PRACTICAL CONSIDERATIONS OF ROUTING PROTOCOLS IN AD HOC NETWORKS

Except where reference is made to the work of others, the work described in this dissertation is my own or was done in collaboration with my advisory committee. This dissertation does not include proprietary or classified information.

_____

Junmo Yang

Certificate of Approval:

_____	_____
David Umphress Associate Professor Department of Computer Science and Software Engineering	Min-Te Sun, Chair Assistant Professor Department of Computer Science and Software Engineering
_____	_____
Yu Wang Assistant Professor Department of Computer Science and Software Engineering	Joe F. Pittman Interim Dean Graduate School

PRACTICAL CONSIDERATIONS OF ROUTING PROTOCOLS IN AD HOC NETWORKS

Junmo Yang

A Dissertation

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Doctor of Philosophy

Auburn, Alabama
December 15, 2006

PRACTICAL CONSIDERATIONS OF ROUTING PROTOCOLS IN AD HOC NETWORKS

Junmo Yang

_____

Signature of Author

_____

Date of Graduation

DISSERTATION ABSTRACT

PRACTICAL CONSIDERATIONS OF ROUTING PROTOCOLS IN AD HOC NETWORKS

Junmo Yang

Doctor of Philosophy, December 15, 2006
(M.S., Auburn University, 2002)
(B.A., Kyunghee University, 1996)

141 Typed Pages

Directed by Min-Te Sun

In this dissertation, we investigate different routing approaches in an attempt to improve the network performance by considering how wireless networks operate in the realistic environment. Our work is centered around two primary focuses: In the first one, we have found that disruptive links appear quite frequently due to the presence of obstacles and node mobility. In the presence of disruptive links, we studied how geographic routing protocols, such as GPSR, should be properly adapted and proposed the Disruption Tolerant Geographical Routing protocol (DTGR). In the second half, we consider the routing problem in multi-radio multi-hop wireless mesh networks. To maximize the overall throughput of such a wireless mesh network, the interference between mesh routers need to be taken into account. We formulate the impact of the interference into a cost function and proposed the Cost Aware Route Selection scheme (CARS).

In a wireless ad hoc networks where temporary link disruptions occur frequently, a node may have incorrect perception of its neighbor set. Since the neighbor set is constructed via beacon sampling, beacon collisions may result in the removal of a node from the neighbor set even though the node is still within the transmission range. Such a behavior can adversely affect the performance of position based routing algorithms as it may lead to inefficient routing or packet dropping. To address this, we propose a scheme that allows each node to associate each of its neighbor with a reachability value that is a measure of the stability of the link. We then apply our scheme to Greedy Perimeter Stateless Routing (GPSR) and design two new routing algorithms, namely Disruption Tolerant Geographic Routing-Simple Forwarding (DTGR-SF) and Disruption Tolerant Geographic Routing-Waiting before Forwarding (DTGR-WF), in which nodes utilize reachability values to make appropriate forwarding decisions. We compare the performances of DTGR-SF and DTGR-WF with that of GPSR in various simulation settings. Our simulation results show that our proposed algorithms perform better in settings where link disruptions are present. In networks with few occurrences of disruptions, our schemes achieve the same high performance as that of GPSR.

Many applications of wireless mesh networks, such as WLAN, video conference, and VoIP, demand more bandwidth and the support of more active users. By installing multiple radio interfaces at each mesh router, a wireless mesh network is able to better utilize the available wireless spectrum for such applications. However, the presence of multiple radio also complicates the selection of route in wireless mesh networks. To address this issue,

we first use a cost function to capture the degree of interference for a given route quantitatively. We then propose a novel metric that measures the bandwidth and cost ratio of each route. Based on this metric, a Cost-Aware Route Selection (CARS) scheme is proposed to improve the overall throughput of a mesh network. The simulation results confirm that our scheme is able to better utilize the limited wireless resource and improves the overall network throughput by more than 95% with different types of traffic and communication patterns when it is compared against the past route selection schemes.

Style manual or journal used Journal of Approximation Theory (together with the style known as "aums"). Bibliograpy follows van Leunen's *A Handbook for Scholars*.

Computer software used The document preparation package TEX (specifically LATEX) together with the departmental style-file `auphd.sty`.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

A wireless ad hoc network is a collection of wireless devices (referred to as nodes or stations) such as handheld devices, mobile phones, and automotive telematics systems that communicate with each other by forming a multi hop radio network. An ad hoc network is formed without the need for an infrastructure over a large geographical area. In such a network, each device plays the role of not only a router for relaying packets to their destinations, but also a host for the source and a sink for traffic flows. Without any type of centralized control, a node in the network should be able to select the best route among candidate routes. In reality, the network topology in an ad hoc network is highly complicated and meshed due to temporary link failures and the emergence of multi-radio and multi-channel in Wireless Mesh Networks (WMNs).

The major advantage of an ad hoc network is that a multi-hop network can be formed without any need for a fixed infrastructure. This makes an ad hoc network a strong candidate for Wi-Fi, WLAN, WMAN, or Wi-Max solutions. Since the University of California at Berkeley introduced ”smart dust” as a sensor in the late 1990s [1, 2], the application of an ad hoc network has been extended to emergency services (e.g. E911 w/ Global Positioning System), disaster recovery, communications on the battle field, underwater research, and many others.

To take full advantage of ad hoc networks, many research issues such as virtual backbone construction [3, 4], cross-layer design [5, 6], overlay design [7, 8], adaptive rate control [9], and fault tolerant [10] remain to be addressed. Among these issues, routing is considered one of the most important issues limiting ad hoc networks. In addition, routing process should be able to deal with real-world problems such as temporary link failures and multi radio and multi channel in WMNs to improve network performance:

1. Successful routing protocol must be able to recover immediately from temporal link failures. In wireless ad hoc and sensor networks, the transmission media (i.e. the radio signal) is less stable than in wired networks. This may mislead nodes and cause them to behave as if temporary link disruptions are permanent link failures. Discovering and establishing a new route in an ad hoc network increase costs in terms of time and network resources.

2. In WMNs, the route bandwidth should be large enough for multiple users to access the Internet simultaneously. This type of utilization of multi-radio and multi-channels has emerged recently in WMNs. The optimal route selection in such a multi-radio network is an NP-hard problem because the degree of complexity in network formation is much higher than that for single radio networks.

This dissertation is to investigate two different routing approaches in an attempt to improve the network performance by considering the practical way ad hoc networks operates in the real world. The first disruption tolerant geographic routing (DTGR) will allow node i to associate each of its neighbors j with a reachability value that is a measure of

the stability of the link between i and j, thus making the routing disruption tolerant. An alternative approach, the cost aware route selection scheme (CARS), will calculate the cost of the interference and the path bandwidth, thus improving the throughput in multi-radio and multi-channel WMNs.

The remaining chapters are organized as follows. Chapter 2 presents the background and general issues that affect current wireless networks. Chapter 3 introduces the existing routing protocols and Chapter 4 examines the practical considerations involved. In Chapter 5, our disruption tolerant geographic routing protocol (DTGR) is presented. A cost aware route selection (CARS) is introduced in Chapter 6. A summary of the dissertation, the conclusions reached, the contributions to the field, and suggestions for future research are given in Chapter 7.

CHAPTER 2

BACKGROUND AND GENERAL ISSUES

This chapter provides an overview of wireless ad hoc networks. There are two types of wireless ad hoc networks: mesh networks, and sensor networks. Each has important applications and supports a different degree of cover area for wireless devices. Section 2.1 discusses the relevant background and gives an overview of wireless ad hoc networks. Sections 2.2, and 2.3 provide the definition, the standards, and research issues for each type of ad hoc network.

2.1 Overview of wireless ad hoc networks

Typically wireless networks are made up of two types of components: wireless devices (e.g., routers and hosts), and wireless transmission media (e.g., radio frequency bands).[1] Generally routers are responsible for forwarding packets and hosts are responsible for the source or the sink of traffic flows. At present, radio frequency (RF) is the most popular medium in wireless communication. Since different RF spectra have different properties in terms of their transmission range, transmission rate, power consumption, and propagation model (e.g., omnidirectional or directional) [11, 12], each wireless device is designed to exploit a specific range of the RF spectrum for its specialized purpose.

[1] Note that the wireless networks discussed in this dissertation refer to multi-hop wireless networks rather than single-hop wireless networks (e.g., cellular wireless networks).

An ad hoc network is a special type of wireless networks. Lacking any fixed or centralized infrastructure, ad hoc networks form their own multi-hop networks. Any wireless device, also referred to as a node, must be able to act as both a router and a host by self-configuration. Due to the constraints imposed by power limitations or the standards set by various industry committees, each node in an ad hoc network has only a limited transmission range. As a result, a packet in an ad hoc network is likely to travel through several hops before it reaches its final destination.

In an ad hoc network, the connection is established for the duration of one session and requires no base station. Instead, nodes identify other nodes within transmission range to form a network. Nodes may search for destination nodes that are out of transmission range by a simple flooding approach. After discovering their destination nodes, routing protocols then provide stable connections even if the nodes are moving around. Different types of wireless devices, such as personal digital assistants (PDAs), laptops, and mobile phones in ad hoc networks, may form a network by linking up others equipped with the same type of radios. However, different types of radio have distinctly different attributes, including computation, storage and communication capabilities.

2.2 Mesh Networks

A wireless mesh network (WMN) is a way to route packets between nodes that are capable of self-organization and maintenance. Providing many alternate paths between the source and the destination results in quick re-configuration and continuous connections

when the existing path fails. As a result, this self-healing ability makes WMNs both robust and reliable.

A mesh networking standard 802.11s [13] is the unapproved IEEE 802.11 standard for Extended Service Set (ESS) mesh networking. An IEEE 802.11s standard is a collection of access points (APs) interconnected with wireless links that enable automatic topology learning and dynamic path configuration. It specifies an extension to the IEEE 802.11 MAC standard to solve the inter-operability problem by defining an architecture and protocol.

The choice of radio technology for WMNs is crucial. In a traditional wireless network, where wireless devices connect to a single access point, each device has to share a limited bandwidth. In WMNs, devices with an adaptive radio technology will only connect with other devices within range. The advantage of WMNs is that the more devices are in range the more bandwidth becomes available.

A wireless mesh network is generally built on top of home networks, which are typically wireless local area networks (WLANs). A WLAN provides high data rate connections in a local area to the Internet. Most WLANs operate in unlicensed bands that are free of charge and rigorously regulated. The Institute of Electrical and Electronics Engineers (IEEE), European Telecommunications Standards Institute (ETSI), and HomeRF Working Group (HomeRF WG) have all been involved in developing standards for WLANs, but the ones that dominate the market are from IEEE. Currently there are four IEEE specifications for WLAN: 802.11b, 802.11a, 802.11g, and 802.11n. The WLANs that are based on these specifications are the building blocks for wireless mesh networks.

- IEEE 802.11b (also referred to as Wi-Fi) [14, 15, 16, 17] – IEEE 802.11b supports transmission rates of up to 11 Mbps and uses the unlicensed 2.4 GHz Industrial, Scientific, and Medical (ISM) spectrum. There are two immediate consequences of using an unlicensed band. First, the transmissions in a IEEE 802.11b network are prone to interference from other devices utilizing the same spectrum, such as microwave ovens and cordless phones. Second, the transmission power of a IEEE 802.11b device has to conform with certain regulation so that it will not be harmful to other devices using the same unlicensed band. To deal with these issues, a spread spectrum is primarily used in IEEE 802.11b.

- IEEE 802.11a [14, 15, 16, 18] – IEEE 802.11a supports high transmission rates of up to 54 Mbps by using an orthogonal frequency division multiplexing (OFDM) that operates in the 5 Ghz band rather than a spread spectrum scheme. Due to the choice of the 5 GHz spectrum, IEEE 802.11a devices lead to far less radio frequency (RF) interference. With high data rates and relatively less interference, IEEE 802.11a is especially suited to supporting multimedia applications.

- IEEE 802.11g [14, 15, 16, 19] – Developed more recently than either IEEE 802.11b or 802.11a, IEEE 802.11g is an attempt to benefit from the positive aspects of both the earlier standards. IEEE 802.11g supports a bandwidth of up to 54 Mbps and uses the 2.4 Ghz ISM spectrum. IEEE 802.11g is compatible with 802.11b, meaning that 802.11g access points (AP) will also work with 802.11b wireless network adapters and vice versa.

- IEEE 802.11n [14, 15, 16]- IEEE 802.11n builds upon previous 802.11 standards by incorporating multiple-input multiple-output (MIMO) technology. IEEE 802.11n offers especially high transmission rates of 100Mbps to 200Mbps.

To improve the performance of a wireless mesh network, the access point is usually assumed to be equipped with multiple wireless interfaces built on either the same or different WLAN technologies. The primary research issue in mesh networks is how to take advantage of the availability of multiple wireless interfaces at each access point to maximize the communication throughput [20].

2.3 Sensor Networks

Wireless sensor networks are a special class of ad hoc networks that are used to provide a wireless communication infrastructure among sensors deployed in a specific application domain. A sensor network is composed of a large number of sensor nodes that are densely populated, either inside the coverage area or very close to it. The positions of the sensor nodes need not be engineered or predetermined, allowing random deployment in inaccessible terrain or disaster relief operations.

Each node in a sensor network consists of three subsystems: the sensor subsystem which senses the environment, the processing subsystem which performs local computations on the sensed data, and the communication subsystem which is responsible for message exchanges with neighboring sensor nodes. While individual sensors are limited by their limited sensing region, processing power, and energy, networking a large number of sensors can result in a robust, reliable, and accurate sensor network covering a wide region.

The types of sensors range from small passive microsensors (e.g, smart dust) to larger scale, controllable weather-sensing platforms. At present, there are several standard and proprietary devices that support sensor networks. IEEE 802.15.1 (Bluetooth) and IEEE 802.15.4 (Zigbee) are the most promising standards for wireless sensor networks because Bluetooth and Zigbee devices are generally inexpensive and consume relatively little power, although motes [1], designed primarily by UC-Berkeley, have also been adopted by many sensor network applications.

- IEEE 802.15.1 (Bluetooth)[21] - Initially developed by the Bluetooth special interest group, Bluetooth is a wireless specification for wireless personal area networks (WPANs), which has characteristics such as short-range, low power, and low cost. Operating on the 2.4 GHz unlicensed ISM band, Bluetooth supports data rates up to 2.178 Mbps within distances of up to 100 m.

- IEEE802.15.4 (Zigbee)[21] - Initially developed by the Zigbee alliance, ZigBee is designed to support low data rate, low power consumption, and low cost wireless communications. The primary applications of Zigbee include automation and remote control. It supports a data rate of 250 kbps using 2.4 GHz unlicensed bands within a range of 10 to 75 m.

CHAPTER 3

ROUTING PROTOCOLS IN AD HOC NETWORKS

The absence of a centralized infrastructure makes routing in ad hoc networks one of the most challenging research issues. Nodes in an ad hoc network have no knowledge of network topology. By exchanging information, the route from the source to the destination must be discovered and established for either a long period of time or temporarily. Routing protocols in ad hoc networks can be classified in terms of the presence of a routing table, network formation (clustering or partitioning), specialized purpose (e.g. multicast), and the scope of nodal information (e.g. location). In this chapter, described routing protocols are categorized according to several criteria and their characteristics described. However, the classification of routing protocols in ad hoc networks is not deterministic and a protocol can fall into several categories simultaneously.

3.1 Proactive (Table Driven) Routing Protocols

In proactive routing protocols, each node maintains global topology information in its table. The destination sequence distance vector (DSDV) routing protocol [22, 23, 24], the wireless routing protocol (WRP)[25], and the cluster-head gateway switch routing protocol (CGSR) [26] are all types of proactive routing protocols.

DSDV is one of the first protocols proposed for use in ad hoc networks. Based on the Bellman-Ford algorithm [27], its routing table is composed of destination, next hop,

distance, and sequence number information. To avoid the occurrence of infinite loops, sequence number tags are used. Two timers, which may be either a soft timer (trigger of event) or a hard timer (physical or logical timer) are used to refresh the routing table at each node. The minimal delay incurred by the route setup process is the primary advantage of DSDV, since all the available routes to the destinations are already in the table. However, to maintain all the available routes, the routing table must be updated periodically by exchanging update messages, which leads to an increased overhead in the network. In addition, a node must wait for table update message before initiating a table.

WRP is also based on the Bellman-Ford algorithm. While DSDV maintains only one routing table at each node, WRP keeps sets of tables, such as a distance table, routing table, link cost table, and message retransmission list. This enables WRP to maintain more accurate information on the network topology. The advantages of WRP are the same as for DSDV, which also benefits from fast convergence. However, since WRP maintains multiple tables, computation complexity is high. Also, the use of multiple tables requires more memory in a node. This increase overhead due to updating tables results in poor performance under high mobility condition and in a large network.

CGSR uses a hierarchical network topology. While most proactive routing protocols employ a table driven approach, CGSR adopts flat topology. A node in the network topology belongs to one of the following types: cluster-head, cluster-gateway, and cluster-member. A cluster-head is elected by a least cluster change (LCC) algorithm. Within transmission range of a cluster-head, there should be no other cluster-head. If a node is a member of more than one cluster-head, it is called a gateway. The remaining nodes are

simply cluster-members. The role of a cluster-head includes coordinating of scheduling and bandwidth assignments. Since a network is partitioned by the cluster-heads, the bandwidth can be utilized efficiently. However, the path length could be increased in CGSR. In addition, for high mobility applications, CGSR is not stable because the cluster-heads need to be re-elected frequently, leading to more communication and thus also more energy consumption.

3.2 Reactive (On Demand) Routing Protocols

While proactive routing protocols have a table which includes the information of destination, reactive routing protocols establish the path only when it is demanded. The dynamic source routing protocol (DSR) [28, 29], the ad hoc on demand distance vector routing protocol (AODV) [30, 31], and the temporally ordered routing algorithm (TORA) [32, 33] are all examples of reactive routing protocols.

DSR is one of the most popular routing protocols and is widely used in ad hoc networks. Instead of periodically sending a hello packet or a beacon, it sends a $RouteRequest$ packet in order to discover a path by flooding when it is demanded. As soon as a destination receives a $RouteRequest$ packet, it responds by sending a $RouteReply$ packet to the source. Since a $RouteRequest$ packet retains the information on the traversed path, a $RouteReply$ packet can backtrack the path of the $RouteRequest$ packet. Once a source node gets a $RouteReply$ packet, a path to the destination can thus be established. The elimination of the need to broadcast periodic update messages results in a much reduced overhead in a network. Since a path is established only when it is requested, each node

does not need to have a large memory. The reusability of path information that is cached in intermediate nodes is another advantage of DSR. However, the latency incurred by sending a packet to find a path is a disadvantage of DSR. High mobility can also result in degrade of network performance. In addition, the overhead involved in establishing a path is proportional to the length of that path.

AODV also uses an on-demand approach to discover a path. A path is established only when a source node requests a path. After establishing a path, packets can be transmitted. The basic concept of AODV is that the destination uses sequenced numbers to identify the most recent path. While DSR sends all packets through a pre-determined path, in AODV the source and intermediate nodes keep different routes' information for each corresponding traffic flow. The advantage of AODV is that it can determine the latest path to the destinations. In addition, since one $RouteRequest$ can be used for multiple destination, the delay in establishing a path is likely to be less than for other reactive protocols. However, in AODV, sending a beacon periodically can lead to unnecessary bandwidth consumption. Also, the generation of multiple $RouteReply$ packets for single $RouteReqeust$ packet results in another unnecessary overhead.

TORA, based on a link reversal algorithm, provides multi-paths and loop-free connections. Only one hop local information is kept at each node. TORA has three functions; establishing, maintaining, and erasing routes. Establishing a path is executed only when it is requested. When an intermediate node discovers an invalid link, it sends an update packet to erase the link, enabling the source node to send a clear packet to erase the invalid

link. Since control packets are used for local regions, the overhead due to control packets is much less. However, local reconfiguration of a path often leads to non-optimal routes.

3.3 Hybrid (Proactive/Reactive) Routing Protocols

A combination of proactive and reactive routing protocols is implemented in hybrid routing protocols. In hybrid routing protocols, two different domains (i.e., inter- or intra-zones) use their corresponding types of routing protocols. The zone routing protocol (ZRP) [34, 35], the zone-based hierarchical link state routing protocol (ZHLS) [32], and the core extraction distributed ad hoc routing protocol (CEDAR) [36, 37] are all examples of hybrid protocols.

ZRP divides a network topology into two zones: $intra-zone$ and $inter-zone$. In the $intra-zone$, proactive routing protocols are employed, while in the $inter-zone$, reactive routing protocols are used. The boundary between the $inter-zone$ and $intra-zone$ is determined by the zone radius, which is a predetermined number of hops. A zone radius of 1 indicates one hop distance. If a destination is located within the zone radius, a source can transmit packets directly to the destination using proactive routing protocols. Otherwise, a source node needs to send a $RouteRequest$ packet by reactive routing protocols to find a route. Compared to the simple $RouteRequest$ flooding, ZRP can reduce the overhead of control packets. However, redundant $RouteRequest$s can be produced for a destination node which is located in the $inter-zone$. Also, the size of the zone radius significantly impacts the performance of ZRP.

ZHLS, which utilizes the geographical location information, is another example of a hybrid protocol. In ZHLS, the network topology is divided into non-overlapping zones. The hierarchical addressing scheme consists of a node ID and a zone ID, with the assumption that each node knows its location information through a Global Position System (GPS)[38]. ZHLS can reduce the storage requirements and communication overhead. In addition, ZHLS is robust with regard to node mobility. However, the generation of additional overhead for the zone-level topology is the main disadvantage of this protocol.

CEDAR extracts the core nodes, known as the dominating set, from the network topology. The main purpose of CEDAR is to construct a virtual backbone with dominating nodes. A dominating set (DS) is defined as a subset of nodes in a graph such that each node not in the subset has at least one neighbor in the subset. By constructing a minimal DS, any packet can be transmitted from a source node to a destination. Another purpose of CEDAR is to provide QoS. When establishing a path, CEDAR also considers the required bandwidth. Thus, CEDAR can perform both routing and QoS path computation efficiently with DS. However, since most computation is carried out at core nodes, the movement of the core nodes can seriously degrade the performance of the protocol. In addition, the need to update the information on the core nodes increases the control overhead.

3.4 Hierarchical Routing Protocols

In hierarchical routing protocols, network topology can be partitioned with multi-layered clusters. Fish eye state routing (FSR) [39] and hierarchical state routing (HSR) [40], which is an improved version of FSR, are introduced in this section.

In FSR, each update message does not contain information about all nodes. Instead, FSR exchanges information about closer nodes more frequently, thus reducing the update message size and allowing each node to obtain more accurate information about its neighbors. However, further away from the node the accuracy of information is decreased. Although a node does not have accurate information on distant nodes, packets can be transmitted correctly because the information on the destination becomes more accurate as it gets closer to the destination. FAR is scalable to the size of networks under controlled message overhead.

HSR is a multilevel clustering and logical partitioning routing protocol. In HSR, mobile nodes are grouped into clusters and a cluster-head is elected for each cluster. The cluster-heads of low level clusters then organize themselves into upper level clusters, and so on. Inside a cluster, nodes broadcast their link state information to all others. The cluster-head summarizes the link state information for its cluster and sends the information to its neighboring cluster-heads via gateway nodes. Nodes in upper level hierarchical clusters flood the network with the topology information they have obtained on the nodes in the lower level clusters. In HSR, a hierarchical address is assigned to every node. The hierarchical address reflects the network topology and provides sufficient information for packet deliveries in the network.

3.5 Geographical Routing Protocols

By equipping each node with an inexpensive Global Positioning System (GPS) cpability, it is possible to track their location. Geographical routing protocols discover and

establish a route by utilizing this location information. The location aided routing protocol (LAR) [41] and the distance routing effect algorithm for mobility (DREAM) [42] are all examples of geographical routing protocols.

LAR is a reactive routing scheme. LAR utilizes the position information obtained by GPS and is expected to improve the efficiency of the route discovery procedure by limiting the scope of route request flooding. In LAR, a source node estimates the current location range of the destination based on information on the last reported location and mobility pattern of the destination. In LAR, the route request flooding is limited to a request zone where the destination is expected to be currently located. The size of a request zone can be adjusted according to the mobility pattern of the destination. When the speed of the destination is low, the request zone is small; and when it moves fast, the request zone is large. The advantage of LAR is that it reduces the control message overhead. However, it is not scalable to network size due to the directional flooding.

DREAM exploits the location and speed information for mobile nodes to discover a route. As with LAR, DREAM utilizes location information not only to discover a route, but also to flood data packets to a small region. However, unlike LAR DREAM is a proactive routing protocol. In DREAM, a routing table contains location information for all the other nodes. To maintain this information, every mobile node must send location updates. The frequency of the location update is determined by the distance and node mobility. DREAM also is not scalable to network size because of the directional flooding.

3.6 Multicast Routing Protocols

Due to dynamic changes in the network topology, a multicast structure must be reconstructed continuously as connectivity changes. The bandwidth efficient multicast routing protocol (BEMRP)[43] and the ad hoc multicast routing protocol (AMRoute)[44] are examples of multicast routing protocols.

BEMRP is designed to achieve both low communication overhead and high multicast efficiency. BEMRP employs on-demand invocation of the route setup and route recovery processes to avoid periodic transmissions of control packets. The route setup process allows a newly joining node to find the nearest forwarding node to minimize the number of forwarding nodes, and a route optimization process detects and removes unnecessary forwarding nodes to eliminate redundant and inefficient routes. The main advantage of BEMRP is that it saves bandwidth due to the reduction in the number of data packet retransmissions. However, when a node joins the multicast group, it selects its nearest forwarding node, which may result in increasing the distance between source and receiver. This leads to a high incidence of path breaks and delay.

AMRoute has been proposed for robust IP Multicasts in mobile ad hoc networks by exploiting user-multicast trees and dynamic logical cores. It creates a bidirectional, shared tree for data distribution using only group senders and receivers as tree nodes. Unicast tunnels are used as tree links to connect neighbors on the user-multicast tree. Certain tree nodes are designated by AMRoute as logical cores, and are responsible for initiating and managing the signaling component of AMRoute, such as the detection of group members

and tree setup. The disadvantage of AMRoute is that under mobility conditions, the hop count of the unicast tunnels can increase, thus decreasing throughput.

CHAPTER 4

PRACTICAL CONSIDERATIONS IN THE DESIGN OF WIRELESS PROTOCOLS

An ad hoc networks is a special type of wireless network. Compared to wired transmission media (e.g. cable), wireless transmission media (e.g. radio frequencies) form less stable network. In this chapter, practical issues affecting wireless networks will be discussed after an overviewing the physical characteristics of wireless media. Since the physical constraints of wireless media directly affect not only the performance of networks, but also the design of medium access control (MAC) and routing protocols, the physical characteristics of wireless media need to be clearly understood.

4.1 Antennas and Radio Propagation

An antenna is defined as an electrical conductor that is used either for radiating electromagnetic energy or for collecting electromagnetic energy. Generally, the performance of antennas can be characterized by their radiation patterns. An isotropic antenna [45] is an idealized antenna that is known to produce the simplest pattern. Two idealized radiation patterns of isotropic antennas are commonly used in ad hoc networks research: the omnidirectional antenna [45] and the directional antenna [45]. In Figure 4.1, the properties of both are illustrated for 2-dimensional space. In Figure 4.1, the distance from the antenna to a point within the radiation pattern is proportional to the radiated power from the antenna in that direction. In the omnidirectional antenna in Figure 4.1 A, vectors A and B receive equal radiated power. However, in the directional antenna in Figure 4.1 B, the radiated

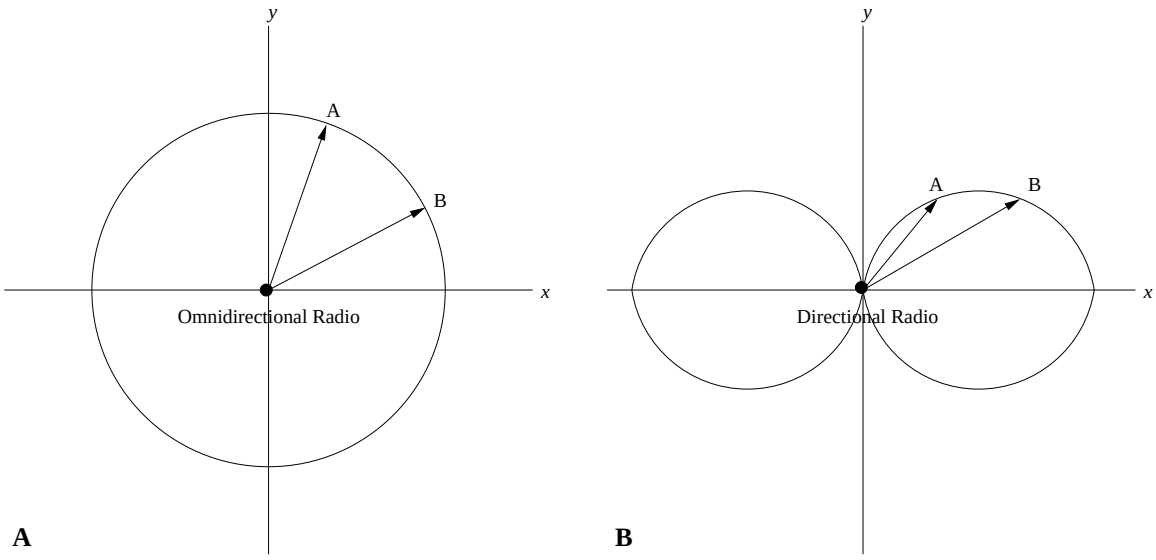


Figure 4.1: A: Omnidirectional antenna B: Directional antenna

power received by vector B is greater than that of vector A. In reality, the size of the radiation pattern is arbitrary. However, in ad hoc network research, it is commonly assumed that an antenna is either omnidirectional or directional for simplicity.

A signal can traverse one of three modes: ground waves, sky waves, and line of sight (LOS). Ground waves [46] are radio waves that follow the contour of the earth. Since the ground is not a perfect electrical conductor, ground waves are attenuated as they travel. An AM radio is a well-known example of ground wave propagation. Sky wave propagation [47] includes any of the modes that rely on refraction of the radio waves in the ionosphere. Sky wave propagation is used for amateur radio, and international broadcasts such as the BBC. Above 30 MHz, neither ground wave propagation nor sky wave propagation can be used since this frequency cannot be reflected by the ionosphere. Since the study of ad hoc networks focuses on the the signals that are much higher than 30 MHz, LOS

mode [48] is assumed to be used in the propagation model. Thus, a detailed discussion on LOS will be provided in the next section.

4.2 Impairments and Fading in LOS

In wireless communication, impairments often occur between the transmitted signal and the received signal. For example, a binary 1 at a transmitter may be transformed into a binary 0 at a receiver or vice versa. This is called a bit error in digital signal communication. These impairments are caused by the following:

- Attenuation: Attenuation is defined as the reduction in the signal strength (i.e., amplitude and intensity) with respect to the traversed distance over a transmission media. In general, attenuation is measured in decibels per unit distance. In order to overcome the attenuation, a signal must have sufficient strength to allow the receiver to correctly interpret the original signal. In addition, a signal should maintain a higher level than the background noise for error avoidance.

- Free space loss: Free space loss is considered a special type of attenuation. Although normal attenuation does not happen, transmitted signals are attenuated over long distances due to the beam divergence and the inverse square law of electromagnetic radiation. Free space power loss is proportional to the square of the distance between the transmitter and receiver and is also proportional to the square of the frequency of the radio signal. Hence, the power of a transmitter must be sufficient to send a receivable signal to a suitably sensitive receiver.

- Noise: In wireless communication, noise is defined as an unwanted signal inserted somewhere between the transmitter and the receiver. Interference is the main type of radio noise. Radio noise can be caused by virtually any electromagnetic source, from lightning to man-made electronics, including the receiver itself. Transmitter power must be increased to overcome radio noise over long distances.

- Atmospheric absorption: An additional loss between the transmitter and the receiver is atmospheric absorption. This is a phenomenon where electromagnetic energy is absorbed by a substance in the atmosphere such as water vapor and oxygen.

- Multi-path: In wireless communications, multi-path is a propagation phenomenon where a radio signal reaches the receiving antenna by two or more paths. This happens due to atmospheric ducting, ionospheric reflection and refraction, reflection from terrestrial objects, and diffraction at the edge of an impenetrable object.

In addition to impairments, fading is one of the most challenging issues in wireless communication engineering. Fading refers to the time variation of the received signal power caused by changes in the transmission medium or path(s). The most common types of fading are slow fading and fast fading:

- Slow Fading [49]: Shadowing or large-scale fading is a kind of fading caused by larger movements of a mobile node or obstructions within the propagation environment.

- Fast Fading [49]: Multi-path fading or small-scale fading is a kind of fading that occurrs due to small movements of a mobile node.

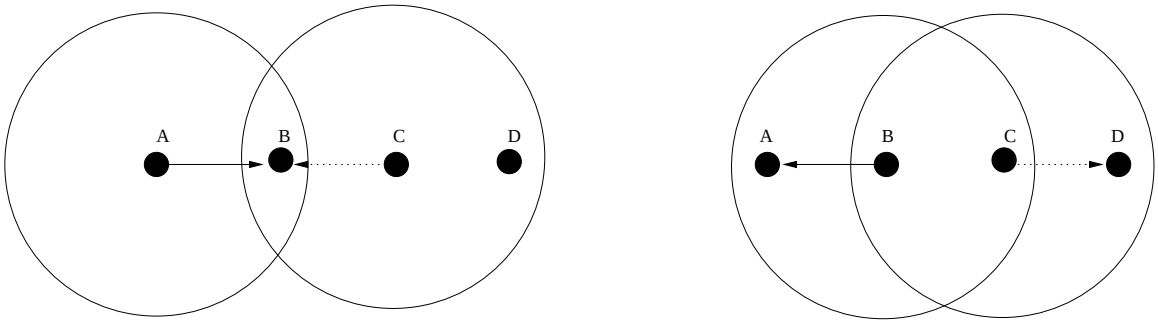
In this section, a brief overview of the physical constraints that affect wireless radio systems has been described. The wireless medium is often unstable due to impairments or fading, which leads to considerable research efforts aimed at the design of better wireless MAC and routing protocols.

4.3 General Issues in Wireless Medium Access Control

In ad hoc networks, the absence of any centralized infrastructure makes the design of a MAC protocol even more difficult. Two general issues in the design of MAC protocols are the hidden terminal problem and the exposed terminal problem. In this section, after the brief overview of two issues common approaches to dealing with these problems will be discussed.

4.3.1 The Hidden Terminal and The Exposed Terminal Problems

Hidden terminals [50] in a wireless network refer to nodes which are out of range of other nodes or a collection of nodes. For instance, in Figure 4.2 A, nodes C and D are hidden terminals with respect to a node A. Within a given transmission range, nodes may also interfere with each other. Suppose that node A is now transmitting data packets to node B by utilizing carrier sense multiple access (CSMA), as depicted in Figure 4.2 A. If node C senses the medium, it may falsely conclude that it should start transmitting data packets to a node B. This will cause interference at node B. The problem that a node is not able to detect potential collisions or interferences because it is out of transmission range is known as a hidden terminal problem. Figure 4.2 B shows the reverse situation, which is referred



A: Hidden Terminal Problem B: Exposed Terminal Problem

Figure 4.2: Hidden Terminal Problem and Exposed Terminal Problem

to as an exposed terminal problem [51]. Suppose that node B is transmitting data packets to node A. Since node C is in close proximity to node B, it will falsely conclude that it cannot transmit data packets to node D. These two problems are the most basic problems that should be avoided during the design of wireless MAC protocols.

4.3.2 The Near-Far Problem and The Capture Effect

The near-far problem [52] is common in wireless communication. Suppose that two transmitters generate signals simultaneously at equal powers. The receiver will receive more power from the nearer transmitter due to the inverse square law. Since the signal-to-noise ratio (SNR) of the farther transmitter is lower than that of the nearer transmitter, a receiver cannot correctly detect the signal from the farther transmitter. This phenomenon is called the near-far problem.

In addition, the capture effect [53] is another phenomenon in wireless communication. This happens with frequency modulation (FM) such that only the stronger of two signals at or near the same frequency will be demodulated.

4.3.3 Carrier Sense Multiple Access With Collision Avoidance (CSMA/CA)

In place of the carrier sense multiple access / collision detection (CSMA/CD) adopted in the IEEE 802.3 (Ethernet) standard, carrier sense multiple access / collision avoidance (CSMA/CA) [54] is adopted by the IEEE 802.11 wireless LAN standard. The basic elements of CSMA/CA are as follows: interframe spacing (IFS), contention window (CW), and a backoff counter. The CW intervals are used for contention and transmission of the packet frames. The IFS is used as an interval between two CW intervals. The backoff counter is used to organized the back-off procedure for transmission of packets.

In addition, IEEE 802.11 WLAN includes an additional scheme to avoid collisions or interferences in advance, namely a request-to-send (RTS) and clear-to-send (CTS) mechanism. Before transmitting data packets, a transmitter sends a short control packet (called an RTS packet) that identifies the source address, destination address, and the length of the data packets. A receiver responds with a CTS packet specifying whether a medium is free or not.

4.4 Practical Issues

In the previous sections of this chapter, the difficulties and constraints of wireless media have been discussed at the physical and MAC layers. In addition, common approaches to avoid those limitation have also been discussed. However, wireless ad hoc networks still demand better solution to overcome those constraints bearing in mind the practical considerations involved in wireless networks. Thus, this dissertation considers two more detailed

and complicated practical issues with respect to the improvement of network performance. Although the protocols and the schemes proposed here have been devised for the improvement of a network layer, this should be done as part of the cross layer design in ad hoc networks.

In wireless ad hoc networks, discovering and establishing a route incur a cost. This directly impacts on the network performance with respect to the throughput, successful delivery rate, number of successfully established connections, and delay. In particular, if each link among the nodes is unstable, the procedure for discovering and establishing a new route must be repeated whenever it is requested. In reality, many obstacles and nodal movements lead to the temporary link disruptions due to the reasons discussed in previous sections. However, most existing routing protocols do not consider that situation. Hence, for temporary link disruptions, most routing protocols must repeat the procedure multiple times, resulting in the degradation of overall network performance. Thus, in this dissertation, a disruption tolerant geographic routing protocol is introduced in Chapter 5.

Another practical issue is encountered for wireless mesh networks (WMNs). In WMNs, each node has multi-radio (or multi-antenna). As a community network, a WMN should support more end-users (referred to as mesh clients). In such an environment, interference is a major reason that degrades the network performance. In general, however, most routing protocols in WMNs do not consider the fact that signal strength is diminished by distance (i.e., the inverse square law). In addition, the physical properties of wireless radios, such as their transmission range, maximum transmission rate, and the number of distinct channels are also simply ignored in most studies of multi-radio and multi-channel WMNs. Thus, in

Chapter 6, a new cost aware route selection (CARS) that considers the physical characteristics of radios and wireless media as much as possible is introduced.

CHAPTER 5

DISRUPTION TOLERANT GEOGRAPHIC ROUTING (DTGR)

Ad hoc networks are formed by a collection of wireless mobile nodes. These networks assume no availability of an established infrastructure or centralized administration, and consist of dynamic wireless links, i.e., new links are constructed and existing links are destroyed or disrupted. The disruptions of the wireless links can either last for a long time, e.g., due to severe network conditions such as an earthquake, on a battlefield, etc.; or only for a short period of time, e.g., due to obstructions in between the communicating nodes. We define the latter as temporary disruptions. Temporary disruptions may occur frequently in ad hoc networks for the following reasons:

- Obstructions present between a sender and a receiver - Obstructions can be stationary or mobile. For instance, a network constructed by vehicles with mounted communication devices may get temporarily disrupted due to the presence of tall buildings.

- Node mobility - Due to mobility, a node j may temporarily move out of the transmission range of its neighboring node i and then move back within the transmission range. This is especially true if j is at the edge of the coverage area of node i.

- Beacon collisions - In many wireless networks, nodes use beacons to validate the availability of a link. Unfortunately, since beacons are commonly exchanged through a shared and uncoordinated channel, the repeated collisions of a beacon from node j with the beacons from other nodes in its vicinity may give its neighboring node

i the wrong impression that the link is down even when it is still up. In general, the increase in probability that a beacon will collide is directly proportional to the increase in node density.

- Noises introduced by other wireless devices - Many wireless technologies, such as IEEE 802.11b and Bluetooth, share the same wireless spectrum. When devices based on different wireless technologies are placed in close range, they tend to interfere with each other. In addition, home/office appliances such as microwave ovens and cordless phones can also generate interference when they are in operation.

Temporary disruptions can affect a node's perception of the status of the links between itself and its neighbors. At any given time, the state of a link between two neighboring nodes can be represented by a binary value of 1 or 0, with 1 indicating that the link is up and, thus, the neighbor is reachable, and 0 indicating that the link is down and, thus, the neighbor is unreachable.[1]

Due to temporary disruptions, the state of a link connecting two nodes can toggle between 1 and 0, where the exact value may not be perceived correctly by the nodes. This is because, in most wireless networks, a node senses its link availability through beacon sampling. If the node receives a beacon from a neighbor, it considers that the state of the link to that neighbor is 1 and, thus, the neighbor can be used for packet forwarding. If a node does not receive a beacon from a neighbor within a certain period of time, it concludes that the state of the link to that neighbor is 0 and does not consider the neighbor

[1] With the state of a link as 1, we do not imply that the transmission error rate of the link is 0, but that the error rate is small enough for the nodes to directly communicate.

for packet forwarding. Although inexpensive, beacon sampling can be misleading because links determined as down may actually be up (e.g., due to beacon collisions) or become up (e.g., due to node mobility) before a node can perceive them.

Due to the incorrect perception of a node about its neighbor set, a neighbor considered by a node as unreachable might actually be reachable. We refer to such neighbors (and the corresponding links) as $unstable$ neighbors (links). $Unstable$ neighbors cannot be captured by the simple binary $0/1$ neighbor set categorization used by the current position-based routing algorithms. In these algorithms, each node i constructs (and maintains) a neighbor table that contains only those neighbors that i perceives as reachable. In order to construct a neighbor table as accurately as possible, these algorithms usually consider unstable neighbors as being unreachable and, thus, do not store them in the neighbor table.

In this chapter, we investigate the effect of node's wrong perception about its neighbor set (in particular, the effect of excluding $unstable$ neighbors in packet forwarding) on the performance of position-based routing algorithms. Since in these algorithms, a node chooses its next hop for packet forwarding solely based on the nodes presented in its neighbor table, we first show, with the help of examples, that not involving $unstable$ neighbors in the decision may either degrade the routing performance of these algorithms or, even worse, render the network disconnected.

We thus propose that node packet forwarding decisions should not completely rule out $unstable$ neighbors. However, because such neighbors have a high probability of being $unreachable$, they should be considered as alternatives when depending on only $stable$ neighbors might result in packet dropping or inefficient routing. We propose associating

each link (neighbor) with a reachability value to accommodate the possible incorrect perception of the node about the state of the link. Associating a reachability value to each neighbor allows a forwarding node to choose from unstable neighbors for packet forwarding when a packet cannot be forwarded to any of the $stable$ neighbors.

To illustrate the effectiveness of our proposed scheme, we apply it to Greedy Perimeter Stateless Routing (GPSR) [55], a well-known position-based routing algorithm, and design two new routing algorithms, Disruption Tolerant Geographic Routing-Simple Forwarding (DTGR-SF) and Disruption Tolerant Geographic Routing-Waiting before Forwarding (DTGR-WF), in which nodes utilize reachability values to make forwarding decisions. We compare the performances of the new routing algorithms with the of GPSR in terms of packet delivery success ratio and packet delivery latency. Through simulations, we show that our schemes achieve better performances than GPSR in cases where disruptions are involved; and in network conditions where GPSR has shown to yield higher packet delivery success ratio and lower average packet delivery latency than other existing routing protocols, our schemes achieve the same high packet delivery success ratio as GPSR with reduced average packet delivery latency.

The remainder of this chapter is organized as follows: Section 5.1 presents literature review; Section 5.2 presents motivation and problem statement; Section 5.3 presents our proposed scheme for link reachability value computation; Section 5.4 applies reachability value to GPSR and presents routing algorithms DTGR-SF and DTGR-WF. Simulation set up and results are shown in Section 5.5.

5.1 Survey of Existing Disruption Tolerant Routing Protocols

Routing in wireless networks in the presence of disruptions can be categorized into two types. The first type is routing under severe network failures, where the network disconnection or disruption is not temporary, but can last for a significant period of time. An example of such networks can be sparse networks composed of mobile robots participating in rescue tasks after an earthquake. Schemes such as Disruption Tolerant Networking (DTN) [56], message ferrying [57], and MV routing [58] fall into this category. To enable communication between disconnected peers under severe network disruptions,, approaches proposed so far require additional capacity and functionality of the peers and/or new relay devices to be deployed in these networks. For example, in [56], mobiles nodes are required to have large storage and power capacity so that messages can be transferred using store and forwarding. In message ferrying schemes proposed by Zhao et al., special mobile devices, referred as ferries are required to travel along predictable routes within the network so that disconnected peers can adjust their movements to meet with the ferries for message upload and download, while in [58], mobile ferries adjust their movements to meet the routing demand of disconnected peers.

The second type is routing under normal conditions, where the network disconnection or disruption is frequent yet temporary. An example of such networks can be ad hoc networks that are temporarily disconnected due to node movements. Topology-based and position-based routing schemes fall into this category.

In topology-based algorithms, an end-to-end path must be constructed either proactively [22] or on demand [28, 31] before the packet is sent from the source. In case of link disruptions, a new path needs to be discovered and updated information needs to be flooded in the network. The resultant overhead makes these algorithms unscalable as the network size increases and less adaptive as the network becomes more dynamic. A survey of topology-based routing algorithms can be found in [59].

On the other hand, using position-based routing protocols such as GPSR, each node makes routing decisions strictly based on its own location, the location of its neighbors, and the location of the destination. Since these protocols do not need to maintain routing table and only require local re-routing in case of link disruptions, they are inherently more robust to network topology changes and scale better as the network size increases than topology-based routing algorithms.

In one recent study [60], assuming that each sensor node knows the packet reception rate (PRR) as well as the locations of its neighbors, the authors propose blacklisting which rules out unreliable neighbors (neighbors whose PRR is below certain value) before making greedy forwarding decision. Since ruling out of neighbors should not disconnect the network, blacklisting is only applicable in networks with high node density.

GPSR is a well-known position-based routing algorithm that combines two forwarding methods - greedy forwarding and perimeter routing. Greedy forwarding allows a packet to be forwarded to the neighbor geographically closest to the destination, while perimeter

routing allows a packet to circumvent a void when there is no neighbor closer to the destination than the current forwarding node. In GPSR, each node actively maintains the following information:

- Its own geographic position obtained through either GPS [55] or other techniques [61];
- A neighbor table containing the addresses and geographic locations of its neighbors; and
- A subset of neighbors that form a planar subgraph. Some planarized graphs, such as Relative Neighborhood Graph (RNG) and Gabriel Graph (GG) can be constructed by each node locally using only the geographic locations of its neighbors. The planar subgraph is required by perimeter routing to route the packet out of a void.

A packet in GPSR can be in either greedy mode or perimeter mode. When a node has a packet to send in greedy mode, it inserts the destination location (obtained through a location service [62]) inside the packet, sets the packet into greedy mode, and forwards it using greedy forwarding. If the packet cannot be forwarded in greedy mode due to the presence of a void, the node employs perimeter routing and forwards the packet in perimeter mode.

In perimeter routing, each forwarding node applies righthand rule to select the next hop so the packet can traverse along planar faces that converge toward the destination. The packet remains in perimeter mode until it reaches a node i closer to the destination than the node where the perimeter forwarding started. In this case, i sets the packet back to greedy mode and starts greedy forwarding procedure. The routing of the packet is terminated when

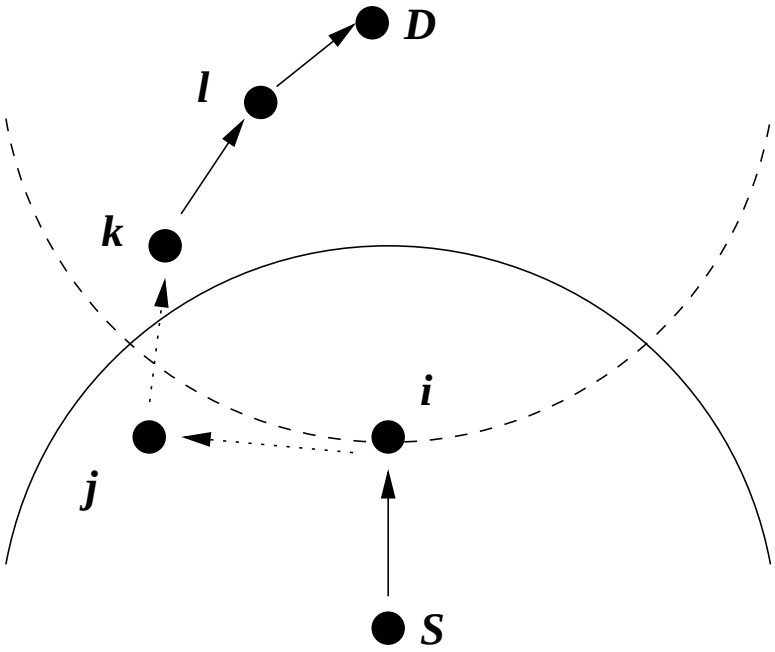


Figure 5.1: GPSR packet routing example, where S is the $source$ node and D is the $destination$ node. The solid line indicates that the packet is forwarded using greedy forwarding, and the dotted line indicates that the packet is forwarded using perimeter routing

the packet reaches the destination or when there is no neighbor to which the packet can be forwarded using either of the above two methods, in which case, the forwarding node drops the packet. Please refer to Fig. A.1 for GPSR routing algorithm.

The greedy forwarding and perimeter routing procedures briefly described above are shown with the help of Fig. 5.1. Source node S first forwards the packet in greedy mode to i, which has no neighbor closer to D than itself, and hence i sets the packet in perimeter mode and sends it to k. When the packet reaches k, which is closer to D than i, k sets the packet back to greedy mode and forwards it to l using greedy forwarding, which in turn forwards the packet to the destination D.

5.2 Motivation and Problem Statement

5.2.1 Motivation

As in position-based algorithms, each node selects the next hop for packet forwarding $solely$ from nodes stored in its neighbor table, the correctness of the neighbor table is crucial in determining the performance of these algorithms. In general, the construction and maintenance of the neighbor table are through a simple beacon sampling mechanism: each node periodically broadcasts beacons to its neighbors to announce its ID (e.g., IP address) and location. When a node i receives a beacon from a node j, i stores j's ID and location in its neighbor table. If i does not receive a beacon from its neighbor j within a time-out interval T , where T is a multiple of beacon interval B, i concludes that j is $unreachable$ and, thus, immediately removes j from its neighbor table. Sometimes node i's own perception that neighbor j is $unreachable$ may not be correct. This is because neighbors usually contend with each other to send beacons over a shared channel [63], and hence, the beacon from a neighbor j can be lost due to the collision with other beacons even if j is indeed $reachable$. The results presented by Huang et al. in [64] further illustrates the above problem. Through analysis and simulation, the authors showed that in IEEE 802.11 ad hoc networks, where all nodes are neighbors of each other, the probability that a specific node successfully sends a beacon in one beacon interval is low even in a small network, e.g., 19.7% for a network of five nodes; and the probability decreases fast as the network size increases, e.g., 4% for a network of twenty five nodes. Given a network with twenty five nodes, even if we set the neighbor time-out interval T to be as long as twenty

beacon intervals, the probability that a $reachable$ neighbor is incorrectly considered as $unreachable$ is still as high as $(1 - 4\%)^{20}$ = 44.2%. The beacon loss ratio is expected to be much higher if we take the transmission of data packets into consideration. The link disruptions resulting from background noise, multipath fading, and obstructions such as moving objects, etc. further worsen the situation.

5.2.2 Definitions and Problem Statement

We consider a large-scale mobile ad hoc network in a 2- D coordinate plane. All nodes are assumed to have the same transmission range R. A link $e(i, j)$ exists between two nodes i and j if and only if the Euclidean distance, $dist(i, j)$, between them is less than or equal to R. Nodes i and j are neighbors if $e(i, j)$ exists. The set of nodes and links are represented by sets V and E, respectively and the resultant undirected graph is represented by G, where $G = (V, E)$. We use $s_{i,j}$ to denote the state of $e(i, j)$, where $s_{i,j} \in \{0, 1\}$. The value 1 indicates that nodes i and j can directly communicate with each other, and 0 indicates that i and j can not directly communicate with each other. We use $p_{i,j}$ to denote the state of $e(i, j)$ perceived by the node i through beacon sampling, where $p_{i,j} \in \{0, 1\}$. $p_{i,j}$ = 1 indicates that i perceives the state of $e(i, j)$ is 1 and thus j is $reachable$, and $p_{i,j}$ = 0 indicates otherwise.[2] A node i has an incorrect perception of the state of the link $e(i, j)$ iff $p_{i,j} \neq s_{i,j}$, e.g., $s_{i,j}$ = 0 and $p_{i,j}$ = 1, or $s_{i,j}$ = 1 and $p_{i,j}$ = 0, with the latter being the focus of the study in this chapter. Temporary disruptions in conjunction with beacon sampling may result in the incorrect perception by a node about its neighbor set.

[2]Note that (i) both $s_{i,j}$ and $p_{i,j}$ may change as time varies and (ii) $p_{i,j}$ can be different from $p_{j,i}$.

The incorrect perception of a node about its neighbor set may result in the removal of a $reachable$ neighbor from the neighbor set of the node, which in turn, may adversely affect the performance of position-based routing algorithms in terms of packet delivery success ratio and packet delivery latency. To address the above problem, our goal is to propose a scheme that (i) improves the performance of position-based routing algorithms in the presence of temporary disruptions; (ii) does not compromise the overall packet delivery success ratio and packet delivery latency in networks with no or few temporary disruptions; and (iii) introduces minimal overhead in terms of space and time requirements.

5.3 Reachability Value Computation Scheme

To achieve the above goal, we propose assigning a reachability value for each link $e(i, j)$. In this section, we first present our scheme for computing the reachability value for a link. We then present how to construct the neighbor table using the computed reachability values.

Each node i associates a link between itself and its neighbor j with a value $r_{i,j} \in [0, 1]$, which represents the stability metric of the link as perceived by node i.[3] For each node i, a link $e(i, j)$ is stable if $e(i, j) \in E$ and $r_{i,j}$ = 1, or $unstable$ if $e(i, j) \in E$ and $r_{threshold} \leq r_{i,j} < 1$, where $r_{threshold}$ is a constant between $(0, 1)$. The link $e(i, j)$ is considered as unreachable if $0 \leq r_{i,j} < r_{threshold}$. For each node i, a neighbor j is $stable$ if $e(i, j)$ is $stable$, $unstable$ if $e(i, j)$ is $unstable$, and $unreachable$ if $e(i, j)$ is $unreachable$.

[3]Note that (i) $r_{i,j}$ may change as time varies and (ii) $r_{i,j}$ can be different from $r_{j,i}$.

Since $unstable$ neighbors have higher probability of being actually $unreachable$ than $stable$ neighbors, they should be considered with lower priority than $stable$ neighbors when a node makes forwarding decisions. These $unstable$ neighbors should be utilized only as alternatives when selecting next hop from only $stable$ neighbors might force the routing algorithm to drop the packet or to route the packet inefficiently. Associating each neighbor with a reachability value allows a node to efficiently utilize both $stable$ neighbors as well as $unstable$ neighbors in packet forwarding.

Each node i, in addition to storing the address and location of each of its neighbor j, also maintains $j's$ reachability value in the neighbor table. Node i uses beacons to approximate $r_{i,j}$ as follows

$$\widehat{r}_{i,j} = \begin{cases} 1, & \text{if } l \leq T \\ \max(1 - 2^{(l-T)} \times 0.1, 0), & otherwise. \end{cases}$$

where T is a given constant representing the time-out interval, l is defined as the number of consecutive beacon intervals during which node i has not received a beacon from j. If node i has received at least one beacon from j in the last T beacon intervals, i assigns j a reachability value of 1. Otherwise, starting from the $(T+1)th$ beacon interval, i decreases $\widehat{r}_{i,j}$ exponentially. We achieve this by exponentially increasing the value by which $\widehat{r}_{i,j}$ is decreased. When the reachability value of neighbor j is below $r_{threshold}$, i removes j from its neighbor table. For example, if $l = 5$ and $T = 3$, then $\widehat{r}_{i,j}$ = 0.6. If $r_{threshold}$ is set as 0.6 and after another one beacon interval ($l = 6$), i has not received a beacon from j, then $\widehat{r}_{i,j} = 0.2$ and i removes j from its routing table. Notice that if i

receives at least one beacon from an $unstable$ node j or from a new node, i assigns that node a reachability value of 1.

5.4 Application: GPSR

The removal of a $reachable$ neighbor may prevent the position-based routing algorithms from choosing an efficient neighbor for packet forwarding. In this section, using GPSR as an example, we first identify the possible scenarios where the performance of position-based routing algorithms is compromised due to the removal of a $reachable$ neighbor from the neighbor table. We then apply a link reachability scheme on GPSR and present the resultant Disruption Tolerant Geographic Routing algorithms.

5.4.1 Adverse Effect of the Incorrect Removal of a Reachable Neighbor from the Neighbor Table.

In both Fig. 5.2 and Fig. 5.3, i is the forwarding node for a packet destined for node D. $s_{i,j}$ = 1 and $p_{i,j}$ = 0 and, thus, node j is a $reachable$ neighbor of i, but i has removed j from its neighbor table. The removal of j from the neighbor table causes i to either drop the packet (as shown in Fig. 5.2) or to route the packet in perimeter mode (as shown in Fig. 5.3). Dropping the packet requires the packet to be retransmitted, and forwarding the packet in perimeter mode usually introduces a much longer route, higher packet delivery latency, and a higher packet loss ratio than forwarding the packet to j in greedy mode.

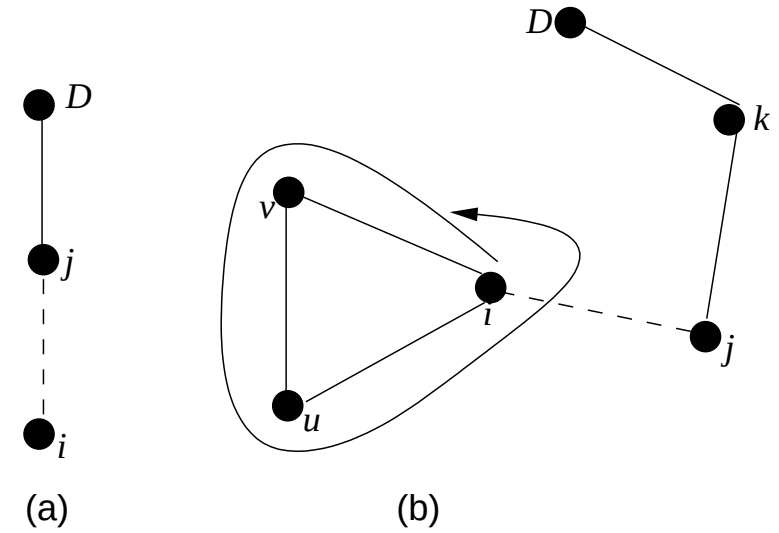


Figure 5.2: Dropping of a packet due to the removal of $reachable$ neighbor j from i's neighbor table

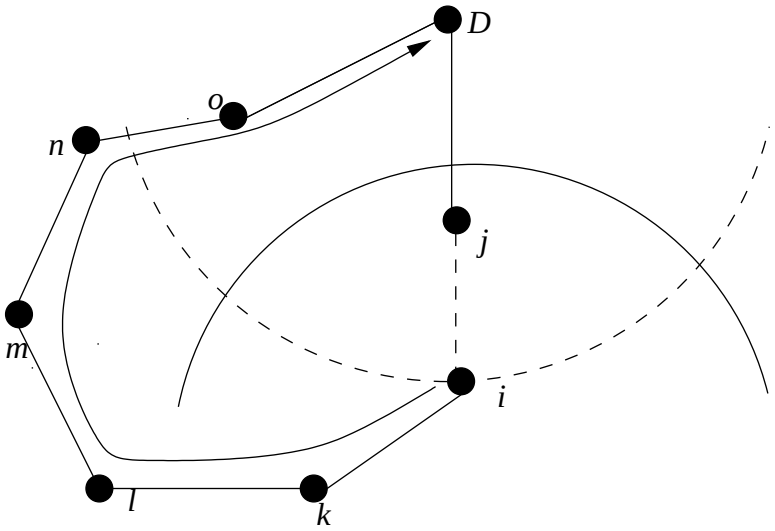


Figure 5.3: Inefficient perimeter routing of a packet destined for D due to the removal of $reachable$ neighbor j from i's neighbor table

5.4.2 Disruption Tolerant Geographic Routing (DTGR)

In DTGR, each node constructs and maintains its neighbor table as presented in Section 5.4. A node i selects a $stable$ neighbor to whom it will forward the packet. It selects $unstable$ neighbors only under the following critical conditions:

- **Before setting a greedy mode packet into perimeter mode.** In GPSR, when a greedy mode packet gets stuck at a node i that does not have a closer neighbor to the destination, i sets the packet into perimeter mode and routes it along a planar subgraph using the righthand rule. Although perimeter routing allows a packet to recover from local maxima, it usually introduces longer routes, higher packet delivery latency, and a higher packet loss ratio, since nodes that can be reached via a path with few hops might become far apart after planarization. In DTGR, we use the reachability value to assist node i to continue forwarding the packet in greedy mode. Thus when node i receives a packet m in greedy mode, i first tries to forward m in greedy mode using neighbors with reachability value of 1, as shown in $step1$ of $DTGR\_Greedy.i.module$ in Fig. A.2. If all i's neighbors with reachability value of 1 are further to the destination than i, then, instead of setting the packet into perimeter routing mode immediately, i tries to select a neighbor from those with a reachability value of less than 1 to continue forwarding the packet in greedy mode, as shown in $step2$ of $DTGR\_Greedy.i.module$ in Figure A.2.

- **Before dropping a packet.** In GPSR, a forwarding node i drops a packet when there is no neighbor j to which i can forward the packet in either greedy mode or perimeter

mode. In DTGR, instead of dropping the packet, forwarding node i tries to utilize the $unstable$ neighbors to forward the packet. In greedy mode, $unstable$ neighbors will be tried as described above. If no such $unstable$ neighbors are available, i will try to forward the packet in perimeter mode on the local planar subgraph constructed from its $stable$ neighbors. If the packet cannot be forwarded to any of the $stable$ neighbors in perimeter mode either, then instead of dropping the packet, i will first construct a planar subgraph consisting of both $stable$ and $unstable$ nodes, and then forward the packet in perimeter mode to an $unstable$ node, if available.

- **DTGR graph planarization.** The routing of a packet out of the local maxima using perimeter routing requires the removal of cross links from the underlying connectivity graph. In GPSR, each node has only one level local graph, i.e, the node maintains local graph formed by links between the node and its $stable$ neighbors only. Assuming that the underlying network is always connected, each node runs planarization algorithms such as RNG or GG in a distributed manner to remove local cross links. However, in a network where temporary link disruptions occur frequently, the underlying network constructed only by $stable$ nodes and links might not be connected all the time, and thus, the resultant planar subgraph also might not be connected, which may result in packet dropping (see Fig 5.2(b)).

  In DTGR, each node has at least two level local graphs: the first level is formed by links between the node and its $stable$ neighbors (neighbors with $\widehat{r}_{i,j} = 1$), and the second level by the links between the node and all its neighbors with $r_{threshold} \leq$

$\widehat{r}_{i,j} < 1$, and thus two level planar subgraphs. After determining that a packet (either in greedy mode or perimeter mode) has to be forwarded in perimeter mode, the node first checks if there exists a next hop neighbor, by applying the right-hand rule on the first level planar subgraph as shown in $step2$ of $DTGR\_Greedy.i.Module$ and $step2$ of $DTGR\_Perimeter.iModule$ in Fig. A.2. If no such neighbor exists, the node then tries to select the next hop neighbor by applying the right-hand rule on the second level planar subgraph as shown in $step4$ of $DTGR\_Greedy.i.Module$ and $step3$ of $DTGR\_Perimeter.iModule$ in Fig. A.2. Since (i) long links have high probability of being $unstable$; and (ii) both RNG and GG favor shorter links than longer ones, the probability that a $stable$ cross link is removed due to short $unstable$ links is low.

5.4.3 Schemes for Forwarding a Packet to a Selected Unstable Neighbor

Once node i has selected an $unstable$ neighbor j to which it will forward the packet, it can utilize the following forwarding schemes:

- **Simple forwarding (DTGR-SF):** In this scheme, i forwards the packet to j immediately after j is selected.
- **Wait before forwarding (DTGR-WF):** In this scheme, i waits for a certain time t_{wait} before forwarding the packet to j. After t_{wait}, i checks if there is a better candidate k than j, e.g., a $stable$ neighbor to which the packet can be forwarded in greedy mode, and forwards the packet to k accordingly. Otherwise, i forwards the packet to j.

In the above two schemes, if node i could not forward the packet to j successfully, i selects another node to whom it can forward the packet. If i fails to forward the packet to this node too, i drops the packet.

DTGR-SF is simpler to implement and introduces no extra delay, yet it might result in a lower packet forwarding success ratio due to its best effort property. DTGR-WF should yield a higher packet delivery success ratio while introducing longer packet delivery latency, since it waits for a certain time for a $stable$ neighbor to be available before forwarding the packet to the selected $unstable$ neighbor.

5.5 Simulation Result and Analysis

5.5.1 Simulation Setting and Parameter Consideration

Both DTGR-SF and DTGR-WF are implemented in Network Simulator (NS) [65] and their performance is compared with GPSR using the wireless extensions provided by [66] under various models of disruption, traffic, mobility, network density, and topology. The performance metrics used are (i) packet delivery success ratio, which is defined as the ratio of the total number of packets received by all destinations in the network over the total number of packets sent by all sources in the network; and (ii) average packet delivery latency, which is defined as the average delivery time of all packets that are successfully received by all destinations in the network. In the following part of this section, we present the implementation of DTGR Routing Agent, various Timers, the temporary link disruption simulation model, various simulation configurations, and the simulation results.

DTGR routing protocol is implemented as an Agent in NS. When a node (implemented as an MobileNode object in NS) receives a packet, it passes the packet to its DTGR Routing Agent. The Routing Agent handles the packet based on the type and forwarding mode of the packet, and the current routing algorithm type (passed from the simulation TCL file). For example, if the packet is a broadcast packet, then the Routing Agent will forward the packet to all its neighbors in the neighbor table, if it is a unicast packet and the node is not the packets destination, then the Routing Agent selects a neighbor as the next hop for packet forwarding based on the packets current forwarding mode, the forwarding nodes own location, the packet destination location, as well as the location and reachability of each of the neighbors of the current forwarding node.

Timers in NS are used to schedule events, please refer to chapter 11 of NS document [65] for more detailed explanation of the usage of Timers. Several important Timers implemented in DTGR are: 1) Beacon Timer, which is used by a node to schedule the next beacon event, 2) Neighbor time out Timer, which is used to fire the neighbor removal event, and 3) DTGR-WF timer, which is used by DTGR-WF to fire the next hop neighbor selection event if an $unstable$ neighbor is selected.

To simulate temporary link disruptions resulting from obstructions, noises, and beacon collisions, each mobile node is configured with a multistate error model, where each nodes state is either on (0 packet loss ratio) or off (100% packet loss ratio) at various periods. The on and off time durations follow the normal distribution. The link-based error model is not used, since, as far as we know, NS does not have a link object to support this function in the wireless network setting. For each simulation setup, three different

randomly-generated motion patterns and topologies are run, and the average of their performance is calculated as our final result.

In all of the simulations, the time-out interval T is $4.5B$, where B is the beacon interval. Similar as in [55], each beacon's transmission is jittered by 50% of the beacon interval B in order to avoid possible neighbor beacon synchronization. The random waypoint model [67] is used to simulate node mobility. The value of $r_{threshold}$ is 0.6.[4]

5.5.2 Networks with Temporary Disruptions Resulting from Obstructions, Noises or Beacon Collisions

The simulations are performed in both static and mobile networks with 50 nodes randomly placed in a 1500×600 rectangular area. In the mobile networks, the maximum speed of each node is $30m/s$. For both static and mobile network settings, the beacon interval is $0.8s$ and the t_{wait} of DTGR-WF is $1.6s$. Each simulation is run for $500s$, during which 15 CBR sources transmit $64B$ packets at $1Kb/s$. In all simulations, each node is periodically on and off at varying start times. The average on time is fixed as $20s$ with a standard deviation of $5s$. The average off time varies from $2.0s$ to $3.5s$, with a fixed standard deviation of $0.5s$.

Fig. 5.4 and Fig. 5.5 show the performance of both DTGR-WF and GPSR in a static network when the average node off duration is varied.[5] Fig. 5.4 shows that as node off

[4]Note that in our simulations, we fix $r_{threshold}$ to 0.6 as it shows the best performance in most of our simulation settings. It's value should be adjusted based on different network settings such as node density, speed and the various disruption models.

[5]Note that the performance of DTGR-SF is not shown in Fig. 5.4 - Fig. 5.7 because compared with DTGR-WF, DTGR-SF's performance gain over GPSR is trivial.

duration increases, the packet delivery success ratio of both schemes decreases. However, compared with GPSR, DTGR-WF achieves a much higher packet delivery success ratio under all cases. For example, when the average node failure time is $2s$, the delivery success ratio of DTGR-WF is 86.8%, which is 34.2% higher than that of GPSR. Also, the performance of DTGR-WF degrades more gracefully than DTGR-SF; thus, DTGR-WF is more resistant to temporary network disruptions. Fig. 5.5 shows that DTGR-WF has a longer average delivery latency due to its wait and forwarding property.

Fig. 5.6 shows the performance of the schemes in a dynamic network, in which each node moves at a maximum speed of $30m/s$. Here the packet delivery success ratio has the same trend as that in the static network, with DTGR-WF showing a significant gain over GPSR. As shown in Fig. 5.7, (i) the delivery latency obtained from both schemes is higher than that found in the static network and (ii) the delivery latency of DTGR-WF is comparable to that of GPSR due to the possible reason that when nodes are mobile, the extra wait time of DTGR-WF allows a new $stable$ neighbor to be used for forwarding the packet in greedy mode.

5.5.3 Networks with Temporary Disruptions Resulting from Node Mobility

We are also interested in finding out how these protocols perform in a sparse network as the degree of node mobility increases. The disruptions in such networks can be seen as frequent link insertions and deletions resulting from the frequent joining and leaving of neighbors. In a sparse network, designing a routing protocol to be resilient to high random mobility is more challenging as each node has a small neighbor set. The simulations are

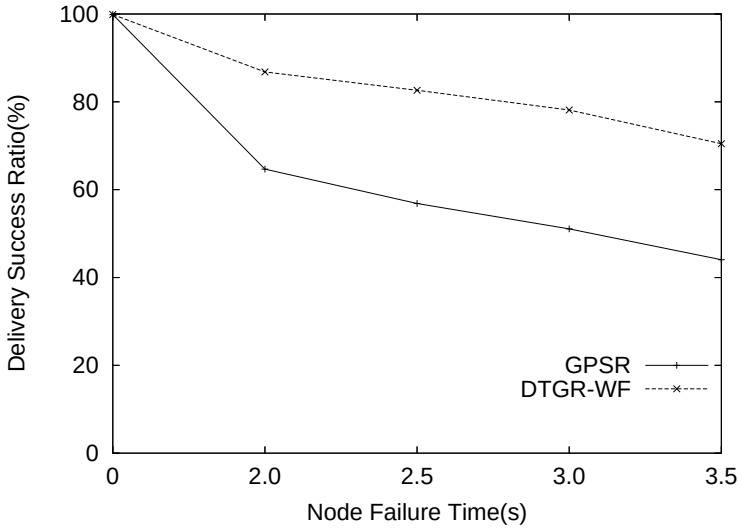


Figure 5.4: Packet delivery success ratio under different node failure durations in a static network

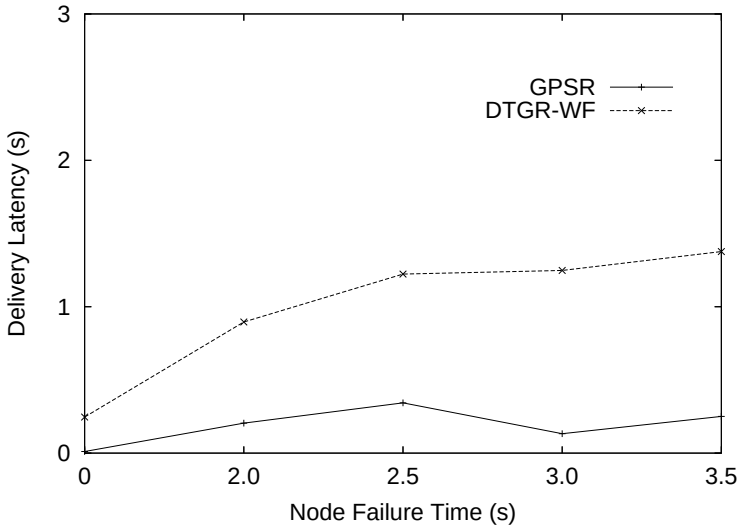


Figure 5.5: Packet delivery latency under different node failure durations in a static network

set such that there is a pause time of $0s$, 50 nodes move randomly in a 1340×1340 grid, and there are 5 neighbors/node on average. Each simulation lasts $900s$, during which 30 CBR sources transmit $64B$ packets at $2Kbps$. The beacon interval is $1s$ and the t_{wait} in DTGR-WF is $2s$. We do not count the packets dropped by the sender at the beginning of the simulation when all nodes are stationary, because the low density of the network can easily result in a situation where the sender is truly an isolated node.

Fig. 5.8 and Fig. 5.9 show the performance of the various schemes as the maximum speed of the node varies. Again, DTGR-WF exhibits a higher packet delivery success ratio compared to GPSR and DTGR-SF. DTGR-WF has an average gain of 37% over GPSR. The packet delivery success ratio of DTGR-SF lies between those of DTGR-WF and GPSR, with an average gain of 14.5% over that of GPSR. For example, at $40m/s$, the packet delivery success ratio is 69.1% , 55.5%, and 49.1% for DTGR-WF, DTGR-SF, and GPSR, respectively. The performance gain of DTGR-WF over GPSR is 40.7% and the performance gain of DTGR-WF over DTGR-SF is 24.5%. As for the data delivery latency, it is interesting that, despite the extra $2.0s$ that DTGR-WF might spend before forwarding a packet, it achieves the lowest average latency in almost all cases. GPSR, on the other hand, has the highest average latency. DTGR-SFs performance falls between those of DTGR-WF and GPSR. DTGR-WF has better performance than the other schemes because new nodes may become neighbors of a DTGR-WF forwarding node during the wait period, thus allowing the packet to be routed in greedy mode. In other words, DTGR-WF allows the packet to be routed using a smaller number of hops, thus results in lower latency.

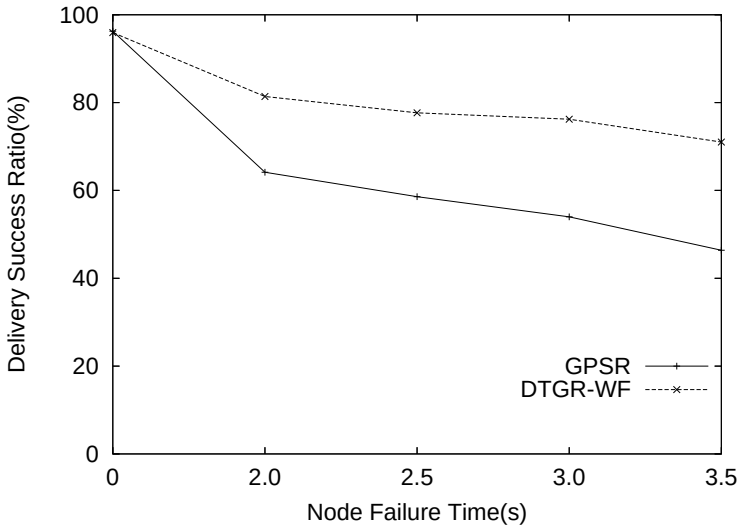


Figure 5.6: Packet delivery success ratio under different node failure durations in a mobile network where maximum node speed = 30 m/s

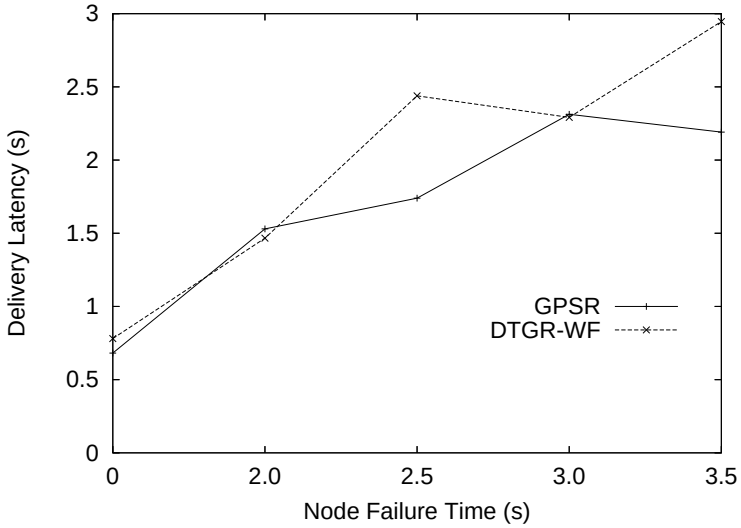


Figure 5.7: Packet delivery latency under different node failure durations in a mobile network where maximum node speed = 30 m/s

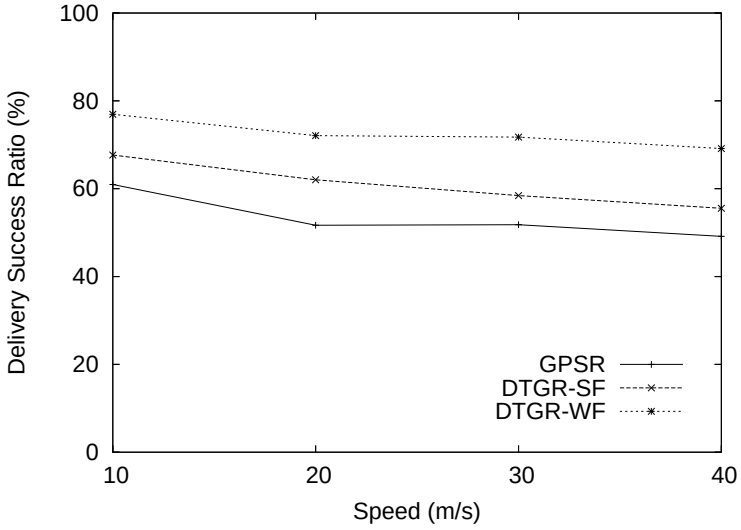


Figure 5.8: Packet delivery success ratio in sparse, highly mobile networks

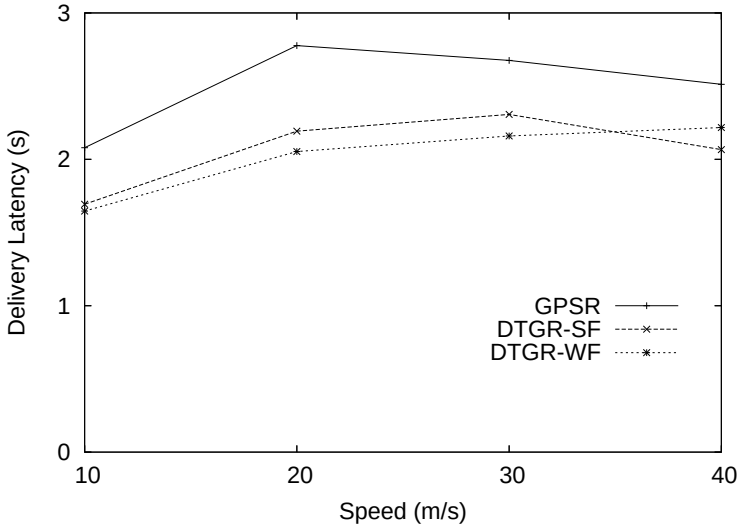


Figure 5.9: Packet delivery latency in sparse, highly mobile networks

5.5.4 Networks with Fewer Occurrences of Disruptions

The above results indicate that DTGR-SF and DTGR-WF perform better than GPSR when disruptions are caused either by temporary node unreachability or by frequent link insertions and deletions. Next we investigate how these protocols perform in the same settings as those used in [55], in which GPSR has been shown to provide a higher packet delivery success ratio and a lower latency than other ad hoc routing protocols. As was the case in [55], we randomly place 50 nodes in a grid of size 1500×300 (20 neighbors/node on average), where each node moves with a maximum speed of $20m/s$, the beacon interval is set as $1.5s$, and the t_{wait} in DTGR-WF is set as $3.0s$. Each simulation has 30 CBR flows originating from 22 source nodes. In addition to the 0-, 60-, and $120s$ pause times used in [55], we also include 300-, 600-, and $900s$ pause times. Each simulation lasts for $900s$.

Table 5.1 and Fig. 5.10 present the packet delivery success ratio and delivery latency. Table 5.1 reveals that all three schemes show similar packet delivery success ratios of near 100%. In terms of the data delivery latency, DTGR-SF yields the lowest latency, followed by DTGR-WF and GPSR. This is because (i) in a dense network where each node has an average of 20 neighbors and $temporary\ disruptions$ are not considered, the possibility that a packet gets stuck in a local maxima is rare, thus allowing a packet to be forwarded in greedy mode most of the time; and (ii) the high node density increases the possibility that beacons from a reachable neighbor will collide with other beacons. In this case, GPSR removes the neighbor from the neighbor table, while DTGR-SF and DTGR-WF keep the neighbor in the neighbor table with a low reachability value so that it can still be used to forward packets in greedy mode.

Pause time(s)	GPSR	DTGR-SF	DTGR-WF
0	98.65	98.91	98.46
10	99.08	99.14	99.16
20	95.90	96.12	96.33
30	99.59	99.58	99.51
40	99.66	99.71	99.63
50	99.98	99.98	99.98

Table 5.1: Packet delivery success ratio (%) in networks with fewer occurrences disruptions

5.5.5 Discussions

In comparison with GPSR, the higher packet delivery success ratio of DTGR incurs the following overhead: (i) higher storage cost because of the larger number of entries stored in the neighbor table; and (ii) possible longer delivery latency when DTGR-WF is used. The above overhead is justifiable because:

- Similar to GPSR, DTGR is nearly stateless, in the sense that with DTGR, each node only stores in the neighbor table the location and address of each of its neighbors. If we use sixteen bytes to store each neighbor (eight bytes floating point values for position coordinates, four bytes for address, and four bytes for reachability value), then the storage space required for a 100-entry neighbor table is only 1.6 KB. Considering that a network with twenty neighbors per node is already very dense[55], we believe that the extra storage space required for $unstable$ neighbors is negligible.

- The extra wait time of DTGR-WF before forwarding the packet to an $unstable$ neighbor might result in longer average packet delivery latency. As shown in Fig.5.5, when the disruptions are due to obstructions, beacon collisions, etc., the average packet delivery latency of DTGR-WF is higher than that of GPSR. However, under some

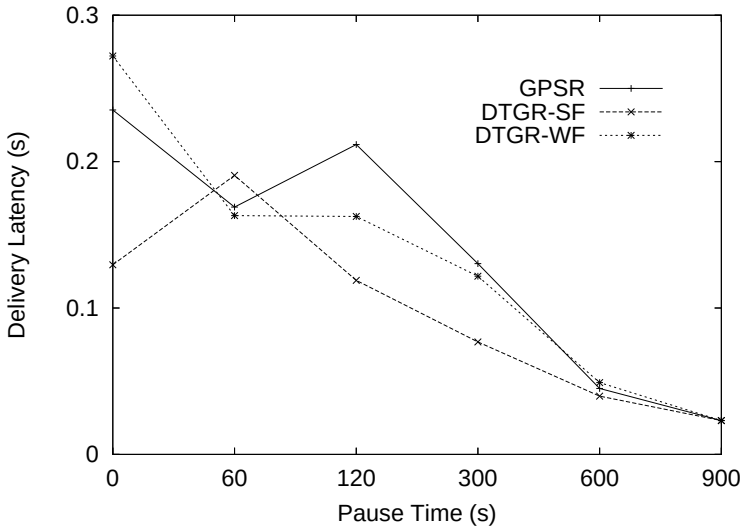


Figure 5.10: Packet delivery latency in Packet delivery success ratio (networks with fewer occurrences of disruptions

cases, for example, when the network is sparse and the nodes are mobile, the average packet delivery latency of DTGR-WF is the lowest, as shown in Fig.5.9, This might be due to the fact that the extra wait time of DTGR-WF allows a new $stable$ neighbor to be used for forwarding the packet in greedy mode. Also, it is worth mentioning again that in DTGR-WF, $unstable$ neighbors are considered only under critical conditions when considering reliable neighbors only might result in packet dropping or inefficient routing.

5.6 Summary

In this chapter, we showed that temporary link disruptions in conjunction with neighbor table construction via beacon sampling can result in an incorrect perception by a node

about its neighbor set, which, in turn, can adversely affect the performance of position-based routing protocols. We then proposed a scheme that allows each node to associate each of its neighbors with a reachability value to accommodate the incorrect perception.We designed two new routing algorithms, Disruption Tolerant Geographic Routing-Simple Forwarding (DTGR-SF) and Disruption Tolerant Geographic Routing- Waiting before Forwarding (DTGR-WF), in which nodes utilize reachability values to make forwarding decisions. We compared the performances of DTGR-SF and DTGR-WF with that of GPSR in various simulation settings.

Some key results are that: (i) when temporary disruptions due to obstructions, beacon collisions, etc., are present, DTGR-WF performs significantly better than GPSR in terms of the packet delivery success ratio with the tradeoff of the increased packet delivery latency. For example, in network settings where nodes are static, DTGR-WF achieves an average 46.8% higher packet delivery success ratio than GPSR; (ii) when temporary disruptions due to node mobility are present, DTGR-WF performs the best in terms of packet success ratio with an average gain of 37% over GPSR and 20% over DTGR-SF. It also achieves the lowest average latency in almost all the cases. GPSR achieves the lowest packet delivery success ratio as well as the highest packet delivery latency; and (iii) in network conditions where GPSR has shown to yield higher packet delivery success ratio and lower average packet delivery latency than existing routing protocols, our schemes achieve the same high packet delivery success ratio as GPSR with reduced average packet delivery latency. Some directions we are currently exploring are: (i) investigating the relationship between various types of link disruptions (e.g., non-deterministic fading models) and the performance of

ad hoc and sensor network routing protocols; (ii) designing an adaptive algorithm that constructs and maintains a neighbor set based on the current network conditions, e.g., node density, mobility, etc.; and (iii) deriving the reachability function using the feedback from link level information such as the packet loss ratio, etc.

CHAPTER 6

COST-AWARE ROUTE SELECTION (CARS)

Wireless Mesh Networks (WMNs) have emerged as one of the most promising applications of ad hoc networks. By connecting inexpensive mesh routers with multiple radios wirelessly, WMNs can quickly provide broadband networking infrastructure for large business enterprizes and bring Internet access to residence in rural areas. An example of WMNs is depicted in Figure 6.1.

To take full advantage of WMNs, many research issues, such as backbone construction, cross-layer design, multi-channel MAC, and fault tolerance [68], are yet to be addressed. Among them, routing is perhaps one of the most important topics. At first glance, since WMNs are considered as a special type of ad hoc network, it seems appropriate to use one of the routing protocols originally developed for ad hoc networks [69] for WMNs. However, such an approach overlooks the following three key differences between research in WMNs and traditional ad hoc networks, and is thus likely to result in poor performance.

- Node classification - Traditional ad hoc networks are formed by nodes that are commonly assumed to be homogeneous in terms of the hardware/software configuration and degree of mobility. In contrast, wireless mesh networks are composed of two distinct types of nodes - mesh routers and mesh clients. Mesh routers, similar to conventional wireless access points, are generally assumed to be built using inexpensive parts, to be stationary, and to be connected to an external power supply. Notice

that most mesh routers are not connected directly to the wired backbone. If a mesh router is connected to the wired backbone, we referred it as the gateway or gateway mesh router in particular. The mesh clients, such as laptops and handheld PDAs with wireless LAN [16] capability, run on their own batteries and move at moderate speed.

- Multiple antennas - To increase the capability of WMNs, mesh routers can be equipped with multiple radio interfaces. Each interface can adopt one of the three wireless standards: IEEE 802.11a [18], 802.11b [17], and 802.11g [19]. The different standards present distinct by different physical characteristics, particularly with regard to their radio spectrum, transmission rate, and transmission radius. This immediately presents two challenges for the protocol design. First, the topology of WMNs is no longer a simple graph. Depending on which radio interfaces are available, a mesh router can have several different sets of neighbors. Second, the channels used by different radio interfaces can interfere with each other if the portion of the radio spectrum used by these interfaces overlap with each other.

- Adaptive transmission rate - In most research on ad hoc networks, the unit disk model is used [70, 71]. In this model, the transmission rate between two nodes within a predefined transmission range is assumed to be a constant. However, it is known that the transmission rate between two wireless LAN entities can automatically step down if the quality of the link between them degrades. For instance, depending on the distance between two nodes, the transmission rate of a IEEE 802.11b link can be

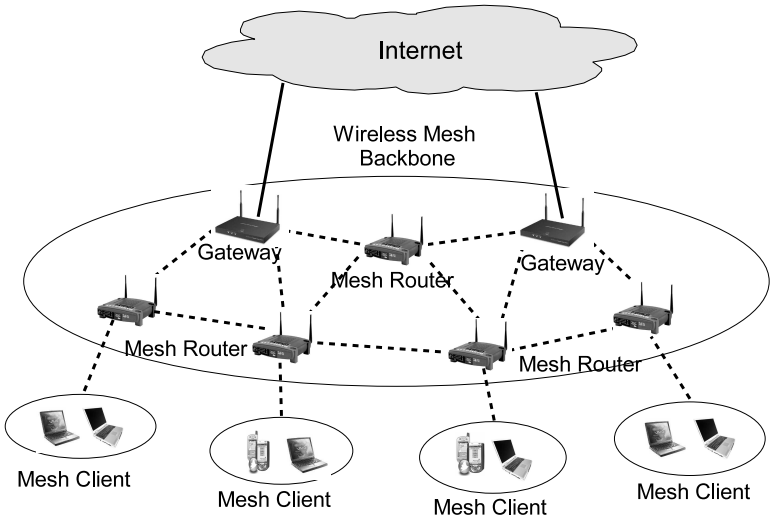


Figure 6.1: An Example of Wireless Mesh Network

either 11Mbps(0m - 50m), 5.5Mbps(51m - 62m), 2Mbps(62m - 68m), or 1Mbps(68m - 85m) [72].

These differences further complicate the issue of routing in WMNs. In [73], it was shown that finding the optimal route in a multi-radio WMN is NP-hard. As the first step toward solving this problem, most previous proposals [73, 74, 75, 76, 77] suggested different metrics that can be used to help identify the best one out of a set of candidate routes. It is expected that by using these metrics for route selection, the overall throughput of the mesh network can be improved.

In this chapter, we propose a novel route selection scheme is proposed, namely Cost-Aware Route Selection (CARS), for WMNs. Unlike the past route selection schemes, which are primarily based on the quality of links in a route, the new scheme takes the interference cost and traffic aggregation into consideration. By selecting the route with

the best bandwidth and cost ratio from a set of candidates, the limited wireless resources (i.e., the available channels for mesh routers) can be better utilized. This will automatically lead to better overall network throughput. The simulation results show that the proposed CARS scheme significantly improves the overall network throughput by more than 150% in the case of burst traffic and the number of connections by more than 95% in the case of constant bit rate traffic.

The remainder of this chapter is organized as follows. In Section 6.1 we review the existing route metrics and route selection schemes for WMNs are required. The proposed route metric and scheme are described in Section 6.2 and the simulation results and analysis are provided in Section 6.3. Finally, the chapter concludes by summarizing the research and pointing out the future research directions in Section 6.4.

6.1 Survey of Existing Route Selection Schemes in WMNs

Routing is one of the most fundamental issues in WMNs. In the past, several metrics were proposed for multi-hop wireless networks in order to measure the quality of a route. In [76], the Expected Transmission Count (ETX), which is based on link layer frame loss rates, was used to locate a path with higher throughput in a multi-hop wireless network. However, ETX does not take into account the bandwidth of links in a path. In addition, ETX does not give preference to channel diversity.

In [78], a link quality source routing (LQSR) protocol was proposed which selects a route according to a specified link quality metric. LQSR is an extension of the dynamic source routing protocol [29]. In [78], three different link quality metrics: ETX, per-hop

round-trip time, and per-hop packet pair, were evaluated and compared, along with LQSR. However, LQSR was designed primarily for nodes with a single radio interface.

In [77], the authors promoted the uses of multiple radio interfaces at each mesh router for the improvement of network capacity. Since then, most research on WMNs has adopted this idea. However, while such configurations enable a mesh router to simultaneously transmit and receive packets, it also complicates the selection of routes. It has been shown that finding the optimal route for a given source-destination pair with the best radio and channel in a multi-radio WMN is an NP-hard problem [73].

In [77], a multi-radio LQSR (MR-LQSR) was proposed for mesh routers with multiple radio interfaces. MR-LQSR incorporates several performance metrics. The Expected Transmission Time (ETT), which is essentially the expected time to transmit a packet of a certain size over a link, is introduced to measure the quality of a link. ETT accounts for both packet loss rate and link bandwidth. The Weighted Cumulative Expected Transmission Time (WCETT) is used to measure the quality of a path. WCETT is a combination of the Summation of ETT (SETT) and Bottleneck Group ETT (BG-ETT), which is the sum of expected transmission time of a bottleneck channel. WCETT takes into account both link quality metric and the minimum hop-count. Depending on the parameter set for SETT and BG-ETT in WCETT, MR-LQSR generally achieves a good tradeoff between delay and throughput. However, MR-LQSR does not consider interference, as the authors assumed that all the radio interfaces on each mesh router are tuned to non-interfering channels. In reality, the number of available channels is limited, so when multiple traffic flows are running on the network the impact of interference should not be overlooked.

In [74], a centralized channel assignment and routing algorithm were proposed. The proposed heuristic improves the aggregate throughput of WMNs and balance loads among gateways. For the channel assignment algorithm, load balancing is the first criterion assessed. The routing algorithm used both shortest path routing and randomized routing.

In [73], in order to solve a joint channel assignment and routing problem, a traffic flow based channel assignment was proposed to maximize the bandwidth allocated to each traffic aggregation point, subject to the fairness constraint. Unlike the heuristic approach in [73, 74] took into account the interference constraints at each mesh router in the formulation of the joint channel assignment and routing. As a result, the proposed algorithm was able to increase overall throughput.

The authors in [73, 74, 75] do not consider the use of scheduling in the event of multiple links being assigned to the same channel. In their algorithms, the mesh routers may need to buffer data packets, introducing extra hardware requirements for mesh routers. Moreover, these algorithms do not consider some of the physical characteristics inherent in the IEEE 802.11 standards, such as an adaptive transmission rate and the existence of multiple neighboring sets for a multi-radio mesh router due to the different transmission ranges of the radios.

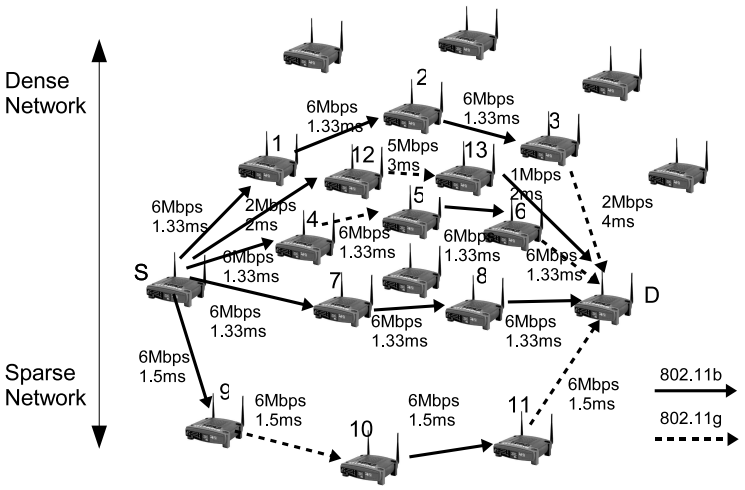


Figure 6.2: 5 candidate routes from Source S to Destination D

6.2 Cost-Aware Route Selection

6.2.1 Motivation

Prior studies [74, 77] have pointed out the shortcomings of the shortest-path routing approach in WMNs. As a result, most of the proposed route selection schemes for WMNs such as [76, 77] are instead based on the quality of links in a route. While these schemes favor routes with higher throughput, they do not take into account the cost of a route. As a result, in cases where multiple active connections are present, these schemes do not scale up well and tend to produce lower overall throughput in multi-radio WMNs.

Figure 6.2 shows 5 candidate routes between source S and destination D. The values of various metrics for these candidate routes, including the shortest-path, SETT, and BG-ETT, are presented in Table 6.1. As can be seen, the shortest-path will select path 2 or 4, since these two routes consist of only 3 hops; SETT will favor path 4 because it has the

Table 6.1: Performance metrics in Figure 6.2

path ID	route	hop	throughput	SETT	BG-ETT
1	s-1-2-3-d	4	2Mbps	8ms	4ms
2	s-12-13-d	3	1Mbps	7.0ms	4.0ms
3	s-4-5-6-d	4	3Mbps	5.32ms	2.66ms
4	s-7-8-d	3	2Mbps	4ms	4ms
5	s-9-10-11-d	4	3Mbps	6.0ms	3.0ms

lowest value of SETT among all routes (In general, SETT tends to favor shorter paths.); and BG-ETT will favor path 3 because of its radio diversity. However, none of these metrics considers the channel diversity, as the impact of interference has not been treated as one of the primary factors for route selection. Another drawback of these metrics is that their route selections are based solely on the individual traffic flow instead of multiple simultaneous flows. As a result, none of these metrics will be in favor of traffic aggregation, which will lead to better utilization of wireless resources (i.e., channels).

Given a set of candidate routes, the problem of route selection in WMNs can be considered as a resource allocation problem, where the limited resource is the wireless medium. When a route is an active, it prevents the mesh routers close to it from accessing the channels used by the active route due to the impact of co-channel interference. This limits the available routes for nearby mesh routers for other connections. In this chapter, a new route selection scheme, namely Cost-Aware Route Selection (CARS) will be proposed. In this scheme, the interference cost of a route is measured quantitatively. By choosing the route with the highest bandwidth and cost ratio, the overall network throughput can be improved. In the following subsections, this approach will be explained in detail.

Table 6.2: Physical Characteristics

	802.11b	802.11g	802.11a
Maximum rate	11Mbps	54Mbps	54Mbps
Transmission range (outdoor)	300 feet	250 feet	175 feet
Transmission range (indoor)	100 - 150 feet	100-150 feet	100-150 feet
Non-overlapping ch	1, 6, 11	1, 6, 11	1 - 12
Spectrum	2.4GHz	2.4GHz	5GHz

6.2.2 Problem Formulation

The assumptions below were made for the WMN in which the route selection scheme is expected to operate. Note that these assumptions do not conflict with any of the IEEE 802.11 specifications, and the frame format in the specifications is never changed.

- All mesh routers in WMNs are stationary.
- Assume that each mesh router has a set of 802.11 radio interfaces. The type of a radio interface can be either 802.11a [18], 802.11b [17], or 802.11g [19].
- A radio interface is always in one of four MAC states: SENDING, RECEIVING, IDLE and IDLE with TIMER. The transitions between these states are illustrated in Figure 6.3. The state IDLE with TIMER means that the radio is unused, but some neighboring mesh routers are using the same type of radio.

- Each type of radio has a number of available channels. If a nearby mesh router is using a channel for communications, the state of the channel is set to be IN-USE. Otherwise, the state of the channel is set to be UNUSED.

- Assume that the primary cause of packet loss is co-channel interference. The other factors that may affect the packet transmissions, such as multi-path fading [69], are assumed to be fixed by incorporating simple error recovery techniques (e.g., CRC [79]).

- To simplify the hardware design and lower the cost, assume a mesh router does not have a large data buffer. Packets received by an intermediate mesh router are always quickly forwarded to the next hop.

- The communications between mesh clients and their associated mesh router are assumed to be handled separately by a different set of wireless radio interfaces. In other words, in this research a route consists of only mesh routers.

In WMNs, since each mesh router can be equipped with multiple radio interfaces, the traditional graph denotation is not sufficient to describe the network topology of a WMN. Before formulating the problem, consider the nomenclature used in this chapter.

A mesh-graph, $G_M = (V_R, E_R)$, is composed of a set of node vectors V_R and a set of link vectors E_R. A node vector is defined as $v =< n, r >$, where n is a mesh router and r is one of n's radio interfaces. A type function $type(v)$ is defined to take a node vector $v =< n, r >$ as its argument and return the type of radio r (i.e., 802.11a, b, or g) of mesh router n. A link vector $< v_1, v_2 >$, where $v_1 =< n_i, r_p >$ and $v_2 =< n_j, r_q >$, represents

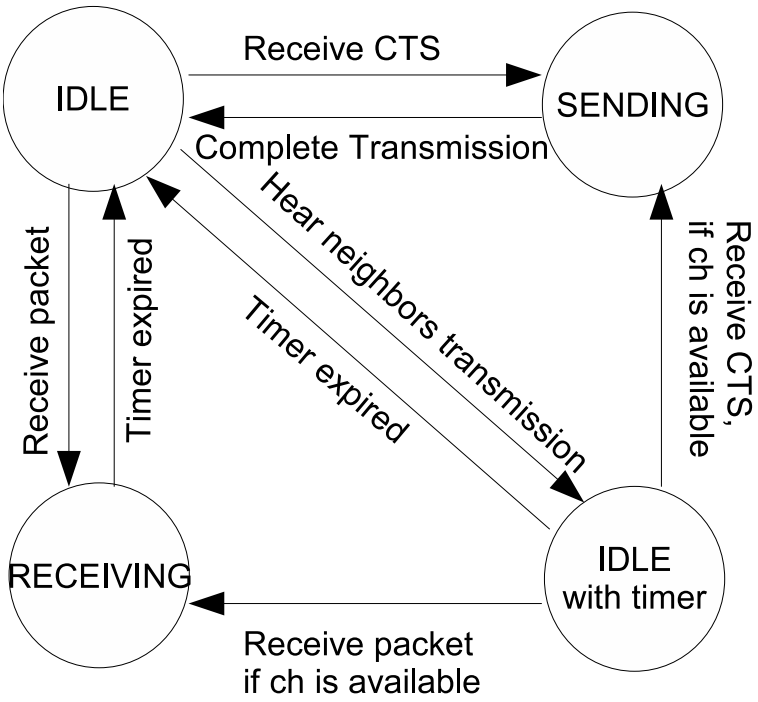


Figure 6.3: The State Transition Diagram for a radio interface

a mesh link between sender mesh router n_i using radio interface r_p to communicate with receiver mesh router n_j using radio interface r_q. Note that for a given mesh link $< v_1, v_2 >$, the constraint $type(v_1) = type(v_2)$ must be satisfied.

The open neighbor set of a node vector $N(v)$ ($v =< n, r >$) is defined as a set of mesh routers within transmission range of the radio transmission r of mesh router n, excluding n itself.

As mentioned earlier, the transmission rate of an 802.11 wireless link may step down automatically when the signal strength is weakened. Since the signal strength is closely related to the distance between sender and receiver, the available bandwidth of a mesh link is formulated as follows. Given a link $< v_1, v_2 >$ where $v_1 =< n_i, r_p >$ and $v_2 =< n_j, r_q >$, the available bandwidth of link $< v_1, v_2 >$, denoted as $b(< v_1, v_2 >)$, is defined by Equation 6.1. In Equation 6.1, the available bandwidth of a link is inversely proportional to the physical distance between two ends of the link.

$$b(< v_i, v_j >) = Maximum\ rate \cdot (1 - \frac{dist(n_i, n_j)}{R}) \quad (6.1)$$
$$(\text{refer to Table 6.2 for maximum rate \& transmission range, R})$$

In WMNs, a path χ connects the source node vector v_s and the destination node vector v_d and is composed of a set of ordered link vectors as follows.

$$\chi = \{< v_s, v_1 >, < v'_1, v_2 >, \cdots, < v'_h, v_d >\}$$

In a path, any two adjacent links $< v'_{k-1}, v_k >$ and $< v'_k, v_{k+1} >$, where $v_k =< n_i, r_p >$ and $v'_k =< n_j, r_q >$, should satisfy the constraint $(n_i = n_j) \wedge (r_p \neq r_q)$.

The path bandwidth of a path χ, denoted as $B(\chi)$, is defined as a function of the available bandwidths of the links in the path. Depending on the type of traffic along a path, the function may be defined differently. (Note that the terms *path bandwidth* and *path throughput* are identical and are used interchangeably in this chapter.) Two types of traffic are considered in this chapter: Burst Traffic (BT) and Constant Bit Rate (CBR) traffic. For the burst traffic, path bandwidth function is defined as the minimum available bandwidth of links in the path as, shown in Equation 6.2. If a path is assigned more bandwidth than the available bandwidth of any link in the path, an intermediate mesh router will have to buffer the data packets and this violates the no data buffer assumption. For instance, in Figure 6.2, path 2 has 3 links with 2Mbps, 5Mbps, and 1Mbps available bandwidth. Consequently, the path bandwidth of path 2 is 1Mbps.

$$\begin{aligned} B(\chi) &= \min\{b(< v_s, v_1 >), b(< v'_1, v_2 >), \cdots, \\ &\quad b(< v'_{h-1}, v_h >), b(< v'_h, v_d >)\} \end{aligned} \tag{6.2}$$

For CBR traffic, $B(\chi)$ is a constant value b_c. Note that the available bandwidth of any of the links in the path has to be larger than b_c.

Let a set of all active connections be S, so the overall throughput B_{all} is defined as the sum of the path bandwidths of all the paths in S, as shown in Equation 6.3. The goal

of this research is to design a route selection scheme to maximize overall throughput B_{all} of a WMN, which is the number of bits the WMN can transport between all source and destination pairs simultaneously. The higher the overall throughput B_{all} allows a WMN to support more end-user flows.

$$B_{all} = \sum_{\forall \chi \in S} B(\chi) \tag{6.3}$$

6.2.3 Physical Layer Constraints

To help formulate the physical layer constraints, we first define the radio state function and the channel state function must be defined. The radio state function *rs*(v) takes a node vector $v = (n, r)$ as input parameter and returns the state (i.e., SENDING, RECEIVING, IDLE, and IDLE w/ TIMER) of the radio interface r of node n. The channel state function *cs*(v, c) takes a node vector $v = (n, r)$ and a channel c as its input parameters and returns the state of the channel c (i.e., IN-USE or UNUSED) for radio interface r of node n. Note that even if a radio is in IDLE or IDLE w/ TIMER state, a channel may still be the in IN-USE state if it is used by one of n's neighbors.

To establish a link $< v_1, v_2 >$, where $v_1 =< n_i, r_p >$ and $v_2 =< n_j, r_q >$, the physical layer constraints can be formulated as follows :

1. ***Before establishing link*** $< v_1, v_2 >$

   Resource Allocation:

   $(rs(v_1) = \text{IDLE} \vee rs(v_1) = \text{IDLE w/ TIMER}) \wedge$

$(rs(v_2) = \text{IDLE} \vee rs(v_2) = \text{IDLE w/ TIMER}) \wedge$

$\exists\, c_k\ cs(v_1, c_k) = cs(v_2, c_k) = \text{UNUSED}$

2. ***After link*** $< v_1, v_2 >$ ***is established using channel*** c_k

   Resource Allocation:

   $rs(v_1) = \text{SENDING} \wedge rs(v_2) = \text{RECEIVING}$

   Interference Avoidance:

   $\forall\, n \in N(v_1) \cup N(v_2)\ cs(< n, r >, c_k) = \text{IN-USE} \wedge$

   $\forall\, n \in N(v_1) \cup N(v_2) \setminus \{n_1, n_2\}$

   $\forall\, r$ if $type(n, r) = type(n_i, r_p) \Rightarrow$

   $rs(< n, r >) = \text{IDLE w/ TIMER}$

To successfully establish a link $< v_i, v_j >$, the components of the link vector and the neighboring routers should satisfy the above constraints. Some of the constraints need to be enforced by the DCF function (e.g., exchange RTS and CTS so the radio interfaces of neighbors will be in the IDLE w/ TIMER state) defined in the 802.11 specification [16].

6.2.4 Cost and Throughput Metrics

The purpose of WMN research is to facilitate rapid Internet access for a large number of mesh clients. Hence, network throughput should be the primary performance measurement. Since the number of radios and channels in WMNs is limited, if the impact of interference can be reduced when routing a traffic flow, the overall throughput can naturally be increased. In this subsection, two metrics used in our CARS scheme to evaluate a path are introduced, the cost metric that measures the degree of interference of a path, and the bandwidth metric that measures the throughput of a path.

To measure the degree of interference of a path, compute the number of mesh routers that will experience interference along the path if the path is chosen for a connection and becomes active. If all candidate routes provide the same amount of bandwidth between source and destination, by selecting the path which creates the least interference, more network resources (e.g. radios and channels) can be utilized by other traffic flows.

Given an active link $< v_1, v_2 >$ where $v_1 =< n_i, r_p >$ and $v_2 =< n_j, r_q >$, the mesh routers in $N(n_i, r_p)$ cannot use the channel currently occupied by radio r_p of node n_i for communications (see interference avoidance physical layer constraint in Subsection 6.2.3). Hence, the cost of using the link $< v_1, v_2 >$ can be defined as $|N(v_1)|$. For a given path $\chi = \{e_0, e_1, \cdots, e_k\}$ where $e_i =< v_i', v_{i+1} >$, the cost of a path $C(\chi)$ is defined as follows:

$$C(\chi) = \sum_{i=0}^{k} |N(v_i')| \tag{6.4}$$

The throughput of a path, on the other hand, is measured by the path bandwidth, $B(\chi)$, which has been defined in Subsection 6.2.2. In general, we prefer a path with lower cost and higher path bandwidth is preferable.

When mesh routers are distributed uniformly, a shorter path contains a smaller number of hops and thus is likely to suffer less interference from neighbors. In addition, if a specific region has too many active communications, a path traversing that region is likely to result in a lower available path bandwidth. By choosing a path with a higher path bandwidth, a path that goes through a lighter traffic area can implicitly gain priority and load balancing can be achieved.

6.2.5 Traffic Aggregation

In addition to path metrics, route selection also takes into account traffic aggregation. If a link is simultaneously used by multiple active connections, we say that traffic is aggregated on that link. Suppose that a link $< v_1, v_2 >$ is already a portion of an active connection. If the same link is reused by another connection, This will not create additional interference. In other words, the cost function of a path should take traffic aggregation into consideration. For a given path $\chi = \{e_0, e_1, \cdots, e_k\}$ where $e_i =< v_i', v_{i+1} >$, if a subset of links in the path S have already been used by other active connections, the cost function should be modified as in follows:

$$C(\chi) = \sum_{e_i \in \chi \setminus S} |N(v_i')| \tag{6.5}$$

Additionally, the definition of the available bandwidth of a link needs to be modified so that the remaining bandwidth of a link can be utilized by aggregated traffic. Given a link $< v_1, v_2 >$, let S be a set of active connections that includes the link, so the available bandwidth of link $< v_1, v_2 >$, where $v_1 =< n_i, r_p >$ and $v_2 =< n_j, r_q >$, is defined as follows:

$$b(< v_1, v_2 >) = Maximum\ Rate \cdot (1 - \frac{dist(n_i, n_j)}{R}) - \sum_{\forall\ \chi \in S} B(\chi) \tag{6.6}$$

($Maximum\ rate$ and transmission radius R are shown in Table 6.2)

For instance, suppose that in Figure 6.2 the links in path 2 have already been used by an active connection and the bandwidth of that path is 1Mbps. The available bandwidth of the first, second and third links of path 2 will then be 1Mbps, 4Mbps, and 0Mbps, respectively.

6.2.6 Proposed Cost-Aware Route Selection Scheme

In this subsection, the proposed Cost-Aware Route Selection (CARS) scheme for WMNs is presented. The new scheme consists of two steps: radio selection and path selection. For a given source-destination pair and the sequence of intermediate mesh routers between them, the first step is to select the radio and channel to be used for the adjacent mesh routers in the sequence. (Note that according to our path definition, even with the same sequence of intermediate mesh routers, if the radio interface used by any intermediate mesh router is changed the path is considered to be different.) After the radio and channel

used for have been intermediate mesh router are identified, a new path metric called CARS is then used to identify the path with the best bandwidth-cost ratio for communications.

Given two adjacent mesh routers n_i and n_j in a sequence within close proximity, up to three sets of radio and channel will be returned as candidates for path consideration. First, the radio n_i with the smallest transmission range (i.e., the smallest number of that neighbors interfere) and one of its unused channels is returned. If no channel of that radio channel is available or n_j does not have an available radio channel with the matched type, the radio n_i with the next smallest transmission range and one of its available channels is returned. This process continues until a set of radio and channel is found. Second, the radio n_i with the highest available bandwidth and one of its unused channels is returned. Similarly, if no channel of that radio is available or n_j does not have an available radio channel with the matched type, the radio of n_i with the next highest available bandwidth and one of its available channel is returned. This process continues until a set of radio and channel is found. Last, these choices are examined to determine if there is an active link from n_i to n_j. If there is, the radio and channel used by the active link with the most remaining available bandwidth will be returned.

After the radio selection step, each sequence of mesh routers between source and destination will produce a number of candidate routes. Given a pair of source and destination, the candidate routes (i.e., the sequence of intermediate mesh routers) are found by doing breadth-first search starting from the shortest path until the number of candidate routes reaches 10000. In Subsection 6.2.4, two metrics that measure the cost and bandwidth of a

path have been introduced. In Equation 6.7, these two metrics are combined into one single Cost-Aware Route Selection (CARS) metric for path evaluation:

$$CARS(\chi) = \frac{(B(\chi))^{\beta}}{(C(\chi))^{\alpha}} \tag{6.7}$$

In Equation 6.7, β is assumed to be $1 - \alpha$ and $0 \leq \alpha, \beta \leq 1$. The greater the value of α, the more weight is put on cost for path selection. On the other hand, the greater the value of β, the more weight is put on path bandwidth for path selection. When $\alpha = \beta$, the CARS metric represents the amount of earned bandwidth for a unit of interference cost. By comparing the CARS metrics for the candidate routes, it is possible to identify the most efficient path that produces the most bandwidth per unit of interference. Hence, Equation 6.7 captures our design goals.

For instance, in Figure 6.2, if α is assigned a larger value (i.e., cost is heavily weighted), CARS will tend to favor path 5 as the sparse network area path 5 traverses has fewer neighbors to cause interference. On the other hand, if β is assigned a larger value (i.e., more weight is given to path bandwidth), CARS will tend to favor path 3. This is because, according to Equation 6.1, the available link bandwidth is inversely proportional to the distance between sender and receiver, so the dense network area that path 3 traverses, will tend to have a higher link bandwidth.

Note that for CBR traffic, path bandwidth is a fixed value b_c. Thus, Equation 6.7 can be simplified as $CARS(\chi) = \frac{1}{C(\chi)}$. Consequently, the CARS metric will give priority to the path with the lower interference cost.

6.3 Simulation Result and Analysis

This section presents the simulation results in order to evaluate the performance of the proposed CARS scheme. For the purpose of comparison, the other route selection schemes, including the shortest path and WCETT with different values of α and β, are implemented along with CARS by C++ on different hardware and environment configurations.

6.3.1 Simulation Setting and Parameter Consideration

The simulations are conducted on a $400m$ by $400m$ two dimensional square. Mesh routers are randomly placed within this square region. Each mesh router in our simulation has a small number of radio interfaces. Each interface has a number of available channels. The channels from different types of radio can be either shared or exclusive. Two channels from different types of radio with the same ID are said to be shared if both radios utilize the same spectrum i.e., only one channel can be used at a time. Two channels from different types of radio with the same ID are said to be exclusive if radios are using different spectra i.e., both channels can be used simultaneously. The transmission rate of a link is determined by Equation 6.6 based on the type of radio and the physical distance between the two ends of the link. The transmission range of a radio is set according to the type of the radio and the location mesh routers (i.e., indoors or outdoors). The values used for the computation of the transmission rate and the transmission range can be found in Table 6.2.

Two types of traffic flow, BT and CBR, are generated in the simulation. For a BT traffic flow, the rate is computed by Equation 6.2. For a CBR traffic flow, the rate is set

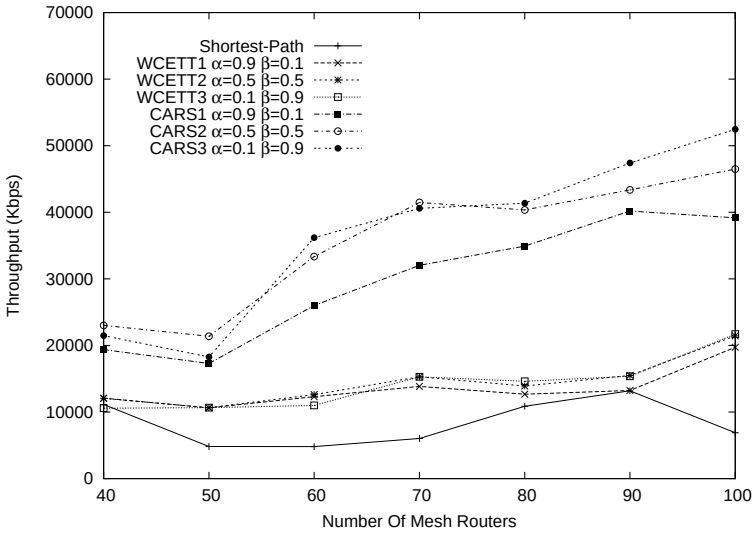


Figure 6.4: Throughput w/ BT, 3 NICs & 3 exclusive channels, P2P, indoors

to be 1024Kbps. Additionally, two different network flow patterns, Peer-to-Peer (P2P) and gateway-oriented, are simulated. In a P2P connection, source and destination are mesh routers randomly selected in WMNs. In a gateway-oriented connection, one of the few sinks are used as one end of the traffic flow. Since the primary cause of packet loss is co-channel interference, the ETT of a link in these candidate routes can simply be calculated as the inverse of the available bandwidth of the link.

6.3.2 Throughput of Traffic Patterns

In this subsection, the simulation results of different route selection schemes on BT and CBR traffic are presented.

Figure 6.4 shows the overall network throughput B_{all} for the different route selection schemes for the burst traffic scenario based on the number of mesh routers in the simulated

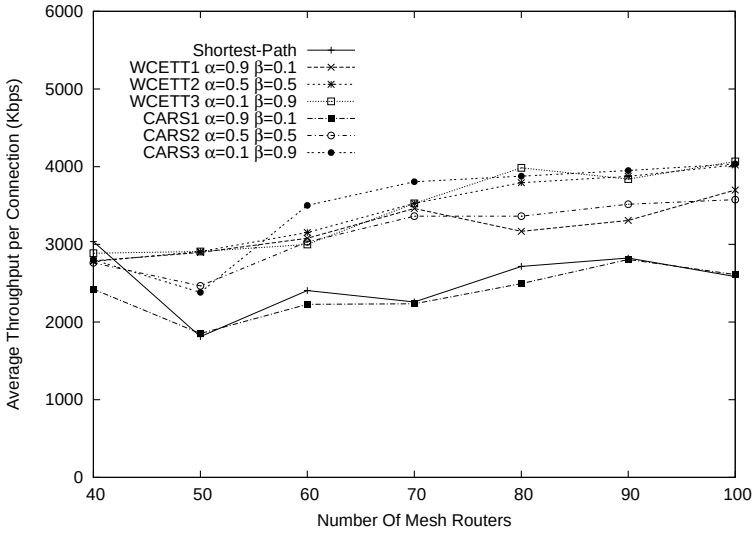


Figure 6.5: Average Throughput per Connection w/ BT, 3 NICs & 3 exclusive channels, P2P, indoors

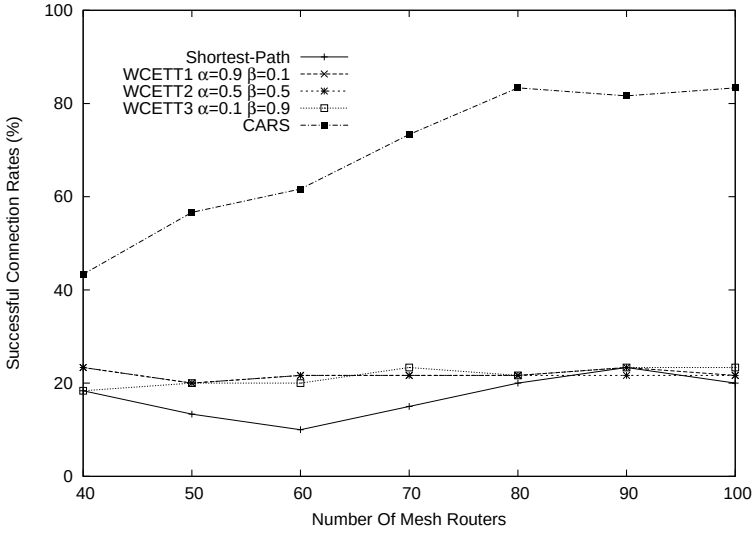


Figure 6.6: Successful Connection Rates w/ CBR, 3 NICs & 3 exclusive channels, P2P, indoors

region. In the simulations, a mesh router is set to have 3 Network Interface Card (NIC) radios, and each radio has 3 exclusive channels. P2P connections are generated until the network is saturated. The location of the simulated WMN is assumed to be indoors.

As illustrated in Figure 6.4, no matter what values of α and β are used in CARS and WCETT, CARS can always produce more than twice as much of the overall network throughput as WCETT's and the shortest path's. This is a big improvement over the past route selection schemes. While all three different CARS versions have similar performance, the that with $\alpha = 0.1$ and $\beta = 0.9$ is slightly better than the other two. This suggests that the path bandwidth metric is slightly more important than the path cost metric. Additionally, the overall network throughput of the three different CARS versions is a lot more responsive to an increase of the number of mesh routers in any of the simulated region than the other route selection schemes. This suggests that the new CARS scheme is more scalable in terms of overall throughput. This feature is especially important for WMNs. In addition, Figure 6.5 shows the average path throughput of different route selection schemes with respect to the number of mesh routers in the simulated region under the same simulation settings. As illustrated in Figure 6.5, the schemes that produce the highest average path throughput are CARS with $\alpha = 0.1$ and $\beta = 0.9$, WCETT with $\alpha = 0.1$ and $\beta = 0.9$, and WCETT with $\alpha = 0.5$ and $\beta = 0.5$. The average path throughput of CARS with $\alpha = 0.5$ and $\beta = 0.5$ and WCETT with $\alpha = 0.9$ and $\beta = 0.1$ is just slightly lower than the highest group. It is interesting to observe from Figure 6.5 that CARS with $\alpha = 0.1$ and $\beta = 0.9$ achieves a significant improvement in the overall network throughput without sacrificing individual path throughput.

Figure 6.6 shows the successful connection rates for different route selection schemes in the case of the CBR traffic scenario with respect to the number of mesh routers. In these simulations, each mesh router is set to have 3 NIC radios, and each radio has 3 exclusive channels. 20 P2P-type connections are attempted. The location of the simulated MWN is also assumed to be indoors. Note that for the CBR scenario, the CARS metric is essentially reduced to the path cost function.

As illustrated in Figure 6.6, no matter what values of α and β are used in WCETT, CARS can successfully establish more than twice as many connections as either WCETT as the shortest path. This suggests that the cost metric still plays a crucial role in route selection. Additionally, the results of this simulation suggest that CARS allows more mesh clients to be supported than either WCETT or the shortest path. This feature is also very important for WMNs.

6.3.3 Successful Connection Rates of Shared and Exclusive Channels

In this subsection, the simulation results of different route selection schemes for shared and exclusive channels are presented. Here, each mesh router is set to have 2 NIC radios, and each radio has 3 channels. The network size is fixed at 80 routers with the assumption that 3 of them work as gateways to connect to the Internet. Gateway-oriented CBR connections are used and the WMN is assumed to be indoors.

Figure 6.7 shows the successful connection rates of different route selection schemes with respect to the number of generated connections in the simulated region in the case of

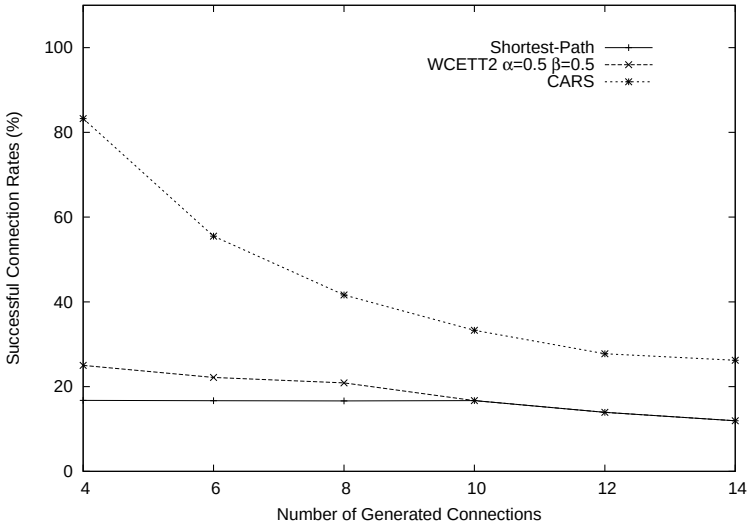


Figure 6.7: Successful Connection Rates w/ 80 mesh routers (3 work as gateway), CBR, 2 NICs & 3 shared channels, indoors

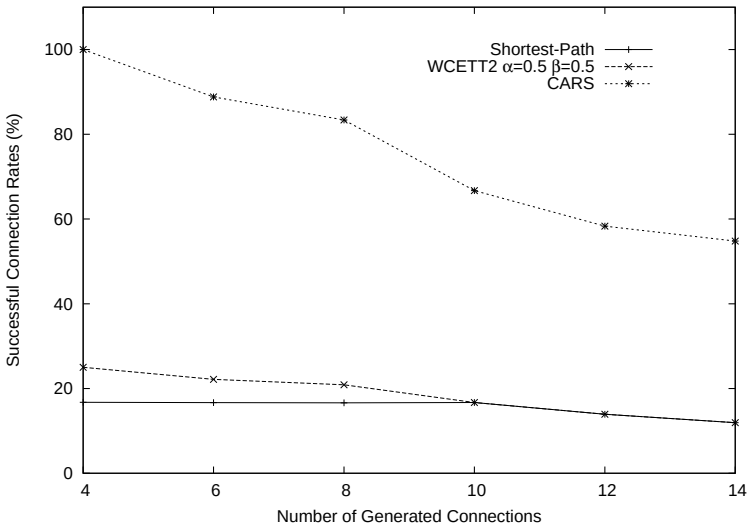


Figure 6.8: Successful Connection Rates w/ 80 mesh routers (3 work as gateway), CBR, 2 NICs & 3 exclusive channels, outdoors

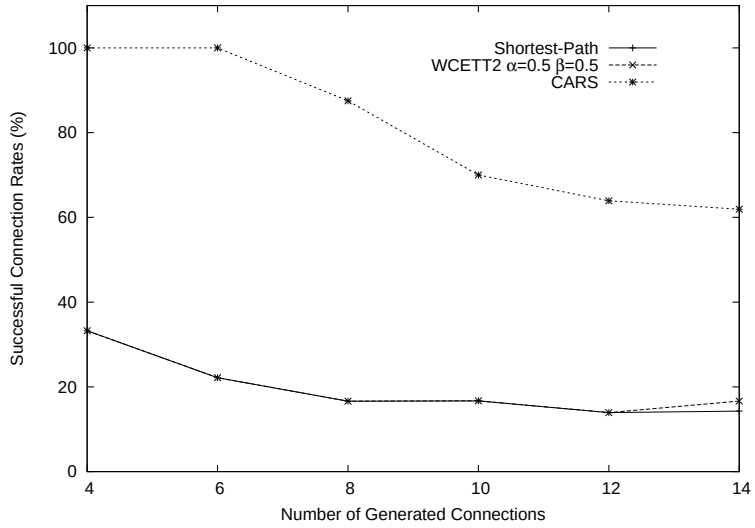


Figure 6.9: Successful Connection Rates w/ 80 mesh routers (3 work as gateway), CBR, 2 NICs & 3 exclusive channels, indoors

the shared channels. As illustrated in Figure 6.7, CARS has more than twice of the successful connection rate of either WCETT's or the shortest path. This suggests that CARS also performs well for the gateway-oriented connections. However, when the number of generated connections increases, the successful connection rates for CARS decreases. This is because the wireless resource (i.e., radios and channels) close to the gateways is quickly exhausted.

Figure 6.9 shows the successful connection rates of different route selection schemes with respect to the number of generated connections in the simulated region in case of the exclusive channels. In Figure 6.9, the successful connection rates for CARS are approximately three times the rates for WCETT and the shortest path. This is because the assumption of exclusive channels actually means less possibility of interference. In other words, more resources are available in the case of the exclusive channels. When Figure 6.7

and Figure 6.9 are compared together, it can be seen that the successful connection rates of the other route selection schemes are not sensitive to the extra resources than become available when the channel type switches from shared to exclusive. This suggests that CARS can better utilize the extra resources in the network.

For the purpose of comparison, Figure 6.8 shows the successful connection rates of different route selection schemes with respect to the number of generated connections in the simulated region for the case of exclusive channels under the same configuration, with the only difference being that the network is located outdoors. As can be seen, the rates in Figure 6.8 are slightly lower than the rates in Figure 6.9. Because in the outdoor case, the transmission range is increased, as indicated in Table 6.2. At the same time, more interference will be created when a path is established.

6.3.4 Network Throughput of Different Number of Radios and Channels

In this subsection, the simulation results of different route selection schemes on different number of NIC radios and channels are presented.

Figure 6.10 shows the overall network throughput B_{all} of different route selection schemes with respect to the number of NIC radios at each mesh router. In this set of simulations, the hardware and environment settings are similar to those used to for Figure 6.4 except that each NIC radio is set to have 2 channels and the network size is set to be 80.

As illustrated in Figure 6.10, no matter how many NIC radios are available at a mesh router, the overall network throughput of any CARS is always more than 1.7 times that of either WCETT or the shortest path. In addition, as the number of radios at each mesh router

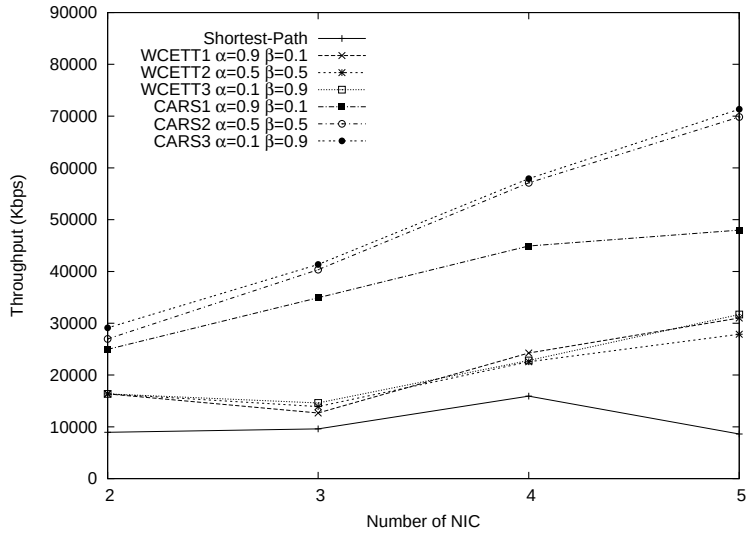


Figure 6.10: Throughput w/ 80 mesh routers, BT, 2 exclusive channels, P2P, indoors

increases, the overall network throughput of both CARS with $\alpha = 0.1$ and $\beta = 0.9$ and CARS with $\alpha = 0.5$ and $\beta = 0.5$ increases faster than the other route selection schemes. This again suggests that with proper selection of the values of α and β, CARS is more scalable in terms of the number of available NIC radios at each mesh router.

Figure 6.11 shows the overall network throughput B_{all} of the different route selection schemes with respect to the number of available channels for each radio. In this set of simulations, the hardware and environment settings are the same as those used for Figure 6.10.

As illustrated in Figure 6.11, no matter how many channels are available at each radio, the overall network throughput of CARS is always more than twice that of either WCETT or the shortest path. Again in Figure 6.11, CARS with $\alpha = 0.1$ and $\beta = 0.9$ and CARS with $\alpha = 0.5$ and $\beta = 0.5$ outperform the other route selection schemes.

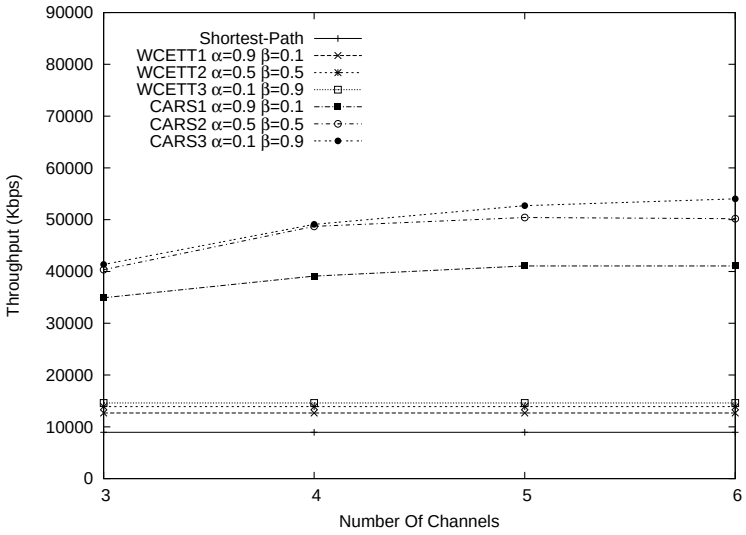


Figure 6.11: Throughput w/ 80 mesh routers, BT, 3 NICs, P2P, indoors

6.4 Summary

In this chapter, a novel route selection scheme, namely Cost-Aware Route Selection, is proposed for WMNs to improve the overall throughput. The scheme incorporates a path metric which captures the bandwidth and cost ratio and the introduces idea of traffic aggregation. Simulation results show that the new CARS scheme improves the overall throughput by up to 165% in the case of the burst traffic and boosts the number of connections by up to 300% in the case of constant bit rate traffic and is also more scalable in terms of the size of the network compared to both WCETT and the shortest path route selection schemes.

Although for a given set of candidate routes this scheme is able to identify the best choice to improve overall network throughput, it has yet to completely solve the routing issue as no protocol is provided to locate those candidate routes. In addition, the new route

selection scheme is centralized in the sense that the source node must collect and process all the necessary information. While the nature of WMNs (i.e., mesh routers are fixed and connected to external power supplies) allows this assumption to hold, future research on a distributed routing protocol that runs only on the basis of localized information would definitely be of interest.

CHAPTER 7

CONCLUSION

This chapter summarizes the research, highlights its contributions to the field of mobile ad hoc networks, and discusses potential future work.

7.1 Summary and Contribution

As a special type of wireless networks, wireless ad hoc networks have emerged as a potential candidate for wireless broadband communication. In addition, advances in ad hoc and sensor networks encourage the development of new applications such as wireless mesh networks (WMNs), and underwater sensor networks. In such multi-hop networks, the route selection scheme has a significant impact on the overall network performance. To obtain better and more stable network performance, research on the practical considerations that limit the design of network protocols is strongly demanded.

In this dissertation, two distinct approaches to routing in ad hoc networks are suggested by considering practical issues: DTGR and CARS. In DTGR, DTGR-SF and DTGR-WF utilize both neighbor tables via beacon sampling to improve network performance and reachability values to make forwarding decisions. Since wireless signals are on unstable transmission medium, temporary disruption in wireless communication is common due to interference from obstacles or nodal movement. The simulation results show that DTGR can help to improve the network performance. CARS is a specialized route selection scheme for multi-radio and multi-channel wireless mesh networks. The provision of

more resources naturally leads to better performance, but also increases the complexity of both network formation and route selection. First, a network formed with multi-radio and multi-channel is mathematically formulated with physical constraints. A cost metric, bandwidth metric, and combined metric are then defined. Hence, CARS selects the route that will cause the lowest interference and have the higher path bandwidth by utilizing the multi radio and multi channel. The simulation results show that the network throughput for burst traffic is significantly improved and more connections can also be established to gateways (i.e., Internet access).

Beyond this laboratory research, more commercial applications are emerged in wireless ad hoc networks due to its properties of self-organizing, self-configuring, and self-healing. The DTGR and CARS proposed in this dissertation can enhance those properties, which is one of the most important contribution of this dissertation.

7.2 Future Research Directions

As discussed in Chapter 6, wireless mesh networks have received increasing attention in recent years due to its ability to quickly provide broadband networking infrastructure for large business enterprizes and Internet access to residence in rural areas. However, many research issues in WMNs, such as virtual backbone construction, guaranteed QoS for multimedia streams, and optimal path selection [68], remain to be addressed. In addition, although CARS is one of the better heuristic schemes of the route selection in WMNs, it needs further refinement to become a fully distributed routing protocol. Thus, our future research directions include the followings:

- Virtual backbone construction - The connected dominating set (CDS) [3] is one of the well-known virtual backbones in ad hoc networks. We will design and implement a better CDS construction protocol tailored specifically for multi-radio and multi-channel WMNs.

- QoS Guarantee - Multimedia streaming in Internet is one of the most popular and dominant traffic patterns nowadays. The quality of service (QoS) guarantees are crucial in multimedia streaming. To provide certain QoS guarantees, the quality of both link and path should be considered in WMNs when routing the traffic.

- Distributed routing protocols - The current version of CARS is a centralized route selection scheme. It requires a node to obtain the current traffic snapshot of the whole WMN to select a route. This assumption is considered impractical. To address this issue, we are investigating to refine CARS so that it is based strictly on the localized traffic information.

BIBLIOGRAPHY

[1] J. M. Kahn, R. H. Katz and K. S. J. Pister, "Next Century Challenges: Mobile Networking for Smart Dust," *Proc. ACM/IEEE international conference on Mobile computing and networking (MobiCom)*, pp. 271-278, Aug. 1999.

[2] J. M. Kahn, R. H. Katz and K. S. J. Pister, "Emerging challenges: Mobile Networking for Smart Dust," *Journal of Communications And Networks*, Vol. 2, No. 3, pp. 188-196, Sep. 2000.

[3] D. Zhou, M. Sun, and T. Lai, "A Timer-based Protocol for Connected Dominating Set Construction in IEEE 802.11 Multihop Mobile Ad Hoc Networks," *IEEE Symposium on Applications and the Internet (SAINT)*, pp.2-8, 2005.

[4] M. Sun, S. Wang, C.K. Chang, T.H. Lai, S. Hiroyuki, and H. Okada, "Interference-Aware MAC Scheduling and SAR Policies for Bluetooth Scatternets," *Proc. IEEE GLOBECOM*, pp. 11-15, Nov. 2002.

[5] Srivastava, V. Motani, M., "Cross-layer design: a survey and the road ahead," *IEEE Communications Magazine*, Vol. 43, No. 12, pp. 112-119, Dec. 2005.

[6] S. Shakkottai, T.S. Rappaport, and P.C. Karlsson, "Cross-layer design for wireless networks," *IEEE Communications Magazine*, Vol. 41, No. 18, pp. 74-80, Oct. 2003.

[7] M. Kwon and S. Fahmy, "Topology aware overlay networks for group communication," *Proc. ACM NOSSDAV*, May 2002.

[8] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller. "Construction of an efficient overlay multicast infrastructure for real-time applications," *Proc. IEEE Conference on Computer Communcation (INFOCOM)*, Apr. 2003.

[9] G. Ramamurthy, and Q. Ren, "Analysis of the adaptive rate control for ABR service in ATM networks," *IEEE GLOBECOM*, pp.1083-1088, 1995.

[10] H. Kawahigashi, Y. Terashima, N. Miyauchi, T. Nakakawaji, "Designing Fault Tolerant Ad Hoc Networks," *Military Communications Conference*, pp. 1 - 8, Oct. 2005.

[11] RF and Microwave Fiber-Optic Design Guide, "Application Note", Apr. 2001.

[12] R. KATZ, “Adaptation and Mobility in Wireless Information Systems.” *IEEE Personal Communications Magazine*, Vol. 1, pp. 6–17, 1995.

[13] http://www.ieee802.org/11/.

[14] http://standards.ieee.org/getieee802/, IEEE 802 working group.

[15] IEEE LAN MAN Standards Committee, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE Std. 802.11-1997, The Institute of Electrical and Electronics Engineers, 1997.

[16] http://standards.ieee.org, IEEE standards organization.

[17] Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications, IEEE Std 802.11b-1999/Cor 1-2001 (R2003).

[18] IEEE LAN MAN Standards Committee, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE Std. 802.11a-1999, The Institute of Electrical and Electronics Engineers, Dec. 1999.

[19] IEEE 802.11g Standard, standards.ieee.org/getieee802/download/802.11g-2003.pdf.

[20] F. Andre, J.-M. Bonnin, B. Deniaud, K. Guillouard, N. Montavont, T. Noel, L. Suciu, “Optimized Support of Multiple Wireless Interfaces within an IPv6 End-Terminal,” *Smart Object Conference (SOC)*, May 2003.

[21] http://www.ieee802.org/15/, IEEE 802.15 Working Group for WPAN.

[22] C. E. Perkins and P. Bhagwat, “Highly Dynamic Destination- Sequenced Distance-Vector Routing (DSDV) for Mobile Computers,” *Proc. ACM Special Interest Group on Data Communications Conference (SIGCOMM)*, pp. 234-244, 1994.

[23] C. Perkins and P. Bhagwat, “Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers,” *Proc. ACM Special Interest Group on Data Communications Conference (SIGCOMM)*, pp. 234-244, 1994.

[24] C.E. Perkins and E.M. Royer, “Ad-hoc on-demand distance vector routing,” *Proc. WMCSA*, pp. 90-100, Feb. 1999.

[25] S. MURTHY, and J.J. GARCIA-LUNA-AVECES, “A Routing Protocol for Packet Radio Networks,” *Proc. ACM International Conference on Mobile Computing and Networking*, pp. 86-95, Nov. 1995.

[26] C.C. CHIANG, H.K. WU, W. LIU, and M. GERLA, "Routing in Clustered Multihop, Mobile Wireless Networks with Fading Channel," *IEEE Singapore International Conference on Networks (SICON)*, pp. 197-211, Apr. 1997.

[27] R. Bellman, "On a Routing Problem," *Quarterly of Applied Mathematics*, Vol. 16, No. 1, pp. 87-90, 1958.

[28] D. B. Johnson and D. A. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks," *Mobile Computing*, Vol. 353, 1996.

[29] S. Basagni, I. Chlamtac, V.R. Syrotiuk, "Dynamic source routing for ad hoc networks using the global positioning system," *Proc. IEEE Wireless Communications and Networking Conference (WCNC)*, Sep. 1999.

[30] C. E. Perkins, "Ad-hoc on-demand distance vector routing," *MILCOM panel on Ad Hoc Networks*, Nov. 1997.

[31] C. E. Perkins and E. M. Royer, "Ad Hoc On-Demand Distance Vector (AODV) Routing," *IETF, Internet-Draft*, Ver. 03, 1999.

[32] M Joa-Ng and I.-T. Lu, "A Peer-to-Peer zone-based two-level link state routing for mobile Ad Hoc Networks," *IEEE Journal on Selected Areas in Communications, Special Issue on Ad-Hoc Networks*, Vol. 17, No. 8, pp.1415-1425, Aug. 1999.

[33] V. Park and M. Corson, "Temporally-Ordered Routing Algorithm (TORA): Version 1 Functional Specification," *Internet-Draft, IETF*, Jul. 2001.

[34] Zygmunt J. Haas and Marc R. Pearlman, "The Performance of Query Control Schemes for the Zone Routing Protocol," *IEEE/ACM Transactions on Networking*, Aug 2001.

[35] Z. Haas and M. Pearlman, "The Zone Routing Protocol for Highly Reconfigurable Ad-Hoc Networks," *Proc. ACM SIGCOMM*, Aug. 1998.

[36] R. Sivakumar, P. Sinha, and V. Bharghavan, "CEDAR: A Core-Extraction Distributed Ad Hoc Routing Algorithm," *IEEE Journal on Selected Area in Communication*, Vol. 17, No. 8, Aug. 1999.

[37] P. Sinha, R. Sivakumar, and V. Bharghavan, "MCEDAR: Multicast Core-Extraction Distributed Ad hoc Routing," *Proc. of IEEE WCNC*, Sep. 1999.

[38] http://www.navcen.uscg.gov/pubs/gps/sigspec/gpsspsa.pdf, Global Positioning System Standard Positioning Service Specification, 2nd Edition, Jun. 1995.

[39] G. Pei, M. Gerla and T.-W. Chen, "Fisheye State Routing in Mobile Ad Hoc Networks," *Proc. ICDCS Workshops*, pp. D71-D78, Apr. 2000.

[40] A. Iwata, C.-C. Chiang, G. Pei, M. Gerla, and T.-W. Chen, "Scalable routing strategies for ad hoc wireless networks," *IEEE Journal on Selected Areas in Communications, Special Issue on Ad-Hoc Networks*, p1369-p1379, Aug. 1999.

[41] Y.-B. KO, V. N. H., "Location-Aided Routing in mobile Ad hoc networks," *Proc. ACM/IEEE Mobicom*, pp. 66-75, Oct. 1998.

[42] S. BASAGNI, I. CHLAMTAC, V. R. SYROTIUK, B. A. WOODWARD, "A Distance Routing Effect Algorithm for Mobility (DREAM)," *Proc. ACM/IEEE Mobicom*, pp. 76-84, Oct. 1998.

[43] T. Ozaki, J. Kim, and T. Suda, "Bandwidth Efficient Multicast Routing Protocol for Ad hoc Networks," *Proc. IEEE ICCCN*, pp. 10-17, Oct. 1999.

[44] Mingyan Liu, Rajesh R. Talpade, Anthony McAuley, and Ethendranath Bommaiah, "AMRoute: Adhoc Multicast Routing Protocol," *University of Maryland CSHCN Technical Report*, 1999-1, 1999.

[45] R. Ramanathan, "On the performance of ad hoc networks with beamforming antennas," *Proc. ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC)*, Oct. 2001.

[46] De, S., Qiao, C., and Wu, H. "Meshed multipath routing with selective forwarding: An efficient strategy in wireless sensor networks," *Elsevier Computer Communications Journal*. Vol. 26, No. 4, 2003.

[47] WHP 004 Cumulative effects of distributed interferers Spectrum Planning Group, BBC Research Development Department White Paper, 2001.

[48] Th. Fritsch, K. Tutschku und K. Leibnitz, "Field Strength Prediction by Ray-Tracing for Adaptive Base Station Positioning in Mobile Communication Networks," Aug. 1995.

[49] P.Viswanath, D.N.C.Tse, and R.Laroia, "Opportunistic Beamforming using Dumb Antennas," *IEEE Trans. Information Theory*, vol. 48, pp. 1277-1294, Jun. 2002.

[50] S. Singh and C. S. Raghavendra, "PAMAS: Power Aware Multi-Access protocol with Signalling for Ad Hoc Networks," *ACM Computer Communications Review*, 1999.

[51] A. Tanenbaum, Computer Networks, Third Edition, Prentice Hall, 1996.

[52] L. K. Rasmussen, T. J. Lim, and A.-L. Johansson, "A matrixalgebraic approach to successive interference cancellation in CDMA," *IEEE Trans. Commun.*, Vo. 48, No. 1, pp. 145-151, Jan. 2000.

[53] M. Zorzi and R. Rao, "Capture and retransmission control in mobile radio.," *IEEE Journal on Selected Areas in Communications*, Vol. 12, No. 8, pp. 1289-1298, 1994.

[54] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang, "MACAW: A medium access protocol for wireless LANs," *Proc. ACM SIGCOMM*, Aug. 1994.

[55] B. Karp and H. T. Kung, "Greedy Perimeter State Routing for Wireless Networks," *Proc. the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pp. 243-254, 2000.

[56] BAA04-13, Disruption Tolerant Networking (DTN), http://www.darpa.mil/ato/solicit/DTN/, 2004.

[57] W. Zhao, M. Ammar, and E. Zegura, "A Message Ferrying Approach for Data Delivery in Sparse Mobile Ad Hoc Networks," *Proc. ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pp. 187-198, 2004.

[58] B. Burns, O. Brock, and B. N. Levine, "MV Routing and Capacity Building in Disruption Tolerant Networks," *Proc. Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pp. 398-408, 2005.

[59] E. M. Royer and C.-K. Toh, "A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks," *IEEE Personal Communications*, pp. 46-55, 1999.

[60] K. Seada, M. Zuniga, B. Krishnamachari, and A. Helmy, "Energy Efficient Forwarding Strategies for Geographic Routing in Lossy Wireless Sensor Networks," *Proc. ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pp. 108-121, 2004.

[61] A. Ward, A. Jones, and A. Hopper, "A New Location Technique for the Active Office," *IEEE Personnel Communications*, vol. 4, No. 5, pp. 42-47, 1997.

[62] J. Li, J. Jannotti, D. DeCouto, D. Karger, and R. Morris, "A Scalable Location Service for Geographic Ad Hoc Routing," *Proc. of the 6th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pp. 120-130, 2000.

[63] Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification, IEEE Standard for Information Technology, 1999.

[64] L. F. Huang and T. H. Lai, "On the Scalability of IEEE802.11 Ad Hoc Networks," *Proc. ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pp. 173-182, 2002.

[65] Network Simulator, The UCB/LBNL/VINT Network Simulator NS(version 2), http://mash.cs.berkeley.edu/ns.

[66] The CMU MONARCH Group, Wireless and Mobility Extensions to ns-2, http://www.monarch.cs.cmu.edu/cmu-ns.html.

[67] J. Broch, D. Maltz, D. Johnson, Y. Hu, and J. Jetcheva, "A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols," *Proc. Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pp. 85-97, 1998.

[68] Ian F. Akyildiz, and Xudong Wang, "A Survey on Wireless Mesh Networks," *IEEE Radio Communications*, Sep. 2005.

[69] C.Siva Ram Murthy and B.S. Manoj, "Ad Hoc Wireless Networks :Architechtures and Protocols," Peason Education, ISBN 013147023X, 2004

[70] K. M. Alzoubi, P.-J. Wan, O. Frieder, "Message-Optimal Connected-Dominating-Set Construction for Routing in Mobile Ad Hoc Networks," *Proc. ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2002.

[71] F. Kuhn, R. Wattenhofer and A. Zollinger, "Ad-hoc networks beyond unit disk graphs," *Proc. joint workshop on Foundations of mobile computing citation*, 2003.

[72] K. Siwiak, "Advances in Ultra-Wide Band Technology," *Radio Solutions*, Nov. 2001.

[73] M. Alicherry, R. Bhatia, and L. Li, "Joint channel assignment and routing for throughput optimization in multi-radio wireless mesh networks," *Proc. Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, Aug. 2005.

[74] A. Raniwala, K. Gopalan, and T. Chiueh, "Centralized Channel Assignment and Routing Algorithms for Multi-Channel Wireless Mesh Networks," *ACM Mobile Computing and Communications Review (MC2R)*, Vol 8, No 2, Apr. 2004.

[75] A. Raniwala and T. Chiueh, "Architecture and Algorithms for an IEEE 802.11-Based Multi-Channel Wireless Mesh Network," *Proc. IEEE Conference on Computer Communcation (INFOCOM)*, Vol 3, pp. 2223- 2234, Mar. 2005.

[76] D.S.J. De Couto, D. Aguayo, J. Bicket, and R. Morris, "A High-Throughput Path Metric for Multi-Hop Wireless Routing," *Proc. ACM International Conference on Mobile Computing and Networking Review*, Sep. 2003.

[77] R. Draves, J. Padhye, and B. Zill, "Routing in multi-radio, multi-hop wireless mesh networks," *Proc. ACM International Conference on Mobile Computing and Networking (MobiCom)*, pp. 114-128, Sep. 2004.

[78] R. Draves, J. Padhye, and B. Zill, "Comparisons of Routing Metrics for Static Multi-Hop Wireless Networks," *Proc. ACM Annual Conf. Special Interest Group on Data Communication (SIGCOMM)*, pp. 133-144, Aug. 2004.

[79] D.V. Sarwate, "Computation of Cyclic Redundancy Check via Table Look-Up," *Communications of the ACM*, Vol.31, No.8, Aug. 1988.

[80] Infopeople Webcasts, http://www.infopeople.org/training/webcasts/03-01-05/Wireless_Webcast.ppt.

APPENDIX

APPENDIX A

PACKET FORWARDING PROCEDURES

Definitions

m: packet currently to be forwarded
$m.L_p$: location that m entered perimeter mode
$m.e_0$: first edge m traversed on the current face, used to avoid looping along the same face
i: m's forwarding node
j: next hop selected by i for forwarding m
N_i: set of nodes in i's neighbor table
P_i: set of neighbors in i's planar subgraph, which i obtains by applying RNG or GG locally

/* **when node** i $(i \neq D)$ **receives a packet** m **destined for node** D **in greedy mode** */

GPSR_Greedy.i Module

Step 1: /* greedy forwarding: forward m to j, the node closest to D among i and i's neighbors*/
if $(\exists j: (\forall k : k \in N_i \wedge k \neq j : dist(j,D) < dist(k,D) \wedge dist(j,D) < dist(i,D)))$
forward m to j;
else /* route m using perimeter routing (right hand rule face routing) */
Step 2: if $(j = Right\_Hand\_Face\_Routing(P_i).i)$
$m.mode, m.e_0, m.L_p := perimeter, e_{i,j}, L_i$;
forward m to j;
Step 3: else drop m;

/* **when node** i $(i \neq D)$ **receives a packet** m **destined for node** D **in perimeter mode** */

GPSR_Perimeter.i Module

/*if it is closer to the destination than the node who starts routing m in perimeter mode*/
/*recover m from perimeter mode to greedy mode */
Step1: if $(dist(i,D) < dist(m.L_p, D))$
set $m.mode$ to Greedy;
GPSR_Greedy.i;
else /*continue perimeter routing*/
Step2: if $((j = Right\_Hand\_Face\_Routing(P_i).i) \wedge (e(i,j) \neq m.e_0))$
forward m to j;
Step3: else drop m,
*Note: If i is the *source* node of m, i sets m in greedy mode and starts forwarding m using GPSR_Greedy.i Module.

Figure A.1: GPSR packet forwarding procedure

Definitions
m: packet currently to be forwarded
$m.L_p$: location that m entered perimeter mode
$m.e_0$: first edge m traversed on the current face, used to avoid looping along the same face
i: m's forwarding node
j: next hop selected by i for forwarding m
N_i: set of nodes in i's neighbor table
P_i: nodes in i's planar subgraph constructed from neighbors with $r = 1$
P'_i: nodes in i's planar subgraph constructed from neighbors with $r_{threshold} \leq r < 1$.

/* **when node** i ($i \neq D$) **receives a packet** m **destined for node** D **in greedy mode** */

DTGR_Greedy.i Module

/* GPSR_Greedy.i module in terms of our reachability value */
/* Same as Step 1 of GPSR_Greedy.i module */
Step 1: if ($\exists j$: ($\hat{r}_{i,j} = 1$) $\wedge$ ($\forall k : k \in N_i \wedge \hat{r}_{i,k} = 1 \wedge k \neq j : dist(j,D) < dist(k,D)$) $\wedge dist(j,D) < dist(i,D)$)))
forward m **to** j;

/* Forward m in greedy mode using $unstable$ nodes */
Step 2: else if ($\exists$ j: ($r_{threshold} \leq \hat{r}_{i,j} < 1$)
$\wedge$ ($\forall$ k: $k \in N_i \wedge r_{threshold} \leq \hat{r}_{i,k} < 1 \wedge k \neq j$: $dist(j,D) < dist(k,D)$)
$\wedge$ $dist(j,D) < dist(i,D)$)
forward m **to** j;

else /* route m using perimeter routing (right hand rule planar graph traversal) */
Step 3: Step 2 of GPSR_Greedy.i module;
/* select j from planar subgraph constructed by nodes with $r_{threshold} \leq r < 1$. */
Step 4: **else if** ($j = Right\_Hand\_Face\_Routing(P'_i).i$)
$m.mode$, $m.e_0$, $m.L_p := perimeter$, $e_{i,j}$, L_i;
forward m to j;
Step 5: **else** drop m;

/* **when node** i ($i \neq D$) **receives a packet** m **destined for node** D **in perimeter mode** */

DTGR_Perimeter.i Module

/* recover from perimeter mode into greedy mode */
Step 1: if ($dist(i,D) < dist(m.L_p, D)$)
set $m.mode$ to Greedy;
DTGR_Greedy.i;

else /* continue forwarding in perimeter mode */
Step 2: step 2 of GPSR_Perimeter.i.Module
/* select j from planar subgraph constructed by nodes with $\hat{r}_{threshold} \leq r < 1$. */
/* ensure that the packet does not loop around the current face */
Step 3: **else if** (($j = Right\_Hand\_Face\_Routing(P'_i).i$) $\wedge$ ($e(i,j) \neq m.e_0$))
forward m to j;
Step 4: **else** drop m;

Figure A.2: DTGR packet forwarding procedure

APPENDIX B

SOURCE CODE OF CARS

```
///**************************************************************************/
// File Name: mesh.cpp
// It's used for multi radio and multi channel Wireless Mesh Networks (WMNs)
// Created by Junmo Yang
// Computer Science and Software Engineering
// Auburn University
/**************************************************************************/


#include "stdafx.h"
#include "Node.h"
#include "Link.h"

#include <fstream>      // For input and output file
#include <math.h>
#include <stdlib.h>
#include <math.h>
#include <algorithm>
#include <iostream>
#include <set>
#include <queue>

using namespace std;

//////// Example of path_info
//
// pathList:      1 2 4 5 6 8
// nicIDSender:    1 2 3 2 1
// nicIDReceiver:  1 3 2 1 5
// channel:        1 2 3 2 1
//
//////////////////////////////
struct path_info {
    vector <int> pathList;
    vector <int> nicIDSender;       // Start from Source Node, End before Destination Node; To
tal = (number of pathList)-1
    vector <int> nicTypeSender;     // Type of NIC - sender
    vector <int> nicIDReceiver;     // Start from Source Node+1, End at Destination Node; Tota
l = (number of pathList)-1
    vector <int> nicTypeReceiver;   // Type of NIC - receiver
    vector <int> channelID;         // Total number = (number of pathList) - 1
    vector <double> availableBW;    // Available BW of a link w/ selected NIC and channel init
ialzied as 0
    int sumOfNumberOfInterferingNBR;// Summation of interfering neighbors it this path, initia
lized as 0
    double minBW;                   // Minimum Bandwidth of this path, initialized as 0.0
};

// Structure of best path info. for each traffic.
struct best_path_info {
    int trafficID;
    vector <int> pathList;
    int totalNBR;
    double minBW;
    int numOfHop;
    int bestPathID;
};


// Searching space
path_info path1[MAX_PATH];
path_info path2[MAX_PATH];
path_info availablePath[MAX_AVAILABLE_PATH];

best_path_info bestPath[MAX_TRAFFIC];

struct traffic_info {
    int id;
    int sourceID;
    int destID;
    int size;
};

traffic_info traffic[MAX_TRAFFIC];
int total_number_of_traffics = 0;
int total_number_of_links = 0;
// Global variable for input parameters (should be obtained by batch files)
int     numOfNode;
int     numOfNIC_a;
int     numOfNIC_b;
int     numOfNIC_g;
double  x_max;
```

```
double  y max;
int     numOfChannel;
double  alpha;
double  beta;
int     protocol;
int     scenario;
int     inOut;
int     gateway;

// Define Functions
void    ReadFile(ifstream & inFile, char * trFile);
void    GenerateNode(int seed);
void    Initialize();
int     CheckConnectivity ();
void    SetNBR();
void    PrintSetsOfNeighbors();
int     FindAvailablePath(int t);
int     checkNBR(int value, int index, int pathID);
int     FindBestPathByShortest(int trafficID, int availablePathNum);
int     FindBestPathByCARS(int trafficID, int availablePathNum);
int     FindBestPathByMicrosoft(int trafficID, int availablePathNum);
void    AfterEstablishedConnection(int bestPathID, int trafficID);
void    PreventNeighborChannel(int prevID, int currID, int type, int channelID);
int     IsLinkAvailable(int pathNum, int prevID, int currID, int trafficID);
int     IsLinkAvailable2(int pathNum, int prevID, int currID, int trafficID);
int     IsChannelAvailable(int prevID, int currID, int nicIDPrev, int nicIDCurr, int pathNum,
int type);
int     CheckAggregation(int prevID, int currID, int pathNum, int trafficID);
int     FindLink(int prevID, int currID, int nicIDPrev, int nicIDCurr, int type);
double  LinkBandwidth(int prevID, int currID, int type);
void    PrintOutResult(int i, ofstream & outFile, char * outFileName);
void    ArrangeDestOfTraffic(int trafficID);

// Define Node objects and Link objects
Node* node[MAX NODE];
Link* link[MAX_LINK];

int _tmain(int argc, _TCHAR* argv[])
{
    int i,seed;
    int execution;
    int availablePathNum    = -1;
    int findoutbestpath     = -1;
    int tempGatewayNum = 0;
    int indicator2 = 0;

    char trFile[200];
    char outFileName[200];


    ifstream inFile1;
    ofstream outFile1;


    if (argc != 16) {
        cout <<endl;
        cout << "Usage: " << endl;
        cout << "1. Routing Protocol (1:Shortest, 2:WMS, 3:Microsoft)" << endl;
        cout << "2. Number of Nodes (200 is Max)" <<endl;
        cout << "3. Max X" <<endl;
        cout << "4. Max Y" <<endl;
        cout << "5. Seed Number (1000, 2000, 3000, ...)" << endl;
        cout << "6. Scenario (1: Burst 2:CBR)" << endl;
        cout << "7. Number of NIC (802.11 a)" <<endl;
        cout << "8. Number of NIC (802.11 g)" <<endl;
        cout << "9. Number of NIC (802.11 b)" <<endl;
        cout << "10. 1 for ourdoor, 2 for indoor" <<endl;
        cout << "11. 0. Peer2Peer, positive integer: number of gateways(9 is max number)" << e
ndl;
        cout << "12. alpha for bandwidth only for CBR" << endl;
        cout << "13. beta for the number of neighbors or 1-beta only for CBR" << endl;
        cout << "14. Traffic File (source destinantion paketSize(Kb))" <<endl;
        cout << "15.File Name for output" << endl;
        exit(1);
    }

    protocol = atoi(argv[1]);
    numOfNode = atoi(argv[2]);
    x max = atoi(argv[3]);
    y max = atoi(argv[4]);
    seed = atoi(argv[5]);
    scenario = atoi(argv[6]);
```

```
numOfNIC a = atoi(argv[7]);
numOfNIC g = atoi(argv[8]);
numOfNIC b = atoi(argv[9]);
inOut = atoi(argv[10]);
gateway = atoi(argv[11]);
alpha = atof(argv[12]);
beta = atof(argv[13]);
strcpy(trFile,argv[14]);
strcpy(outFileName,argv[15]);

ReadFile(inFile1, trFile);          // Read Traffic File.

execution = seed + 1;               // To execute at least one with connected graph
for(; seed < execution; seed++) {


    // If true gateway connections, else then Peer 2 Peer connections
    if(gateway > 0){
        numOfNode = numOfNode + gateway;
    }


    GenerateNode(seed);
    Initialize();
    SetNBR();                       // Setup Neighbors by type of Radio (802.11b, g, a)
    PrintSetsOfNeighbors();         // Print Set of Neighbors

    // Check Network Connection, if not connected, increment seed by 1
    if (CheckConnectivity () == 0) {
        execution++;
        continue;
    }

    for(i = 0; i < total_number_of_traffics; i++) {
        //
        //
        // May need to initialize followings ????
        // And then save best Path onto best_path_info bestPath[MAX_TRAFFIC];
        //
        //path info path1[MAX PATH];
        //path info path2[MAX PATH];
        //path_info availablePath[MAX_AVAILABLE_PATH]
        //
        //

        cout << "For Traffic " << i << ", system is looking for best path." << endl;

        if(gateway > 0){
            tempGatewayNum = i % gateway;
            if(tempGatewayNum == 0) {
                ArrangeDestOfTraffic(i);
                indicator2 = 0;
            }else {
                indicator2++;
            }
        }

        // Construct paths for a traffic ( from source to destination)
        availablePathNum = FindAvailablePath(i);
        if(availablePathNum > 1) {
            if (protocol == 1) {
                findoutbestpath = FindBestPathByShortest(i, availablePathNum);
                if (findoutbestpath != -1) {
                    PrintOutResult(i,outFile1,outFileName);
                }

                if (findoutbestpath == 0 && gateway > 0) {
                    i =i + (gateway - 1 - indicator2);
                }
            }
            else if (protocol == 2) {
                findoutbestpath = FindBestPathByCARS(i, availablePathNum);

                if (findoutbestpath != -1) {
                    PrintOutResult(i,outFile1,outFileName);
                }

                if (findoutbestpath == 0 && gateway > 0) {
                    i =i + (gateway - 1 - indicator2);
                }
            }
            else if (protocol == 3) {
```

```
                    findoutbestpath = FindBestPathByMicrosoft(i, availablePathNum);
                    if (findoutbestpath != -1) {
                        PrintOutResult(i,outFile1,outFileName);
                    }

                    if (findoutbestpath == 0 && gateway > 0) {
                        i =i + (gateway - 1 - indicator2);
                    }
                }
            }
        }
    }

    return 0;
}

// For gateway connection,
// It will change the destination in traffic file...
void ArrangeDestOfTraffic(int trafficID){

    double  distance[10];
    int     idHolder[10];
    double  tempNum1;
    int     tempNum2;

    // Assign distance
    for(int i =0; i < gateway; i++){
        distance[i] = DISTANCE(node[traffic[trafficID].sourceID]->x,node[traffic[trafficID].so
urceID]->y, node[numOfNode - gateway + i]->x, node[numOfNode - gateway + i]->y);
        idHolder[i] = i;
    }

    // Sort array
    for (int j = 0 ; j < gateway-1; j++){
        for(int k = j + 1; k < gateway; k++){
            if(distance[k] < distance[j]){
                tempNum1 = distance[k];
                distance[k] = distance[j];
                distance[j] = tempNum1;

                tempNum2 = idHolder[k];
                idHolder[k] = idHolder[j];
                idHolder[j] = tempNum2;
            }
        }
    }

    for(int i = 0; i < gateway; i++){
        traffic[trafficID].destID = numOfNode - gateway + idHolder[i];
        trafficID++;
    }

}


// Read Trffic File (Source Destination PacketSize)
void ReadFile(ifstream & inFile, char * trFile) {

    int id=0;
    int sourceID;
    int destID;
    int size;

    inFile.open(trFile);

    if(!inFile) {
        cout << "Could not open a file" << endl;
        exit(1);
    }

    while(1)
    {
        inFile >> sourceID;
        if (inFile.eof())
        {
            break;
        }
        total number of traffics++;
        inFile >> destID >> size;
        inFile.ignore(256,'\n');
        if (inFile.eof())
        {
```

```
            break;
        }
        traffic[id].id  = id;
        traffic[id].sourceID = sourceID;
        traffic[id].destID = destID;
        traffic[id].size = size;

        id++;
    }

    inFile.close();
}

// Create Network Topology w/ nodes
void GenerateNode(int seed){

    srand(unsigned(seed));

    // Create Node objects
    for (int i = 0; i < numOfNode; i++) {
        node[i] = new Node(i, (double)x_max*(double)rand()/(double)RAND_MAX, (double)y_max*(do
uble)rand()/(double)RAND_MAX);
    }

     // Print created Node objects
    for (int i = 0; i < numOfNode; i++) {
        cout << "Node ID: " << node[i]->id << " X_POS: " << node[i]->x << " Y_POS: " << node[i
]->y << endl;
    }
}

// Initialize Node, NIC, and Channel
void Initialize(){

    for (int i = 0; i < numOfNode; i++) {
        for(int j = 0; j < numOfNIC b; j++) {
            node[i]->nic802b[j].isUsed = 0;
            node[i]->nic802b[j].recevierID = -1;
            for (int k = 0; k < 3; k++) {
                node[i]->nic802b[j].channel[k] = 0;
            }
        }

        for(int j = 0; j < numOfNIC g; j++) {
            node[i]->nic802g[j].isUsed = 0;
            node[i]->nic802g[j].recevierID  = -1;
            for (int k = 0; k < 3; k++) {
                node[i]->nic802g[j].channel[k] = 0;
            }
        }

        for(int j = 0; j < numOfNIC a; j++) {
            node[i]->nic802a[j].isUsed = 0;
            node[i]->nic802a[j].recevierID = -1;
            for (int k = 0; k < 12; k++) {
                node[i]->nic802a[j].channel[k] = 0;
            }
        }
    }
}

// connectivity check
int CheckConnectivity (){

    int curr = 0; // first node of the queue

    // mark all nodes not visited
    for (int i = 0; i < numOfNode; i++) {
        node[i]->visited = 0;
    }

    queue<int> q;
    q.push(curr);
    node[curr]->visited = 1;

    while ( !q.empty() ) {
        curr = q.front();
        q.pop();
        for (set<int>::const_iterator siter=(node[curr]->nbr802a).begin();siter!=(node[curr]->
nbr802a).end(); ++siter) {
            if (node[*siter]->visited == 0) {
                node[*siter]->visited = 1;
```

```
                q.push(*siter);
            }
        }
    }

    for (i = 0; i < numOfNode; i++) {
        if (node[i]->visited == 0) {
            return 0;
        }
    }

    return 1;
}


// Creating Sets of Neighbors for 802.11a, b, and g respectively for OUTDOOR or INDOOR.
void SetNBR() {

    double distance;

    if (inOut == 1) {// For Outdoor, different transmission radius for 802.11b,g,a
        for (int i = 0; i < numOfNode-1; i++) {
            for (int j = i+1; j < numOfNode; j++) {
                distance = DISTANCE(node[i]->x, node[i]->y,node[j]->x, node[j]->y);

                if (distance <= RADIUS1) {      // if neighbor of 802.11b,
                    node[i]->nbr802b.insert(j);
                    node[j]->nbr802b.insert(i);
                }

                if (distance <= RADIUS2) {      // if neighbor of 802.11g,
                    node[i]->nbr802g.insert(j);
                    node[j]->nbr802g.insert(i);
                }

                if (distance <= RADIUS3) {      // if neighbor of 802.11a,
                    node[i]->nbr802a.insert(j);
                    node[j]->nbr802a.insert(i);
                }
            }
        }

    }else{// For Indoor, same transmission radius for 802.11b,g,a
        for (int i = 0; i < numOfNode-1; i++) {
            for (int j = i+1; j < numOfNode; j++) {
                distance = DISTANCE(node[i]->x, node[i]->y,node[j]->x, node[j]->y);

                if (distance <= RADIUS3) {      // if neighbor of 802.11b, g, or a,
                    node[i]->nbr802b.insert(j);
                    node[j]->nbr802b.insert(i);

                    node[i]->nbr802g.insert(j);
                    node[j]->nbr802g.insert(i);

                    node[i]->nbr802a.insert(j);
                    node[j]->nbr802a.insert(i);
                }
            }
        }
    }
}

// Print the sets of neighbors by radio types (802.11b, g, and a)
void PrintSetsOfNeighbors(){

    // Print out the node and its neighbors
    for (int i = 0; i < numOfNode; i++) {
        cout << "Node " << node[i]->id << " 802.11b: ";
        for(int j = 0; j < numOfNIC_b; j++){
            cout << " B-ID: " << j << " ";
            for (set<int>::const_iterator it = (node[i]->nbr802b).begin(); it != (node[i]->nbr
802b).end(); ++it) {
                cout << " " << node[*it]->id;
            }
        }
        cout << endl;

        cout << "Node " << node[i]->id << " 802.11g: ";
        for(int j = 0; j < numOfNIC_g; j++){
            cout << " G-ID: " << j << " ";
            for (set<int>::const_iterator it = (node[i]->nbr802g).begin(); it != (node[i]->nbr
802g).end(); ++it) {
```

```
                cout << " " << node[*it]->id;
            }
        }
        cout << endl;

        cout << "Node " << node[i]->id << " 802.11a: ";
        for(int j = 0; j < numOfNIC_a; j++){
            cout << " A-ID: " << j << " ";
            for (set<int>::const_iterator it = (node[i]->nbr802a).begin(); it != (node[i]->nbr
802a).end(); ++it) {
                cout << " " << node[*it]->id;
            }
        }
        cout << endl;
    }
}


// This function is to set up "availablePath[i]"
// && return the number of available paths
int FindAvailablePath(int t) {

    int i = 0;
    int j = 0;
    int forq1 = 0;              // Stopping indication of sub-tree

    int source;
    int destination;
    int execution1=0;
    int execution2=0;
    int availablePathID = 0;
    int curr;

    // Indication for Breadth-First Search, in order to move next sub-tree.....
    execution1 = 0;
    execution2 = 0;

    // Initialize vectors of path1 and path2 (Temporal Holder)
    for(int w=0; w < MAX_PATH; w++) {
        path1[w].pathList.clear();
        path2[w].pathList.clear();
    }

    // Initialize vector of actual available path
    for(int w=0; w < MAX_AVAILABLE_PATH; w++) {
        availablePath[w].pathList.clear();
        availablePath[w].channelID.clear();
        availablePath[w].nicIDSender.clear();
        availablePath[w].nicIDReceiver.clear();
        availablePath[w].availableBW.clear();
        availablePath[w].nicTypeSender.clear();
        availablePath[w].nicTypeReceiver.clear();
        availablePath[w].sumOfNumberOfInterferingNBR = 0;
        availablePath[w].minBW = 999999999999.0;
    }

    // Source and Destination
    source = traffic[t].sourceID;
    destination = traffic[t].destID;

    // Start from Source
    curr = source;


    //clear and insert first node (source) onto the root vector
    path1[execution1].pathList.clear();
    path1[execution1].pathList.push_back(curr);

    execution1++;   // move to next tree
    forq1 = 0;      // Another indication in order to stop one layer in sub-tree

    // execute until it reaches MAX_PATH or there is no more existing paths
    while( execution1 <= MAX_PATH && execution2 <= MAX_PATH) {

        if( forq1 == 0)
        {
            execution2 = 0;
            for(i = 0; i < execution1; i++) {

                // Checking it has been reached to MAX_NUMBER of searching space
                if (path1[i].pathList.at(0) != source || execution2 >= MAX_PATH) {
                    execution1 = 0; // in order to stop
```

```
                    break;
                }
                curr = path1[i].pathList.at(path1[i].pathList.size() - 1);

                for (set<int>::const_iterator siter=(node[curr]->nbr802b).begin();siter!=(node
[curr]->nbr802b).end(); ++siter) {

                    // If found path, put it onto available_path
                    if (*siter == destination) {
                        availablePath[availablePathID].pathList.clear();

                        for (vector<int>::const_iterator it2=(path1[i].pathList).begin();it2!=
(path1[i].pathList).end(); ++it2) {
                            availablePath[availablePathID].pathList.push_back(*it2);
                        }

                        availablePath[availablePathID].pathList.push_back(*siter);
                        availablePathID++;

                        continue;
                    }

                    if (!(checkNBR(*siter, i, 1))) {

                        path2[execution2].pathList.clear();

                        // Copy All elemetns in Path1[i] to Path2[j]
                        for (vector<int>::const_iterator it2=(path1[i].pathList).begin();it2!=
(path1[i].pathList).end(); ++it2) {
                            path2[execution2].pathList.push_back(*it2);
                        }

                        path2[execution2].pathList.push_back(*siter);
                        execution2++;
                    }
                }
            }
            forq1 = 1;
        }
        else
        {
            execution1 = 0;
            for(i = 0; i < execution2; i++) {
                if (path2[i].pathList.at(0) != source || execution1 >= MAX_PATH) {
                    execution2 = 0; // in order to stop
                    break;
                }

                curr = path2[i].pathList.at(path2[i].pathList.size() - 1);

                for (set<int>::const_iterator siter=(node[curr]->nbr802b).begin();siter!=(node
[curr]->nbr802b).end(); ++siter) {

                    // If found path, put it onto available_path
                    if (*siter == destination) {
                        availablePath[availablePathID].pathList.clear();

                        for (vector<int>::const_iterator it2=(path2[i].pathList).begin();it2!=
(path2[i].pathList).end(); ++it2) {
                            availablePath[availablePathID].pathList.push_back(*it2);
                        }

                        availablePath[availablePathID].pathList.push_back(*siter);
                        availablePathID++;

                        continue;
                    }

                    if (!(checkNBR(*siter, i, 2))) {

                        path1[execution1].pathList.clear();

                        // Copy All elemetns in Path2[i] to Path1[j]
                        for (vector<int>::const_iterator it2=(path2[i].pathList).begin();it2!=
(path2[i].pathList).end(); ++it2) {
                            path1[execution1].pathList.push_back(*it2);
                        }

                        path1[execution1].pathList.push_back(*siter);

                        execution1++;
                    }
```

```
                }
            }
            forq1 = 0;
        }

        if (execution1 == 0 || execution2 == 0 || execution1 == execution2 || availablePathID
== MAX_AVAILABLE_PATH - 1) break;
    }

    return availablePathID;
}

// Check out it it neighbor and it should not be in the path vector.
int checkNBR(int value, int index, int pathID) {
    if (pathID == 1) {
        for (vector<int>::const iterator siter=(path1[index].pathList).begin(); siter!=(path1[
index].pathList).end(); ++siter) {
            if(value == *siter) return 1;
        }
    }else {
        for (vector<int>::const iterator siter=(path2[index].pathList).begin(); siter!=(path2[
index].pathList).end(); ++siter) {
            if(value == *siter) return 1;
        }
    }
    return 0;
}

// Find out Best Path from available paths for Shortest, previously called as WMS)
int FindBestPathByShortest(int trafficID, int availablePathNum) {

    int     tempNum;    // in order to start after reading source
    int     prev;       // sender
    int     curr;       // receiver

    int     nbr;        // number of neighbor, -1: indicate link failure

    double  cost;
    int     bestSumOfNBR;
    double  bestMinbw;
    double  bestCost = 100000000.00;

    int bestPathID = -1;

    // Start from first availablePath
    for(int i = 1; i < availablePathNum; i++) {

        int indi = 0;

        // Start form first link in availablePath[i]
        for (vector<int>::const_iterator it2=availablePath[i].pathList.begin();it2!=availableP
ath[i].pathList.end(); ++it2) {
            if (tempNum == 0) {
                curr = *it2;
                tempNum = 1;
            }
            else {
                prev = curr;
                curr = *it2;

                //IsLinkAvailabel returns number of interfering neighbors
                // nbr does not need to be calculated, since availablePath[i].sumOfNumberOfInt
erferingNBR includes info..
                nbr = IsLinkAvailable2(i, prev, curr, trafficID);

                if(nbr == -1){  // if true, a link can NOT be established, it should stop, and
 go to next available path..
                    indi = 1;
                    break;
                }
            }
        }

        if (indi == 1) {
            indi = 0;
            continue;
        }

        // Not Actually USED... It's just copy of CARS
        if (scenario == 1) {        // BURST Traffic
            cost = pow((double)(availablePath[i].sumOfNumberOfInterferingNBR), beta)/pow((doub
le)(availablePath[i].minBW), alpha);
```

```
        }
        else if (scenario == 2) {   // CBR Traffic
            cost = (double)(availablePath[i].sumOfNumberOfInterferingNBR)/1.0;
        }

        // If found, setup the value and then exit......
        if (cost < bestCost) {
            bestSumOfNBR = availablePath[i].sumOfNumberOfInterferingNBR;
            bestMinbw = availablePath[i].minBW;
            bestCost = cost;
            bestPathID = i;
            break;
        }
    }

    if (bestPathID != -1) {         // if found best path from availablePath[i]

        //AfterBestPathSelection(trafficID, bestNICList, bestChannelList);
        bestPath[trafficID].minBW = bestMinbw;
        bestPath[trafficID].totalNBR = bestSumOfNBR;
        bestPath[trafficID].trafficID = trafficID;

        for (vector<int>::const iterator it2=availablePath[bestPathID].pathList.begin();it2!=a
vailablePath[bestPathID].pathList.end(); ++it2) {
            bestPath[trafficID].pathList.push_back(*it2);
        }

        bestPath[trafficID].numOfHop = bestPath[trafficID].pathList.size() - 1;
        bestPath[trafficID].bestPathID = bestPathID;


        /****************************************
        - After finding out one path........, then do followings..........
        - Need to setup links.......
        - Need to change NIC and Channels in each NODE or LINK object
        - Create print out solution
        - How to avoid hidden terminal problems or others
        - Make sure BURST and CBR Traffic
        - Make sure all input parameters are considered correctly
        ****************************************/

        AfterEstablishedConnection(bestPathID, trafficID);

        return 0;
    }
    else {                          // if not found bestPath[i]
        bestPath[trafficID].bestPathID = -1;
        return -1;
    }
}

// Find out Best Path from available paths for CARS, previously called as WMS)
int FindBestPathByCARS(int trafficID, int availablePathNum) {

    int     tempNum;    // in order to start after reading source
    int     prev;       // sender
    int     curr;       // receiver

    int     nbr;        // number of neighbor, -1: indicate link failure

    double  cost;
    int     bestSumOfNBR;
    double  bestMinbw;
    double  bestCost = 100000000.00;

    int bestPathID = -1;

    // Start from first availablePath
    for(int i = 1; i < availablePathNum; i++) {

        int indi = 0;

        // Start form first link in availablePath[i]
        for (vector<int>::const_iterator it2=availablePath[i].pathList.begin();it2!=availableP
ath[i].pathList.end(); ++it2) {
            if (tempNum == 0) {
                curr = *it2;
                tempNum = 1;
            }
            else {
                prev = curr;
                curr = *it2;
```

```
                //IsLinkAvailabel returns number of interfering neighbors
                // nbr does not need to be calculated, since availablePath[i].sumOfNumberOfInt
erferingNBR includes info..
                nbr = IsLinkAvailable(i, prev, curr, trafficID);

                if(nbr == -1){  // if true, a link can NOT be established, it should stop, and
 go to next available path..
                    indi = 1;
                    break;
                }
            }
        }

        if (indi == 1) {
            indi = 0;
            continue;
        }

        if (scenario == 1) {        // BURST Traffic
            cost = pow((double)(availablePath[i].sumOfNumberOfInterferingNBR), beta)/pow((doub
le)(availablePath[i].minBW), alpha);
        }
        else if (scenario == 2) {   // CBR Traffic
            cost = (double)(availablePath[i].sumOfNumberOfInterferingNBR)/1.0;
        }

        if (cost < bestCost) {
            bestSumOfNBR = availablePath[i].sumOfNumberOfInterferingNBR;
            bestMinbw = availablePath[i].minBW;
            bestCost = cost;
            bestPathID = i;
        }
    }

    if (bestPathID != -1) {         // if found best path from availablePath[i]

        //AfterBestPathSelection(trafficID, bestNICList, bestChannelList);
        bestPath[trafficID].minBW = bestMinbw;
        bestPath[trafficID].totalNBR = bestSumOfNBR;
        bestPath[trafficID].trafficID = trafficID;

        for (vector<int>::const iterator it2=availablePath[bestPathID].pathList.begin();it2!=a
vailablePath[bestPathID].pathList.end(); ++it2) {
            bestPath[trafficID].pathList.push_back(*it2);
        }

        bestPath[trafficID].numOfHop = bestPath[trafficID].pathList.size() - 1;
        bestPath[trafficID].bestPathID = bestPathID;


        /****************************************
        - After finding out one path........, then do followings..........
        - Need to setup links.......
        - Need to change NIC and Channels in each NODE or LINK object
        - Create print out solution
        - How to avoid hidden terminal problems or others
        - Make sure BURST and CBR Traffic
        - Make sure all input parameters are considered correctly
        ****************************************/

        AfterEstablishedConnection(bestPathID, trafficID);

        return 0;
    }
    else {                          // if not found bestPath[i]
        bestPath[trafficID].bestPathID = -1;
        return -1;
    }
}

int FindBestPathByMicrosoft(int trafficID, int availablePathNum) {

    int     tempNum;    // in order to start after reading source
    int     prev;       // sender
    int     curr;       // receiver

    int     nbr;        // number of neighbor, -1: indicate link failure
    double sett;            // sum of ett (= Packet Size / Bandwidth) Expected Transmission Ti
me
    double bgett[3];    // ( == grouped sum of ett), Largest value will be selected for cost.
lower value has better throughput
```

```
    double  cost;
    int     bestSumOfNBR;
    double  bestMinbw;
    double  bestCost = 100000000.00;

    int bestPathID = -1;

    // Start from first availablePath
    for(int i = 1; i < availablePathNum; i++) {

        int indi = 0;
        sett = 0;
        for(int e=0; e < 3; e++) {
            bgett[e] = 0;
        }

        // Start form first link in availablePath[i]
        for (vector<int>::const_iterator it2=availablePath[i].pathList.begin();it2!=availableP
ath[i].pathList.end(); ++it2) {
            if (tempNum == 0) {
                curr = *it2;
                tempNum = 1;
            }
            else {
                prev = curr;
                curr = *it2;

                //IsLinkAvailabel returns number of interfering neighbors
                // nbr does not need to be calculated, since availablePath[i].sumOfNumberOfInt
erferingNBR includes info..
                nbr = IsLinkAvailable2(i, prev, curr, trafficID);

                if(nbr == -1){  // if true, a link can NOT be established, it should stop, and
 go to next available path..
                    indi = 1;
                    break;
                }
            }
        }

        if (indi == 1) {
            indi = 0;
            continue;
        }

        //SETT
        for (vector<double>::const iterator it2=availablePath[i].availableBW.begin();it2!=avai
lablePath[i].availableBW.end(); ++it2) {
            sett = sett + traffic[trafficID].size/(*it2);
        }

        //BGETT
        vector<double>::const iterator it3=availablePath[i].availableBW.begin();
        for (vector<int>::const iterator it2=availablePath[i].nicTypeSender.begin();it2!=avail
ablePath[i].nicTypeSender.end(); ++it2) {
            if (*it2 == TYPE B) {
                bgett[*it2 - 1] = bgett[*it2 - 1] + traffic[trafficID].size/(*it3);
            }else if ( *it2 == TYPE G){
                bgett[*it2 - 1] = bgett[*it2 - 1] + traffic[trafficID].size/(*it3);
            }else {
                bgett[*it2 - 1] = bgett[*it2 - 1] + traffic[trafficID].size/(*it3);
            }
            ++it3;
        }

        if (scenario == 1) {        // BURST Traffic
            double largestBgett = 0.0;
            for(int e = 0; e < 3; e++) {
                if(bgett[e] > largestBgett) {
                    largestBgett = bgett[e];
                }
            }

            cost = pow((double)sett, beta) + pow(largestBgett, alpha);
        }
        else if (scenario == 2) {   // CBR Traffic
            double largestBgett = 0.0;
            for(int e = 0; e < 3; e++) {
                if(bgett[e] > largestBgett) {
                    largestBgett = bgett[e];
```

```
                }
            }

            cost = pow((double)sett, beta) + pow(largestBgett, alpha);
        }

        if (cost < bestCost) {
            bestSumOfNBR = availablePath[i].sumOfNumberOfInterferingNBR;
            bestMinbw = availablePath[i].minBW;
            bestCost = cost;
            bestPathID = i;
        }
    }

    if (bestPathID != -1) {         // if found best path from availablePath[i]

        bestPath[trafficID].minBW = bestMinbw;
        bestPath[trafficID].totalNBR = bestSumOfNBR;
        bestPath[trafficID].trafficID = trafficID;

        for (vector<int>::const iterator it2=availablePath[bestPathID].pathList.begin();it2!=a
vailablePath[bestPathID].pathList.end(); ++it2) {
            bestPath[trafficID].pathList.push_back(*it2);
        }

        bestPath[trafficID].numOfHop = bestPath[trafficID].pathList.size() - 1;
        bestPath[trafficID].bestPathID = bestPathID;


        /****************************************
        - After finding out one path........, then do followings..........
        - Need to setup links.......
        - Need to change NIC and Channels in each NODE or LINK object
        - Create print out solution
        - How to avoid hidden terminal problems or others
        - Make sure BURST and CBR Traffic
        - Make sure all input parameters are considered correctly
        ****************************************/

        AfterEstablishedConnection(bestPathID, trafficID);

        return 0;
    }
    else {                          // if not found bestPath[i]
        bestPath[trafficID].bestPathID = -1;
        return -1;
    }
}

// Setup NIC and Channel after establishing connection
void AfterEstablishedConnection(int bestPathID, int trafficID){

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

    Need to consider Aggregation.....
    Need to consider sender or receiver's NICs and Channels
    for connection and for hidden terminal
    See Paper.....................

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

    int prevID, currID;
    int linkID;
    double minBW;
    double availableBW;

    // if true, BURST, else CBR TRaffic
    if (scenario == 1){
        minBW = availablePath[bestPathID].minBW;
    }else{
        minBW = traffic[trafficID].size;
    }

    vector<int>::const iterator it1=availablePath[bestPathID].pathList.begin();
    vector<int>::const iterator it2=availablePath[bestPathID].nicIDSender.begin();
    vector<int>::const iterator it3=availablePath[bestPathID].nicTypeSender.begin();
    vector<int>::const iterator it4=availablePath[bestPathID].nicIDReceiver.begin();
    vector<int>::const iterator it5=availablePath[bestPathID].nicTypeReceiver.begin();
    vector<int>::const iterator it6=availablePath[bestPathID].channelID.begin();
    vector<double>::const_iterator it7 = availablePath[bestPathID].availableBW.begin();

    // Header of results
```

```
    cout << "Sender ID\t" << "Receiver ID\t" <<"NIC-Type\t" << "Sender-NIC-ID\t" << "Receiver-
NIC-ID\t" << "Channel-ID" << endl;

    currID = *it1;
    ++it1;
    for (;it1!=availablePath[bestPathID].pathList.end(); ++it1, ++it2, ++it3, ++it4, ++it5, ++
it6, ++it7) {
        prevID = currID;
        currID = *it1;

        linkID = FindLink(prevID, currID, *it2, *it4, *it3);
        if(linkID != -1){
            link[linkID]->availableBW = link[linkID]->availableBW - minBW;
            continue;       // skip rest of process and then goto top
        }else {
            availableBW = *it7 - minBW;

            // Create a link between sender and receiver w/ selected radio and channel..
            link[total number of links] = new Link(total_number_of_links, prevID, currID, *it3
, *it2, *it4, *it6, *it6, availableBW);

            // Prevent neighbor's channel from being used....in order to avoid collision
            // Radios still can be used.......
            PreventNeighborChannel(prevID, currID, *it3, *it6);
        }

        // For Sender
        if(*it3 == TYPE_B){          // 802.11b
            node[prevID]->nic802b[*it2].isUsed = 2;
            node[prevID]->nic802b[*it2].recevierID = currID;
            for(int i = 0; i < numOfNIC_b; i++){
                node[prevID]->nic802b[i].channel[*it6] = 1;
            }

            for(int i = 0; i < numOfNIC_g; i++){
                node[prevID]->nic802g[i].channel[*it6] = 1;
            }
        }else if(*it3 == TYPE_G){    // 802.11g
            node[prevID]->nic802g[*it2].isUsed = 2;
            node[prevID]->nic802g[*it2].recevierID = currID;
            for(int i = 0; i < numOfNIC_g; i++){
                node[prevID]->nic802g[i].channel[*it6] = 1;
            }

            for(int i = 0; i < numOfNIC_b; i++){
                node[prevID]->nic802b[i].channel[*it6] = 1;
            }
        }else{                       // 802.11a
            node[prevID]->nic802a[*it2].isUsed = 2;
            node[prevID]->nic802a[*it2].recevierID = currID;
            for(int i = 0; i < numOfNIC_a; i++){
                node[prevID]->nic802a[i].channel[*it6] = 1;
            }
        }

        // For Receiver
        if(*it5 == TYPE_B){          // 802.11b
            node[prevID]->nic802b[*it4].isUsed = 3;
            node[prevID]->nic802b[*it2].senderID = prevID;
            for(int i = 0; i < numOfNIC_b; i++){
                node[currID]->nic802b[i].channel[*it6] = 1;
            }

            for(int i = 0; i < numOfNIC_g; i++){
                node[currID]->nic802g[i].channel[*it6] = 1;
            }
        }else if(*it5 == TYPE_G){    // 802.11g
            node[prevID]->nic802g[*it4].isUsed = 3;
            node[prevID]->nic802g[*it2].senderID = prevID;
            for(int i = 0; i < numOfNIC_g; i++){
                node[currID]->nic802g[i].channel[*it6] = 1;
            }

            for(int i = 0; i < numOfNIC_b; i++){
                node[currID]->nic802b[i].channel[*it6] = 1;
            }
        }else{                       // 802.11a
            node[prevID]->nic802a[*it4].isUsed = 3;
            node[prevID]->nic802a[*it2].senderID = prevID;
            for(int i = 0; i < numOfNIC_a; i++){
                node[currID]->nic802a[i].channel[*it6] = 1;
            }
```

```
        }

        // In order to debug result
        cout << prevID << "\t" << currID << "\t" << *it3 << "\t" <<*it2 << "\t" << *it4 << "\t
" << *it6 << endl;
    }
}

// After creating a link, it should prevent neighbor's same channel from being used by others.
// in order to avoid collision
void PreventNeighborChannel(int prevID, int currID, int type, int channelID){

    // For neighbor of sender
    if(type == TYPE_B){
        for (set<int>::const_iterator it=node[prevID]->nbr802b.begin();it!=node[prevID]->nbr80
2b.end(); ++it) {
            for(int i = 0; i < numOfNIC_b; i++){
                if(node[*it]->nic802b[i].isUsed != 2 && node[*it]->nic802b[i].isUsed != 3){
                    node[*it]->nic802b[i].isUsed = 1;
                }

                node[*it]->nic802b[i].channel[channelID] = 1;
            }

            for(int i = 0; i < numOfNIC_g; i++){
                node[*it]->nic802g[i].channel[channelID] = 1;
            }
        }
    }else if(type == TYPE_G){
        for (set<int>::const_iterator it=node[prevID]->nbr802g.begin();it!=node[prevID]->nbr80
2g.end(); ++it) {
            for(int i = 0; i < numOfNIC_g; i++){
                if(node[*it]->nic802g[i].isUsed != 2 && node[*it]->nic802g[i].isUsed != 3){
                    node[*it]->nic802g[i].isUsed = 1;
                }

                node[*it]->nic802g[i].channel[channelID] = 1;
            }

            for(int i = 0; i < numOfNIC_b; i++){
                node[*it]->nic802b[i].channel[channelID] = 1;
            }
        }
    }else{
        for (set<int>::const_iterator it=node[prevID]->nbr802a.begin();it!=node[prevID]->nbr80
2a.end(); ++it) {
            for(int i = 0; i < numOfNIC_a; i++){
                if(node[*it]->nic802a[i].isUsed != 2 && node[*it]->nic802a[i].isUsed != 3){
                    node[*it]->nic802a[i].isUsed = 1;
                }

                node[*it]->nic802a[i].channel[channelID] = 1;
            }
        }
    }

    // For neighbor of receiver
    if(type == TYPE_B){
        for (set<int>::const_iterator it=node[currID]->nbr802b.begin();it!=node[currID]->nbr80
2b.end(); ++it) {
            for(int i = 0; i < numOfNIC_b; i++){
                node[*it]->nic802b[i].channel[channelID] = 1;
            }

            for(int i = 0; i < numOfNIC_g; i++){
                node[*it]->nic802g[i].channel[channelID] = 1;
            }
        }
    }else if(type == TYPE_G){
        for (set<int>::const_iterator it=node[currID]->nbr802g.begin();it!=node[currID]->nbr80
2g.end(); ++it) {
            for(int i = 0; i < numOfNIC_g; i++){
                node[*it]->nic802g[i].channel[channelID] = 1;
            }

            for(int i = 0; i < numOfNIC_b; i++){
                node[*it]->nic802b[i].channel[channelID] = 1;
            }
        }
    }else{
        for (set<int>::const_iterator it=node[currID]->nbr802a.begin();it!=node[currID]->nbr80
2a.end(); ++it) {
```

```
            for(int i = 0; i < numOfNIC_a; i++){
                node[*it]->nic802a[i].channel[channelID] = 1;
            }
        }
    }
}

// It's only for CARS (previously called as WMS), not shortest path nor Microsoft
// It must return number of interfering neighbors, if available. if not available. Returns 0,
// This function also should setup the NIC and Channel for a link in availablepath[i]...
// It also calculate number of interfering neighbors and availble bandwidth, and min bandwidth
..
// however available bandwidth should be calculated after establish a connection
// since we are not sure what is the minimum bandwidth for BURST traffic......
int IsLinkAvailable(int pathNum, int prevID, int currID, int trafficID) {

    int checking = -1;
    int linkID = -1;
    int channelID = -1;
    double bwLink;

    //NIC ---  0: Idle 1: Hearing  2: Sending  3: Receiving, -1: cannot be used
    //Channel ---  0: Unused  1: Used   -1: cannot be used  -2:aggregation

    linkID = CheckAggregation(prevID, currID, pathNum, trafficID);
    if(linkID == -1){           //if TRUE, then can NOT be aggregate. if false can be aggregat
e

        // Check 802.11a can make a link
        // There exist same type of radio and available channel for both sender and receiver.
        for(int i = 0; i < numOfNIC_a; i++){
            if(node[prevID]->nic802a[i].isUsed == 0 || node[prevID]->nic802a[i].isUsed == 1){
                for(int j = 0; j < numOfNIC_a; j++){
                    if(node[currID]->nic802a[j].isUsed == 0 || node[currID]->nic802a[j].isUsed
 == 1){

                        // Check that there is any available channel for both sender and recei
ver.
                        // The channel ID should be same for both
                        // Before selecting a radio and channel, it should check out that
                        // radio should be differ from the previously used one.
                        // For 802.11b and 802.11g should check out channel also.....

                        // In order to avoid using same NIC at a node
                        if(availablePath[pathNum].nicIDReceiver.size()){
                            if(availablePath[pathNum].nicTypeReceiver.at(availablePath[pathNum
].nicIDReceiver.size()-1) == TYPE_A && availablePath[pathNum].nicIDReceiver.at(availablePath[p
athNum].nicIDReceiver.size()-1) == i){
                                continue;
                            }
                        }

                        channelID = IsChannelAvailable(prevID, currID, i, j, pathNum, TYPE_A);
                        if (channelID == -1){
                            // No more available channel in selected NIC
                            // Cannot establishing a link
                            // However, it should try to find out another combination of NIC
                            // Should not stop searching for other combination
                        }else{
                            bwLink = LinkBandwidth(prevID, currID, TYPE_A);
                            if(bwLink >= traffic[trafficID].size){
                                availablePath[pathNum].nicIDSender.push_back(i);
                                availablePath[pathNum].nicTypeSender.push_back(TYPE_A);
                                availablePath[pathNum].nicIDReceiver.push_back(j);
                                availablePath[pathNum].nicTypeReceiver.push_back(TYPE_A);
                                availablePath[pathNum].channelID.push_back(channelID);
                                availablePath[pathNum].availableBW.push_back(bwLink);

                                availablePath[pathNum].sumOfNumberOfInterferingNBR = available
Path[pathNum].sumOfNumberOfInterferingNBR + node[prevID]->nbr802a.size();

                                if (bwLink < availablePath[pathNum].minBW){
                                    availablePath[pathNum].minBW = bwLink;
                                }

                                return node[prevID]->nbr802a.size();

                            }else{
                                // Bandwidth is not enough
                                // Therefore, cannot establish a link using such NIC
                                // However, should not stop searching for other combination
                            }
```

```
                        }
                    }
                }
            }
        }

        // Check 802.11g can make a link
        // There exist same type of radio and available channel for both sender and receiver.
        for(int i = 0; i < numOfNIC g; i++){
            if(node[prevID]->nic802g[i].isUsed == 0 || node[prevID]->nic802g[i].isUsed == 1){
                for(int j = 0; j < numOfNIC g; j++){
                    if(node[currID]->nic802g[j].isUsed == 0 || node[currID]->nic802g[j].isUsed
 == 1){

                        // Check that there is any available channel for both sender and recei
ver.
                        // The channel ID should be same for both
                        // Before selecting a radio and channel, it should check out that
                        // radio should be differ from the previously used one.
                        // For 802.11b and 802.11g should check out channel also.....

                        // In order to avoid using same NIC at a node
                        if(availablePath[pathNum].nicIDReceiver.size()){
                            if(availablePath[pathNum].nicTypeReceiver.at(availablePath[pathNum
].nicIDReceiver.size()-1) == TYPE G && availablePath[pathNum].nicIDReceiver.at(availablePath[p
athNum].nicIDReceiver.size()-1) == i){
                                continue;
                            }
                        }

                        channelID = IsChannelAvailable(prevID, currID, i, j, pathNum, TYPE_G);
                        if (channelID == -1){
                            // No more available channel in selected NIC
                            // Cannot establishing a link
                            // However, it should try to find out another combination of NIC
                            // Should not stop searching for other combination
                        }else{
                            bwLink = LinkBandwidth(prevID, currID, TYPE_G);
                            if(bwLink >= traffic[trafficID].size){
                                availablePath[pathNum].nicIDSender.push back(i);
                                availablePath[pathNum].nicTypeSender.push back(TYPE_G);
                                availablePath[pathNum].nicIDReceiver.push back(j);
                                availablePath[pathNum].nicTypeReceiver.push back(TYPE G);
                                availablePath[pathNum].channelID.push back(channelID);
                                availablePath[pathNum].availableBW.push_back(bwLink);

                                availablePath[pathNum].sumOfNumberOfInterferingNBR = available
Path[pathNum].sumOfNumberOfInterferingNBR + node[prevID]->nbr802g.size();

                                if (bwLink < availablePath[pathNum].minBW){
                                    availablePath[pathNum].minBW = bwLink;
                                }

                                return node[prevID]->nbr802g.size();

                            }else{
                                // Bandwidth is not enough
                                // Therefore, cannot establish a link using such NIC
                                // However, should not stop searching for other combination
                            }
                        }
                    }
                }
            }
        }


        // Check 802.11b can make a link
        // There exist same type of radio and available channel for both sender and receiver.
        for(int i = 0; i < numOfNIC b; i++){
            if(node[prevID]->nic802b[i].isUsed == 0 || node[prevID]->nic802b[i].isUsed == 1){
                for(int j = 0; j < numOfNIC b; j++){
                    if(node[currID]->nic802b[j].isUsed == 0 || node[currID]->nic802b[j].isUsed
 == 1){

                        // Check that there is any available channel for both sender and recei
ver.
                        // The channel ID should be same for both
                        // Before selecting a radio and channel, it should check out that
                        // radio should be differ from the previously used one.
                        // For 802.11b and 802.11g should check out channel also.....
```

```
                        // In order to avoid using same NIC at a node
                        if(availablePath[pathNum].nicIDReceiver.size()){
                            if(availablePath[pathNum].nicTypeReceiver.at(availablePath[pathNum
].nicIDReceiver.size()-1) == TYPE_B && availablePath[pathNum].nicIDReceiver.at(availablePath[p
athNum].nicIDReceiver.size()-1) == i){
                                continue;
                            }
                        }

                        channelID = IsChannelAvailable(prevID, currID, i, j, pathNum, TYPE_B);
                        if (channelID == -1){
                            // No more available channel in selected NIC
                            // Cannot establishing a link
                            // However, it should try to find out another combination of NIC
                            // Should not stop searching for other combination
                        }else{
                            bwLink = LinkBandwidth(prevID, currID, TYPE_B);
                            if(bwLink >= traffic[trafficID].size){
                                availablePath[pathNum].nicIDSender.push_back(i);
                                availablePath[pathNum].nicTypeSender.push_back(TYPE_B);
                                availablePath[pathNum].nicIDReceiver.push_back(j);
                                availablePath[pathNum].nicTypeReceiver.push_back(TYPE_B);
                                availablePath[pathNum].channelID.push_back(channelID);
                                availablePath[pathNum].availableBW.push_back(bwLink);

                                availablePath[pathNum].sumOfNumberOfInterferingNBR = available
Path[pathNum].sumOfNumberOfInterferingNBR + node[prevID]->nbr802b.size();

                                if (bwLink < availablePath[pathNum].minBW){
                                    availablePath[pathNum].minBW = bwLink;
                                }

                                return node[prevID]->nbr802b.size();

                            }else{
                                // Bandwidth is not enough
                                // Therefore, cannot establish a link using such NIC
                                // However, should not stop searching for other combination
                            }
                        }
                    }
                }
            }
        }
    }else{
        return checking; // It must be 0 due to the aggregation
    }

    return checking;
}

// It's only for Microsoft and Shortest Path
int IsLinkAvailable2(int pathNum, int prevID, int currID, int trafficID) {

    int checking = -1;
    int linkID = -1;
    int channelID = -1;
    double bwLink;

    //NIC ---  0: Idle 1: Hearing  2: Sending  3: Receiving, -1: cannot be used
    //Channel ---  0: Unused  1: Used   -1: cannot be used  -2:aggregation

    linkID = CheckAggregation(prevID, currID, pathNum, trafficID);
    if(linkID == -1){           //if TRUE, then can NOT be aggregate. if false can be aggregat
e

        // Check 802.11a can make a link
        // There exist same type of radio and available channel for both sender and receiver.
        for(int i = 0; i < numOfNIC_a; i++){
            if(node[prevID]->nic802a[i].isUsed == 0){
                for(int j = 0; j < numOfNIC_a; j++){
                    if(node[currID]->nic802a[j].isUsed == 0){

                        // Check that there is any available channel for both sender and recei
ver.
                        // The channel ID should be same for both
                        // Before selecting a radio and channel, it should check out that
                        // radio should be differ from the previously used one.
                        // For 802.11b and 802.11g should check out channel also.....

                        // In order to avoid using same NIC at a node
                        if(availablePath[pathNum].nicIDReceiver.size()){
```

```
                            if(availablePath[pathNum].nicTypeReceiver.at(availablePath[pathNum
].nicIDReceiver.size()-1) == TYPE_A && availablePath[pathNum].nicIDReceiver.at(availablePath[p
athNum].nicIDReceiver.size()-1) == i){
                                continue;
                            }
                        }

                        channelID = IsChannelAvailable(prevID, currID, i, j, pathNum, TYPE_A);
                        if (channelID == -1){
                            // No more available channel in selected NIC
                            // Cannot establishing a link
                            // However, it should try to find out another combination of NIC
                            // Should not stop searching for other combination
                        }else{
                            bwLink = LinkBandwidth(prevID, currID, TYPE_A);
                            if(bwLink >= traffic[trafficID].size){
                                availablePath[pathNum].nicIDSender.push_back(i);
                                availablePath[pathNum].nicTypeSender.push_back(TYPE_A);
                                availablePath[pathNum].nicIDReceiver.push_back(j);
                                availablePath[pathNum].nicTypeReceiver.push_back(TYPE_A);
                                availablePath[pathNum].channelID.push_back(channelID);
                                availablePath[pathNum].availableBW.push_back(bwLink);

                                availablePath[pathNum].sumOfNumberOfInterferingNBR = available
Path[pathNum].sumOfNumberOfInterferingNBR + node[prevID]->nbr802a.size();

                                if (bwLink < availablePath[pathNum].minBW){
                                    availablePath[pathNum].minBW = bwLink;
                                }

                                return node[prevID]->nbr802a.size();

                            }else{
                                // Bandwidth is not enough
                                // Therefore, cannot establish a link using such NIC
                                // However, should not stop searching for other combination
                            }
                        }
                    }
                }
            }
        }

        // Check 802.11g can make a link
        // There exist same type of radio and available channel for both sender and receiver.
        for(int i = 0; i < numOfNIC_g; i++){
            if(node[prevID]->nic802g[i].isUsed == 0){
                for(int j = 0; j < numOfNIC_g; j++){
                    if(node[currID]->nic802g[j].isUsed == 0){

                        // Check that there is any available channel for both sender and recei
ver.
                        // The channel ID should be same for both
                        // Before selecting a radio and channel, it should check out that
                        // radio should be differ from the previously used one.
                        // For 802.11b and 802.11g should check out channel also.....

                        // In order to avoid using same NIC at a node
                        if(availablePath[pathNum].nicIDReceiver.size()){
                            if(availablePath[pathNum].nicTypeReceiver.at(availablePath[pathNum
].nicIDReceiver.size()-1) == TYPE_G && availablePath[pathNum].nicIDReceiver.at(availablePath[p
athNum].nicIDReceiver.size()-1) == i){
                                continue;
                            }
                        }

                        channelID = IsChannelAvailable(prevID, currID, i, j, pathNum, TYPE_G);
                        if (channelID == -1){
                            // No more available channel in selected NIC
                            // Cannot establishing a link
                            // However, it should try to find out another combination of NIC
                            // Should not stop searching for other combination
                        }else{
                            bwLink = LinkBandwidth(prevID, currID, TYPE_G);
                            if(bwLink >= traffic[trafficID].size){
                                availablePath[pathNum].nicIDSender.push_back(i);
                                availablePath[pathNum].nicTypeSender.push_back(TYPE_G);
                                availablePath[pathNum].nicIDReceiver.push_back(j);
                                availablePath[pathNum].nicTypeReceiver.push_back(TYPE_G);
                                availablePath[pathNum].channelID.push_back(channelID);
                                availablePath[pathNum].availableBW.push_back(bwLink);
```

```
                                availablePath[pathNum].sumOfNumberOfInterferingNBR = available
Path[pathNum].sumOfNumberOfInterferingNBR + node[prevID]->nbr802g.size();

                                if (bwLink < availablePath[pathNum].minBW){
                                    availablePath[pathNum].minBW = bwLink;
                                }

                                return node[prevID]->nbr802g.size();

                            }else{
                                // Bandwidth is not enough
                                // Therefore, cannot establish a link using such NIC
                                // However, should not stop searching for other combination
                            }
                        }
                    }
                }
            }
        }


        // Check 802.11b can make a link
        // There exist same type of radio and available channel for both sender and receiver.
        for(int i = 0; i < numOfNIC b; i++){
            if(node[prevID]->nic802b[i].isUsed == 0){
                for(int j = 0; j < numOfNIC b; j++){
                    if(node[currID]->nic802b[j].isUsed == 0){

                        // Check that there is any available channel for both sender and recei
ver.
                        // The channel ID should be same for both
                        // Before selecting a radio and channel, it should check out that
                        // radio should be differ from the previously used one.
                        // For 802.11b and 802.11g should check out channel also.....

                        // In order to avoid using same NIC at a node
                        if(availablePath[pathNum].nicIDReceiver.size()){
                            if(availablePath[pathNum].nicTypeReceiver.at(availablePath[pathNum
].nicIDReceiver.size()-1) == TYPE B && availablePath[pathNum].nicIDReceiver.at(availablePath[p
athNum].nicIDReceiver.size()-1) == i){
                                continue;
                            }
                        }

                        channelID = IsChannelAvailable(prevID, currID, i, j, pathNum, TYPE_B);
                        if (channelID == -1){
                            // No more available channel in selected NIC
                            // Cannot establishing a link
                            // However, it should try to find out another combination of NIC
                            // Should not stop searching for other combination
                        }else{
                            bwLink = LinkBandwidth(prevID, currID, TYPE_B);
                            if(bwLink >= traffic[trafficID].size){
                                availablePath[pathNum].nicIDSender.push back(i);
                                availablePath[pathNum].nicTypeSender.push back(TYPE_B);
                                availablePath[pathNum].nicIDReceiver.push back(j);
                                availablePath[pathNum].nicTypeReceiver.push back(TYPE B);
                                availablePath[pathNum].channelID.push back(channelID);
                                availablePath[pathNum].availableBW.push_back(bwLink);

                                availablePath[pathNum].sumOfNumberOfInterferingNBR = available
Path[pathNum].sumOfNumberOfInterferingNBR + node[prevID]->nbr802b.size();

                                if (bwLink < availablePath[pathNum].minBW){
                                    availablePath[pathNum].minBW = bwLink;
                                }

                                return node[prevID]->nbr802b.size();

                            }else{
                                // Bandwidth is not enough
                                // Therefore, cannot establish a link using such NIC
                                // However, should not stop searching for other combination
                            }
                        }
                    }
                }
            }
        }
    }else{
        return checking; // It must be 0 due to the aggregation
    }
```

```
    return checking;
}

// If channel is available for that radio of link, returns channel ID....
int IsChannelAvailable(int prevID, int currID, int nicIDPrev, int nicIDCurr, int pathNum, int
type){

    int skipChannel1 = -1;      // For one hop before
    int skipChannel2 = -1;      // For two hop before

    // if prevents same channel assignment upto two hops distance, in order to avoid hidden te
rminal problem
    // 802.11b and g shares same channel
    if (type == TYPE_B || type == TYPE_G){  // For 802.11b or g
        if (availablePath[pathNum].nicTypeSender.size() == 1){
            if(availablePath[pathNum].nicTypeSender.at(availablePath[pathNum].nicTypeSender.si
ze() - 1) == TYPE_B || availablePath[pathNum].nicTypeSender.at(availablePath[pathNum].nicTypeS
ender.size() - 1) == TYPE_G){
                skipChannel1 = availablePath[pathNum].channelID.at(availablePath[pathNum].chan
nelID.size() - 1);
            }
        }else if (availablePath[pathNum].nicTypeSender.size() >= 2){
            if(availablePath[pathNum].nicTypeSender.at(availablePath[pathNum].nicTypeSender.si
ze() - 1) == TYPE_B || availablePath[pathNum].nicTypeSender.at(availablePath[pathNum].nicTypeS
ender.size() - 1) == TYPE_G){
                skipChannel1 = availablePath[pathNum].channelID.at(availablePath[pathNum].chan
nelID.size() - 1);
            }

            if(availablePath[pathNum].nicTypeSender.at(availablePath[pathNum].nicTypeSender.si
ze() - 2) == TYPE_B || availablePath[pathNum].nicTypeSender.at(availablePath[pathNum].nicTypeS
ender.size() - 2) == TYPE_G){
                skipChannel2 = availablePath[pathNum].channelID.at(availablePath[pathNum].chan
nelID.size() - 2);
            }
        }
    }else{                      // For 802.11a
        if (availablePath[pathNum].nicTypeSender.size() == 1){
            if(availablePath[pathNum].nicTypeSender.at(availablePath[pathNum].nicTypeSender.si
ze() - 1) == TYPE_A){
                skipChannel1 = availablePath[pathNum].channelID.at(availablePath[pathNum].chan
nelID.size() - 1);
            }
        }else if (availablePath[pathNum].nicTypeSender.size() >= 2){
            if(availablePath[pathNum].nicTypeSender.at(availablePath[pathNum].nicTypeSender.si
ze() - 1) == TYPE_A){
                skipChannel1 = availablePath[pathNum].channelID.at(availablePath[pathNum].chan
nelID.size() - 1);
            }

            if(availablePath[pathNum].nicTypeSender.at(availablePath[pathNum].nicTypeSender.si
ze() - 2) == TYPE_A){
                skipChannel2 = availablePath[pathNum].channelID.at(availablePath[pathNum].chan
nelID.size() - 2);
            }
        }
    }

    // find out availalbe channel without duplication upto two hop distance
    if(type == TYPE_B){
        for(int i = 0; i < 3; i++){
            if(i != skipChannel1 && i != skipChannel2 && node[prevID]->nic802b[nicIDPrev].chan
nel[i] == 0 && node[currID]->nic802b[nicIDCurr].channel[i] == 0){
                return i;
            }
        }
    }else if(type == TYPE_G){
        for(int i = 0; i < 3; i++){
            if(i != skipChannel1 && i != skipChannel2 && node[prevID]->nic802g[nicIDPrev].chan
nel[i] == 0 && node[currID]->nic802g[nicIDCurr].channel[i] == 0){
                return i;
            }
        }
    }else{
        for(int i = 0; i < 12; i++){
            if(i != skipChannel1 && i != skipChannel2 &&  node[prevID]->nic802a[nicIDPrev].cha
nnel[i] == 0 && node[currID]->nic802a[nicIDCurr].channel[i] == 0){
                return i;
            }
        }
    }
```

```
    return -1;
}
// Return bandwidth of sender and receiver using radio TYPE B(1), TYPE G(2), TYPE A(3)
// This only can be used for initial link connections, cannot be used for aggregation
double LinkBandwidth(int prevID, int currID, int type){

    double distance;
    double bw;

    if(inOut == 1){     // For OUTDOOR
        if(type == TYPE_B){          //802.11b
            distance = DISTANCE(node[prevID]->x, node[prevID]->y,node[currID]->x, node[currID]
->y);
            bw = double((1024-MAX_BW1)/RADIUS1)*distance + MAX_BW1;

        }else if (type == TYPE_G){  //802.11g
            distance = DISTANCE(node[prevID]->x, node[prevID]->y,node[currID]->x, node[currID]
->y);
            bw = double((1024-MAX_BW2)/RADIUS2)*distance + MAX_BW2;
        }else{                  //802.11a
            distance = DISTANCE(node[prevID]->x, node[prevID]->y,node[currID]->x, node[currID]
->y);
            bw = double((1024-MAX_BW3)/RADIUS3)*distance + MAX_BW3;
        }
    }else{              // For INDOOR
        if(type == TYPE_B){          //802.11b
            distance = DISTANCE(node[prevID]->x, node[prevID]->y,node[currID]->x, node[currID]
->y);
            bw = double((1024-MAX_BW1)/RADIUS3)*distance + MAX_BW1;

        }else if (type == TYPE_G){  //802.11g
            distance = DISTANCE(node[prevID]->x, node[prevID]->y,node[currID]->x, node[currID]
->y);
            bw = double((1024-MAX_BW2)/RADIUS3)*distance + MAX_BW2;
        }else{                  //802.11a
            distance = DISTANCE(node[prevID]->x, node[prevID]->y,node[currID]->x, node[currID]
->y);
            bw = double((1024-MAX_BW3)/RADIUS3)*distance + MAX_BW3;
        }
    }

    return bw;
}


int CheckAggregation(int prevID, int currID, int pathNum, int trafficID){

    int linkID = -1;

    for(int i = 0; i < numOfNIC a; i++){
        if(node[prevID]->nic802a[i].isUsed == 2 && node[prevID]->nic802a[i].recevierID == curr
ID){
            for(int j = 0; j < numOfNIC a; j++){
                if(node[currID]->nic802a[j].isUsed == 3 && node[currID]->nic802a[j].senderID =
= prevID){
                    linkID = FindLink(prevID, currID, i, j, TYPE A);
                    if(link[linkID]->availableBW >= traffic[trafficID].size){
                        availablePath[pathNum].nicIDSender.push back(i);
                        availablePath[pathNum].nicTypeSender.push back(TYPE_A);
                        availablePath[pathNum].nicIDReceiver.push back(j);
                        availablePath[pathNum].nicTypeReceiver.push back(TYPE A);
                        availablePath[pathNum].channelID.push back(link[linkID]->channel1ID);
                        availablePath[pathNum].sumOfNumberOfInterferingNBR = availablePath[pat
hNum].sumOfNumberOfInterferingNBR + 0;

                        if (link[linkID]->availableBW < availablePath[pathNum].minBW){
                            availablePath[pathNum].minBW = link[linkID]->availableBW;
                        }

                        availablePath[pathNum].availableBW.push_back(link[linkID]->availableBW
);
                        return linkID;
                    }
                }
            }
        }
    }

    for(int i = 0; i < numOfNIC g; i++){
        if(node[prevID]->nic802g[i].isUsed == 2 && node[prevID]->nic802g[i].recevierID == curr
ID){
```

```
            for(int j = 0; j < numOfNIC_g; j++){
                if(node[currID]->nic802g[j].isUsed == 3 && node[currID]->nic802g[j].senderID =
= prevID){
                    linkID = FindLink(prevID, currID, i, j, TYPE_G);
                    if(link[linkID]->availableBW >= traffic[trafficID].size){
                        availablePath[pathNum].nicIDSender.push_back(i);
                        availablePath[pathNum].nicTypeSender.push_back(TYPE_G);
                        availablePath[pathNum].nicIDReceiver.push_back(j);
                        availablePath[pathNum].nicTypeReceiver.push_back(TYPE_G);
                        availablePath[pathNum].channelID.push_back(link[linkID]->channel1ID);
                        availablePath[pathNum].sumOfNumberOfInterferingNBR = availablePath[pat
hNum].sumOfNumberOfInterferingNBR + 0;

                        if (link[linkID]->availableBW < availablePath[pathNum].minBW){
                            availablePath[pathNum].minBW = link[linkID]->availableBW;
                        }

                        availablePath[pathNum].availableBW.push_back(link[linkID]->availableBW
);
                        return linkID;
                    }
                }
            }
        }
    }

    for(int i = 0; i < numOfNIC_b; i++){
        if(node[prevID]->nic802b[i].isUsed == 2 && node[prevID]->nic802b[i].recevierID == curr
ID){
            for(int j = 0; j < numOfNIC_b; j++){
                if(node[currID]->nic802b[j].isUsed == 3 && node[currID]->nic802b[j].senderID =
= prevID){
                    linkID = FindLink(prevID, currID, i, j, TYPE_B);
                    if(link[linkID]->availableBW >= traffic[trafficID].size){
                        availablePath[pathNum].nicIDSender.push_back(i);
                        availablePath[pathNum].nicTypeSender.push_back(TYPE_B);
                        availablePath[pathNum].nicIDReceiver.push_back(j);
                        availablePath[pathNum].nicTypeReceiver.push_back(TYPE_B);
                        availablePath[pathNum].channelID.push_back(link[linkID]->channel1ID);
                        availablePath[pathNum].sumOfNumberOfInterferingNBR = availablePath[pat
hNum].sumOfNumberOfInterferingNBR + 0;

                        if (link[linkID]->availableBW < availablePath[pathNum].minBW){
                            availablePath[pathNum].minBW = link[linkID]->availableBW;
                        }

                        availablePath[pathNum].availableBW.push_back(link[linkID]->availableBW
);
                        return linkID;
                    }
                }
            }
        }
    }

    return linkID;
}

// Return LINK ID (it should be matched by sender and receiver w/ their NIC ID)
int FindLink(int prevID, int currID, int nicIDPrev, int nicIDCurr, int type){

    for(int i = 0; i < total number of links; i++){
        if(link[i]->node1ID == prevID && link[i]->node2ID == currID && link[i]->nic1ID == nicI
DPrev && link[i]->nic2ID == nicIDCurr && link[i]->typeOfNIC == type){
            return i;   // if found, return link ID
        }
    }

    return -1; // if not found
}


void PrintOutResult(int i, ofstream & outFile, char * outFileName) {

    if (bestPath[i].bestPathID != -1 && bestPath[i].bestPathID != 0) {
        outFile.open(outFileName,  ios::out | ios::app);
        outFile << "Path ID\t";
        outFile << bestPath[i].bestPathID;
        outFile <<"\t MinBW\t";
        outFile << bestPath[i].minBW ;
        outFile << "\t NumberOfHops\t";
        outFile << bestPath[i].numOfHop;
```

```
        outFile << endl;
        outFile.close();
    }

}
```