES / number of PARENT_NODES)

- If any parent has a total number of child nodes greater than CP_ceil, then we try to assign the extra child nodes (total children – CP_floor) to some other parent.
- This switching can be done only if the new parent can offer 'connectivity_value' of at least 75% of the previously associated BS. The 'connectivity_value' can be any parameter like signal strength.
- Since we have assigned few SS nodes to a new parent, this new parent shall also check if the total number of child nodes is greater than CP_ceil.
- We keep track of a node's previous parent, such that it doesn't again associate with the previous parent.

This is repeated at each hop, from bottom to up till the root node is reached.

Time complexity of this is O (n* n * logn).

Interference aware Least Used channel allocation method (IALU):

First, we sequentially assign all of the 'M' available channels from the frequency spectrum, until the 'M' links are exhausted.

Then for the link in question, select a channel, which would already in use for at least one link, we consider the following criteria in order:

- Select a set of channels whose cone of interference [14] doesn't overlap with the current links
    - From that set select a channel which is Least Used among the set
- If the cones overlap, then select a set of channel with less overlap area among the others
    - Select a channel whose traffic load is minimized in the set, hence reducing the interference traffic.

6.4 OPNET Modeler Implementation

- Simulations of these algorithms done with OPNET Modeler 14.5 software, includes standard and customized models for SS, BS and relay nodes.
- WiMAX multi-hop relay network scenario, covering an area of 2 km X 2 km.
- Traffic demands for every node are considered the same.

- Settings for the simulations is as follows:

Parameter	Value
Omni-directional antenna gain	2 dBi
Directional antenna beam width --- 90 degrees	14 dBi (approx.)
Directional antenna beam width--- 45 degrees	20 dBi (approx.)
Directional antenna beam width--- 5 degrees	40 dBi (approx.)
Request packet size	1500 Bytes
Response packet size	500 Bytes
Requests per second	10
Transport protocol	UDP
SS transmitter power (W)	0.1
BS transmitter power (W)	0.158
Number of channels	5
Spectrum band	5.0 GHz
Bandwidth of a channel	20 MHz
Duplexing technique	TDD
UL/DL frame boundary	75 % (UL) – 25% (DL)

Fig. 6.1 – Simulation settings and parameter values.

6.5 Results

For the small scenario-A, we got the following results.

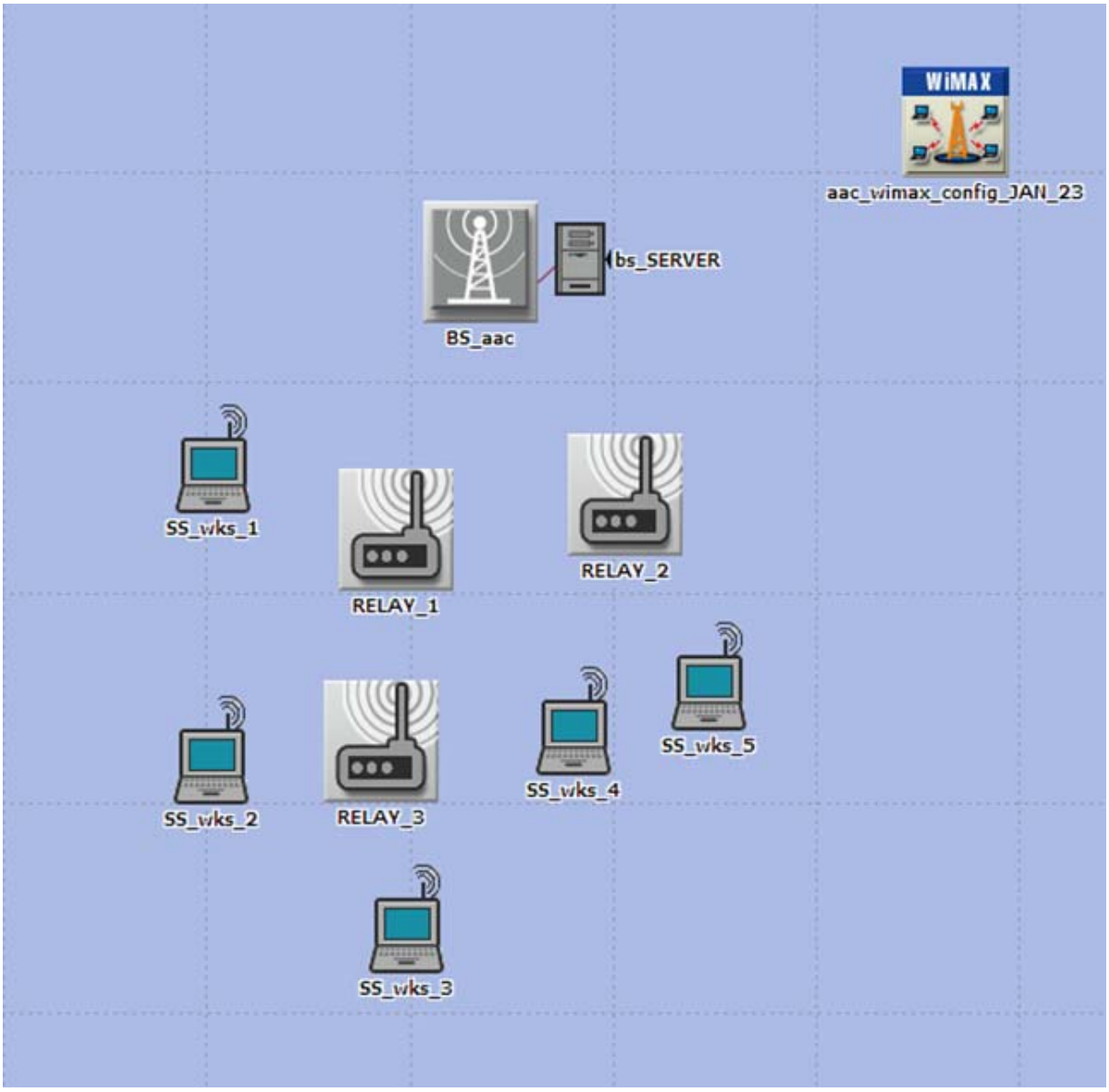


Fig. 6.2 - shows a small multi-hop scenario with 1 BS, 3 RSs and 5 SSs.

The **Total Throughput of BS** without implementing the proposed algorithm. This scenario follows the general approach of the 802.16 e standard.

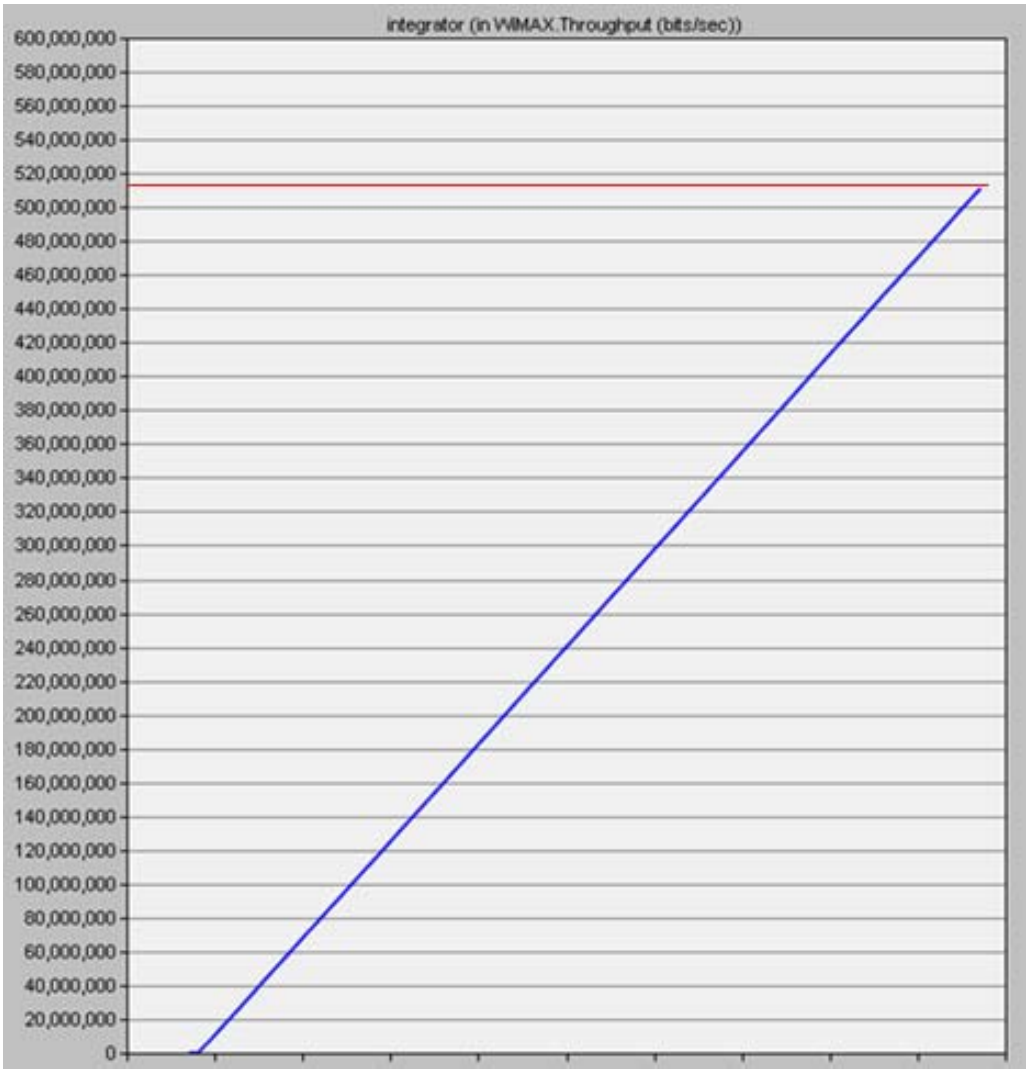


Fig. 6.3 – Total Throughput (BS) without proposed algotihms

The above figure shows a graph of total BS throughput when the network simulation was run for a period of 15 minutes. Here, the network parameters were set and configured as per the 802.16e standard. The SS and RS resource scheduling was done based on distance (nearest) between the radios and no node balancing method was employed for constructing the topology. Also, no channel assignment algorithm was used and all the links were operating on same frequency channel. All the radio antennas were omni-directional.

In this case, the value of total BS throughput over a period of 15 minutes is about 510Mb.

The **Total Throughput of BS** after implementation of BFS+BHNB and IALU methods with directional antennas.

Fig. 6.4 – Total Throughput (BS) after implementing proposed algotihms

The above figure shows a graph of total BS throughput when the network simulation was run for a period of 15 minutes. Here, the network parameters were set and configured after implementing BFS based tree construction algorithm and our proposed BHNB algorithm for constructing the network topology. In this case the number of nodes at each level was balanced and since we have equal traffic demand for each SS node, the traffic load was balanced. This provided a more stable and balanced topology contributing to better performance (higher throughput as we can see). Also, our proposed IALU algorithm was used for channel assignment, for providing more resources and reducing the co-channel interference. All the antennas used were directional antennas.

In this case, the value of total BS throughput over a period of 15 minutes is more than 540Mb. So, in this small network using our proposed algorithms and approach the total BS throughput was improved by about 30Mb in network communication simulated over a period of 15 minutes.

The **Total Delay** without implementing the proposed algorithm. This scenario follows the general approach of 802.16 e standard.

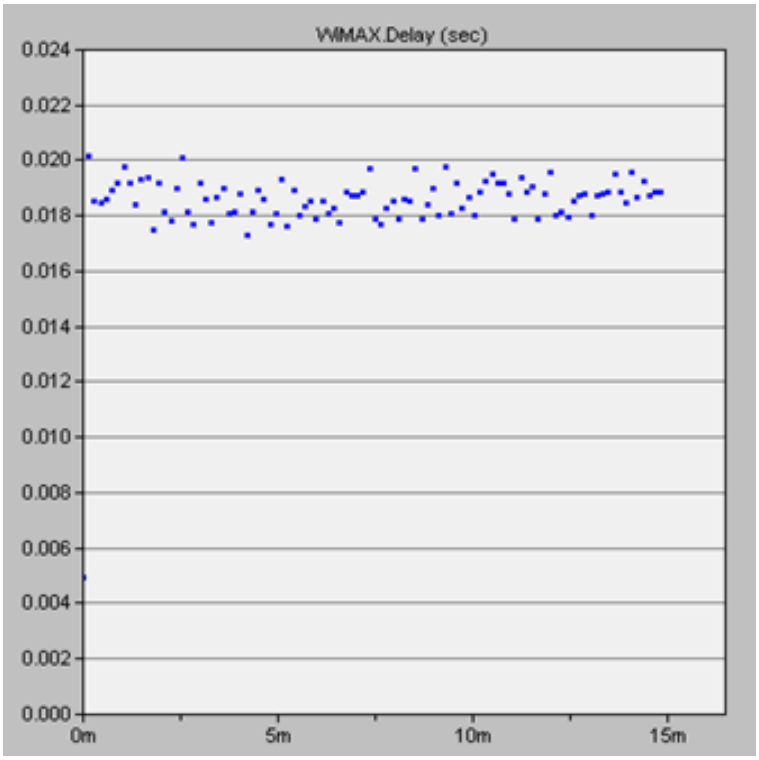


Fig. 6.5 – Total Delay (average) without implementing proposed algotihms

The above figure shows a graph of total network delay (average) when the network simulation was run for a period of 15 minutes. Here, the network parameters were set and configured as per the 802.16e standard.

In this case, the network delay ranges from 18ms to 20 ms.

The **Total Delay** after implementation of BFS+BHNB and IALU methods with directional antennas.

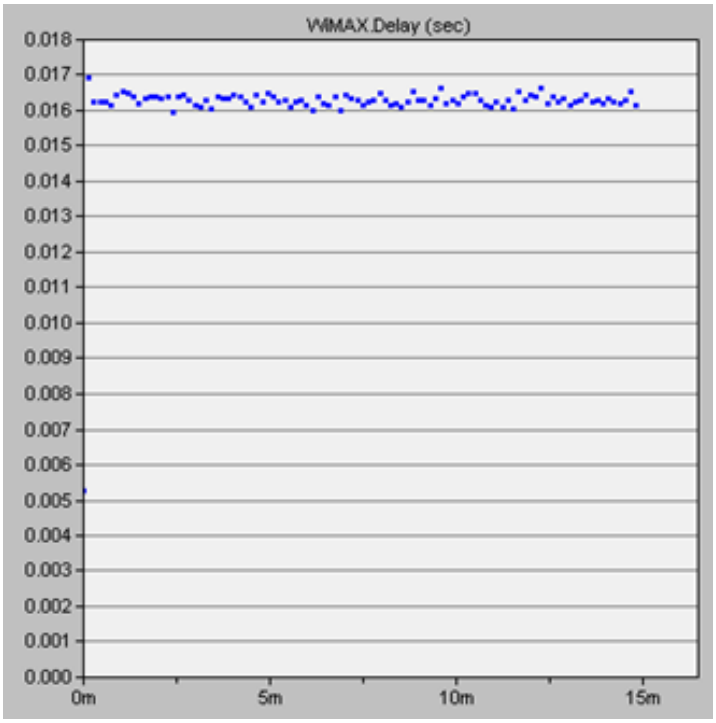


Fig. 6.6 – Total Delay (average) after implementing proposed algotihms

The above figure shows a graph of total BS throughput when the network simulation was run for a period of 15 minutes. Here, the network parameters were set and configured after implementing BFS based tree construction algorithm and our proposed BHNB algorithm for constructing the network topology, along with IALU algorithm using directional antennas.

In this case, the network delay is less than 16.5ms on an average which is less than delay of 19ms on average when standard (802.16e) approach and configuration was used.

The **Instantaneous Throughput of BS** without implementing the proposed algorithm. This scenario follows the general approach of 802.16 e standard.

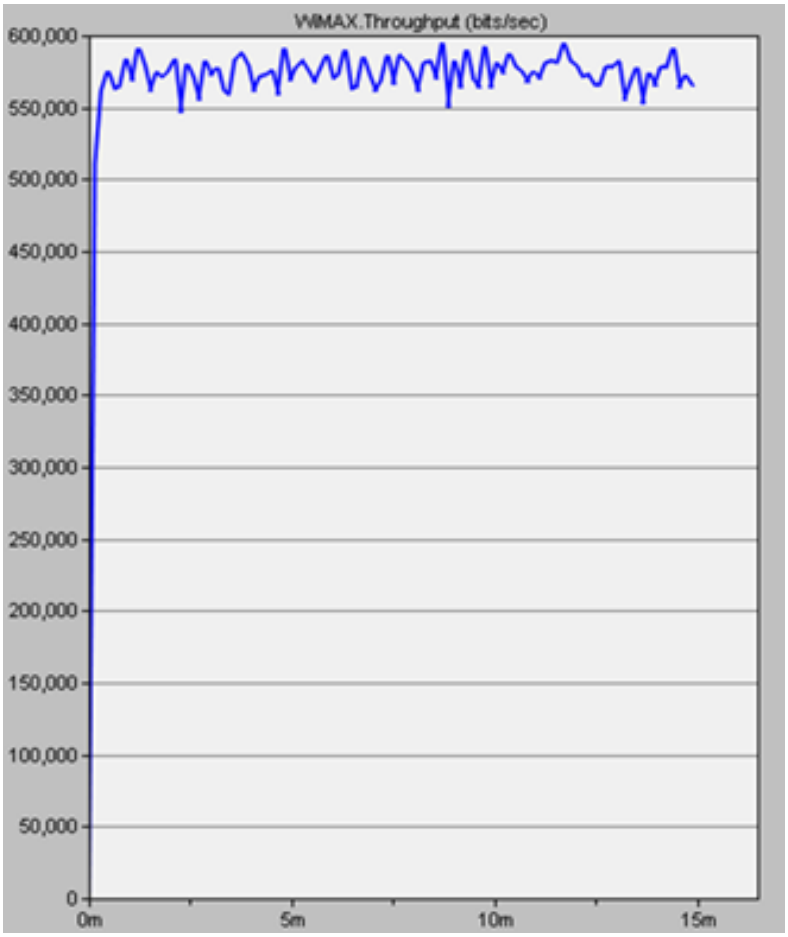


Fig. 6.7 – Instantameous Throughput (BS) without implementing proposed algotihms

The instantaneous throughput of the BS is about 5.7 Mbps with the 802.16e standard approach, not using the proposed algorithms and methods.

The **Instantaneous Throughput of BS** after implementation of BFS+BHNB and IALU methods with directional antennas.

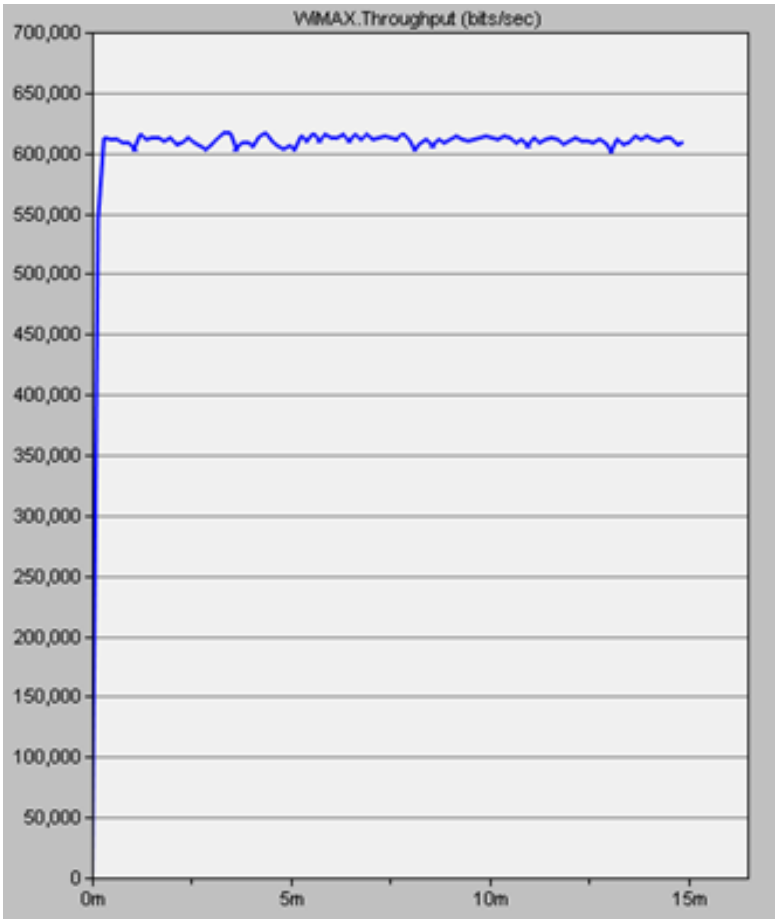


Fig. 6.7 – Instantameous Throughput (BS) after implementing proposed algotihms

The instantaneous throughput of the BS is about 6.1 Mbps after implementing the proposed BFS based tree construction and BHNB algorithms and using IALU algorithm with directional antennas. There is a gain of about 0.4 Mbps in BS throughput for a same given traffic demand value.

For the bigger scenario-B, we go the following results.

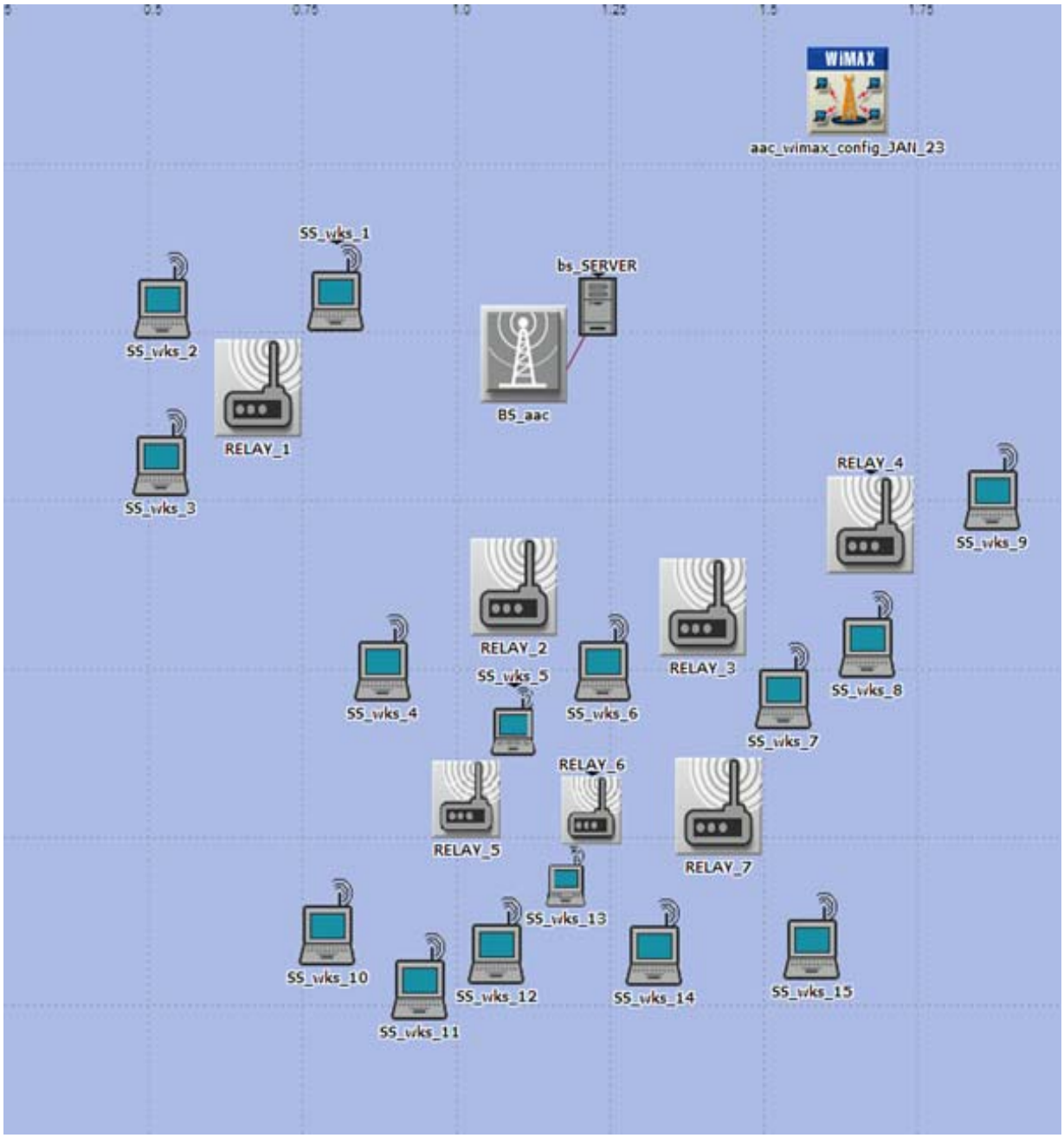


Fig. 6.8 - shows a larger scenario with 1 BS, 7 RSs and 15 SSs.

The efficiency and effectiveness of the proposed BFS based tree construction algorithm and BHNB algorithm is more evident in larger scenarios where there are several hops and nodes at

each hop. In the above scenario where we have about 7 RSs and 15 SSs, the implementation of the proposed algorithms shall yield much different and efficient network topology.

In the figure below we have a 5-hop service link from BS to an SS. The node *SS_wks_15* need 5 hops to reach the BS. There are several such 5-hops links in the network when only 802.16e standard approach is used for resource allocation and topology constructions.

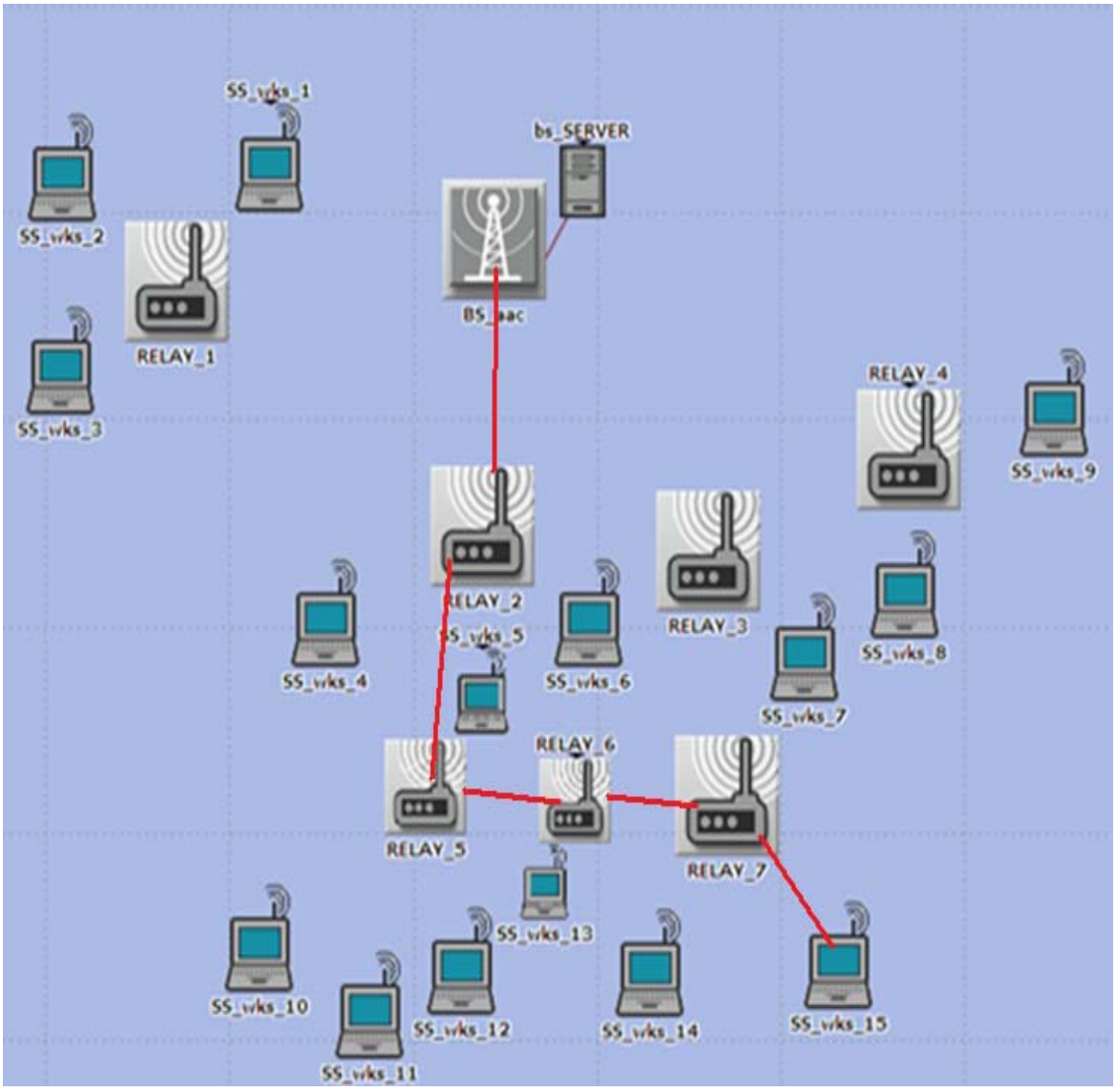


Fig. 6.9 - shows a service link in the topology construction for the given scenario using standard approach (connecting to nearest radio).

In the below figure, service links for the same node as in the above figure are shown. Our proposed BFS based topology construction algorithm was implemented, now node *SS_wks_15* can reach the BS in 3 hops.

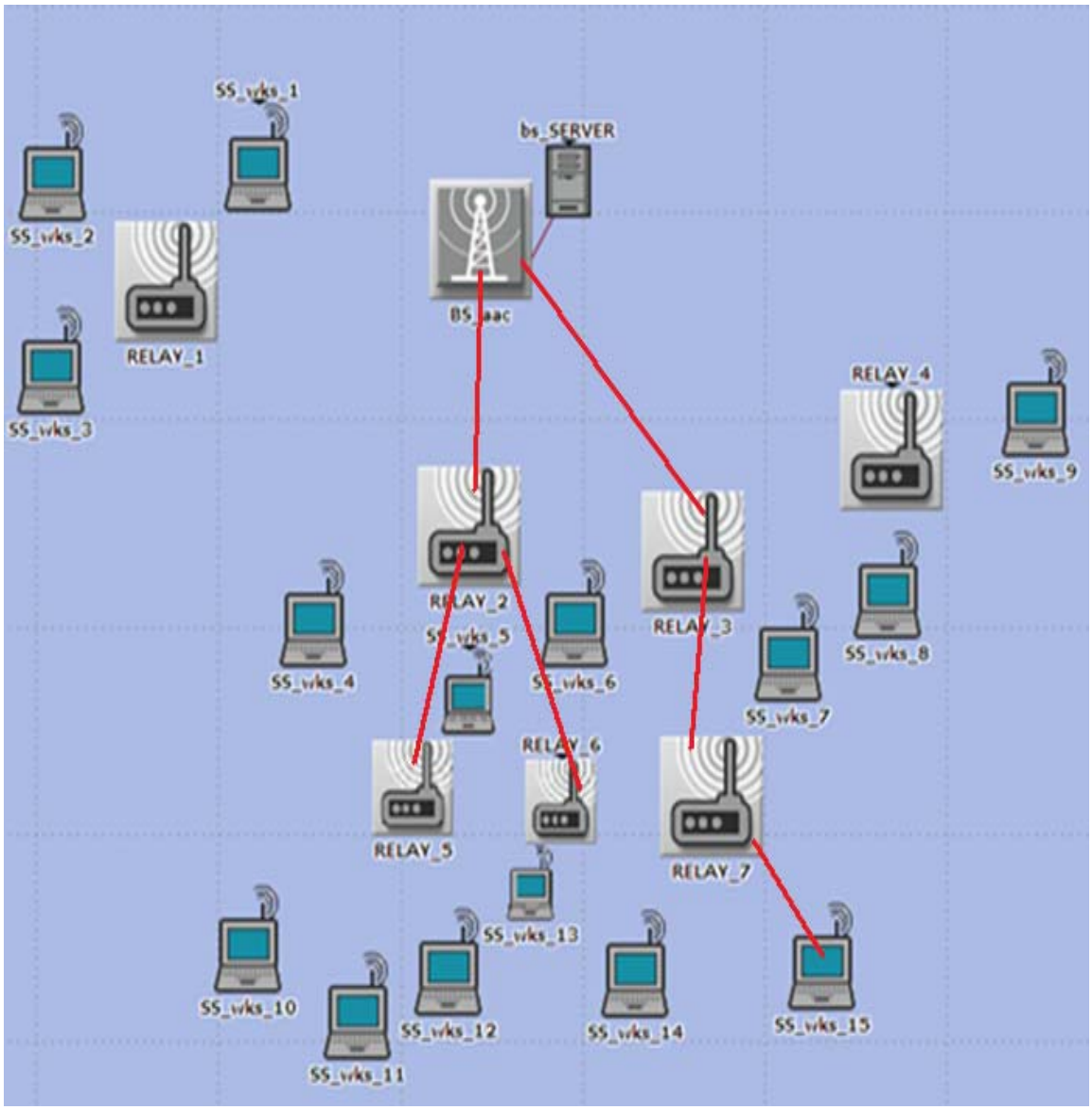


Fig. 6.10 - shows the service links (for same nodes as in Fig. 6.9) in the network topology for the given scenario using BFS based tree construction algorithm.

The effect of BFS based tree construction algorithm is that the number of hops in the network is reduced:

The number of nodes with 5 hops = 0; previous value (with 802.16e standard approach) = 3.

The number of nodes with 4 hops = 0; previous value (with 802.16e standard approach) = 4.

In the figure below we can see that there are 5 child node for *Relay_2* and 4 child nodes for *Relay_5*. Here, BHNB algorithm for node balancing is not implemented.

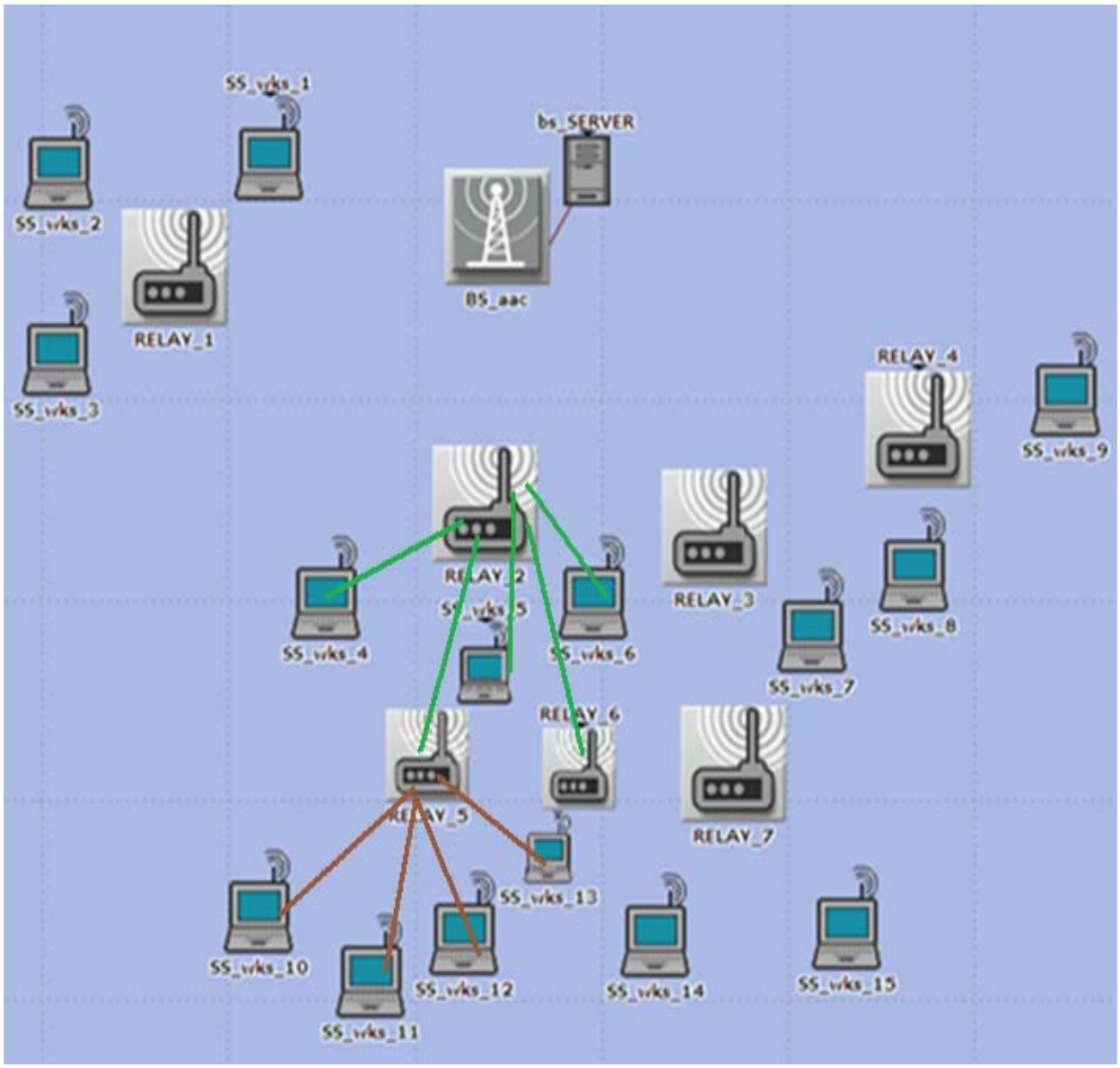


Fig. 6.11 – Resource allocation schedule for *Relay_2* and *Relay_5* without implementing BHNB algorithm.

In the figure below we can see that there are 3 child node for *Relay_2* and 3 child nodes for *Relay_5*. Here, BHNB algorithm for node balancing is implemented and hence at each hop number of child nodes is tried to be distributed evenly to the parent nodes giving a more stable and balanced network topology.

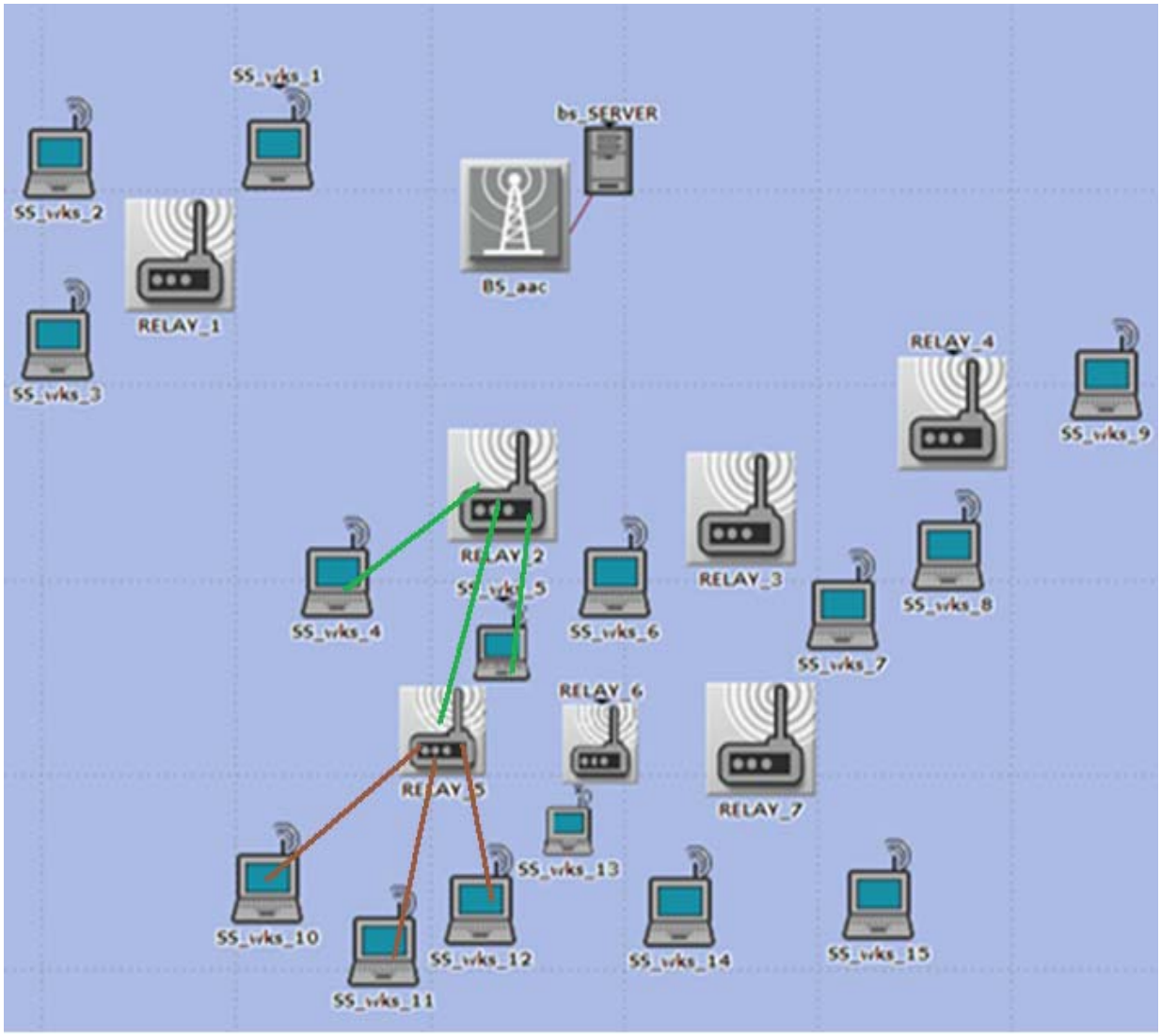


Fig. 6.12 – Resource allocation schedule for *Relay_2* and *Relay_5* after implementing BHNB algorithm.

The effect of BHNB algorithm for balancing the network load on the large network scenario is as follows:

BHNB minimizes the maximum number of child nodes of a parent in the network from 5 to 3. At each hop the number of child nodes and balanced.

At hop 2:

- Relay_1 has 2 child nodes; previous value (without implementing BHNB) = 2
- Relay_2 has 3 child nodes; previous value (without implementing BHNB) = 5
- Relay_3 has 3 child nodes; previous value (without implementing BHNB) = 3
- Relay_4 has 3 child nodes; previous value (without implementing BHNB) = 1

At hop 3:

- Relay_5 has 2 child nodes; previous value (without implementing BHNB) = 4
- Relay_6 has 2 child nodes; previous value (without implementing BHNB) = 1
- Relay_7 has 2 child nodes; previous value (without implementing BHNB) = 1

Results for the larger scenario (scenario-B) with 7 RSs and 15 SSs also show similar effect and efficiency of our proposed tree construction algorithms and channel assignment algorithms. The comparison of network parameters with 802.16e standard only approach and while implementing the proposed algorithm show that the efficiency of the network is improved. The graphs for throughput and delay parameters are as follows.

The **Total Throughput of BS** without implementing the proposed algorithm. This scenario follows the general approach of 802.16 e standard.

Fig. 6.13 – Total throughput (BS) for large scenario without implementing BFS+BHNB and IALU methods with directional antennas.

The **Total Throughput of BS** after implementation of BFS+BHNB and IALU methods with directional antennas.

Fig. 6.14 – Total throughput (BS) for large scenario after implementing BFS+BHNB and IALU methods with directional antennas.

The **Total Delay of BS** without implementing the proposed algorithm. This scenario follows the general approach of 802.16 e standard.

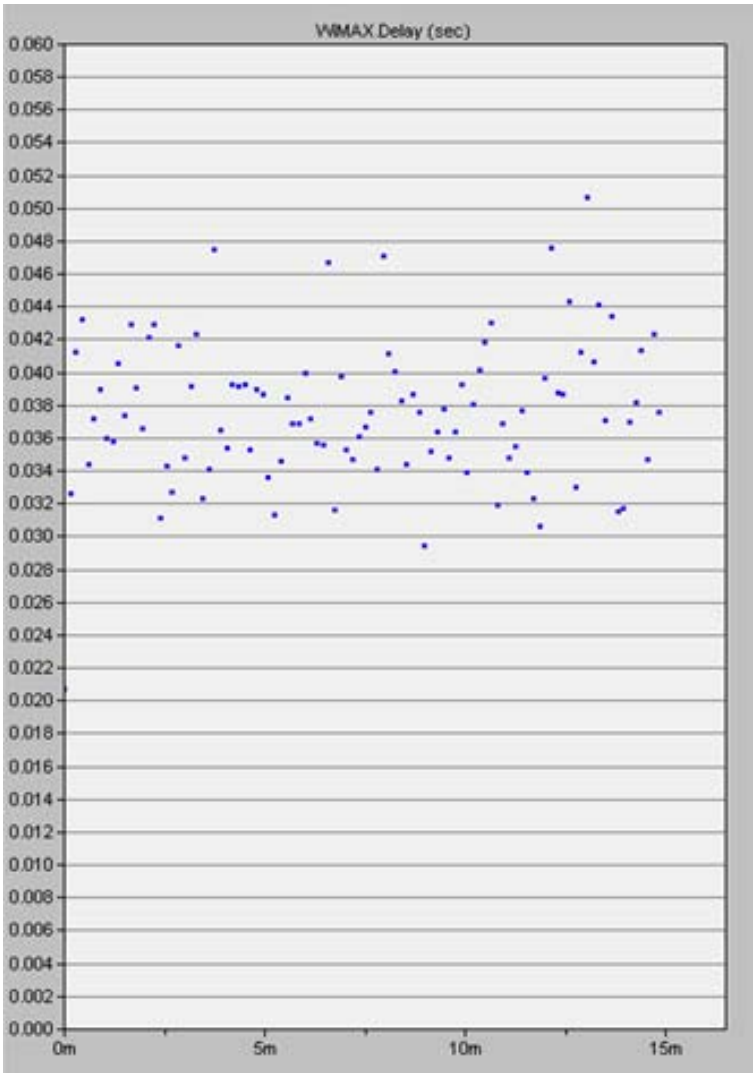


Fig. 6.15 – Total Delay (BS) for large scenario without implementing BFS+BHNB and IALU methods with directional antennas.

The **Total Delay of BS** after implementation of BFS+BHNB and IALU methods with directional antennas.

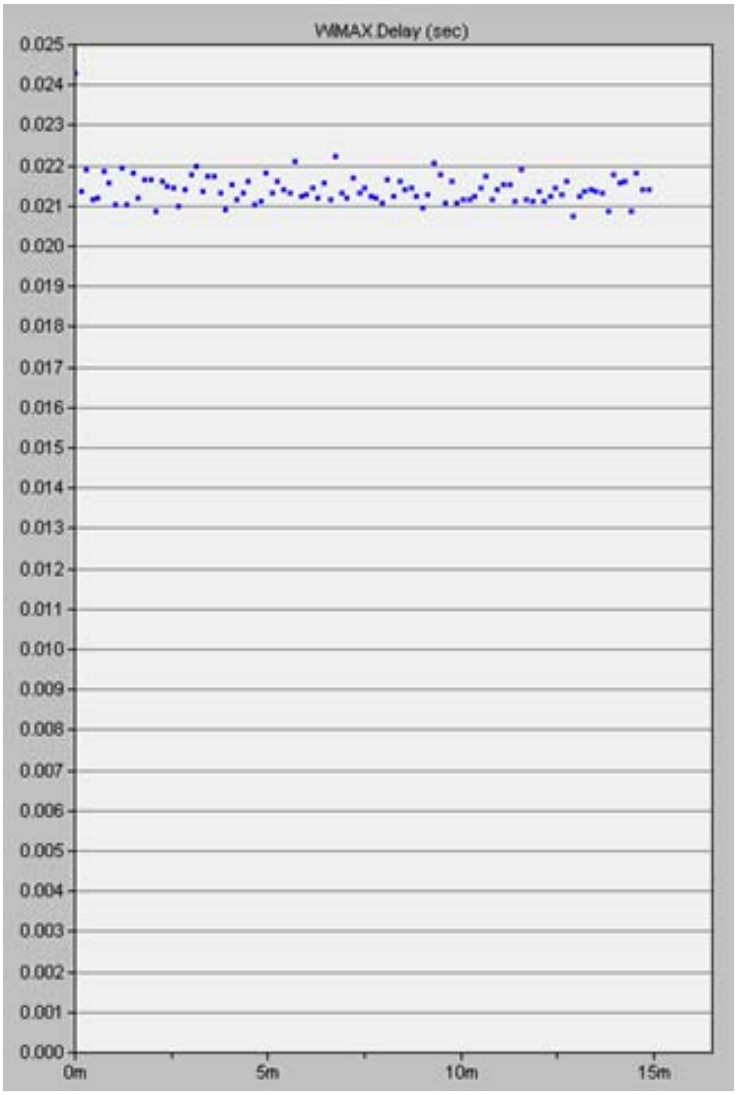


Fig. 6.16 – Total Delay (BS) for large scenario after implementing BFS+BHNB and IALU methods with directional antennas.

The **Instantaneous Throughput of BS** without implementing the proposed algorithm. This scenario follows the general approach of 802.16 e standard.

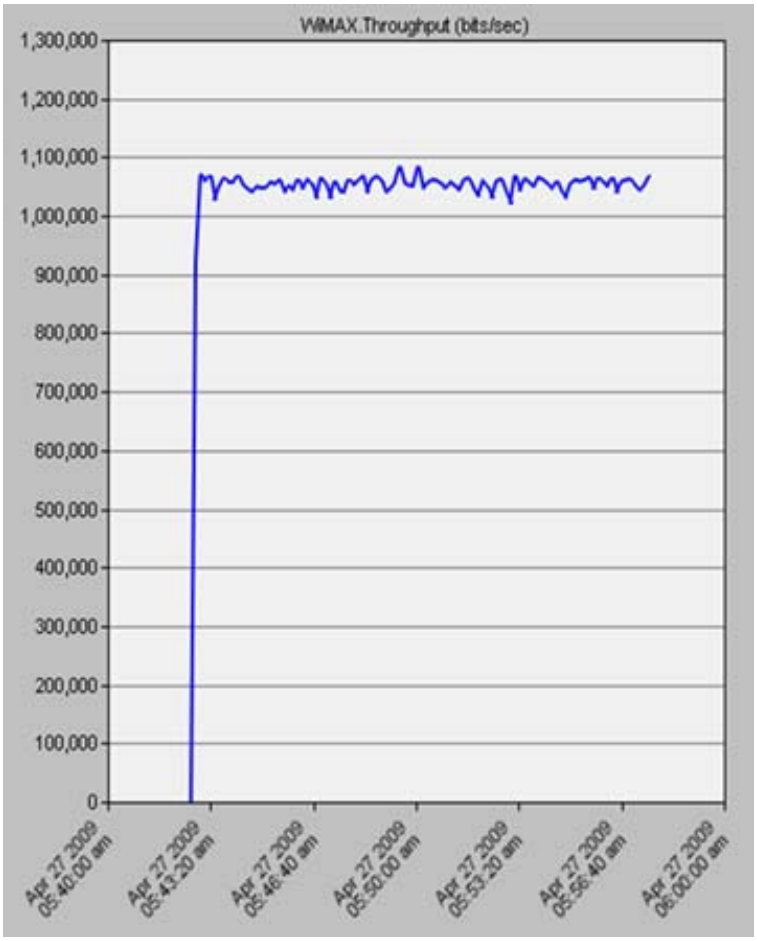


Fig. 6.17 – Instantaneous Throughput (BS) for large scenario without implementing BFS+BHNB and IALU methods with directional antennas.

The **Instantaneous Throughput of BS** after implementation of BFS+BHNB and IALU methods with directional antennas.

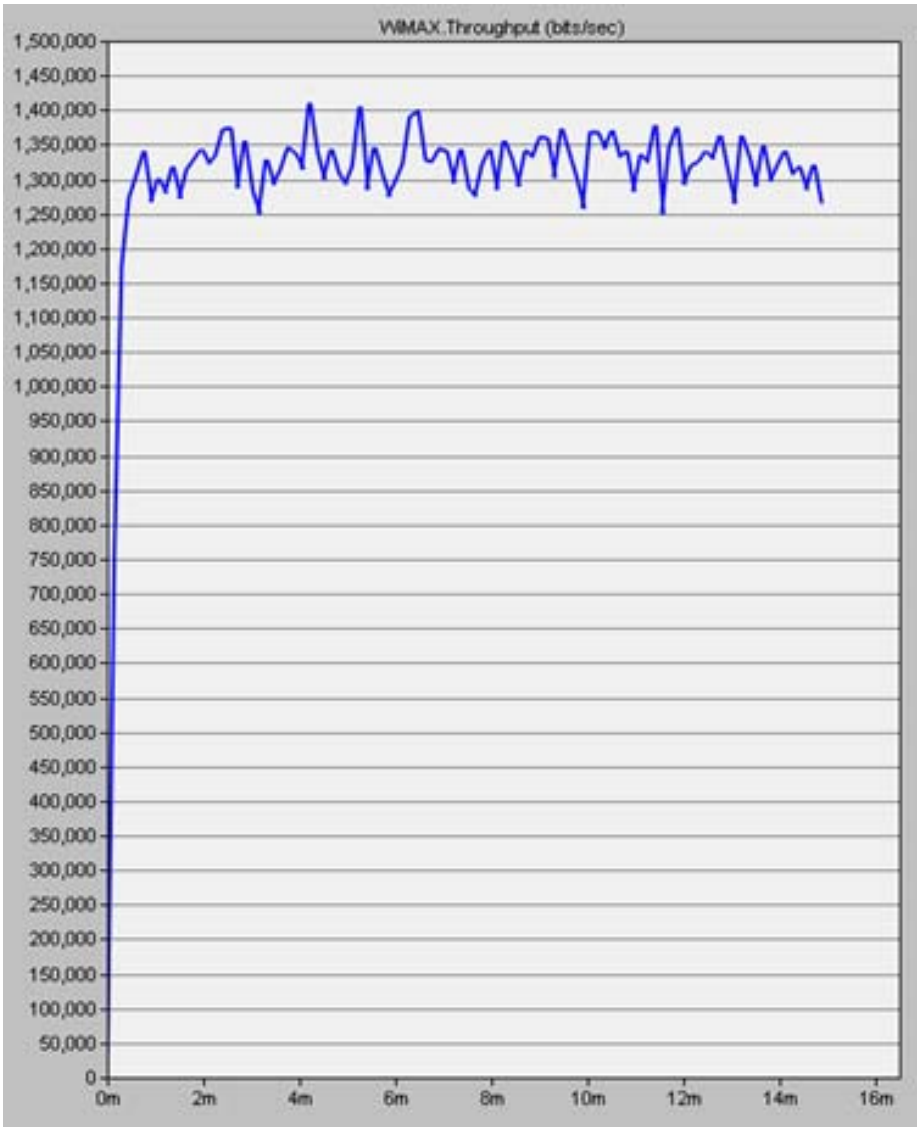


Fig. 6.18 – Instantaneous Throughput (BS) for large scenario after implementing BFS+BHNB and IALU methods with directional antennas.

The **Total Delay** without implementing the proposed algorithm. This scenario follows the general approach of 802.16 e standard.

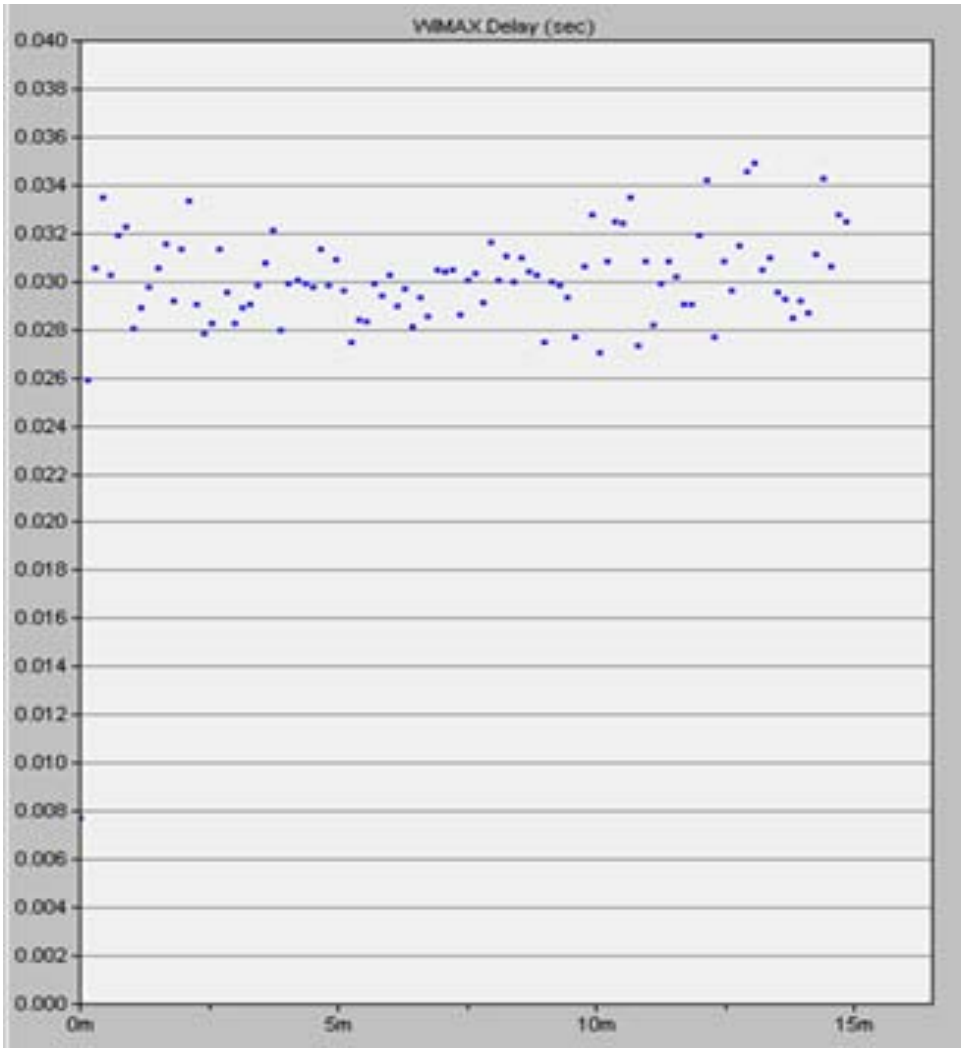


Fig. 6.19 – Total Delay (Network) for large scenario without implementing BFS+BHNB and IALU methods with directional antennas.

The **Total Delay** after implementation of BFS+BHNB and IALU methods with directional antennas.

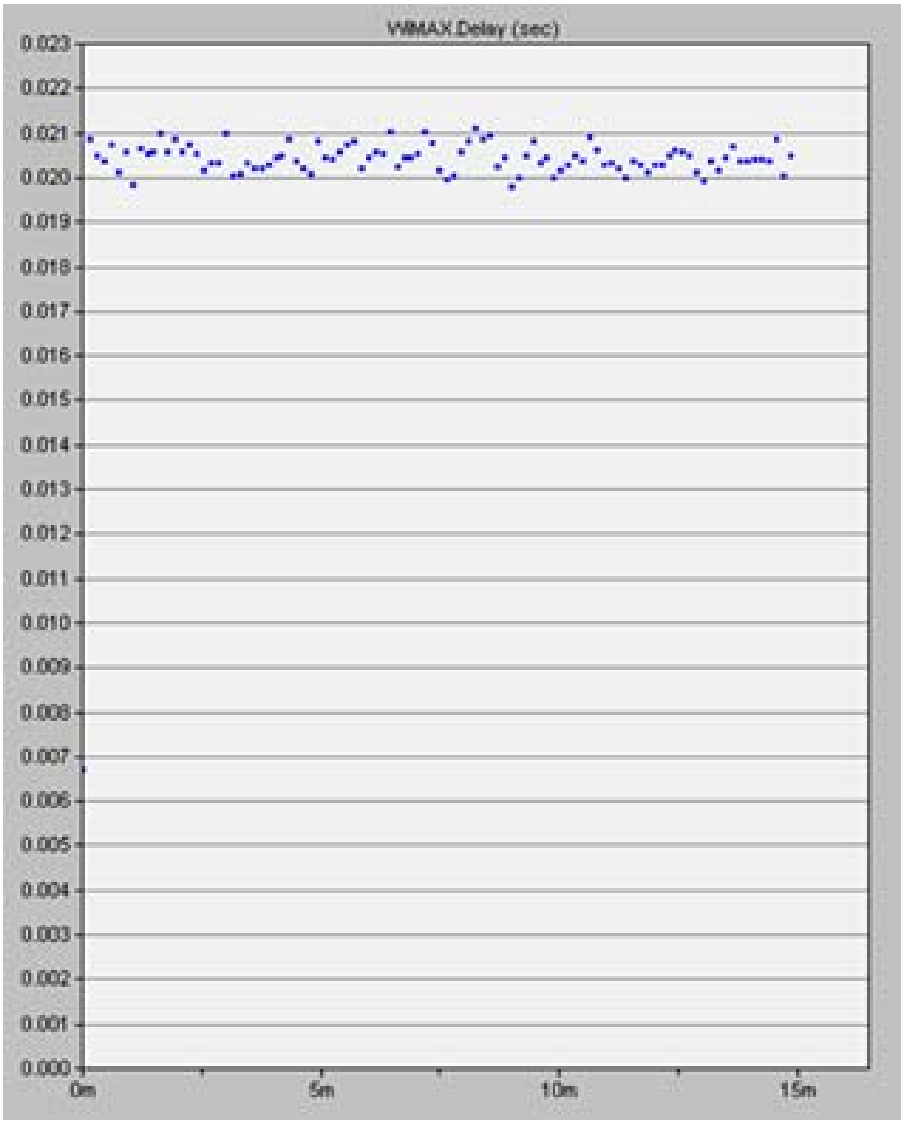


Fig. 6.20 – Total Delay (Network) for large scenario after implementing BFS+BHNB and IALU methods with directional antennas.

The **Average Throughput of BS** without implementing the proposed algorithm. This scenario follows the general approach of 802.16 e standard.

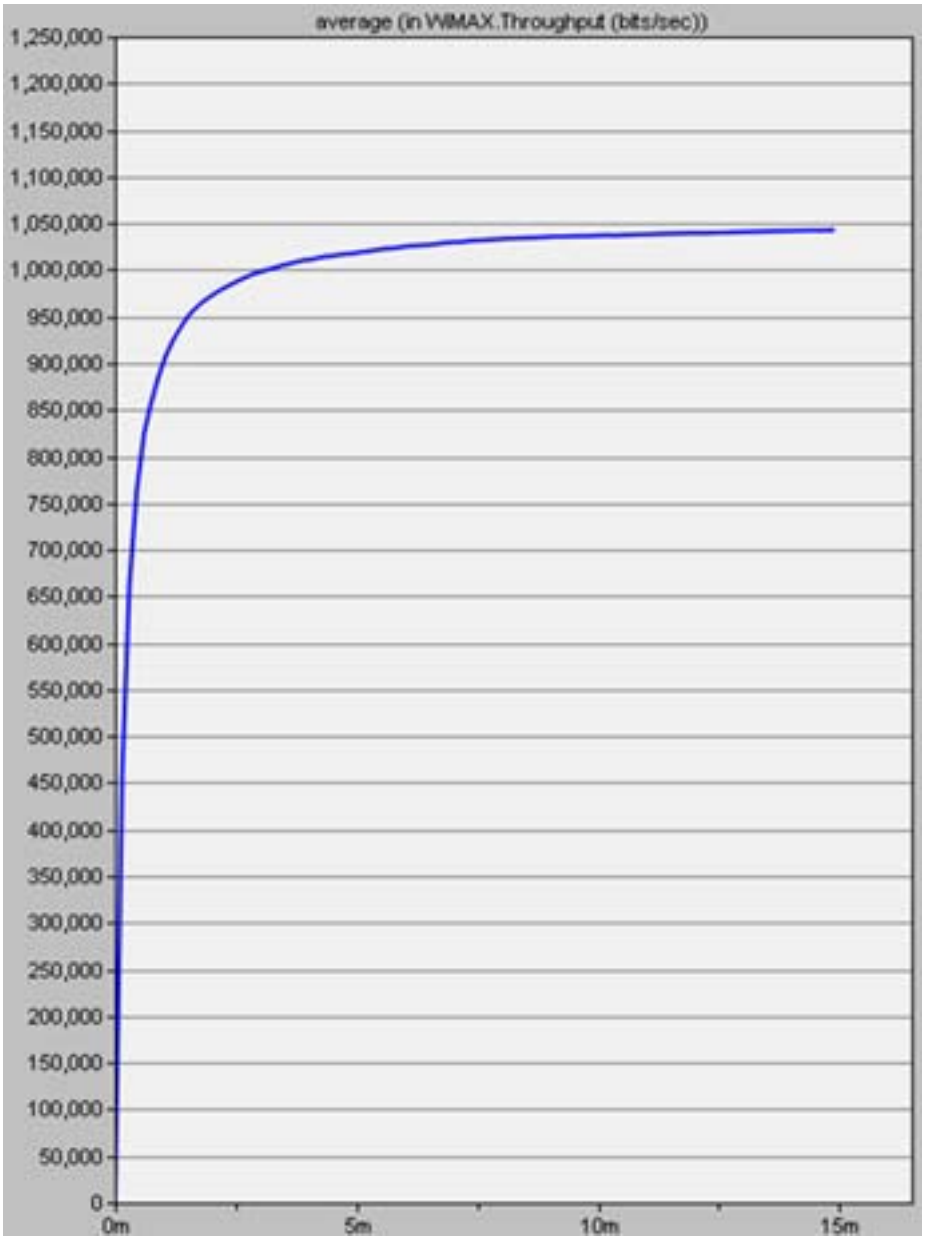


Fig. 6.21 – Average Throughput (BS) for large scenario without implementing BFS+BHNB and IALU methods with directional antennas.

The **Average Throughput of BS** after implementation of BFS+BHNB and IALU methods with directional antennas.

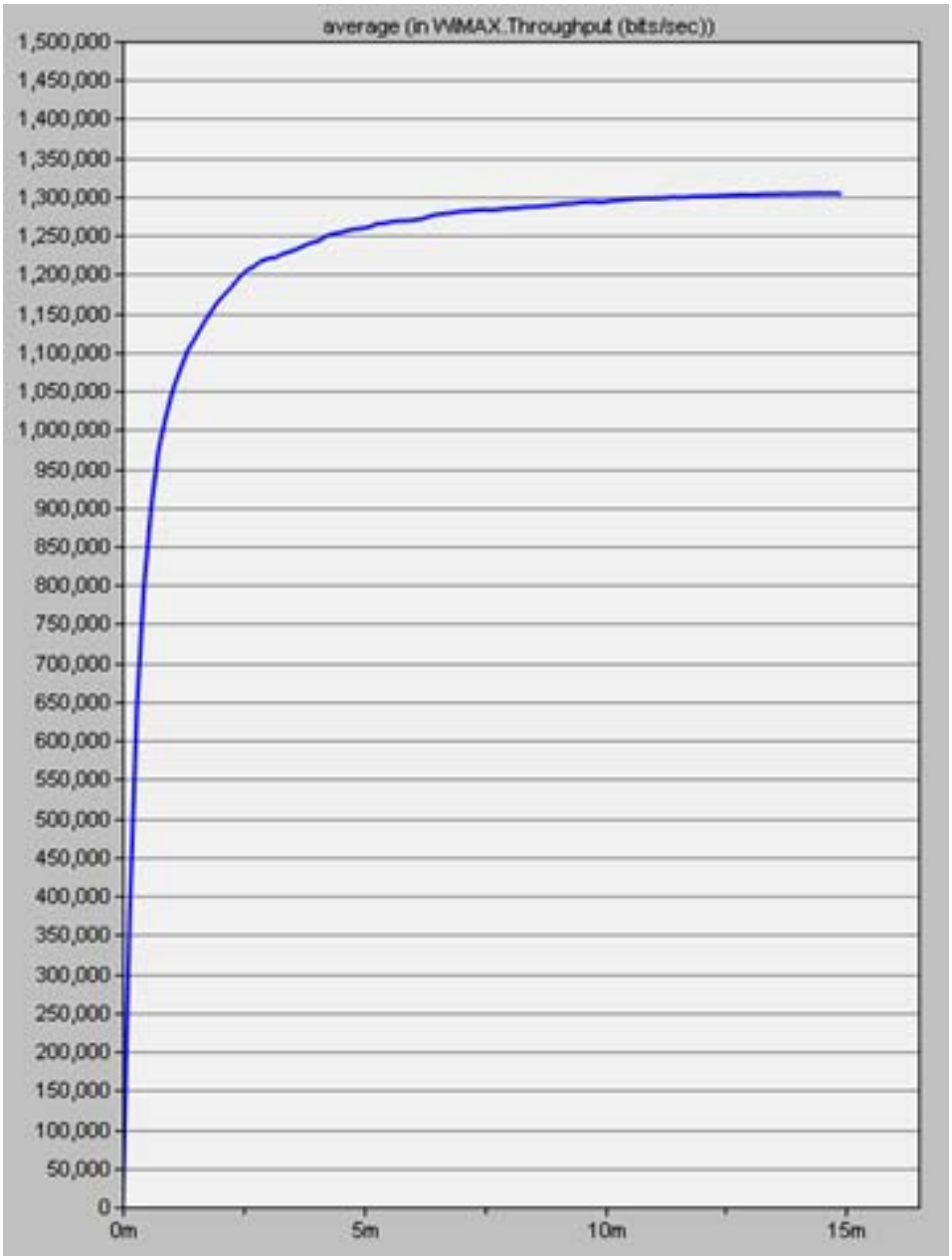


Fig. 6.22 – Average Throughput (BS) for large scenario after implementing BFS+BHNB and IALU methods with directional antennas.

Chapter 7

Network initiated handover or "forced handover" and handover in a multi-hop scenario implementing in an RS

7.1 Network initiated handover

In a WiMAX network, generally a handover is carried out when either a BS or an SS foresee the need of doing it. When a mobile SS moves and tends to go out of the communication range of its current serving BS, it is tried to handover the SS to another BS which can serve it. Handover may also happen if a BS is no longer able to serve any of its SSs or meet their traffic requirements. In these cases handover depends on issues like the mobility of the node, signal strength, QoS and availability of BS resources in the network. It is directly related to the issues like connectivity and QoS requirements of either the SS or the BS. But in some wireless mobile networks there may a requirement to process handover even when the SSs and BSs have no connectivity or QoS issues and they continue to function and communicate normally. Thus, the handover may be initiated and controlled by the network itself [8].

This network initiated handover may be implemented on static nodes also, where the SS and the BS in consideration both don't move but due to the influence of other nodes may force the handover.

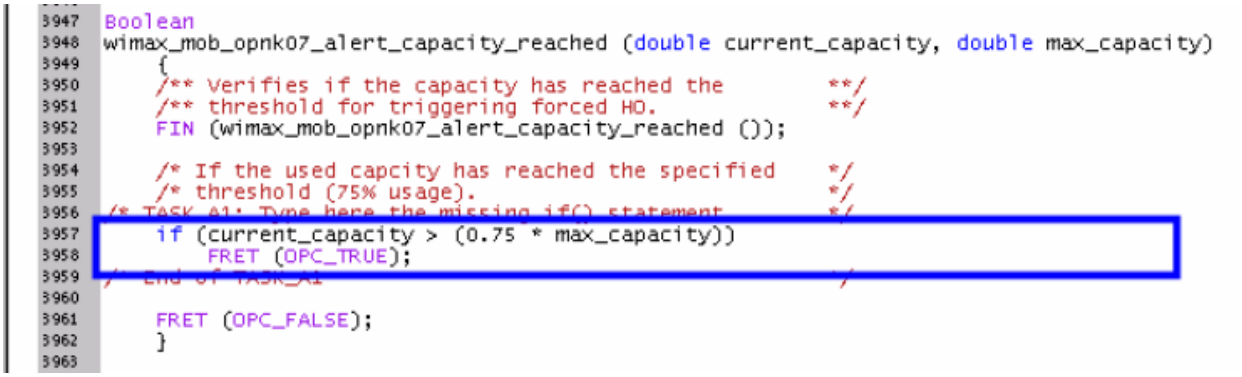
This handover is then called network initiated handover, and this implementation depends on the needs of the network and its resources and could be vendor specific. For instance, there may be a network where the BS to which an SS needs to report back might change over a period of time even though it is normally able continue to report to the current BS. Here, neither the BS nor the SS would see any need for handover and would also not know how to incur this desired change in communication.

Hence, it is the responsibility of the network to initiate the handover and provide the guidance to the relevant nodes in the network. For this, there is a need to have a central control in the network to carry out the required procedure and to maintain consistency and stability in all the nodes that would be part of this procedure or be affected by its consequences.

In a WiMAX network, where there are many BSs and a set of SS nodes which lie in communication range of more than one BS this forced handover can be implemented. The WiMAX network in consideration has few BSs and some mobile and non-mobile SSs. Here, the basic goal and requirement of the network would be to balance the network load on the BSs. The network may not prefer to load the major share of network traffic on just one BS when this can possibly be shared by other BSs. A plausible constraint would be that if a BS is serving more than certain number of SSs or handling more than certain amount of traffic or percentage of

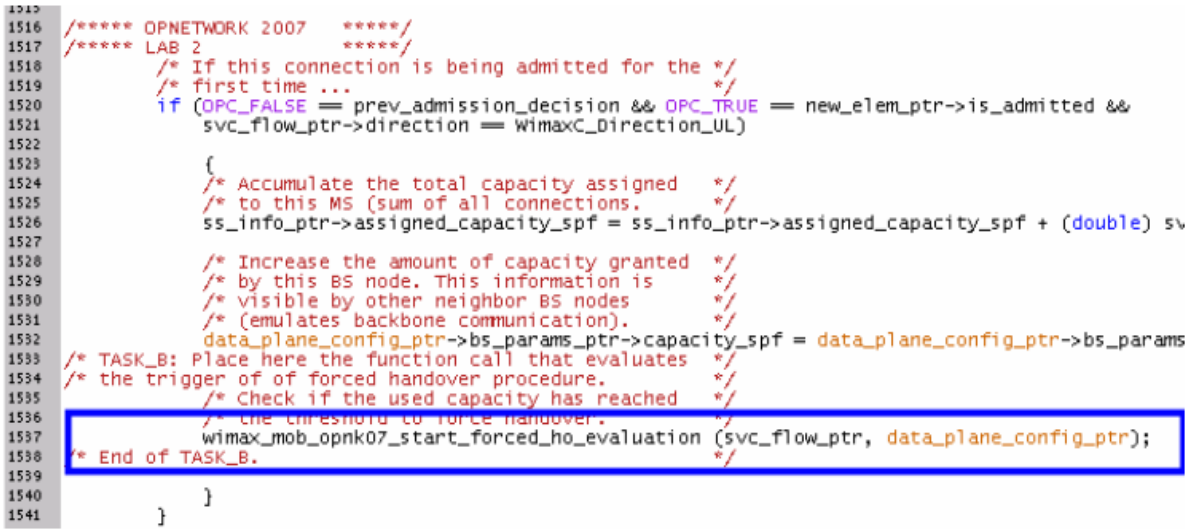
overall traffic at a point of time, it would look to do a forced handover to balance the network load.

A condition or a consequence which prompts the network to do a handover in order to satisfy its constraints of network balance is a trigger for the forced handover. The traffic generation of the SSs and the mobility of the SSs may contribute to this trigger. A handover may be triggered when all the nodes are static but their traffic load changes significantly over a period of time, or even when nodes move making few SSs join or leave a BS causing network imbalance.



```
3947 Boolean
3948 wimax_mob_opnk07_alert_capacity_reached (double current_capacity, double max_capacity)
3949     {
3950     /** Verifies if the capacity has reached the        **/
3951     /** threshold for triggering forced HO.             **/
3952     FIN (wimax_mob_opnk07_alert_capacity_reached ());
3953
3954     /* If the used capcity has reached the specified    */
3955     /* threshold (75% usage).                           */
3956 /* TASK_A1: Type here the missing if() statement        */
3957     if (current_capacity > (0.75 * max_capacity))
3958         FRET (OPC_TRUE);
3959 /* End of TASK_A1                                       */
3960
3961     FRET (OPC_FALSE);
3962     }
3963
```

Fig. 7.1 - shows a trigger for forced handover [8].



```
1516 /***** OPNETWORK 2007    *****/
1517 /***** LAB 2             *****/
1518         /* If this connection is being admitted for the */
1519         /* first time ...                               */
1520         if (OPC_FALSE == prev_admission_decision && OPC_TRUE == new_elem_ptr->is_admitted &&
1521             svc_flow_ptr->direction == WimaxC_Direction_UL)
1522
1523             {
1524             /* Accumulate the total capacity assigned   */
1525             /* to this MS (sum of all connections.      */
1526             ss_info_ptr->assigned_capacity_spf = ss_info_ptr->assigned_capacity_spf + (double) sv
1527
1528             /* Increase the amount of capacity granted  */
1529             /* by this BS node. This information is     */
1530             /* visible by other neighbor BS nodes       */
1531             /* (emulates backbone communication).       */
1532             data_plane_config_ptr->bs_params_ptr->capacity_spf = data_plane_config_ptr->bs_params
1533 /* TASK_B: Place here the function call that evaluates  */
1534 /* the trigger of of forced handover procedure.         */
1535             /* Check if the used capacity has reached   */
1536             /* the threshold to force handover.         */
1537             wimax_mob_opnk07_start_forced_ho_evaluation (svc_flow_ptr, data_plane_config_ptr);
1538 /* End of TASK_B.                                       */
1539
1540             }
1541         }
```

Fig. 7.2 - shows the code where the evaluation for forced handover is done [8]. This is where the network makes a BS look to make a possible forced handover, if needed.

In this scenario, few SSs move and associate to new BSs following the procedure of standard 802.16e handover [2]. This causes the change in amount of traffic load on the BSs. So, the network tries to balance the network load by forcing certain mobile SSs and static SSs to associate to another BS to bring down the load on the BSs and maintaining the network balance.

The network initiated is implemented in OPNET Modeler 14.0, where some special OPNET files are included to implement and reflect the specified functionality and behavior associated with the 'forced' handover. The modules and code are modified and a triggering condition is set according to the requirement of the network.

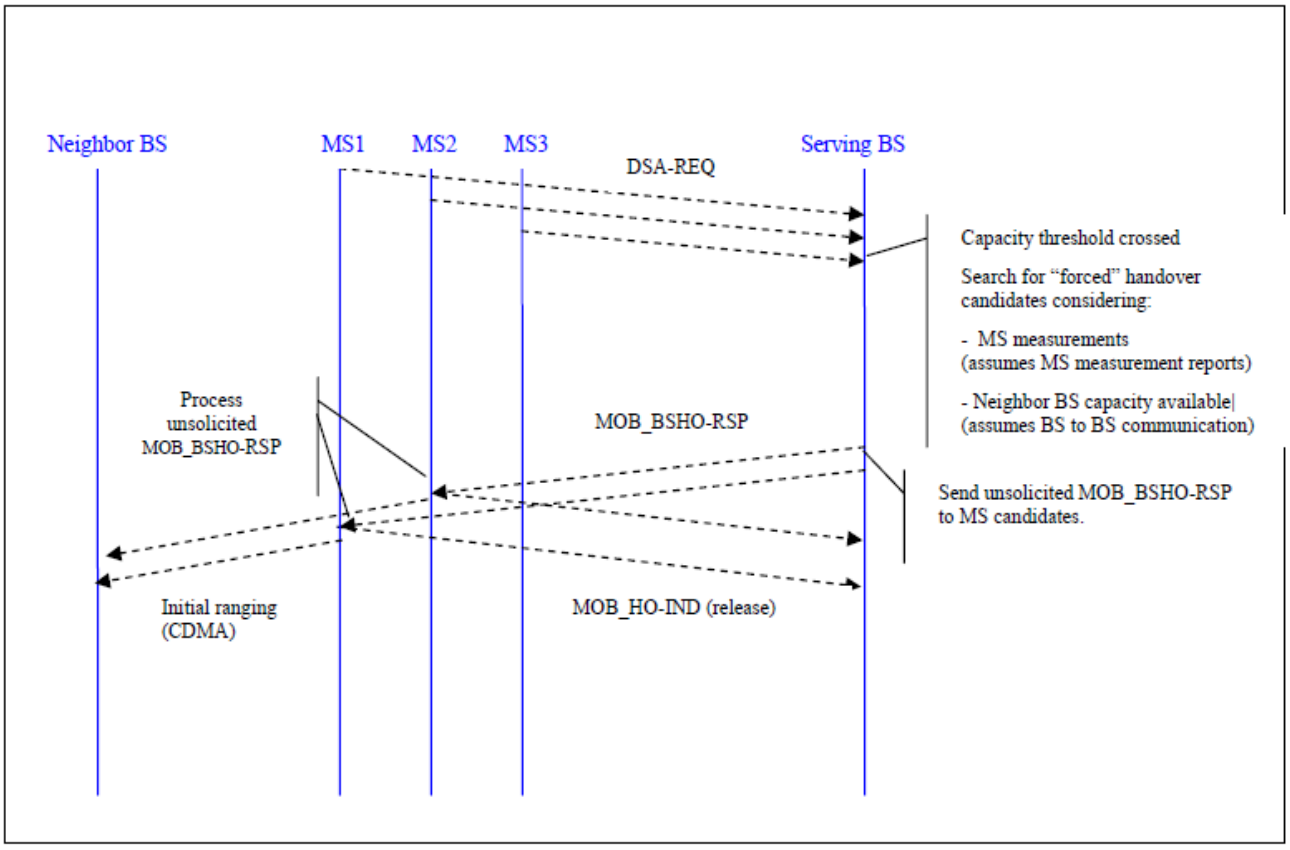


Fig. 7.3 – illustrates [8] the control plane procedure for BS initiated handover.

The network send out DSA messages to the BS and asks the BS to choose a few SSs and find another BS for these SS [19]. It keeps continues until the network load of an individual BS is below the threshold value in percentage of overall node.

The module is changed in OPNET Modeler of a WiMAX SS MAC where there is a new state for forced handover. This allows and handles the forced handover, following the requests of the network.

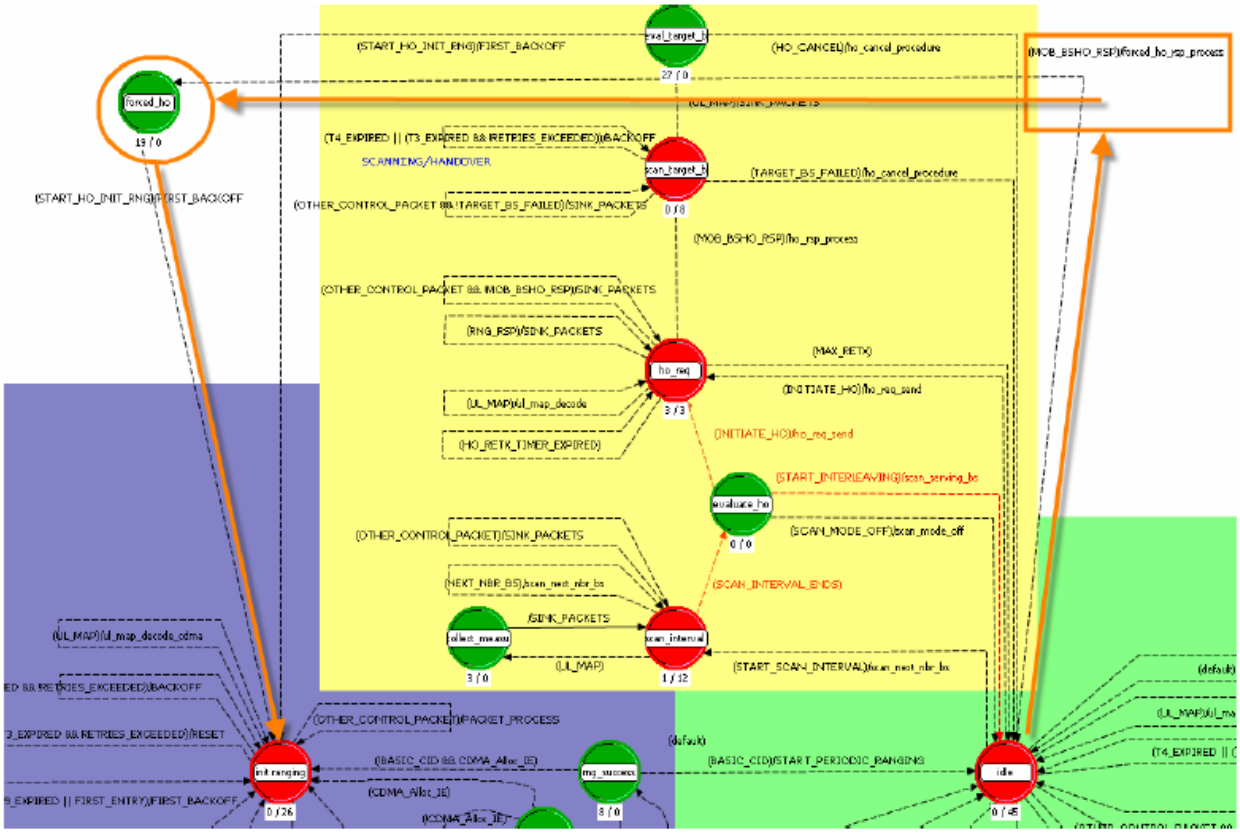


Fig. 7.4 - shows the module [8] of the SS WiMAX_MAC which is modified for ‘forced handover’.

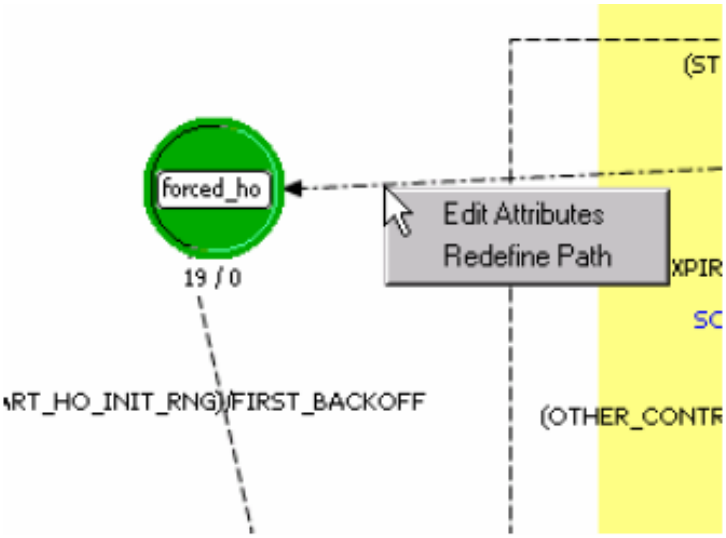


Fig 7.5 - shows a zoom-in view [8] of the previous figure to show an additional state called *forced_ho*, where the code and state transition for forced handover is implemented.

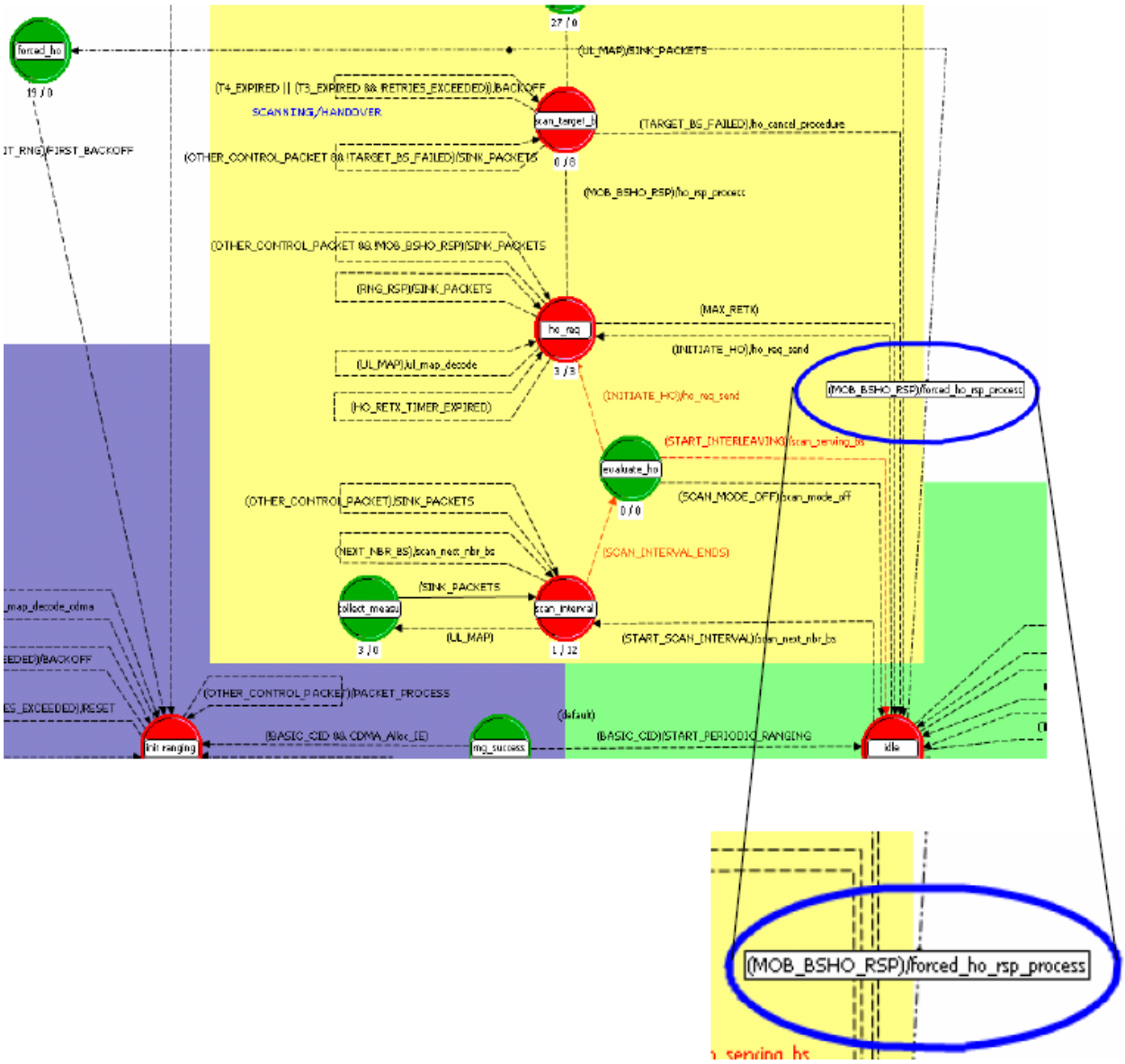


Fig. 7.6 - shows module view [8] of SS WiMAX_MAC, where the transitions, conditions and function calls that lead to *forced_ho* state from *idle* state, such that subsequent 'forced handover' procedure is carried out.

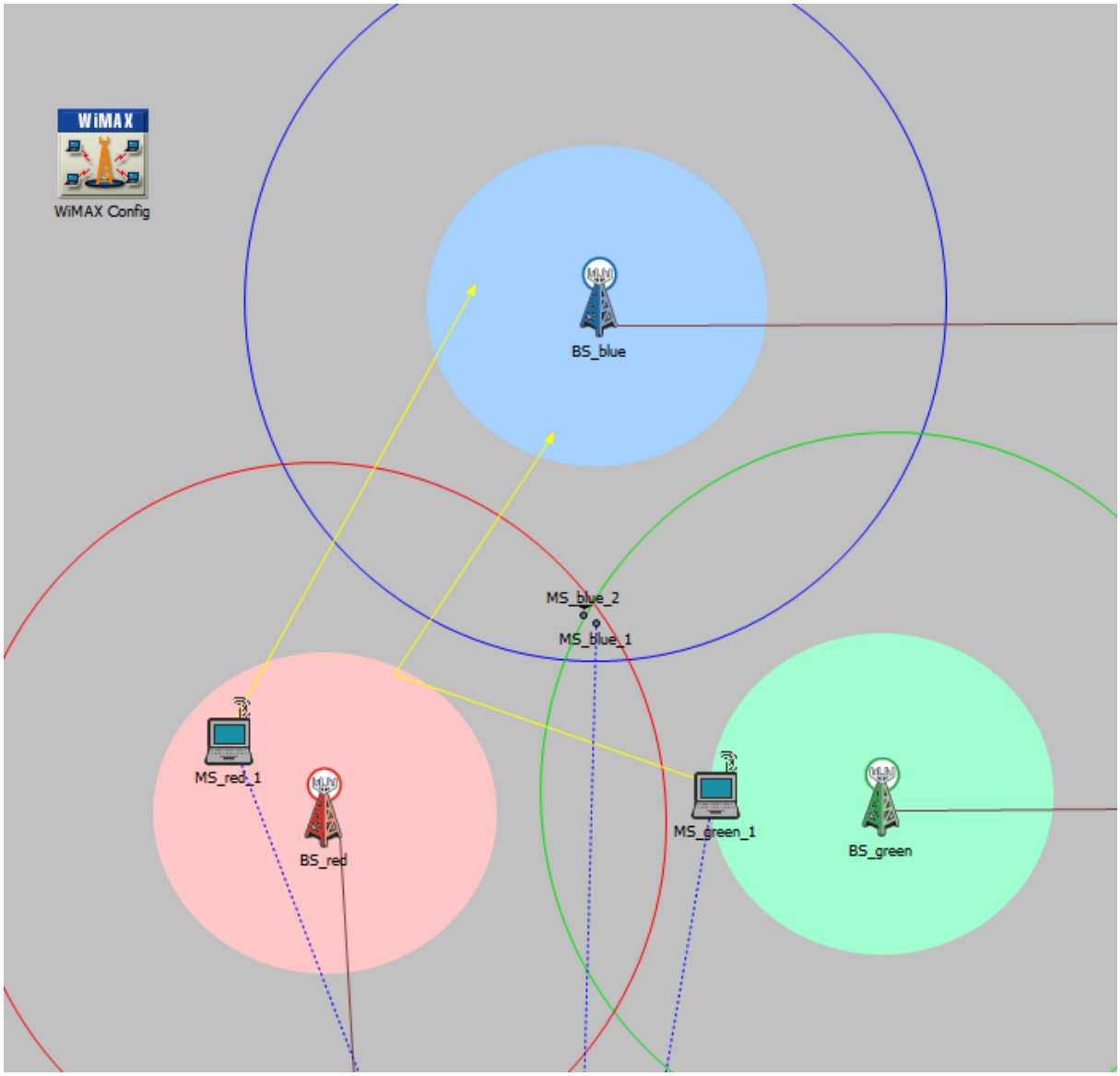


Fig 7.7 - shows a scenario where 'forced' handover is implemented on two mobile nodes *MS_Blue_1* and *MS_Blue*.

Consider the WiMAX scenario described in the above figure. There are 3 BSs and 4 Mobile Subscriber (MS) Stations. These SSs are initially associated with the BSs of color according to which they are named. *MS_blue_1* and *MS_blue_2* are associated to *BS_blue*, *MS_green_1* is associated to *BS_green* and *MS_red_1* is associated to *BS_red*.

Now, only two SS nodes move during the simulation, *MS_green_1* and *MS_red_1* along their respective trajectories as shown in the figure. The BS_ID of *BS_red* is 1, *BS_green* is 2 and *BS_blue* is 5.

The forced handover can be shown using the following graph obtained by running the simulation.

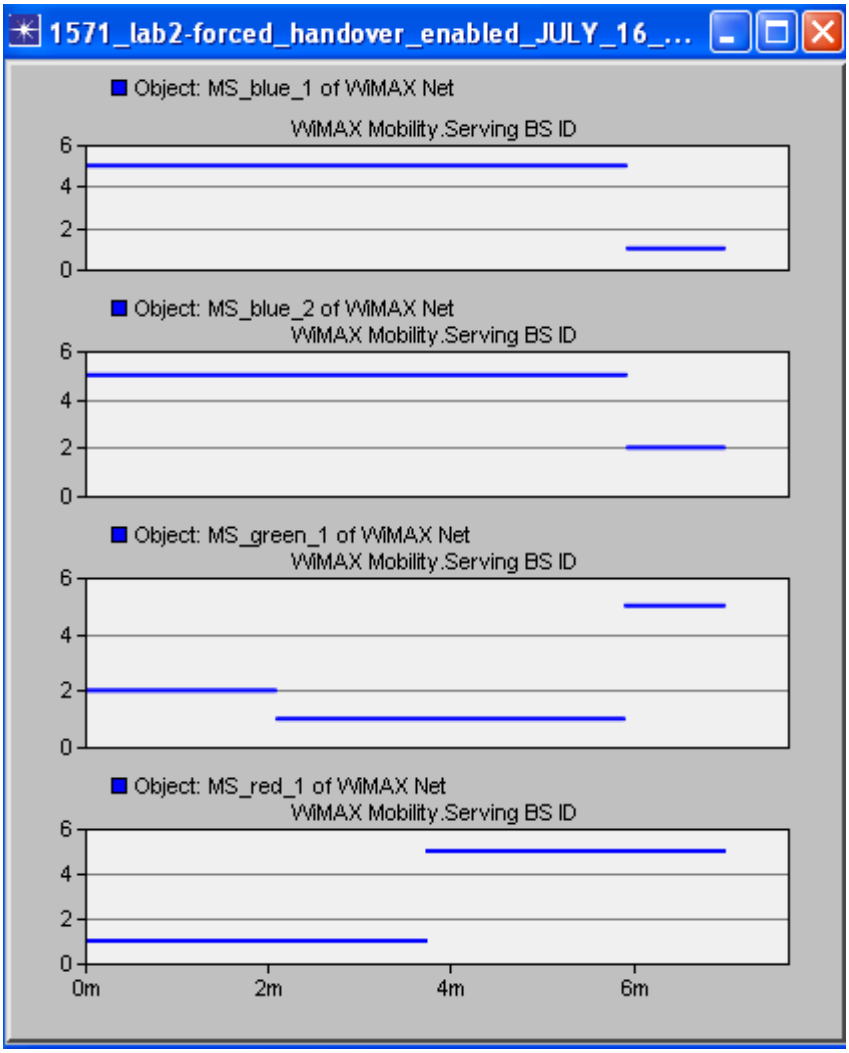


Fig. 7.8 - shows the SSs with the BS_ID values they associate with during the simulation.

It can be noticed that when the two mobile node *MS_red_1* and *MS_green_1* move they associated to other BSs. *MS_red_1* associated to *BS_blue* (BS_ID = 5) and *MS_green_1* is associated to *BS_red* (BS_ID = 1) and then *BS_blue* (BS_ID = 5).

Here none of the *blue* SSs were moving but they still changed their association to other BSs after some time in simulation, when the *BS_blue* was handling more traffic than the threshold value because *MS_red_1* and *MS_green_1* both got associated to it.

To balance the traffic, *BS_blue* has to force some SS nodes to other BSs. It could choose *MS_blue_1* and *MS_blue_2* since they were within communication range of other BSs (*BS_red* and *BS_green*) as well. Hence, even though none of *MS_blue_1* or *MS_blue_2* was moving they underwent forced handover to balance the network traffic.

It can be seen from the above figure that *MS_blue_1* gets associated to *BS_red* (BS_ID = 1) and *MS_blue_2* gets associated to *BS_green* (BS_ID = 2).

This triggering condition could be altered to achieve different goals of the network. It is observed that each triggering condition and requirement for forced handover should be handled in different way and the modification in the WiMAX models might be of different kind. The sequence of events, trigger and the point of modification in module of the WiMAX MAC may vary depending on the procedure and behavior required by the network for processing the 'forced handover'.

7.2 Multi-hop handover with mobile relay station

In order to comprehensively study and solve the issues related to multi-hop WiMAX scenarios, there is a requirement that all features are tried in every hop. For implementing and simulating mobility in a multi-hop scenario, the relay station should be able to move and participate in handover.

Traditionally, the SSs perform handover with BSs, but in a scenario where RSs provide the communication and extend service to the SSs, the RSs should be able to handle handover of SSs. This functionality is desired for improving network balance, changing topology during runtime, providing more flexibility to SSs for communication and above all network stability and connectivity.

Consider a general case, where a SS moves out of a building and enters another building few meters away and users of each of these distant building are connected to the network via different RSs, it is convenient and preferable for the moving user to be able to go through handover rather than losing the connection and setting up a fresh connection for the same service.

Consider a case where one of the RSs has to be temporarily shut down due to issues such as security, failure, maintenance or other reasons, and then the SSs associated with the RS would lose the connectivity to the network. If there is a functionality of handover in RSs, the other RSs would be able to maintain the connectivity and network stability.

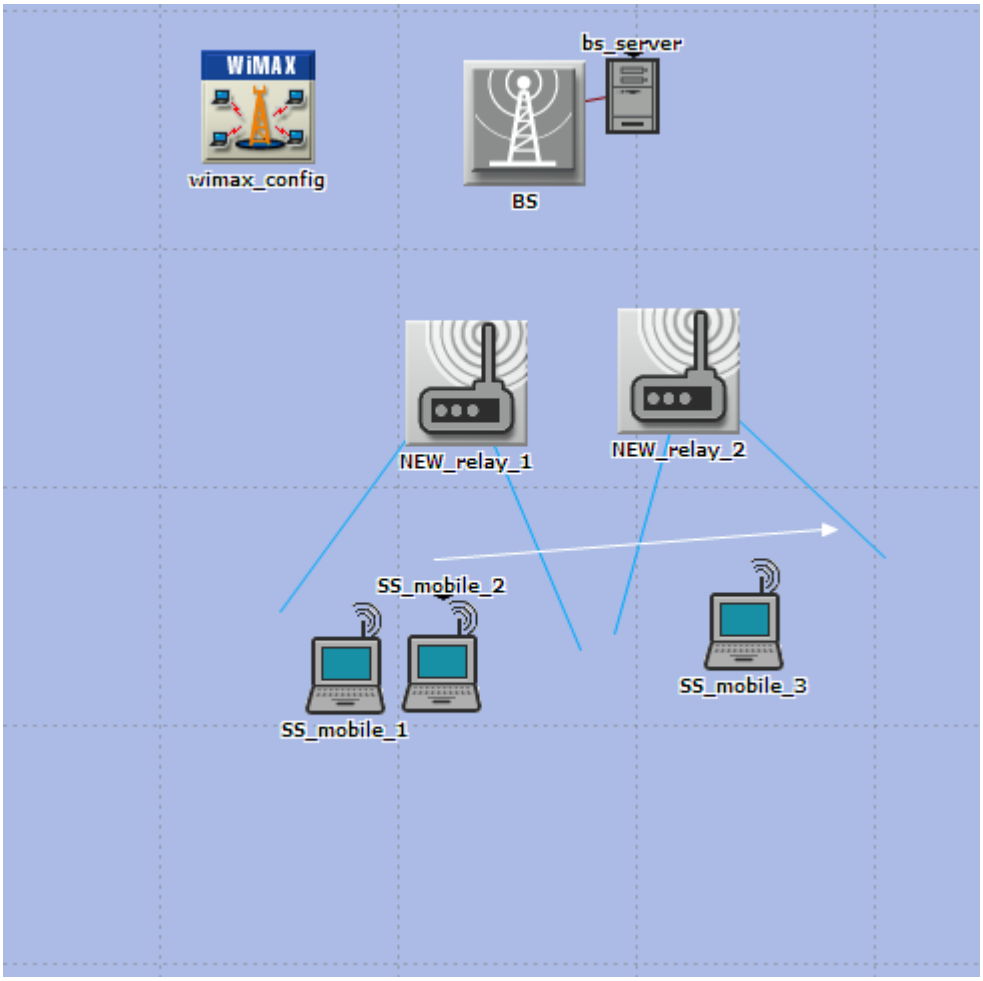
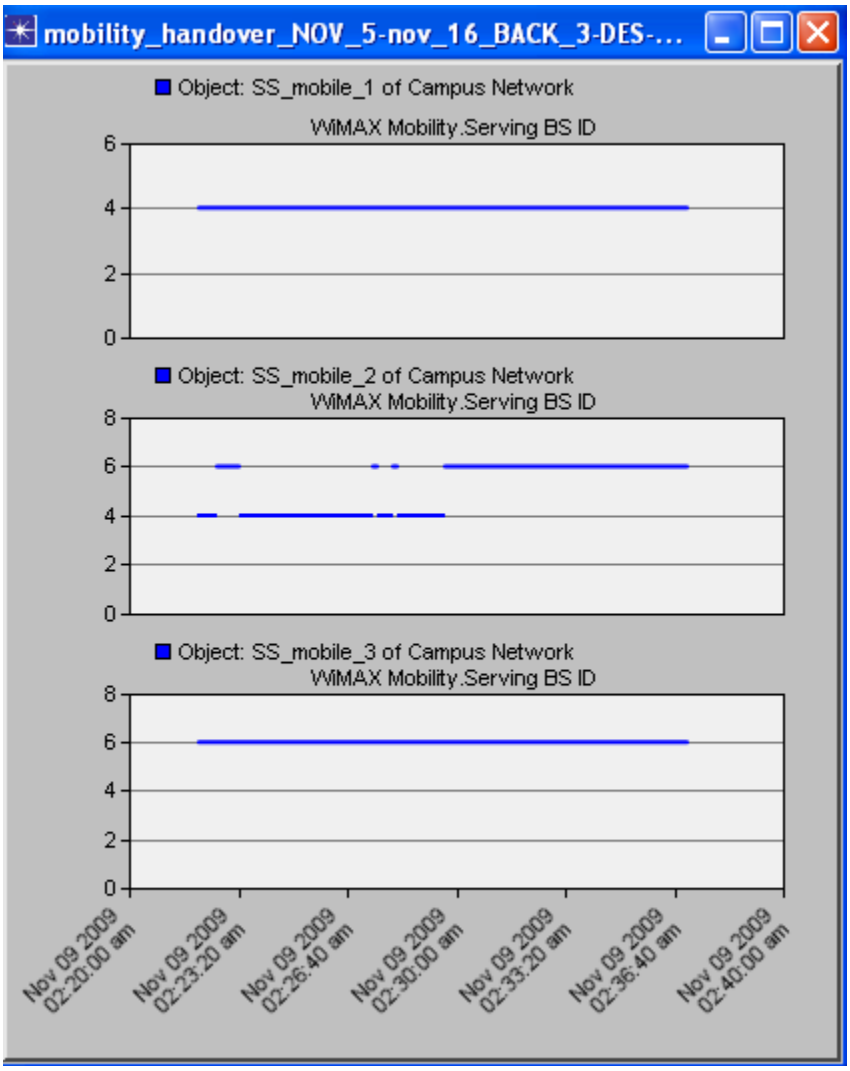


Fig. 7.8 - shows the WiMAX multi-hop network scenario with mobile relays doing handover.

From the above figure, we can see that *SS_mobile_2* moves out of the communication zone of *NEW_relay_1* and moves into the zone of *NEW_relay_2*. Here, the handover is carried out by *NEW_relay_1* to find a new communication for *SS_mobile_2*.

The BS_ID of the *NEW_relay_1* is 4 and BS_ID of *NEW_relay_2* is 6.



Fi.g 7.10 – shows mobile SSs with their serving BS ID values during the simulation time.

Above figure shows the SSs with their associated RSs during the simulation. The *SS_mobile_2* node is eventually associated to *NEW_relay_2* (BS_ID = 6) indicating the implementation of handover in WiMAX multi-hop scenario.

References

[1] http://www.wimaxforum.org/

[2] IEEE 802.16 Working Group, "IEEE Standard for Local and Metropolitan Area Networks– Part 16: Air Interface for Fixed Broadband Wireless Access Systems," IEEE Std. 802.16-2004, October 2004.

[3] —, "Part 16: Air Interface for Fixed Broadband Wireless Access Systems–Amendment 2: Medium Access Control Modifications and Additional Physical Layer Specifications for 2-11 GHz," IEEE Std. 802.16a, Apr. 2003.

[4] —, "Part 16: Air Interface for Fixed Broadband Wireless Access Systems–Amendment 2: Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands," IEEE Std.802.16e, Dec. 2005.

[5] www.opnet.com

[6] Quality of Seervice in WiMAX Mesh Networks, by Dustin Dunkle April 2009

[7] Amendment to IEEE Standard for Local and Metropolitan Area Networks - Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems - Multihop Relay Specification

[8] Introduction to WiMAX Modeling for Network R&D and Planning (Draft 1571 from OPNET)

[9] Interference-Aware Topology Control and QoS Routing in Multi-Channel Wireless Mesh Networks by Jian Tang, Guoliang Xue and Weiyi Zhang.

[10] K. Yap, W.Yeow, M. Motani and C. Tham, "Simple Directional Antennas: Improving Performance in wireless Multihop Networks".

[11] Z. Huang and C. Shen, "Multibeam Antenna-Based Topology Control with Directional Power Intensity for Ad Hoc Networks", IEEE Transaction on Mobile Computing.

[12] U. Kumar, H. Gupta and S. Das, "A Topology Control Approach to Using Directional Antennas in Wireless Mesh Networks". IEEE ICC.

[13] A. Subramanian, H. Gupta and S. Das, "Minimum Interference Channel Assignment in Multi-Radio Wireless Mesh Networks". IEEE SECON.

[14] “DMesh: Incorporating Practical Directional Antennas in Multi-Channel Wireless Mesh Networks” by Saumitra M. Das, Himabindu Pucha, Dimitrios Koutsonikolas, Y. Charlie Hu and Dimitrios Peroulis, August 2005.

[15] “Quality of Service Support in IEEE 802.16 Networks” by Claudio Cicconetti, Luciano Lenzini, and Enzo Mingozzi.

[16] “Deployment and Radio Resource Reuse in IEEE 802.16j Multi-hop Relay Network in Manhattan-like Environment” by I-Kang Fu and Wern-Ho Sheen, Fang-Ching Ren.

[17] “Resource Allocation Algorithm Using Directional Antennas in WiMAX” by Neeraj Gurdasani, November 2009.

[18] Communication Networks: Fundamentals Concepts and Key Architectures by Alberto Leon-Garcia and Indra Widjaja.

[19] “Support of load balancing to enhance service availability”, Jaejeong Shim, KiBack Kim, Aeri Lim, Vikas Mehrotra, Apurv Mathur, Simon Jing, Rishi Arora, Joseph Schumacher, Tommi Rantanen, Giovanni Maggi, Aik Chindapol.

Appendices

Appendix A

Implementation of directional antenna at PHY layer in OPNET Modeler

(Tracking, Dynamic/Adaptive antenna selection)

```
FIN (wimax_phy_mcarrier_pk_send (<args>));



      //=======================CODE_for_BS=============

      if( strcmp(my_node_name,"BS_aac") == 0)
      {

      double latitude,longitude,altitude;
      double x_pos,y_pos,z_pos;

      double lat_1, lat_2, long_1, long_2;

      static int print_count=0;

                  //printf("\n********* This is BS: %d\n", my_mac_address);

      //tx_node_id = op_topo_parent (op_id_self ());
//printf("%d\n",tx_node_id);
      /* id of antenna module of transmitter node */
//    ant_node_id = op_id_from_name (tx_node_id, OPC_OBJTYPE_ANT,
"wimax_ant_8_0");
//printf("%d\n",ant_node_id);

      /* id of subnet containing transmitter node */
      //subnet_id = op_topo_parent (tx_node_id);

      //ant_node_id1 = op_id_from_name (subnet_id, OPC_OBJTYPE_NODE_FIX,
"BS");
   //ant_node_id2 = op_id_from_name (ant_node_id1, OPC_OBJTYPE_ANT,
"wimax_ant_8_0");


////  printf("%d\n",ant_node_id2);
      /* id of receiver within subnet */
      //rx_node_id = op_id_from_name (subnet_id, OPC_OBJTYPE_NODE_MOB, "SS");
//printf("%d\n",rx_node_id);

      /* position of rx loaded into x_pos, y_pos, z_pos */
      //op_ima_obj_pos_get (rx_node_id, &latitude, &longitude,&altitude,
&x_pos, &y_pos, &z_pos) ;
```

```
    //==============for calculating the 'angle of cone'......================
      //if  (strcmp(my_node_name,"BS_aac") == 0)
      //{
      //    op_ima_obj_pos_get (op_id_from_name (op_topo_parent
(op_topo_parent (op_id_self ())), OPC_OBJTYPE_NODE_FIX, my_node_name),
&latitude, &longitude,&altitude, &x_pos, &y_pos, &z_pos);
            //printf("\n\n\n BS_aac ==== Latitude: %lf    ======= Longitude:
%lf \n\n\n", latitude, longitude);
            //printf("\n\n\n X_pos: %lf    ======= Y_pos: %lf \n\n\n", x_pos,
y_pos);
      //}
      //=========================================================================
===

      //op_ima_obj_pos_get (op_id_from_name (op_topo_parent (op_topo_parent
(op_id_self ())), OPC_OBJTYPE_NODE_FIX, "RELAY_1"), &latitude,
&longitude,&altitude, &x_pos, &y_pos, &z_pos);

    op_ima_obj_pos_get (op_id_from_name (op_topo_parent (op_topo_parent
(op_id_self ())), OPC_OBJTYPE_NODE_FIX, "RELAY_2"), &latitude,
&longitude,&altitude, &x_pos, &y_pos, &z_pos);
      lat_1=latitude;
      long_1=longitude;

    op_ima_obj_pos_get (op_id_from_name (op_topo_parent (op_topo_parent
(op_id_self ())), OPC_OBJTYPE_NODE_MOB, "SS_wks_5"), &latitude,
&longitude,&altitude, &x_pos, &y_pos, &z_pos);
      lat_2=latitude;
      long_2=longitude;

      latitude = (lat_1 + lat_2)/2;
      longitude = (long_1 + long_2)/2;


      //printf("\n\n********* This position of 'x': %d\n",x_pos);
      //printf("\n********* This position of 'y': %d\n",y_pos);
      //printf("\n********* This position of 'z': %d\n\n",z_pos);

      //printf("\n$$$$$$$$$$$$ This is TX id: %d\n\n", tx_phy_info_ptr-
>tx_module_objid);

      /* set target of antenna toward receiver rx */
      //op_ima_obj_attr_set (ant_node_id2, "target x", x_pos);

      print_count++;

      if(print_count == 100)
      {
      //printf("\n\n\n HERE IS THE =======  BS BS BS BS BS BS BS BS BS BS
====== IMPLEMENTATION OF 'directional antenna'... \n");

      //printf("\n latitude Postion: %lf   ------  longitude Position: %lf
------   altitude Position : %lf \n\n\n", latitude, longitude, altitude);

      print_count = 0;
```

```
      }

      op_ima_obj_attr_set (op_id_from_name (op_id_from_name (op_topo_parent
(op_topo_parent (op_id_self ())), OPC_OBJTYPE_NODE_FIX, "BS_aac"),
OPC_OBJTYPE_ANT, "wimax_ant_32_0"), "target latitude", latitude);

      //printf("\n********* INSIDE HEADER... before line 2 \n\n");

    //op_ima_obj_attr_set (ant_node_id2, "target y", y_pos);

      op_ima_obj_attr_set (op_id_from_name (op_id_from_name (op_topo_parent
(op_topo_parent (op_id_self ())), OPC_OBJTYPE_NODE_FIX, "BS_aac"),
OPC_OBJTYPE_ANT, "wimax_ant_32_0"), "target longitude", longitude);

      //printf("\n********* INSIDE HEADER... before line 3 \n\n");

      op_ima_obj_attr_set (op_id_from_name (op_id_from_name (op_topo_parent
(op_topo_parent (op_id_self ())), OPC_OBJTYPE_NODE_FIX, "BS_aac"),
OPC_OBJTYPE_ANT, "wimax_ant_32_0"), "target altitude", altitude);

    //op_ima_obj_attr_set (ant_node_id2, "target z", z_pos);

      }

      //=========================CODE_END_BS====================


      //==============================CODE_FOR_RELAY=========================

    ////////////////////=====BS_RADIO_RELAY==================/////////

      //if  (tx_module_objid == 4927 || tx_module_objid == 6612  ||
tx_module_objid == 8297)

      //printf("\n I'm fine...\n");

      if( strcmp(my_mac_name,"wimax_mac_1_0")==0 )
      {
      if ( (strcmp(my_node_name,"RELAY_1")==0) ||
(strcmp(my_node_name,"RELAY_2")==0) || (strcmp(my_node_name,"RELAY_3")==0) ||
(strcmp(my_node_name,"RELAY_4")==0) || (strcmp(my_node_name,"RELAY_5")==0) ||
(strcmp(my_node_name,"RELAY_6")==0) || (strcmp(my_node_name,"RELAY_7")==0) ||
(strcmp(my_node_name,"RELAY_8")==0) || (strcmp(my_node_name,"RELAY_9")==0) ||
(strcmp(my_node_name,"RELAY_10")==0))
      {

      double latitude,longitude,altitude;
      double x_pos,y_pos,z_pos;

      static int print_count=0;
      char* target_node;
      char* source_node;

    char* target_node_1;
    char* target_node_2;
      double lat_1, lat_2, long_1, long_2;
```

```
      //printf("\n********* This is RELAY: %d\n", my_mac_address);
      print_count++;
      if(print_count == 100)
      {
      //printf("\n\n\n HERE IS THE =======  RS RS RS RS RS RS RS RS RS RS RS
RS RS RS  RS RS RS====== IMPLEMENTATION OF 'directional antenna'... \n");
      //printf("\n latitude Postion: %lf   ------  longitude Position: %lf
------   altitude Position : %lf \n\n\n", latitude, longitude, altitude);
      print_count = 0;
      }


      if  (strcmp(my_node_name,"RELAY_1") == 0)
      {
            target_node_1 = "SS_wks_25";
            target_node_2 = "RELAY_3";

            source_node = "RELAY_1";
      }

      if  (strcmp(my_node_name,"RELAY_2") == 0)
      {
            target_node_1 = "SS_wks_11";
            target_node_2 = "SS_wks_3";

            source_node = "RELAY_2";
      }

      if  (strcmp(my_node_name,"RELAY_3") == 0)
      {
            target_node_1 = "SS_wks_1";
            target_node_2 = "SS_wks_2";

            source_node = "RELAY_3";
      }

      if  (strcmp(my_node_name,"RELAY_4") == 0)
      {
            target_node_1 = "RELAY_6";
            target_node_2 = "RELAY_5";

            source_node = "RELAY_4";
      }

      if  (strcmp(my_node_name,"RELAY_5") == 0)
      {
            target_node_1 = "SS_wks_8";
            target_node_2 = "SS_wks_6";

            source_node = "RELAY_5";
      }

      if  (strcmp(my_node_name,"RELAY_6") == 0)
      {
            target_node_1 = "SS_wks_24";
            target_node_2 = "SS_wks_9";
```

```
            source_node = "RELAY_6";
      }

      if  (strcmp(my_node_name,"RELAY_7") == 0)
      {
            target_node_1 = "SS_wks_14";
            target_node_2 = "SS_wks_15";

            source_node = "RELAY_7";
      }

      if  (strcmp(my_node_name,"RELAY_8") == 0)
      {
            target_node_1 = "RELAY_7";
            target_node_2 = "RELAY_9";

            source_node = "RELAY_8";
      }

      if  (strcmp(my_node_name,"RELAY_9") == 0)
      {
            target_node_1 = "SS_wks_18";
            target_node_2 = "SS_wks_19";

            source_node = "RELAY_9";
      }
      if  (strcmp(my_node_name,"RELAY_10") == 0)
      {
            target_node_1 = "SS_wks_17";
            target_node_2 = "SS_wks_12";

            source_node = "RELAY_10";
      }

      //this is for dynamic antenna assigning...
   // if  (tx_module_objid == 4902)
      //{
      //    target_node = "SS_wks_1";
      //    source_node = "RELAY_1";
      //}

      //this is for dynamic antenna assigning...
    //if  (tx_module_objid == 6587)
      //{
            //target_node = "SS_wks_2";
            //source_node = "RELAY_2";
      //}


    //==============For calculating the 'angle of cone'......================
      //if  (strcmp(my_node_name,"RELAY_2") == 0)
      //{
      //    op_ima_obj_pos_get (op_id_from_name (op_topo_parent
(op_topo_parent (op_id_self ())), OPC_OBJTYPE_NODE_FIX, source_node),
&latitude, &longitude,&altitude, &x_pos, &y_pos, &z_pos);
```

```
      //    printf("\n\n\n RELAY_2 ==== Latitude: %lf    ======= Longitude:
%lf \n\n\n", latitude, longitude);
      //    printf("\n\n\n X_pos: %lf    ======= Y_pos: %lf \n\n\n", x_pos,
y_pos);
      //}
      //=====================================================

   //if( strcmp(my_node_name,"RELAY_1") == 0 )
      //op_ima_obj_pos_get (op_id_from_name (op_topo_parent (op_topo_parent
(op_id_self ())), OPC_OBJTYPE_NODE_FIX, target_node), &latitude,
&longitude,&altitude, &x_pos, &y_pos, &z_pos);
  // else
   //{
      //}
   if( strcmp(my_node_name,"RELAY_3") == 0 || strcmp(my_node_name,"RELAY_2")
== 0  || strcmp(my_node_name,"RELAY_5") == 0  ||
strcmp(my_node_name,"RELAY_6") == 0  || strcmp(my_node_name,"RELAY_7") == 0
|| strcmp(my_node_name,"RELAY_9") == 0 || strcmp(my_node_name,"RELAY_10") ==
0 )
   {
      op_ima_obj_pos_get (op_id_from_name (op_topo_parent (op_topo_parent
(op_id_self ())), OPC_OBJTYPE_NODE_MOB, target_node_1), &latitude,
&longitude,&altitude, &x_pos, &y_pos, &z_pos);
      lat_1=latitude;
      long_1=longitude;

    op_ima_obj_pos_get (op_id_from_name (op_topo_parent (op_topo_parent
(op_id_self ())), OPC_OBJTYPE_NODE_MOB, target_node_2), &latitude,
&longitude,&altitude, &x_pos, &y_pos, &z_pos);
      lat_2=latitude;
      long_2=longitude;

   }
   else if ( strcmp(my_node_name,"RELAY_4") == 0 ||
strcmp(my_node_name,"RELAY_8") == 0)
   {
    op_ima_obj_pos_get (op_id_from_name (op_topo_parent (op_topo_parent
(op_id_self ())), OPC_OBJTYPE_NODE_FIX, target_node_1), &latitude,
&longitude,&altitude, &x_pos, &y_pos, &z_pos);
      lat_1=latitude;
      long_1=longitude;

    op_ima_obj_pos_get (op_id_from_name (op_topo_parent (op_topo_parent
(op_id_self ())), OPC_OBJTYPE_NODE_FIX, target_node_2), &latitude,
&longitude,&altitude, &x_pos, &y_pos, &z_pos);
      lat_2=latitude;
      long_2=longitude;
   }
   else if ( strcmp(my_node_name,"RELAY_1") == 0 )
   {
      op_ima_obj_pos_get (op_id_from_name (op_topo_parent (op_topo_parent
(op_id_self ())), OPC_OBJTYPE_NODE_MOB, target_node_1), &latitude,
&longitude,&altitude, &x_pos, &y_pos, &z_pos);
      lat_1=latitude;
      long_1=longitude;
```

```
    op_ima_obj_pos_get (op_id_from_name (op_topo_parent (op_topo_parent
(op_id_self ())), OPC_OBJTYPE_NODE_FIX, target_node_2), &latitude,
&longitude,&altitude, &x_pos, &y_pos, &z_pos);
      lat_2=latitude;
      long_2=longitude;
   }


      latitude = (lat_1 + lat_2)/2;
      longitude = (long_1 + long_2)/2;


      //====CODE_INSERT==Dynamic_Directional_Antenna_SWITCHING=======

      //if( tx_module_objid == 4902 &&  op_sim_time() >= 150)  //this can be
the switching condition and parameter(cone_TYPE).....
            //printf("*****: %lf",op_sim_time());

    if( strcmp(my_node_name,"RELAY_2") == 0  ||
strcmp(my_node_name,"RELAY_6") == 0 || strcmp(my_node_name,"RELAY_7") == 0)
      op_ima_obj_attr_set (op_id_from_name (op_id_from_name (op_topo_parent
(op_topo_parent (op_id_self ())), OPC_OBJTYPE_NODE_FIX, source_node),
OPC_OBJTYPE_ANT, "wimax_ant_1_0"), "pattern", "cone_35");
      else if ( strcmp(my_node_name,"RELAY_5") == 0  ||
strcmp(my_node_name,"RELAY_9") == 0)
    op_ima_obj_attr_set (op_id_from_name (op_id_from_name (op_topo_parent
(op_topo_parent (op_id_self ())), OPC_OBJTYPE_NODE_FIX, source_node),
OPC_OBJTYPE_ANT, "wimax_ant_1_0"), "pattern", "cone_45");
    else if ( strcmp(my_node_name,"RELAY_4") == 0  ||
strcmp(my_node_name,"RELAY_8") == 0)
    op_ima_obj_attr_set (op_id_from_name (op_id_from_name (op_topo_parent
(op_topo_parent (op_id_self ())), OPC_OBJTYPE_NODE_FIX, source_node),
OPC_OBJTYPE_ANT, "wimax_ant_1_0"), "pattern", "cone_90");
      else if ( strcmp(my_node_name,"RELAY_1") == 0)
    op_ima_obj_attr_set (op_id_from_name (op_id_from_name (op_topo_parent
(op_topo_parent (op_id_self ())), OPC_OBJTYPE_NODE_FIX, source_node),
OPC_OBJTYPE_ANT, "wimax_ant_1_0"), "pattern", "cone_40");
    else if ( strcmp(my_node_name,"RELAY_3") == 0)
    op_ima_obj_attr_set (op_id_from_name (op_id_from_name (op_topo_parent
(op_topo_parent (op_id_self ())), OPC_OBJTYPE_NODE_FIX, source_node),
OPC_OBJTYPE_ANT, "wimax_ant_1_0"), "pattern", "cone_60");
      else if ( strcmp(my_node_name,"RELAY_10") == 0)
    op_ima_obj_attr_set (op_id_from_name (op_id_from_name (op_topo_parent
(op_topo_parent (op_id_self ())), OPC_OBJTYPE_NODE_FIX, source_node),
OPC_OBJTYPE_ANT, "wimax_ant_1_0"), "pattern", "cone_50");

      //================================================================

      op_ima_obj_attr_set (op_id_from_name (op_id_from_name (op_topo_parent
(op_topo_parent (op_id_self ())), OPC_OBJTYPE_NODE_FIX, source_node),
OPC_OBJTYPE_ANT, "wimax_ant_1_0"), "target latitude", latitude);

      op_ima_obj_attr_set (op_id_from_name (op_id_from_name (op_topo_parent
(op_topo_parent (op_id_self ())), OPC_OBJTYPE_NODE_FIX, source_node),
OPC_OBJTYPE_ANT, "wimax_ant_1_0"), "target longitude", longitude);
```

```
      op_ima_obj_attr_set (op_id_from_name (op_id_from_name (op_topo_parent
(op_topo_parent (op_id_self ())), OPC_OBJTYPE_NODE_FIX, source_node),
OPC_OBJTYPE_ANT, "wimax_ant_1_0"), "target altitude", altitude);

      }
      }


     //////////////=====SS_RADIO_RELAY==================///////

      //if  (tx_module_objid == 5043 || tx_module_objid == 6728  ||
tx_module_objid == 8413)

      if( strcmp(my_mac_name,"wimax_mac_2_0")==0 )
      {
      if ( (strcmp(my_node_name,"RELAY_1")==0) ||
(strcmp(my_node_name,"RELAY_2")==0) || (strcmp(my_node_name,"RELAY_3")==0) ||
(strcmp(my_node_name,"RELAY_4")==0) || (strcmp(my_node_name,"RELAY_5")==0) ||
(strcmp(my_node_name,"RELAY_6")==0) || (strcmp(my_node_name,"RELAY_7")==0) ||
(strcmp(my_node_name,"RELAY_8")==0) || (strcmp(my_node_name,"RELAY_9")==0) ||
(strcmp(my_node_name,"RELAY_10")==0))
      {

      double latitude,longitude,altitude;
      double x_pos,y_pos,z_pos;

      //static int print_count=0;
      char* target_node;
      char* source_node;

    if  (strcmp(my_node_name,"RELAY_1") == 0)
      {
            target_node = "BS_aac";
            source_node = "RELAY_1";
      }

      if  (strcmp(my_node_name,"RELAY_2") == 0)
      {
            target_node = "BS_aac";
            source_node = "RELAY_2";
      }

      if  (strcmp(my_node_name,"RELAY_3") == 0)
      {
            target_node = "RELAY_1";
            source_node = "RELAY_3";
      }

       if  (strcmp(my_node_name,"RELAY_4") == 0)
      {
            target_node = "BS_aac";
            source_node = "RELAY_4";
      }

      if  (strcmp(my_node_name,"RELAY_5") == 0)
      {
            target_node = "RELAY_4";
```

```
            source_node = "RELAY_5";
      }

      if  (strcmp(my_node_name,"RELAY_6") == 0)
      {
            target_node = "RELAY_4";
            source_node = "RELAY_6";
      }

       if  (strcmp(my_node_name,"RELAY_7") == 0)
      {
            target_node = "RELAY_8";
            source_node = "RELAY_7";
      }

      if  (strcmp(my_node_name,"RELAY_8") == 0)
      {
            target_node = "BS_aac";
            source_node = "RELAY_8";
      }

      if  (strcmp(my_node_name,"RELAY_9") == 0)
      {
            target_node = "RELAY_8";
            source_node = "RELAY_9";
      }

      if  (strcmp(my_node_name,"RELAY_10") == 0)
      {
            target_node = "RELAY_6";
            source_node = "RELAY_10";
      }


        //===========This is for dynamic antenna assigning==========
    //if  (tx_module_objid == 5018)
        //{
        //    target_node = "BS_aac";
              //source_node = "RELAY_1";
        //}

        //==========This is for dynamic antenna assigning===========
    //if  (tx_module_objid == 6703)
        //{
        //    target_node = "BS_aac";
        //    source_node = "RELAY_2";
        //}

      op_ima_obj_pos_get (op_id_from_name (op_topo_parent (op_topo_parent
(op_id_self ())), OPC_OBJTYPE_NODE_FIX, target_node), &latitude,
&longitude,&altitude, &x_pos, &y_pos, &z_pos);

      op_ima_obj_attr_set (op_id_from_name (op_id_from_name (op_topo_parent
(op_topo_parent (op_id_self ())), OPC_OBJTYPE_NODE_FIX, source_node),
OPC_OBJTYPE_ANT, "wimax_ant_2_0"), "target latitude", latitude);
```

```
	op_ima_obj_attr_set (op_id_from_name (op_id_from_name (op_topo_parent
(op_topo_parent (op_id_self ())), OPC_OBJTYPE_NODE_FIX, source_node),
OPC_OBJTYPE_ANT, "wimax_ant_2_0"), "target longitude", longitude);

	op_ima_obj_attr_set (op_id_from_name (op_id_from_name (op_topo_parent
(op_topo_parent (op_id_self ())), OPC_OBJTYPE_NODE_FIX, source_node),
OPC_OBJTYPE_ANT, "wimax_ant_2_0"), "target altitude", altitude);

	}
	}

	//====================CODE_END_RELAY==========




	//============================CODE_for_SS===============

	//if  (tx_module_objid == 2752 || tx_module_objid == 3384  ||
tx_module_objid == 9071 || tx_module_objid == 9703  || tx_module_objid ==
10335)

	if ( strcmp(my_node_name,"SS_wks_1") == 0 ||
strcmp(my_node_name,"SS_wks_2") == 0 || strcmp(my_node_name,"SS_wks_3") == 0
|| strcmp(my_node_name,"SS_wks_4") == 0 || strcmp(my_node_name,"SS_wks_5") ==
0
		|| strcmp(my_node_name,"SS_wks_6") == 0 ||
strcmp(my_node_name,"SS_wks_7") == 0 || strcmp(my_node_name,"SS_wks_8") == 0
|| strcmp(my_node_name,"SS_wks_9") == 0 || strcmp(my_node_name,"SS_wks_10")
== 0
		|| strcmp(my_node_name,"SS_wks_11") == 0 ||
strcmp(my_node_name,"SS_wks_12") == 0 || strcmp(my_node_name,"SS_wks_13") ==
0 || strcmp(my_node_name,"SS_wks_14") == 0 ||
strcmp(my_node_name,"SS_wks_15") == 0
		|| strcmp(my_node_name,"SS_wks_16") == 0 ||
strcmp(my_node_name,"SS_wks_17") == 0 || strcmp(my_node_name,"SS_wks_18") ==
0 || strcmp(my_node_name,"SS_wks_19") == 0 ||
strcmp(my_node_name,"SS_wks_20") == 0
		|| strcmp(my_node_name,"SS_wks_21") == 0 ||
strcmp(my_node_name,"SS_wks_22") == 0 || strcmp(my_node_name,"SS_wks_23") ==
0 || strcmp(my_node_name,"SS_wks_24") == 0 ||
strcmp(my_node_name,"SS_wks_25") == 0 )
    {

	double latitude,longitude,altitude;
	double x_pos,y_pos,z_pos;

	//static int print_count=0;
	char* target_node;
	char* source_node;


	if  (strcmp(my_node_name,"SS_wks_1") == 0)
	{
		target_node = "RELAY_3";
```

```
      source_node = "SS_wks_1";
}

if  (strcmp(my_node_name,"SS_wks_2") == 0)
{
      target_node = "RELAY_3";
      source_node = "SS_wks_2";
}

if  (strcmp(my_node_name,"SS_wks_3") == 0)
{
      target_node = "RELAY_2";
      source_node = "SS_wks_3";
}
if  (strcmp(my_node_name,"SS_wks_4") == 0)
{
      target_node = "RELAY_2";
      source_node = "SS_wks_4";
}

if  (strcmp(my_node_name,"SS_wks_5") == 0)
{
      target_node = "BS_aac";
      source_node = "SS_wks_5";
}


if  (strcmp(my_node_name,"SS_wks_6") == 0)
{
      target_node = "RELAY_5";
      source_node = "SS_wks_6";
}

if  (strcmp(my_node_name,"SS_wks_7") == 0)
{
      target_node = "RELAY_4";
      source_node = "SS_wks_7";
}

if  (strcmp(my_node_name,"SS_wks_8") == 0)
{
      target_node = "RELAY_5";
      source_node = "SS_wks_8";
}
if  (strcmp(my_node_name,"SS_wks_9") == 0)
{
      target_node = "RELAY_6";
      source_node = "SS_wks_9";
}

if  (strcmp(my_node_name,"SS_wks_10") == 0)
{
      target_node = "BS_aac";
      source_node = "SS_wks_10";
}

      if  (strcmp(my_node_name,"SS_wks_11") == 0)
```

```
{
      target_node = "RELAY_2";
      source_node = "SS_wks_11";
}

if  (strcmp(my_node_name,"SS_wks_12") == 0)
{
      target_node = "RELAY_10";
      source_node = "SS_wks_12";
}

if  (strcmp(my_node_name,"SS_wks_13") == 0)
{
      target_node = "RELAY_7";
      source_node = "SS_wks_13";
}
if  (strcmp(my_node_name,"SS_wks_14") == 0)
{
      target_node = "RELAY_7";
      source_node = "SS_wks_14";
}

if  (strcmp(my_node_name,"SS_wks_15") == 0)
{
      target_node = "RELAY_7";
      source_node = "SS_wks_15";
}


if  (strcmp(my_node_name,"SS_wks_16") == 0)
{
      target_node = "RELAY_10";
      source_node = "SS_wks_16";
}

if  (strcmp(my_node_name,"SS_wks_17") == 0)
{
      target_node = "RELAY_10";
      source_node = "SS_wks_17";
}

if  (strcmp(my_node_name,"SS_wks_18") == 0)
{
      target_node = "RELAY_9";
      source_node = "SS_wks_18";
}
if  (strcmp(my_node_name,"SS_wks_19") == 0)
{
      target_node = "RELAY_9";
      source_node = "SS_wks_19";
}

if  (strcmp(my_node_name,"SS_wks_20") == 0)
{
      target_node = "RELAY_8";
      source_node = "SS_wks_20";
}
```

```
      if  (strcmp(my_node_name,"SS_wks_21") == 0)
      {
            target_node = "BS_aac";
            source_node = "SS_wks_21";
      }

      if  (strcmp(my_node_name,"SS_wks_22") == 0)
      {
            target_node = "RELAY_8";
            source_node = "SS_wks_22";
      }

      if  (strcmp(my_node_name,"SS_wks_23") == 0)
      {
            target_node = "RELAY_5";
            source_node = "SS_wks_23";
      }

      if  (strcmp(my_node_name,"SS_wks_24") == 0)
      {
            target_node = "RELAY_6";
            source_node = "SS_wks_24";
      }

      if  (strcmp(my_node_name,"SS_wks_25") == 0)
      {
            target_node = "RELAY_1";
            source_node = "SS_wks_25";
      }


      //this is for dynamic antenna assigning...
      //if  (tx_module_objid == 2727)
      //{
      //    target_node = "RELAY_1";
      //    source_node = "SS_wks_1";
      ///}
      //this is for dynamic antenna assigning...
      //if  (tx_module_objid == 3359)
      //{
      //    target_node = "RELAY_2";
      //    source_node = "SS_wks_2";
      //}


    //==============for calculating the 'angle of cone'......================
      if  (strcmp(my_node_name,"SS_wks_5") == 0)
      {
            op_ima_obj_pos_get (op_id_from_name (op_topo_parent
(op_topo_parent (op_id_self ())), OPC_OBJTYPE_NODE_MOB, source_node),
&latitude, &longitude,&altitude, &x_pos, &y_pos, &z_pos);
            printf("\n\n\n SS_wks_5 ==== Latitude: %lf    ======= Longitude:
%lf \n\n\n", latitude, longitude);
            printf("\n\n\n X_pos: %lf    ======= Y_pos: %lf \n\n\n", x_pos,
y_pos);
      }
```

```
      ========================================================================

      op_ima_obj_pos_get (op_id_from_name (op_topo_parent (op_topo_parent
(op_id_self ())), OPC_OBJTYPE_NODE_FIX, target_node), &latitude,
&longitude,&altitude, &x_pos, &y_pos, &z_pos);

      op_ima_obj_attr_set (op_id_from_name (op_id_from_name (op_topo_parent
(op_topo_parent (op_id_self ())), OPC_OBJTYPE_NODE_MOB, source_node),
OPC_OBJTYPE_ANT, "wimax_ant_0_0"), "target latitude", latitude);

      op_ima_obj_attr_set (op_id_from_name (op_id_from_name (op_topo_parent
(op_topo_parent (op_id_self ())), OPC_OBJTYPE_NODE_MOB, source_node),
OPC_OBJTYPE_ANT, "wimax_ant_0_0"), "target longitude", longitude);

      op_ima_obj_attr_set (op_id_from_name (op_id_from_name (op_topo_parent
(op_topo_parent (op_id_self ())), OPC_OBJTYPE_NODE_MOB, source_node),
OPC_OBJTYPE_ANT, "wimax_ant_0_0"), "target altitude", altitude);

      }

      //==========================SS_END====================


      //=========================CODE_END==============================


      //================Implementation of directional_Antenna==============



      }


      //===============CODE_INSERT_END=============
```

Appendix B

Topology Control algorithm-1

The Projects "Algorithm_1_wimax_project" and "Algorithm_2_wimax_project" have .cpp files for execution. They have been run in Microsoft Visual Studios 2005.

The Project titled "Algorithm_1_wimax_project" contains the soucre code for the implementation of Algorithm_1 of topology control.

The program takes the location of RS nodes and SS nodes as the input along with the 'maximum communication distance', which limits the communication range in the network. Before running the program, we need to know the number of RSs and number of SSs participating in the network. We need to enter these values in the top line sof the code, where we define the values for 'tot_rs' and 'tot_ss'.

We also can change the 'antenna_beam' value to any desire value, which is presently set to 40. We need to change the 'beam_max' value to the 'antenna_beam' value.

We have a variable 'value constant' which used for the purpose of calculating the data rates, we can set this to any convenient value.

The program runs in a while-loop till all the nodes are connected.

The program finds an RS for each SS, running in a for-loop for iterations eual to number of SS nodes.

In each pass, we find an RS for an SS which has the bottleneck rate.

Bottleneck rate is the minimum rate in the list maximum possible rate for each SS.

Funciton used are:

double get_distance(double x1, double y1, double x2, double y2);

It takes the co-ordinates of the Node 1 and Node 2 and returns the distance between them.

double calculate_angle(double x1, double y1, double x_p, double y_p, double x2, double y2);

This is simple function which gives the angle between the 3 points.

double calculate_rs_beam(int cur_rs, int cur_ss);

This function takes the RS numebr and SS numebr as input and gives the angle this RS will need to have to cover the SSs under it, including the cur_ss. This is used to see, whether the COVERAGE_angle get more than the antenna_beam angle.

Key Variables:

ss_candidate_rs[][] gives the value 1 or 0 to indicate if the RS is a candidate for the SS.

ss_rs_association[][] ss_rs_association[][] are maintained to keep track of the assigned SSs and their associated RSs.

rs_beam[] has the curernt coverage angles of all the RSs.

all_connected is an integer variable which works a flag for while-loop to run, till all the nodes get an RS.

If the program does terminate, it means with current location of nodes and antenna_beam, there is no feasible solution such that every SS is covered by an RS.

Otherwise, program will end with printing the each SS number and corresponding assigned RS.

Code for topology control algorithm-1

```
// algorithm_AAA_1.cpp : Defines the entry point for the console application.
//

#include<stdio.h>
#include <math.h>
//#include "fstream.h"


#define tot_bs  1
#define tot_rs  2                    //Give the total number of RSs in your network
#define tot_ss  4                    //Give the total number of SSs in your network
```

```
double get_distance(double x1, double y1, double x2, double y2);                    //function prototype...

double calculate_angle(double x1, double y1, double x_p, double y_p, double x2, double y2); //function prototype...

double calculate_rs_beam(int cur_rs, int cur_ss);                                    //function prototype...



        //========== input parameters...


    //Total number of nodes...


        int tot_nodes = tot_bs + tot_rs + tot_ss;


        //Node Properties...

        struct node

        {

                int id;

                double x,y;

        };


        //Location of BS...

        double bs_x;

        double bs_y;


        //Location of RSs...
```

```
        double rs_x[tot_rs+1];
        double rs_y[tot_rs+1];

        //Location of SSs...
        double ss_x[tot_ss+1];
        double ss_y[tot_ss+1];

        //IDs of nodes...
        int bs_id;
        int rs_id[tot_rs+1];
        int ss_id[tot_ss+1];

        //Number of cadidate RSs for each SS...
        int ss_num_rs[tot_ss+1]={0};
        //List of cadidate RSs...
        int ss_candidate_rs[tot_ss+1][tot_rs+1]={0};

        //SS ---> RS association...
        int ss_rs_association[tot_ss+1][tot_rs+1]={0};
    //RS ---> SS association...
        int rs_ss_association[tot_rs+1][tot_ss+1]={0};

        //SS ---> RS related...
        int ss_rs_related[tot_ss+1][tot_rs+1]={0};
```

```
//Status of connection of an SS...
int ss_connected[tot_ss+1]={0};

//Rate of each SS...
double ss_data_rate[tot_ss+1]={0};

double ss_max_rate[tot_ss+1]={0};


//Beam of antenna at an RS...
double rs_beam[tot_rs+1]={0};

//Data-rates... for bottleneck values...
double max_rate, current_rate, bottleneck_rate,ss_current_rate,ss_min_rate;
int current_rs, current_ss;

//Check if all SS are connected...
int all_connected, connect_flag;

//Distance variable...
double distance=0, rs_distance=0;

//Constant value for calculation of 'Ri'...
double value_constant;
```

```
        //Angle in question....
        double current_angle=360;

        double max_angle;
double update_beam;

        //communication range...
        double d_max;

        //Minimum  and Maximum beamwidth...
        double beam_min;
        double beam_max;

        //Minimum data rate...
        double R_min;

        //Antenna beamwidth...
        double antenna_beam = 40;

        //=================================


        int max_rate_rs;
        double max_rate_distance;
        //double max_rate;
```

```
        int cand_one_flag;



int main(int argc, char* argv[])
{

                printf("\n");
    // printf("Enter BS node X_location, Y_location: ");
                //scanf("%lf  %lf", &bs_x, &bs_y);
                //printf("\n");
                //printf("BS node ========= Location X,Y : ( %lf , %lf )\n",bs_x, bs_y);

        for(int i=1; i<=tot_rs ; i++)
        {
                printf("\n");
                printf("\n");
                printf("Enter RS_%d node X_location, Y_location: ", i);
                scanf("%lf  %lf", &rs_x[i], &rs_y[i]);

        }

        printf("\n");
        printf("The RS node locations are: \n");
        for(int i=1; i<=tot_rs ; i++)
        {
```

```
    printf("\n");
    printf("RS_%d ========= Location X,Y : ( %lf , %lf )\n",i,rs_x[i],rs_y[i]);
    }

    for(int i=1; i<=tot_ss ; i++)
    {
            printf("\n");
            printf("\n");
printf("Enter SS_%d node X_location, Y_location: ", i);
            scanf("%lf  %lf", &ss_x[i], &ss_y[i]);

    }

    printf("The SS node locations are: \n");
    for(int i=1; i<=tot_ss ; i++)
    {
    printf("\n");
    printf("SS_%d ========= Location X,Y : ( %lf , %lf )\n",i,ss_x[i],ss_y[i]);
    }

    printf("\n");
    printf("Enter the value of d_max (maximum communication distance) : ");
    scanf("%lf", &d_max);
    printf("\n");
```

```
printf("\n");

//Giving Parameter values for calculations...
beam_min = 0;
beam_max = 40;
value_constant = 10000000;

//Determine the number of candidate RSs for each SS...

for(int i=1; i<=tot_ss ; i++)
{
double x1 = ss_x[i];
double y1 = ss_y[i];

for(int j=1; j<=tot_rs ; j++)
{
double x2 = rs_x[j];
double y2 = rs_y[j];

distance = get_distance(x1,y1,x2,y2);

if(distance <= d_max)
{
ss_candidate_rs[i][j] = 1;
ss_num_rs[i]++;
```

```
}

}//inner for_loop

}//outer for_loop

//Display the Candidate RSs for each SS...

for(int i=1 ; i<=tot_ss ; i++)
{
        for(int j=1; j<=tot_rs ; j++)
        {
                        if(ss_candidate_rs[i][j] == 1)
                        {
                                printf("\n");
                                printf("The SS#%d has RS#%d as candidate RS\n",i, j);

                        }
        }

}

//Associate the SSs which has only 1 candidate RS...
cand_one_flag = 0;
```

```
for(int i=1 ; i<=tot_ss ; i++)
{
if(ss_num_rs[i] == 1)
        cand_one_flag = 1;
}

if(cand_one_flag == 1)
{
printf("\n");
printf("These SSs have only one Candidate RS\n");
for(int i=1 ; i<=tot_ss ; i++)
{
        if(ss_num_rs[i] == 1)
        {
                for(int j=1; j<=tot_rs ; j++)
                {
                        if(ss_candidate_rs[i][j] == 1)
                        {


                                if(1)
                                {
                                ss_rs_association[i][j] = 1;
                                ss_connected[i]=1;
                                rs_ss_association[j][i] = 1;
```

```
                                        double x1 = ss_x[i];
                                        double y1 = ss_y[i];

                                        double  x_p = rs_x[j];
                                        double  y_p = rs_y[j];

                                        distance = get_distance(x_p,y_p,x1,y1);
                                        ss_data_rate[i] = value_constant / (distance * distance *
antenna_beam);

                                        printf("\n");
                                        printf("The SS#%d connects to RS#%d and has a data_rate
= %lf\n",i,j,ss_data_rate[i]);
                                        }

                                }
                        }

                }

        }//outer for_loop
        }

        printf("\n");
        printf("The following SSs have more than 1 RSs to choose from\n");
```

```
for(int i=1 ; i<=tot_ss ; i++)
{
if(ss_num_rs[i] > 1)
        printf("\n SS#%d",i);

}
printf("\n");

//Trying to get the remaining SSs get connected to an RS... (from the list of candidate RSs)...
//Following the criteria that the angle formed is ---less than --- antenna_beam ---  and --- R_min--- is maximized....

all_connected = 0;
while(all_connected != 1)
{

//Erasing all the dotted lines...
for(int i=1 ; i<=tot_ss ; i++)
        for(int j=1 ; j<=tot_rs ; j++)
        {
                ss_rs_related[i][j] = 0;
        }//for_loop 0.5 ...

        ss_current_rate = 100;
                ss_min_rate = 100;
```

```
//top_for_loop 1
for(int i=1 ; i<=tot_ss ; i++)
{
        if(ss_connected[i] == 1)
        {

        }
        else
        {
                max_rate = 0;
                current_rate = 0;
                current_rs = 0;

                ss_current_rate = 100;
                //ss_min_rate = 100;

                max_rate = 0;

                for(int j=1 ; j<=tot_rs ; j++)
                {
                if(ss_candidate_rs[i][j] == 1)
                {
                current_angle = calculate_rs_beam(j,i);
```

```
//      ========================

                printf("\n");

                printf("The current COVERAGE_angle of RS#%d is %lf \n",j,current_angle);


                        if(current_angle > antenna_beam)

                        {

                                printf("\n$$$$$$$$$$$$$$$$$$$");

                           printf("This combination is NOT POSSIBLE... angle made will exceed
the ANTENNA BEAM!!!\n");

                                printf("$$$$$$$$$$$$$$$$$$$");

                                printf("\n");

                        }


                        else if(current_angle <= antenna_beam)

                        {


                        double x1 = ss_x[i];

                        double y1 = ss_y[i];


                        double  x_p = rs_x[j];

                        double  y_p = rs_y[j];


                        distance = get_distance(x_p,y_p,x1,y1);

                   ss_current_rate = value_constant / (distance * distance * antenna_beam);
```

```
                        printf("\n");

                        printf("SS#%d will have rate = %lf  with  RS#%d at a distance =
%lf\n",i,ss_current_rate,j,distance);


                                        if(ss_current_rate > max_rate)

                                        {

                                                max_rate = ss_current_rate;

                                                max_rate_rs = j;

                                                max_rate_distance = distance;

                                        }

                        }


//          ============================

                        }//if...


                        }//for_loop...



                printf("\n");


        printf("\n=============================================================
==============\n");

                printf("Finally when SS#%d chooses RS#%d which has distance = %lf ---
Max.rate = %lf\n",i,max_rate_rs,max_rate_distance,max_rate);


        printf("\n=============================================================
==============\n");
```

```
            ss_max_rate[i]=max_rate;
            ss_rs_related[i][max_rate_rs]= 1;   //dotted line between SS and RS.... for identifying the max_rate_rs...

            } // else 1...
        } //top_for_loop 1

        //Now, all atleast related to an RS...
        //We find SS with the bottleneck 'Ri' and connect it to its related RS...

        current_rate = 100;
        bottleneck_rate = 100;
        for(int i=1 ; i<=tot_ss ; i++)
        {

            for(int j=1 ; j<=tot_rs ; j++)
            {

            if(ss_rs_related[i][j] == 1)
            {
                current_rate = ss_max_rate[i];
                if(current_rate <= bottleneck_rate)
                {
                    current_ss = i;
                    current_rs = j;
```

```
                        bottleneck_rate = current_rate;
                    }

                }//if 1...

                }//for_loop 2...

        }//for_loop 1...

        printf("\n");
        printf("\n||||||||||||||||||||||||||||||||||||||||||||||||\n");
        printf("The BOTTLENECK RATE is %lf  when SS: %d is CONNECTED with RS#%d\n", ss_max_rate[current_ss],current_ss,current_rs);
        printf("\n||||||||||||||||||||||||||||||||||||||||||||||||\n");

        //Associating the bottleneck SS to its RS...

        ss_rs_association[current_ss][current_rs] = 1;
        ss_connected[current_ss]=1;
        rs_ss_association[current_rs][current_ss] = 1;


        //Updating the rs_beam for current COVERAGE angle...
        rs_beam[current_rs] = calculate_rs_beam(current_rs,current_ss);
```

```
printf("\n");

printf("Just after associating the bottleneck SS to RS.... rs_beam of RS#%d is %lf\n", current_rs, rs_beam[current_rs]);


//Update the ss_data_rate array of all the SSs associated witht the current_RS...

for( int j=1 ; j<=tot_rs ; j++)

{

for( int n=1 ; n<=tot_ss ; n++)

{


                        if(rs_ss_association[j][n] == 1)

                        {

                                double x1 = ss_x[n];

                                double y1 = ss_y[n];


                                double  x_p = rs_x[j];

                                double  y_p = rs_y[j];


                                distance = get_distance(x_p,y_p,x1,y1);

                           ss_data_rate[n] = value_constant / (distance * distance * antenna_beam);

                                printf("\n");

                                printf("The data rate of associated SS#%d with RS#%d is %lf\n",n,j,ss_data_rate[n]);


                        }//if...
```

```
	}
	}//for_loop for UPDATE...

	printf("\n ----------------------------------------------------      \n");

	connect_flag = 1;
	for(int i=1 ; i<=tot_ss ; i++)
	{
			if(ss_connected[i] != 1)
				connect_flag =0;

	}//for_loop 0...

	if(connect_flag == 0)
	{
		printf("\n");
		printf("NOT ALL SSs HAVE AN RS... WILL REPEAT THE STEPS FOR REMAINING SSs...\n");
		printf("\n");
	}
	all_connected = connect_flag;

	printf("\n");
	printf("============================================================================\n");
```

```
        printf("============================================================
=====================\n");


        }//while_loop 1... at top...


        printf("\n");
        printf("AT this point we have found a feasible solution...\n");
        printf("The RS-SS assignment is as follows:\n");
        printf("\n");


    for( int i=1 ; i<=tot_ss ; i++)
        {
        for( int j=1 ; j<=tot_rs ; j++)
        {

                                if(rs_ss_association[j][i] == 1)
                                {
                                        printf("\n");
                                        printf("SS#%d is assigned RS#%d\n",i,j);
                                }
        }
        }


        printf("\n");
        printf("---------- END----------\n");
```

```
	return 0;
} // main function...


//Function to calculate the distance between 2 points...
double get_distance(double x1, double y1, double x2, double y2)
{
double distance = sqrt(((x2-x1) * (x2-x1)) + ((y2-y1) * (y2-y1)));
return distance;
}


//Function for calculating the angle between 3 nodes...
double calculate_angle(double x1, double y1, double x_p, double y_p, double x2, double y2)
{
	double result;

// calculating the 3 distances

double ab = get_distance(x1,y1,x_p,y_p);

double bc = get_distance(x_p,y_p,x2,y2);

double ac = get_distance(x1,y1,x2,y2);
```

```
double cosB = pow(ac, 2) - pow(ab, 2) - pow(bc, 2);

cosB =  cosB / (2 * ab * bc);

result = 180 - ((acos(cosB) * 180) /3.141592653); //(2 * acos(0.0))



return result;
}

double calculate_rs_beam(int cur_rs, int cur_ss)
{
                        int j = cur_rs;
                        int rs_assoc_flag = 0;
                        for(int i=1; i<=tot_ss ; i++)
                        {
                                if(rs_ss_association[j][i] == 1)
                                {
                                        rs_assoc_flag = 1;
                                }
                        }

                        if(rs_assoc_flag == 0)
                        {
                                //rs_beam[j] = beam_min;
```

```
                return beam_min; //minimum beam...
        }
        else
        {
                max_angle=0;
                for( int n=1 ; n<=tot_ss ; n++)
                {

                if(rs_ss_association[j][n] == 1)
                {
                        double x1 = ss_x[n];
                        double y1 = ss_y[n];

                        double  x_p = rs_x[j];
                        double  y_p = rs_y[j];

                        double  x2 = ss_x[cur_ss];
                        double  y2 = ss_y[cur_ss];

                        current_angle = calculate_angle(x1,y1,x_p,y_p,x2,y2);
                        //printf("\n");
                        //printf("The angle between SS#%d --- RS#%d --- SS#%d
is = %lf\n", n,j,cur_ss,current_angle);

                        if(current_angle >= max_angle)
```

```
                {
                        max_angle=current_angle;
                }
        }//if...
        }//for_loop...


                return max_angle;



        }

}
```

Appendix C

Topology Control algorithm-2

The Project titled "Algorithm_2_wimax_project" contains the soucre code for the implementation of Algorithm_2 of topology control.

The input to the program is the locations of the BS, RS and SS nodes. Then we shall also give a valid RS assignment for each SS.

This asignment should be valid in terms of the coverage of the angle. Since, the input ot this program is usually the output of algorithm_1, it complies with all the criteria of validation.

We again give the 'maximum communication distance' as in the case of algorithm_1. All the basic variable and functions are same as the algorithm_1 and does similar jobs. We should follow the same basic steps for using the algorithm_2 as for algorithm_1.

But, not all function or instruction may be used in the running of the program as this algorithm is a special case of the implementation of the program.

Key Functions are:

double fix_cur_ss(int move, int rs);

This function tries for a new assignment for SS(move) which is currently assigned to RS(rs) and returns the value of success as 1 and failure as -1.

double disconnect_ss_move(int ss_move, int rs_leaving);

This function disconnects the SS (ss_move) for the current RS(rs_leaving).

int get_cur_rs(int ss_move);

This funciton returns the number of RS, which is currentlt assigned the SS.

void update_rs_beam(int rs_leaving)

The function updates the angle of coverage for an RS, when a new SS is added to its list.

void delete_update_rs_beam(int rs_leaving);

The function updates the angle of coverage for an RS, when a SS is removed from its list.

void update_SS_data_rates(void);

Thsi updates the date rate of all the SSs in the network.

double calculate_rs_beam_1(int cur_rs, int cur_ss);

This function does the same job as 'calculate_rs_beam' in algorithm_1.

Key Variables:

ss_rs_blue_solid[][] indicates if the SS and RS are assigned to each other and are connected. Gives 1 for success and 0 for failure.

ss_rs_dotted[][] similarly indicates the current approach of an SS fot a candidate RS. The two nodes are not connected, but the RS considers the SS in the coverage list for determining the feasibility.

ss_rs_red[][] this indicates the back-tracking of the SS relation, when it moves from one candidate RS to other for getting improved data rate. If the value for an SS and RS is 1, they

would be connected to each other if and when there is solution found. Consequently, if there is no solution found, then this possible link, is erased setting the value to 0.

coverage_flag is used to check if the coverage angle of the RS is more than the allowed angle, whcih is equal to the antenna_beam.

min_data_rate_flag is used to check, if we can consider this candidate RS for assignment such that it improves the minimum data rate of the network.

Code for topology control algorithm-2

```
// algorithm_BBB_3.cpp : Defines the entry point for the console application.
//

#include<stdio.h>
#include<math.h>

#define tot_bs  1
#define tot_rs  3                    //Give the total number of RSs in your network
#define tot_ss  6                    //Give the total number of SSs in your network

// Function Prototypes...

double fix_cur_ss(int move, int rs);                    // function prototype...
double calculate_SS_R_min(void);
```

```
int select_ss_move(int cur_rs,int cur_ss);
double disconnect_ss_move(int ss_move, int rs_leaving);
int get_cur_rs(int ss_move);
void update_rs_beam(int rs_leaving);
double get_distance(double x1, double y1, double x2, double y2);
void update_SS_data_rates(void);
double calculate_angle(double x1, double y1, double x_p, double y_p, double x2, double y2);
double calculate_rs_beam(int cur_rs, int cur_ss);
double calculate_rs_beam_1(int cur_rs, int cur_ss);

void delete_update_rs_beam(int rs_leaving);
double delete_calculate_rs_beam(int cur_rs, int cur_ss);

//========== input parameters...

        //Total Nodes...
   int tot_nodes = tot_bs + tot_rs + tot_ss;

        //Node Properties...
        struct node
        {
                int id;
                double x,y;
        };
```

```
//Location of BS...
double bs_x;
double bs_y;

//Location of RSs...
double rs_x[tot_rs+1];
double rs_y[tot_rs+1];

//Location of SSs...
double ss_x[tot_ss+1];
double ss_y[tot_ss+1];

//IDs of nodes...
int bs_id;
int rs_id[tot_rs+1];
int ss_id[tot_ss+1];

//ID of SS with bottleneck rate...
int ss_R_min_id;

//current min_rate of all SSs...
double ss_R_min;

//The SS on move.. which we are trying to fix/hook...
int ss_move;
```

```
	//The RS number, which the current SS is leaving from for better Ri...
	int rs_leaving;

	//The number of connected SSs of an RS...
	int num_connected_ss[tot_rs+1]={0};

	//The number of connected SSs of an RS...
	int num_associated_ss[tot_rs+1]={0};



	//Number of cadidate RSs for each SS...
	//int ss_num_candidate_rs[tot_ss+1]={0};
	int ss_num_rs[tot_ss+1]={0};

	//List of candidate RSs...
	int ss_candidate_rs[tot_ss+1][tot_rs+1]={0};

	//If the SS has tried this RS...
	int rs_reached[tot_ss+1][tot_rs+1]={0};

	//SS ---> RS association...
	int ss_rs_association[tot_ss+1][tot_rs+1]={0};
//RS ---> SS association...
	int rs_ss_association[tot_rs+1][tot_ss+1]={0};
```

```
//SS ---> RS... BLUE_SOLID...
int ss_rs_blue_solid[tot_ss+1][tot_rs+1]={0};

//SS ---> RS... BLUE DOTTED...
int ss_rs_dotted[tot_ss+1][tot_rs+1]={0};

//SS ---> RS... RED...
int ss_rs_red[tot_ss+1][tot_rs+1]={0};

//Rate of each SS...
double ss_data_rate[tot_ss+1]={0};

//Beam of antenna at an RS...
double rs_beam[tot_rs+1]={0};

//Data-rates... for bottleneck values...
double R_min;

//Distance variable...
double distance=0;

//Constant value for calculation of 'Ri'...
double value_constant;
```

```
//Angle in question....
double current_angle=360;

double max_angle;

//communication range...
double d_max;

//Minimum  and Maximum beamwidth...
double beam_min;
double beam_max;

//flag for fix_ss return value...
double fix_return = 0;

//Givng the value for antenna beam...
double antenna_beam = 40;

//==================================


double current_rs_coverage;
double current_rate;
```

```
int main(int argc, char* argv[])
{
	//==============================Give
INPUT==================================


		printf("\n");
	printf("Enter BS node X_location, Y_location: ");
		scanf("%lf  %lf", &bs_x, &bs_y);

		printf("\n");
		//printf("The BS node ID is : %d\n",bs_id);
		printf("BS node ========= Location X,Y : ( %lf , %lf )\n",bs_x, bs_y);



	printf("\n");
	printf("BS node ========= Location X,Y : ( %lf , %lf )\n",bs_x, bs_y);

	for(int i=1; i<=tot_rs ; i++)
	{
		printf("\n");
		//printf("Enter RS_%d node ID: ", i);
		//scanf("%d", &rs_id[i]);
		printf("\n");
```

```
            printf("Enter RS_%d node X_location, Y_location: ", i);
            scanf("%lf  %lf", &rs_x[i], &rs_y[i]);

    }

    printf("\n");
    //printf("The RS node IDs and location are: \n");
    printf("The RS node locations are: \n");
    for(int i=1; i<=tot_rs ; i++)
    {
    printf("\n");
    printf("RS_%d ========== Location X,Y : ( %lf , %lf )\n",i,rs_x[i],rs_y[i]);
    }

    for(int i=1; i<=tot_ss ; i++)
    {
            printf("\n");
            //printf("Enter SS_%d node ID: ", i);
            //scanf("%d", &ss_id[i]);
            printf("\n");
printf("Enter SS_%d node X_location, Y_location: ", i);
            scanf("%lf  %lf", &ss_x[i], &ss_y[i]);

    }
```

```
//printf("The SS node IDs and location are: \n");
printf("The SS node locations are: \n");
for(int i=1; i<=tot_ss ; i++)
{
printf("\n");
printf("SS_%d ========= Location X,Y : ( %lf , %lf )\n",i,ss_x[i],ss_y[i]);
}

printf("\n");
printf("Enter the value of d_max (maximum communication distance) : ");
scanf("%lf", &d_max);
printf("\n");

//printf("Enter the value of Constant for calculating 'Ri'  : ");
//scanf("%lf", &value_constant);


//Givng values for parameters for calculation...
beam_min = 0;
beam_max = 40;
value_constant = 10000000;


printf("\n");
```

```
	printf("\n============================================================
===========\n");
	for(int i=1; i<=tot_ss; i++)
	{
		int rs_id;
		printf("\n");
		printf("Enter the parent RS number of SS#%d =  ",i);
		scanf("%d", &rs_id);

		printf("\n");
		printf("i is %d --- rs_id is %d \n",i, rs_id);
		ss_rs_blue_solid[i][rs_id] = 1;
		ss_rs_association[i][rs_id] = 1;
		rs_ss_association[rs_id][i] = 1;
		num_associated_ss[rs_id]++;
		num_connected_ss[rs_id]++;


	}
	printf("\n============================================================
===========\n");
	for(int i=1; i<=tot_ss; i++)
	{
		for(int j=1; j<=tot_rs; j++)
		{
			if(ss_rs_blue_solid[i][j] == 1)
```

```
                {
                        printf("\n");
                        printf("The SS#%d is connected to RS#%d\n",i,j);
                }
        }

}

printf("\n");
printf("\n==========\n");

//==========end_of_INPUTs==========

//Determine the number of candidate RSs for each SS...
for(int i=1; i<=tot_ss ; i++)
{
double x1 = ss_x[i];
double y1 = ss_y[i];

for(int j=1; j<=tot_rs ; j++)
{
double x2 = rs_x[j];
double y2 = rs_y[j];
```

```
distance = get_distance(x1,y1,x2,y2);

if(distance <= d_max)
{
ss_candidate_rs[i][j] = 1;
ss_num_rs[i]++;
}

}//inner for_loop

}//outer for_loop

//Display the Candidate RSs for each SS...

for(int i=1 ; i<=tot_ss ; i++)
{
        for(int j=1; j<=tot_rs ; j++)
        {
                        if(ss_candidate_rs[i][j] == 1)
                        {
                                printf("\n");
                                printf("The SS#%d has RS#%d as candidate RS\n",i, j);

                        }
        }
```

```
}

R_min=100;

for(int j=1; j<=tot_rs ; j++)
  update_rs_beam(j);

update_SS_data_rates();
R_min = calculate_SS_R_min();
ss_move = ss_R_min_id;
rs_leaving = get_cur_rs(ss_move);
//select_ss_move(i,ss_num);

printf("\n");
for(int i=1; i<=tot_ss; i++)
{
        printf("The data rate of SS#%d is = %lf \n", i, ss_data_rate[i]);
}

printf("\n");
for(int j=1; j<=tot_rs ; j++)
{
        printf("The RS#%d beam = %lf \n", j, rs_beam[j]);
}
```

```
printf("\n");

printf("The ss_move ====for the first time=== in MAIN... is SS#%d \n", ss_move);
printf("The min. rate is : %lf \n", R_min);
printf("The leaveing RS is RS#%d \n", rs_leaving);
printf("\n");



//fix_return = 1;
//WHILE(FIX_RETURN != 0)

fix_return = fix_cur_ss(ss_move, rs_leaving);

printf("\n");
printf("This is fix_return in MAIN... fix_return : %lf\n", fix_return);
//if_x2
if(fix_return > 0)
{
        for(int i=1; i<=tot_ss ; i++)
                for(int j=1; j<=tot_rs ; j++)
                {
                        //if_X3
                        if(ss_rs_dotted[i][j] == 1)
                        {
```

```
                    ss_rs_dotted[i][j] = 0;

                    ss_rs_blue_solid[i][j]=1;

                    ss_rs_association[i][j] = 1;
                    rs_ss_association[j][i] = 1;
                    num_associated_ss[j]++;
                    num_connected_ss[j]++;

                   for(int j=1; j<=tot_rs ; j++)
                  {
                        if(ss_rs_red[i][j] == 1)
                        {
                              ss_rs_red[i][j] = 0;
                        }
                  }

                 }//if_X3...
            }


}//if_X2...

//}//for_loop_X1...
```

```
        for(int i=1; i<=tot_ss ; i++)
                        for(int j=1; j<=tot_rs ; j++)
                        {
                                if(ss_rs_red[i][j] == 1)
                                {
                                        ss_rs_blue_solid[i][j]=1;


                                        ss_rs_association[i][j] = 1;
                                        rs_ss_association[j][i] = 1;
                                        num_associated_ss[j]++;
                                        num_connected_ss[j]++;
                                }
                        }



//update_rates...
for(int j=1; j<=tot_rs ; j++)
            update_rs_beam(j);


update_SS_data_rates();


//PRINTF BLUE EDGES RELATIONS.....
for(int i=1; i<=tot_ss ; i++)
                        for(int j=1; j<=tot_rs ; j++)
```

```
                                        printf("The BLUE_SOLID values for SS#%d and RS#%d = %d
\n", i , j, ss_rs_blue_solid[i][j]);


        return 0;

}



double fix_cur_ss(int my_ss_move, int my_rs_leaving)

{

        int my_cur_ss = my_ss_move;

   int my_cur_rs = my_rs_leaving;


        int ss_move;

        double value_return;


        printf("The current_ss is SS#%d\n",my_cur_ss);


        //erase the BLUE_SOLID...

        disconnect_ss_move(my_cur_ss, my_cur_rs);

        //Update the rs_beam.. since we have disconnected one SS node...


        //UPDATING AFTER DELETING A LINK!!!!!!!!!!!

        delete_update_rs_beam(my_cur_rs);
```

```
        //update_SS_data_rates();


        printf("After disconnecting... new COVERAGE beam value of RS#%d is = %lf \n",
my_cur_rs, rs_beam[my_cur_rs]);

        //Make a RED_LINE...

        ss_rs_red[my_cur_ss][my_cur_rs] = 1;


        printf("The my_cur_ss is SS#%d \n", my_cur_ss);

        //Check for the next candidate_RS... for this SS..


        for(int i=1; i<=tot_ss ; i++)

                        for(int j=1; j<=tot_rs ; j++)

                                printf("The BLUE_SOLID values for SS#%d and RS#%d = %d
\n", i , j, ss_rs_blue_solid[i][j]);


        printf("Trying here... current_ss is SS#%d and current_rs is RS#%d \n", my_cur_ss,
my_cur_rs);


    int next_rs = 0;

        int next_rs_flag = 0;

        int coverage_flag = 1;

        int min_data_rate_flag = 1;


        for(int j=1; j<=tot_rs ; j++)

        {
```

```
                printf("\n");

                printf("I'm going inside for_loop for checking the next_RS...\n");

                printf("\n");

                printf("The value of 'candidature' for SS#%d  and RS#%d is %d \n",my_cur_ss, j, ss_candidate_rs[my_cur_ss][j]);

                printf("\n");

                printf("The value of 'ss_rs_red' for SS#%d  and RS#%d is %d \n",my_cur_ss, j, ss_rs_red[my_cur_ss][j]);

                printf("\n");

                printf("The value of 'rs_reached' for SS#%d  and RS#%d is %d \n",my_cur_ss, j, rs_reached[my_cur_ss][j]);


                current_rs_coverage = calculate_rs_beam_1(j,my_cur_ss);


        printf("\n");

           printf("The current COVERAGE_angle of RS#%d is %lf \n",j,current_rs_coverage);


                if(current_rs_coverage > antenna_beam)

                {

                                printf("\n$$$$$$$$$$$$$$$$$$");

                            printf("This combination is NOT POSSIBLE... angle made will exceed the ANTENNA BEAM!!!\n");

                                printf("$$$$$$$$$$$$$$$$$$");

                                printf("\n");

                                coverage_flag = 0;

                }
```

```
                double  x_p = rs_x[j];
                double  y_p = rs_y[j];

                double  x2 = ss_x[my_cur_ss];
                double  y2 = ss_y[my_cur_ss];

                distance = get_distance(x_p,y_p,x2,y2);
                current_rate = value_constant / (distance * distance * antenna_beam);

        if(current_rate < R_min)
        {
                min_data_rate_flag = 0;
        }

        // values printed should be 1 0 0 1 1 ... for entering into the 'for_loop'....

        if((ss_candidate_rs[my_cur_ss][j] == 1) && (ss_rs_red[my_cur_ss][j] != 1) &&
(rs_reached[my_cur_ss][j] != 1) && (coverage_flag == 1) && (min_data_rate_flag == 1))
        {
                printf("\n");
                printf("I'm here at least once... for RS#%d \n", j);
                printf("\n");
                next_rs = j;

                rs_reached[my_cur_ss][j] = 1;
```

```
                printf("Updated the 'rs_reached' for SS#%d  and RS#%d is %d
\n",my_cur_ss, j, rs_reached[my_cur_ss][j]);


                next_rs_flag = 1;

        }


        if(next_rs_flag == 1)

                break;

}


//if_xyz_1

if(next_rs != 0 )

{

        my_cur_rs = next_rs;


        printf("Got a next_rs... now current RS is RS#%d \n", my_cur_rs);


        //making BLUE_DOTTED between my_cur_ss and my_cur_rs...

        ss_rs_dotted[my_cur_ss][my_cur_rs]=1;

        ss_rs_association[my_cur_ss][my_cur_rs]=1;

        rs_ss_association[my_cur_rs][my_cur_ss]=1;

        num_associated_ss[my_cur_rs]++;


        update_rs_beam(my_cur_rs);

        printf("After dotted_BLUE with next_rs... the beam value of RS#%d is = %lf \n",
my_cur_rs, rs_beam[my_cur_rs]);
```

```
update_SS_data_rates();
printf("\n");
for(int i=1; i<=tot_ss; i++)
{
	printf("Now... the data rate of SS#%d is = %lf \n", i, ss_data_rate[i]);
}

ss_R_min = calculate_SS_R_min();

printf("\n");
printf("The lowest data rate now is = %lf \n", ss_R_min);

//if_xyz_2
if( ss_R_min > R_min)
{
	printf("\n");
	printf("The lowest data rate now is GREATER than R_min= %lf \n",
R_min);

	for(int i=1; i<=tot_rs ; i++)
	{
		if(ss_rs_red[my_cur_ss][i] == 1)
		{
```

```
                        ss_rs_red[my_cur_ss][i] = 0;
                }

        }

        ss_rs_dotted[my_cur_ss][my_cur_rs]=0;

        ss_rs_blue_solid[my_cur_ss][my_cur_rs]=1;

        ss_rs_association[my_cur_ss][my_cur_rs]=1;
        rs_ss_association[my_cur_rs][my_cur_ss]=1;
        num_associated_ss[my_cur_rs]++;
        num_connected_ss[my_cur_rs]++;

        printf("RETURNing here... found a new assignment with better
MIN_RATE\n");
        return 1;

    }//if_xyz_2...

    //else_xyz_2
    else
    {
        printf("\n");
        printf("The lowest data rate now is SMALLER than R_min= %lf \n",
R_min);
```

```
                //select an ss from my_cur_rs...
                printf("\n");
                printf("Now.. we need to move another SS... from current RS#%d \n",
my_cur_rs);
                ss_move = select_ss_move(my_cur_rs, my_cur_ss);
                printf("The selected ss_move is SS#%d \n", ss_move);
                value_return = fix_cur_ss(ss_move, my_cur_rs);

        }//else_xyz_2...

        //if_ret_-1
        if(value_return == -1)
        {
                for(int i=1; i<=tot_rs ; i++)
                {
                        if(ss_rs_red[my_cur_ss][i] == 1)
                        {
                                ss_rs_red[my_cur_ss][i] = 0;
                        }

                }

                ss_rs_blue_solid[ss_move][my_cur_rs]=1;

                ss_rs_association[ss_move][my_cur_rs]=1;
```

```
		rs_ss_association[my_cur_rs][ss_move]=1;
		num_associated_ss[my_cur_rs]++;
		num_connected_ss[my_cur_rs]++;

		//update_rs_beam(my_cur_rs);

		ss_rs_dotted[my_cur_ss][my_cur_rs] = 0;

		ss_rs_association[my_cur_ss][my_cur_rs]=0;
		rs_ss_association[my_cur_rs][my_cur_ss]=0;
		num_associated_ss[my_cur_rs]--;

		update_rs_beam(my_cur_rs);

}//if_ret_-1...

//if_ret_1
if(value_return == 1)
{
		for(int i=1; i<=tot_rs ; i++)
		{
			if(ss_rs_red[my_cur_ss][i] == 1)
			{
				ss_rs_red[my_cur_ss][i] = 0;
			}
```

```
                    }


                    ss_rs_dotted[my_cur_ss][my_cur_rs]=0;


                    printf("\n");
                    printf("RETURNing 1 here... found a solution...MIN_RATE is
betterd!!!\n");


                    printf("\n");
                    printf("The SS#%d is now connected to RS#%d \n", my_cur_ss,
my_cur_rs);


                    //got to see and proceed from here...


                    //   .
                    //   .
                    //   .
                    //   .
                    //   .
                    //   .


                    ss_rs_blue_solid[my_cur_ss][my_cur_rs]=1;
```

```
                ss_rs_association[my_cur_ss][my_cur_rs]=1;
                rs_ss_association[my_cur_rs][my_cur_ss]=1;
                num_associated_ss[my_cur_rs]++;
                num_connected_ss[my_cur_rs]++;

                update_rs_beam(my_cur_rs);


            }//if_ret_1...

        }//if_xyz_1...

        //else_xyz_1
        else
        {
            printf("\n");
            printf("Flopping here... current_ss is SS#%d and current_rs is RS#%d \n",
my_cur_ss, my_cur_rs);
            for(int j=1; j<=tot_rs ; j++)
            {
                rs_reached[my_cur_ss][j] = 0;
            }

            return -1;
        }//else_xyz_1...
```

```
}


double calculate_SS_R_min(void)
{
	double ss_R_min = 100;

	//for_loop_1
	for(int i=1; i<=tot_ss; i++)
	{
		if( ss_R_min >= ss_data_rate[i])
		{
		ss_R_min_id = i;
		ss_R_min = ss_data_rate[i];
		}//if...

	}//for_loop_1...
	//ss_move = ss_R_min_id;

	return ss_R_min;


}

int select_ss_move(int cur_rs,int cur_ss)
```

```
{
                int j = cur_rs;
                int i=0;

                max_angle=0;

                for( int n=1 ; n<=tot_ss ; n++)
                {

                        if((rs_ss_association[j][n] == 1) && (n != cur_ss))
                        {
                                double x1 = ss_x[n];
                                double y1 = ss_y[n];

                                double  x_p = rs_x[j];
                                double  y_p = rs_y[j];

                                double  x2 = ss_x[cur_ss];
                                double  y2 = ss_y[cur_ss];

                                current_angle = calculate_angle(x1,y1,x_p,y_p,x2,y2);
                                //printf("\n");
                                //printf("The angle between SS#%d --- RS#%d --- SS#%d
is = %lf\n", n,j,cur_ss,current_angle);
                        if(current_angle >= max_angle)
```

```
                                {
            i = n;
                                        max_angle = current_angle;
                                }

                                }//if...
                        }//for_loop...

                        return i;

}

double disconnect_ss_move(int ss_move, int rs_leaving)
{

        if(ss_rs_association[ss_move][rs_leaving] == 1)
        {

                ss_rs_association[ss_move][rs_leaving]=0;
                rs_ss_association[rs_leaving][ss_move]=0;
                num_associated_ss[rs_leaving]--;
                num_connected_ss[rs_leaving]--;

                ss_rs_blue_solid[ss_move][rs_leaving]=0;
```

```
                return 1;
        }
        else
                return 0;
}

int get_cur_rs(int ss_move)
{

        for( int n=1 ; n<=tot_rs ; n++)
        {

                                if(ss_rs_association[ss_move][n] == 1)
                                {


                                        //ss_rs_association[ss_move][n]=0;
                                        //rs_ss_association[n][ss_move]=0;
                                        //num_connected_ss[n]--;
                                        //ss_rs_red[ss_move][n]=1;

                                        return n;
                                }
        }
```

```
        return 0;
}



void update_rs_beam(int rs_leaving)
{

        int ss_of_rs;

   if(num_associated_ss[rs_leaving] == 1)
        {
                rs_beam[rs_leaving] = beam_min;
        }
        else if(num_associated_ss[rs_leaving] > 1)
        {
                ss_of_rs=0;
                for(int i=1; i<=tot_ss ; i++)
                {
                        if(rs_ss_association[rs_leaving][i] == 1)
                        {
                                ss_of_rs = i;
                        }
                }
                rs_beam[rs_leaving]= calculate_rs_beam(rs_leaving,ss_of_rs);
```

```
        }
        else if(num_associated_ss[rs_leaving] == 0)
        {
                rs_beam[rs_leaving] = 0;
        }



}

void delete_update_rs_beam(int rs_leaving)
{

        int ss_of_rs;

   if(num_connected_ss[rs_leaving] == 1)
        {
                rs_beam[rs_leaving] = beam_min;
        }
        else if(num_connected_ss[rs_leaving] > 1)
        {
                ss_of_rs=0;
                for(int i=1; i<=tot_ss ; i++)
                {
                        if(rs_ss_association[rs_leaving][i] == 1)
                        {
```

```
                        ss_of_rs = i;
                    }
                }
                rs_beam[rs_leaving]= delete_calculate_rs_beam(rs_leaving,ss_of_rs);

        }
        else if(num_connected_ss[rs_leaving] == 0)
        {
                rs_beam[rs_leaving] = 0;
        }

}

//Function to calculate the distance between 2 points...
double get_distance(double x1, double y1, double x2, double y2)
{
double distance = sqrt(((x2-x1) * (x2-x1)) + ((y2-y1) * (y2-y1)));
return distance;
}

void update_SS_data_rates(void)
{
        double current_rate;
```

```
	for(int i=1; i<=tot_ss; i++)
	{
		for(int j=1; j<=tot_rs; j++)
		{
			if(ss_rs_association[i][j] == 1)
			{

			current_angle = calculate_rs_beam(j,i);
			double  x_p = rs_x[j];
			double  y_p = rs_y[j];

			double  x2 = ss_x[i];
			double  y2 = ss_y[i];

			distance = get_distance(x_p,y_p,x2,y2);
			current_rate = value_constant / (distance * distance * antenna_beam);

			ss_data_rate[i] = current_rate;

			}//if...
		}//for...
	}//for...
}
```

```
//Function for calculating the angle between 3 nodes...
double calculate_angle(double x1, double y1, double x_p, double y_p, double x2, double y2)
{
        double result;

// calculating the 3 distances

double ab = get_distance(x1,y1,x_p,y_p);

double bc = get_distance(x_p,y_p,x2,y2);

double ac = get_distance(x1,y1,x2,y2);

double cosB = pow(ac, 2) - pow(ab, 2) - pow(bc, 2);

cosB =  cosB / (2 * ab * bc);

result = 180 - ((acos(cosB) * 180) /3.141592653); //(2 * acos(0.0))


return result;
}


//calculating the RS_beam.... after adding a link...
```

```
//=====================================================

double calculate_rs_beam(int cur_rs, int cur_ss)
{
                int j = cur_rs;
                int rs_assoc_flag = 0;
                for(int i=1; i<=tot_ss ; i++)
                {
                        if(rs_ss_association[j][i] == 1)
                        {
                                rs_assoc_flag = 1;
                        }
                }

                if(rs_assoc_flag == 0)
                {
                        //rs_beam[j] = beam_min;
                        return beam_min; //minimum beam...
                }
                else
                {
                        max_angle=0;
                        for( int n=1 ; n<=tot_ss ; n++)
                        {
```

```
if(rs_ss_association[j][n] == 1)
{
	double x1 = ss_x[n];
	double y1 = ss_y[n];

	double  x_p = rs_x[j];
	double  y_p = rs_y[j];

	double  x2 = ss_x[cur_ss];
	double  y2 = ss_y[cur_ss];

	current_angle = calculate_angle(x1,y1,x_p,y_p,x2,y2);
	//printf("\n");
	//printf("The angle between SS#%d --- RS#%d --- SS#%d is = %lf\n", n,j,cur_ss,current_angle);

	if(current_angle >= max_angle)
	{
		max_angle=current_angle;
	}
}//if...
}//for_loop...

if(max_angle <= beam_max)
{
```

```
                        if(max_angle >= rs_beam[j])
                        {
                                //rs_beam[j]=max_angle;
                                return max_angle;
                        }
                        else return rs_beam[j];

                        }
                        else
                                return -1;

                }
}

//calculating the RS_beam.... after deleting a link...
//=====================================================

double delete_calculate_rs_beam(int cur_rs, int cur_ss)
{
                int j = cur_rs;
                int rs_assoc_flag = 0;
```

```
for(int i=1; i<=tot_ss ; i++)
{
        if(rs_ss_association[j][i] == 1)
        {
                rs_assoc_flag = 1;
        }
}

if(rs_assoc_flag == 0)
{
        //rs_beam[j] = beam_min;
        return beam_min; //minimum beam...
}
else
{
        max_angle=0;
        //for_1...
        for( int n1=1 ; n1<=tot_ss ; n1++)
        {
                //if_1...
        if(rs_ss_association[j][n1] == 1)
        {
        //for_2...
        for( int n2=1 ; n2<=tot_ss ; n2++)
        {
```

```
                //if_2...
            if(rs_ss_association[j][n2] == 1 && n1 != n2)
            {
                double x1 = ss_x[n1];
                double y1 = ss_y[n1];

                double  x_p = rs_x[j];
                double  y_p = rs_y[j];

                double  x2 = ss_x[n2];
                double  y2 = ss_y[n2];

                current_angle = calculate_angle(x1,y1,x_p,y_p,x2,y2);
                //printf("\n");
                //printf("The angle between SS#%d --- RS#%d --- SS#%d
is = %lf\n", n,j,cur_ss,current_angle);

                if(current_angle >= max_angle)
                {
                        max_angle=current_angle;
                }
            }//if_2...
            }//for_loop_2...
            }//if_1...
            }//for_loop_1...
```

```
                        if(max_angle <= beam_max)
                        {

                        //if(max_angle >= rs_beam[j])
                        //{
                                //rs_beam[j]=max_angle;
                                //return max_angle;
                        //}
                                return max_angle;

                        //else return rs_beam[j];

                        }
                        else
                                return -1;



                }
}


double calculate_rs_beam_1(int cur_rs, int cur_ss)
{
                int j = cur_rs;
```

```
int rs_assoc_flag = 0;
for(int i=1; i<=tot_ss ; i++)
{
        if(rs_ss_association[j][i] == 1)
        {
                rs_assoc_flag = 1;
        }
}

if(rs_assoc_flag == 0)
{
        //rs_beam[j] = beam_min;
        return beam_min; //minimum beam...
}
else
{
        max_angle=0;
        for( int n=1 ; n<=tot_ss ; n++)
        {

        if(rs_ss_association[j][n] == 1)
        {
                double x1 = ss_x[n];
                double y1 = ss_y[n];
```

```
                double  x_p = rs_x[j];
                double  y_p = rs_y[j];

                double  x2 = ss_x[cur_ss];
                double  y2 = ss_y[cur_ss];

                current_angle = calculate_angle(x1,y1,x_p,y_p,x2,y2);
                //printf("\n");
                //printf("The angle between SS#%d --- RS#%d --- SS#%d
is = %lf\n", n,j,cur_ss,current_angle);

                if(current_angle >= max_angle)
                {
                        max_angle=current_angle;
                }
        }//if...
        }//for_loop...


                return max_angle;


        }
```

```
}
```

Appendix D

Implementation of BFS based tree construction algorithm

```
//============This is created for Implementing BFS ============
CODE_INSERT================

int my_xxx_mac_ids[50];

struct BFS_struct{


char my_node_names[50][20];

double my_criterion_distance;

int my_marked_nodes[50];

int my_hop_nodes[50];

int my_node_relation[50][50];


}my_xxx_bfs_struct;

static int my_xxx_mac_ids_count;



//================Queue Implementation============================

int MAXSIZE = 50;


struct my_queue_structure

{


int front,rear;

int queue[50];

}q;
```

```
int queue_empty(void);
int queue_full(void);
void queue_add(int);
int queue_pop(void);
void queue_display(void);

//================================================================

//=================Queue Implementation============================

int queue_full(void)
{
if ( q.rear == MAXSIZE)
return(1);
else
return(0);
}

int queue_empty(void)
{
if (q.front == q.rear + 1)
```

```
return(1);
else
return(0);
}

void queue_add(int node_id)
{

if(queue_full() == 1)
{
printf("\n\nQueue Full\n");
}

else
{
q.rear = q.rear + 1;

q.queue[q.rear] = node_id;

if(q.rear == 1) q.front ++;
}

}//end of add method...
```

```
int queue_pop(void)
{
int node_id;

if(queue_empty() == 1)
{
printf("\n\nQueue Empty\n");
return 0;
}

else
{
node_id=q.queue[q.front];
printf("% d Has Been Deleted!",q.queue[q.front]);
q.front = q.front +1;
return node_id;
}
}// end of delete method...


void queue_display(void)
{
int i;
```

```
if(queue_empty () == 1)
printf("\nQueue Empty!!");
else
{
printf("\nDisplaying Queue\n");
for(i = q.front; i <= q.rear; i++)
        printf("%d\n",q.queue[i]);
}
}//end of display method...


//=======================================


int i,j;
char node_name [512];
int my_xxx_mac_address;
int flag_already_in_list =0;


char                                    my_xxx_node_name[128];


//==========IMPLEMENTING_BFS_APPROACH==============
//if(my_mac_address == 1111)
```

```
//{


op_ima_obj_attr_get_str (op_topo_parent (op_id_self ()), "name", 128, my_xxx_node_name);
//if  (strcmp(my_xxx_node_name,"BS_aac") == 0)
//{
printf("\n\n\n\n\n\n\n\n XXXXXXXXXXXXXXXXXXXXXXXX \n");
printf("\n THIS IS :: %s  ===   from.....my_xxx_node_name \n\n", my_xxx_node_name);



//==================MAC_ADDRESS at this point is ZERO for all of the
nodes!!!==============================
//printf("\n\n The mac_address :: %d \n\n", my_mac_address);
//============================================================


//=====CHECKING if the node is already entered into the list.....==============
for(i=1; i <=my_xxx_mac_ids_count; i++)
{
if ( strcmp(my_xxx_node_name, my_xxx_bfs_struct.my_node_names[i]) == 0)
flag_already_in_list = 1;
}


if ( flag_already_in_list == 1)
{
printf("\n\n This node is coming for more than one time ====== %s \n\n",
my_xxx_node_name);
}
```

```
//=====================================================
else
{

op_ima_obj_attr_get (op_id_self (), "Address", &my_xxx_mac_address);
printf("\n\n The mac_address :: %d \n\n", my_xxx_mac_address);

my_xxx_mac_ids[++my_xxx_mac_ids_count]= my_xxx_mac_address;

i=0;
while ( my_xxx_node_name[i] != '\0')
{
my_xxx_bfs_struct.my_node_names[my_xxx_mac_ids_count][i]=my_xxx_node_name[i];
i++;
}

printf("\n The count of mac_ids so far is :: %d\n\n", my_xxx_mac_ids_count);
printf("\n XXXXXXXXXXXXXXXXXX \n\n\n");

if(my_xxx_mac_address == 1111)
my_xxx_bfs_struct.my_marked_nodes[my_xxx_mac_ids_count]= 1;

//i=1;
//if( my_xxx_mac_ids_count == 12)
```

```
//{
//                    printf("\n\n Now.. all of them are stored in the array....\n The list of IDS are \n\n");
//                    for(i=1; i <=my_xxx_mac_ids_count ; i++)
//                    printf("\n ID: %d = %d\n\n", i, my_xxx_mac_ids[i]);


//}


//op_ima_obj_pos_get (op_id_from_name (op_topo_parent (op_topo_parent (op_id_self ())), OPC_OBJTYPE_NODE_MOB, "SS_wks_5"), &latitude, &longitude,&altitude, &x_pos, &y_pos, &z_pos);
//printf("\n\n\n SS_wks_5 ==== Latitude: %lf    ======= Longitude: %lf \n\n\n", latitude, longitude);
//printf("\n\n\n X_pos: %lf    ======= Y_pos: %lf \n\n\n", x_pos, y_pos);
//}


//}


for( i = 0; i < 50 ; i++)
for( j = 0; j < 50 ; j++)
my_xxx_bfs_struct.my_node_relation[i][j]=0;



}//else...
//=========================================================
```

```
//===Here is the implementation of algorithm... see if the array has all the ids...


if( my_mac_address == 1111)   //to disable this.. changing the mac_address value to something else... it was '1111' before...

{

int current_node =0;

int j=0;

int current_hop_cross=0;


printf("\n\n The mac_address :: %d \n\n", my_mac_address);

printf("\n\n Now.. all of the MAC_IDS are stored in the array....\n The list of IDS are \n\n");

for(i=1; i <=my_xxx_mac_ids_count ; i++)

printf("\n ID: %d = %d\n\n", i, my_xxx_mac_ids[i]);


printf("\n\n Now.. all of the NODE_NAMES are stored in the array....\n The list of NAMES are \n\n");

for(i=1; i <=my_xxx_mac_ids_count ; i++)

printf("\n NAME: %d = %s \n\n", i, my_xxx_bfs_struct.my_node_names[i]);


printf("\n\n Now.. all of the MARKED/UNMARKED are stored in the array....\n The list of STATUS are \n\n");

for(i=1; i <=my_xxx_mac_ids_count ; i++)

printf("\n STATUS: %d = %d \n\n", i, my_xxx_bfs_struct.my_marked_nodes[i]);



//=======================================================
```

```
//now... working on this.. distance...


q.front = 0;

q.rear = 0;


my_xxx_bfs_struct.my_criterion_distance =  0.012;    //Setting the criterion distance... for one hop... (Pretty much like... 0.3850 Kms)  ///////// it was 0.005433 for scenarios 1,2 // Setting the criterion distance...(like... 0.5433 Kms)


queue_add(1);   //Adding BS_aac into the queue...

my_xxx_bfs_struct.my_hop_nodes[j]=0;   //Maitaining nodes at each hop...

my_xxx_bfs_struct.my_hop_nodes[++j]=1;

if(my_xxx_bfs_struct.my_hop_nodes[j-1] == 0)

current_hop_cross = my_xxx_bfs_struct.my_hop_nodes[j];


while(!queue_empty()) //do till queue becomes empty....

{


current_node = queue_pop();


if(current_hop_cross == current_node)

my_xxx_bfs_struct.my_hop_nodes[++j]=0;




printf("\n\n The current_node is : %d \n\n", current_node);
```

```
if(my_xxx_bfs_struct.my_node_names[current_node][0] == 'S')

{

printf("\n\n This is as SS node...\n\n");



}


else

{


op_ima_obj_pos_get (op_id_from_name (op_topo_parent (op_topo_parent (op_id_self ())),
OPC_OBJTYPE_NODE_FIX, my_xxx_bfs_struct.my_node_names[current_node]), &latitude,
&longitude,&altitude, &x_pos, &y_pos, &z_pos);

my_node_latitude = latitude;

my_node_longitude = longitude;


for(i=1; i <=my_xxx_mac_ids_count ; i++)

{




if(my_xxx_bfs_struct.my_marked_nodes[i] == 0)

{


printf("\n NAME: %d = %s \n\n", i, my_xxx_bfs_struct.my_node_names[i]);


if(my_xxx_bfs_struct.my_node_names[i][0] == 'S')
```

```
op_ima_obj_pos_get (op_id_from_name (op_topo_parent (op_topo_parent (op_id_self ())),
OPC_OBJTYPE_NODE_MOB, my_xxx_bfs_struct.my_node_names[i]), &latitude,
&longitude,&altitude, &x_pos, &y_pos, &z_pos);

else

op_ima_obj_pos_get (op_id_from_name (op_topo_parent (op_topo_parent (op_id_self ())),
OPC_OBJTYPE_NODE_FIX, my_xxx_bfs_struct.my_node_names[i]), &latitude,
&longitude,&altitude, &x_pos, &y_pos, &z_pos);




other_node_latitude = latitude;

other_node_longitude = longitude;




my_xxx_distance = ( (other_node_latitude - my_node_latitude)*(other_node_latitude -
my_node_latitude) ) + ( (other_node_longitude - my_node_longitude)*(other_node_longitude -
my_node_longitude) );

//my_xxx_bfs_struct.my_criterion_distance = my_xxx_bfs_struct.my_criterion_distance *
my_xxx_bfs_struct.my_criterion_distance;


printf("\n\n The Distance between Current Node: %s  ========and======== Other_Node :
%s ======= is ======== %lf \n\n\n",  my_xxx_bfs_struct.my_node_names[current_node],
my_xxx_bfs_struct.my_node_names[i], my_xxx_distance);


if(my_xxx_distance <= (my_xxx_bfs_struct.my_criterion_distance *
my_xxx_bfs_struct.my_criterion_distance))

{

//Mark this node... this is reached by the current node...

my_xxx_bfs_struct.my_marked_nodes[i] = 1;
```

```
printf("\n\n ################### The node :: %s  ======== IS COVERING ======= node :: %s ###################\n\n",my_xxx_bfs_struct.my_node_names[current_node], my_xxx_bfs_struct.my_node_names[i]);


//Now add this to queue... if the marked_node is not a 'SS'...

//if(my_xxx_bfs_struct.my_node_names[i][0] != 'S')

queue_add(i);


my_xxx_bfs_struct.my_hop_nodes[++j]=i;

if(my_xxx_bfs_struct.my_hop_nodes[j-1] == 0)

current_hop_cross = my_xxx_bfs_struct.my_hop_nodes[j];


//Relate this node as the CHILD node of current node...

my_xxx_bfs_struct.my_node_relation[current_node][i]= 1;  //relation value '1' denotes PARENT---CHILD relation...

my_xxx_bfs_struct.my_node_relation[i][current_node]= -1; //relation value '-1' denotes CHILD-- PARENT relation...


}



}//if...



}//for...


}//else...
```

```
}//while...

//================================

printf("\n\n\n\n $$$$$$$$$$$$$$$$$ The hop nodes list is: \n");
for( j=0; j<=20; j++)
printf("\n Node ID: %d\n",my_xxx_bfs_struct.my_hop_nodes[j]);

}

//====================Code_END================================
```