Mobile Computing Systems

Ravi Prakash[1]
Computer Science Program
Erik Jonsson School of Engineering and Computer Science
University of Texas at Dallas
Richardson, TX 75083-0688.

[1]Copying, storing, or distributing any part of this manuscript in any form without the written permission of the author is prohibited.

Contents

Chapter 1

Introduction

A mobile computing system consists of a set of mobile and fixed computers [22]. The fixed computers communicate with each other through a fixed wireline network. The mobile computers, henceforth referred to as *Mobile Hosts* (MH), communicate with other nodes in the network in two ways:

1. By plugging into a connection point (called a *telepoint*) of the fixed wireline network for the duration of communication [10].
2. Through wireless links.

There is an ever increasing demand for the ability to perform various computing tasks in environments other than the office and home. The desire to be able to compute and communicate, while on the move, has been the primary motivation for mobile computing. The steady reduction in the size and price of devices, advancements in wireless communication technology, and the development of new protocols and applications tailored for mobility support have all contributed towards making mobile computing a reality.

Mobile computing is not the same as wireless networking, even though there is a significant overlap between the two areas. Quite often the only connection that a mobile computer has to the rest of the world is a wireless link. The presence of a wireless link is a sufficient, but not a necessary, requisite for communication between a mobile computer and other computers. Consider a user who carries a laptop computer wherever he goes. From a hotel room the user connects his laptop to his home computer using a dial-up line, transfers some files back and forth, modifies some files, and disconnects. Then the user reconnects after some time from the airport, once again using a dial-up line. This process may repeat several times without ever using a wireless link for connection. Therefore, it is important to keep in mind that some mobile computing systems may never use wireless links.

This brings us to another interesting aspect of mobile computing, namely, *disconnected operation*. What to do if computer A sends a message to computer B when B is not connected to the network? Also, what if A and B concurrently make changes to different replicas of a data item while being mutually unreachable? Such scenarios are likely to occur in mobile computing systems due to a variety of reasons.

For example, consider two people who download the same calendar from a central server onto their mobile computers in the morning and then disconnect from the network. This calendar lists the times during which various conference rooms have been reserved for meetings. User A, noticing that conference room 1 is available on Monday from 10 a.m. until 1 p.m., reserves the room for a meeting from 10 a.m. to 11 a.m. User B, unaware of the reservation request made by A, also finds the conference room to be available between 10 a.m. and 1 p.m. and reserves it from 10:30 a.m. until 11:30 a.m. Each user's reservation request is consistent with his copy of the calendar and is not denied. However, there is obviously a conflict between the two requests. This conflict will not be detected until both users try to synchronize their copies of the calendar with the copy in the central server. Such a situation needs to be addressed when multiple replicas of data, each unreachable from the others, and independently modifiable are allowed to exist.

1.1 System Model

In order to appreciate the issues involved in mobile computing systems it is important that we first understand the configuration of these systems.

We shall consider three models:

1. the cellular network model,
2. the virtual cellular network model, and
3. the ad-hoc network model.

1.1.1 Cellular Network Model

A cellular network covers a certain area that is divided into regions called cells. Neighboring cells usually overlap. Each cell has a fixed base station (BS). The base stations are connected to each other by a wireline network. Several mobile nodes may be present in a cell. Mobile nodes (also referred to as mobile hosts or MHs) can move from one cell to another. An MH always communicates with other nodes in the system through the base station of the cell in which it is present. In order to do so, the MH needs to establish a wireless link with its base station. If the communication partner is also present in the same cell, the base station forwards the message to the partner along another wireless link. If the partner is present in another cell, the base station forwards the message along the wireline backbone to the base station of the partner's cell. The backbone network is connected to the telephone network, and perhaps to other networks.

This architecture is shown in Figure 1.1.

1.1.2 Virtual Cellular Network Model

A virtual cellular network (VCN) is similar to the cellular network, except for one major difference: the base stations of a VCN are also mobile. Therefore, there is no

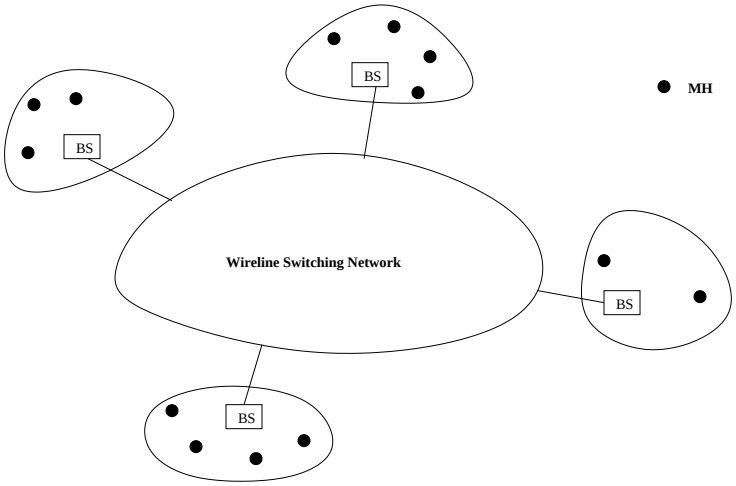


Figure 1.1: A wireless communication network.

wireline backbone. The inter-base station links are also wireless. The mobile base stations continue to coordinate the activities of the MHs in their vicinity. However, unlike cellular networks, the graph of mobile base stations changes with time. As a result, several solutions that work for cellular networks cease to perform correctly. VCNs may be useful for tactical networks where mobile base stations may be installed on tanks, trucks, etc. Individual soldiers would carry the mobile hosts.

1.1.3 Ad-hoc Network Model

In an ad-hoc network all nodes are alike and all are mobile. There are no base stations to coordinate the activities of subsets of nodes. Therefore, all the nodes have to collectively make decisions. Due to mobility, a node's neighborhood changes with time. As the mobility of nodes may not be predictable, changes in network topology over time are arbitrary. All communication is over wireless links. Ad-hoc networks are suitable for tactical missions, emergency response operations, electronic classroom networks, etc.

1.1.4 Network Connectivity

Connectivity has different meanings for each of the network models described above. In a cellular network we assume that the wireline backbone network never gets disconnected and all base stations stay operational. In such a scenario, as long as every MH is operational and is located in some cell, they can reach each other in the network. From a graph theoretic point of view, the base stations are the internal nodes of a graph and the MHs are the leaves. Over time, the only changes that the graph undergoes are the leaf node to internal node interconnections. The subgraph comprising the internal

nodes remains unchanged.

In a virtual cellular network model we assume that the mobile base stations are always reachable from each other along paths that consist exclusively of mobile base stations.[1] Each MH is connected to some mobile base station. However, the subgraph consisting of internal nodes (mobile base stations) changes with time.

In an ad-hoc network a mobile node has a link with another mobile node if they are within wireless range of each other. We assume that a single or multiple wireless hop path exists between every pair of mobile nodes. However, these paths may change over time.

1.1.5 Node Mobility

In a cellular network an MH can enter a cell from any direction. The duration that an MH stays in a cell depends on the path taken by the MH inside the cell, its speed, and the size of the cell. Let us consider an urban area. A mobile node carried by a pedestrian can go almost anywhere in the cell, but the speed is low. On the other hand, a mobile node onboard a car can move at much higher speeds. However, it is restricted to travel on the roads. Given the path along which an MH is moving, it is easy to determine the neighboring cell that the MH will enter after exiting its current cell. This is because the relative positions of cells remains unchanged.

However, in the VCN model, due to the mobility of MBSs the set of neighboring cells changes with time. Hence, the sequence of cells visited by an MH is dependent on the mobility profile of that MH as well as the mobility profile of the MBSs.

In the cellular and VCN models a communicating MH is coupled with the MBS of the cell in which it is present. So, in several situations it is acceptable to model the movement of an MH as the following two-step cycle:

1. When an MH enters an MBS's cell, the MH stays in that cell for a period of time determined by a probability distribution.

2. At the end of this period the MH enters an arbitrarily selected neighboring cell. Note that due to the mobility of MBSs the set of neighboring cells changes with time.

Two realistic scenarios where such a mobility model would be applicable for virtual cellular networks are the battlefield scenario and the public transport scenario.

In a battlefield, soldiers (equivalent to MHs) may move along with a tank (equivalent to an MBS) for a while before moving towards and connecting with a neighboring tank. This switch from one tank's cell to another would continue for the system lifetime. In a civilian scenario, passengers may be traveling on a bus with several passengers carrying their MHs. A bus may have an antenna to communicate with each MH making the inside of the bus into a cell. All the busses, acting as MBSs, may communicate with their neighbors and route messages across their wireless backbone network. From time to time, passengers get off a bus at bus-stops and board another bus, thus moving out of one cell and into another neighboring cell.

[1] A solution to coordinate movement of mobile base stations while maintaining connectivity can be found in [49].

1.1.6 Event Ordering

A mobile computation may be distributed over multiple nodes. These nodes do not have a global clock, nor do they have any shared memory. The only way these nodes can communicate with each other is through messages. Thus a mobile computation has all characteristics of a distributed system.

Several mobile applications require an order to be established between events on different nodes. All the events can be categorized as one of the following: (i) message send event, (ii) message receive event, and (iii) all other events which are also referred to as internal events. Each event is associated with the node at which it occurs.

A *happened before* relation, represented as $\rightarrow$, between events in the computation can be expressed as follows:
Let A and B be two events. $A \rightarrow B$ if:

1. A and B occurred at the same node with A occurring before B, or
2. A is a message send event and B is the message receive event for that message, or
3. $A \rightarrow C$ and $C \rightarrow B$.

In the absence of a global clock it is not possible to express the *happened before* relation between events using the values of the local clocks of their nodes at the time of their occurrence.

Lamport [34] proposed a logical clock system to express the *happened before* relation. Each process has a scalar clock, whose value is initialized to zero. The clock value associated with event a is represented as C_a. Each message m carries with it C_m, the value of the sending process at the time of sending the message. This clock value is referred to as the message timestamp.

The logical clock of a process is updated according to the following rules:

1. Each time an internal event or a send event occurs at a process its clock is incremented by a positive value v. Any value can selected for v. For simplicity, let us assume that $v = 1$.
2. When message m arrives at process P_i, the clock value of P_i associated with the message receive event is equal to $max(C_i, C_m) + v$, where C_i was the value of P_i's clock before receiving the message.

Therefore, if $a \rightarrow b$ then $C_a < C_b$. However, the converse is not true. Mattern [37] and Fidge [21] independently proposed a vector clock system that satisfied the following relation: $a \rightarrow b \Leftrightarrow C_a < C_b$. In an n process system the vector clock is a vector of n integers, one per process.[2] The vector clock of process P_i is represented as VC_i and is initialized to n zeroes. Vector clocks are incremented according to the following rules:

1. If an event occurs on process P_i, $VC_i[i]$ is incremented by v. If this event happens to be a message send, the updated vector is associated with the message.

[2]The n processes are indexed from 0 to $n - 1$.

2. When message m with vector timestamp C_m arrives at process P_j, C_j is updated as follows:
   (a) $C_j[j]$ is incremented by v.
   (b) for all $0 \leq k < n$, $C_j[k] = max(C_j[k], C_m[k]$

Various mobile applications may have their messages timestamped with either their scalar or vector clock.

1.2 Operational Constraints

Mobile computing systems impose certain operational constraints hitherto unseen in a network of static computers.

1. *Limited energy supply:* Mobile hosts have a limited energy source in the form of a battery pack. Advances in energy storage technology have not kept pace with advances in CPU speed and memory density and speed. Wireless communication drains the energy of the mobile hosts. Hence, energy should be conserved at a mobile host by keeping its involvement in various computations to a minimum. This can be achieved by minimizing the number of messages it has to exchange with its mobile service station.

2. *Bandwidth limitations:* The frequency spectrum allocated for wireless communication is not growing at the same rate as the demand. Hence, it is important to develop strategies to maximize the number of supportable wireless communication sessions given the finite number of channels. Such strategies should be scalable as mobile computing systems are growing rapidly. Moreover, wireless channels used by MHs have a significantly lower data rate than the wireline communication links between MSSs. Several communication protocols and applications software that are suitable for distributed systems using wireline networks have unacceptable performance in a mobile computing system due to the low data rate. Hence, communication overheads should be kept as small as possible. This will help in minimizing the communication delays over the low bandwidth channels. Also, there is a need to develop alternative communication protocols suited to mobile environments [6].

3. *Memory limitations:* Generally mobile hosts have a much smaller main memory than static hosts. A mobile host may not have a hard disk, or if it does, the capacity of such a disk may be limited. Hence, it is important that the data structures required to support various applications have as small storage overhead as possible.

4. *Disconnected operation:* A mobile host, participating in a distributed computation, may be temporarily unreachable from other nodes in the network. This may occur due to voluntary disconnection of the mobile host from the network for a period of time, or due to involuntary disconnections when the mobile host is in a region where it cannot communicate. The distributed applications executing in the system should be able to handle such periods of disconnection.

5. *Low CPU speed:* Mobile hosts typically have slower CPUs than static hosts. So, they may not be suitable for computationally intensive tasks. There is a need for migrating computationally intensive tasks to the static nodes or mobile service stations as they have faster CPUs.

6. *Inferior user interface:* Portability concerns mitigate against mobile hosts having I/O devices of the size, speed, and resolution available with static hosts. Mobile applications have to be sensitive to these limitations.

1.3 Overview

In the following chapters we will discuss various aspects of mobile computing. Let us briefly consider each of them.

1.3.1 Channel Allocation

The frequency spectrum allocated for wireless communication is not growing at the same rate as the demand. This spectrum is divided into a finite number of wireless channels, or time slots for Frequency Division Multiplexed Access (FDMA) and Time Division Multiplexed Access (TDMA) schemes, respectively. In Code Division Multiplexed Access (CDMA) different modulation codes are used. For convenience, we will use the term *channel* to imply time slots for TDMA schemes, frequency bands for FDMA, or modulation code or CDMA.

In a cellular network, when a mobile host wishes to establish a communication session it sends a request to the MBS of the cell in which it is present. The communication session can be supported if a wireless channel can be allocated for communication between the mobile host and the mobile base station. If a particular wireless channel is used concurrently by more than one communication session in a cell, or in neighboring cells, the sessions will interfere with each other. Such an interference is called *co-channel interference*. However, the same wireless channel can be used to support concurrent communication sessions in geographically separated cells such that their signals do not interfere with each other. This is known as *frequency reuse* and the corresponding cells are called *co-channel cells*. The set of neighboring cells that are in the co-channel interference range of each other form a *cluster*. A 7-cell cluster for frequency reuse is shown in Figure 1.2. A cell and its six neighbors, each with a different label, belong to a cluster.

At any time, a channel can be used to support at most one communication session in a cluster. Due to the limited number of channels, and an ever growing demand for them, efficient allocation of wireless channels to support communication sessions is of vital importance.

A mobile host may move from one cell to a neighboring cell while participating in a communication session. In such an eventuality, the communication link between the MH and the MSS of the old cell is broken, the corresponding wireless channel released, and a new communication link needs to be established between the MH and the MSS of the new cell by allocating a new wireless channel. Such a transfer of

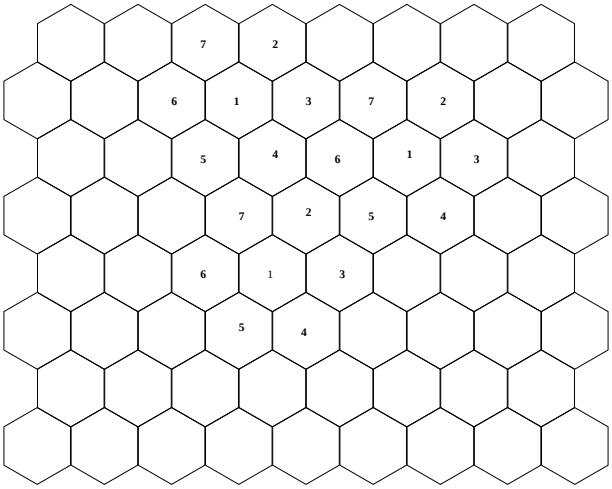


Figure 1.2: A 7-cell cluster (co-channel cells have identical numeric labels)

responsibility for a communication session, from one MSS to another, is referred to as *hand-off*.

An efficient channel allocation strategy should exploit the principle of frequency reuse to increase the availability of wireless channels to support concurrent communication sessions. An efficient channel allocation algorithm should have the following properties:

1. minimize connection set-up time.
2. maximize the number of communication sessions that can be supported simultaneously.
3. ability to adapt to changing load distribution
4. minimize computation and communication overheads for channel selection.

1.3.2 Location Management

A node that wishes to communicate with a mobile host in the network first needs to locate the target MH. However, the ability of MHs to autonomously move from one part of the network to another part sets mobile networks apart from static networks with regard to management of location information. Creating a fixed location directory of all the nodes *a priori* is not a solution. The location directory has to be dynamically managed to account for the mobility of the MHs.

The need for a dynamic location directory raises important issues. Some of them are as follows:

1. In a cellular network the location of an MH can be specified with reference to the mobile base station in whose cell it is present. However, in virtual cellular networks the geographical area covered by a cell changes with time due to MBS mobility. In ad-hoc networks there is no notion of cells. So, what is meant by location of an MH in virtual cellular networks and ad-hoc networks?

2. When should the location directory be updated? If the updates are done each time an MH's location changes, the directory will always have the latest location information. This reduces the time and effort in locating an MH. However, such a policy imposes a heavy burden on the communication network and the location servers, i.e., nodes that maintain the directory.

3. Should the location directory be maintained at a centralized site, or should it be distributed? A central location server has problems with regard to robustness and scalability. Hence, a distributed directory server is preferable. This leads us to the following questions.

4. How should the location information be distributed among the location servers?

5. Should location information about an MH be replicated across multiple location servers?

It is not possible to *a priori* determine the variations in spatial distribution of MHs in the network, and the frequency with which node location will be updated or queried. Hence, a location management strategy should address issues (4) and (5) so as to ensure fair distribution of responsibility among all the location servers, and be scalable.

1.3.3 Routing in Mobile Networks

Once the source of a message has determined the destination's location, the message has to be routed to the destination. Several routing schemes for fixed wireline networks cannot be directly applied to the mobile environment. This is due to the fact that node mobility results in frequent removal of existing links and establishment of new links. Hence, the connectivity graph can change significantly over short periods of time. Routing algorithms for wireline networks do take link disruptions into account. However, they make an implicit assumption that such incidences are rare. When these algorithms are executed in a mobile environment their performance may not be acceptable.

A routing algorithm has to ensure that no message is lost. Some applications may require that there is no duplication of messages, $i.e.$, every message is delivered exactly once to the destination. Also, messages may have to be delivered to the destination in the order they were sent.

The suitability of a given routing algorithm should be evaluated in the context of the specific network model being considered: cellular, virtual cellular or ad-hoc.

In a cellular network the wireline backbone connecting the base stations changes very rarely. The task of routing a message from one mobile host to another mobile host can be mapped onto the problem of routing the message from the source mobile host's

base station to the destination mobile host's base station. For inter base station routing existing wireline routing algorithms may be considered. However, the destination mobile host is capable of moving from one cell to another. So, during a communication session the destination base station may change several times. Assume a stream of messages sent from a source to the destination mobile host which is in the process of moving from one cell to another neighboring cell. There may be a node that lies on the paths from the source to the old and the new base stations. It will take some time for this upstream node to become aware of the destination's movement and the resultant path change. During this time the upstream node is likely to keep routing the messages towards the old base station. One must ensure that these messages are not discarded when the reach the old base station. Either the old base station, or some node upstream from it should ensure that the messages get routed to the new base station.

In a virtual cellular network the mobile base stations may construct a wireless backbone to route messages between mobile hosts. Once again, inter-MH routing can be modeled as routing between the mobile base stations of the source and the destination MHs using the wireless backbone. However, the wireless backbone between the mobile base stations itself changes with time.

In ad-hoc networks there are no base stations to shoulder the responsibility of routing. Every node can be a source and destination for messages, as well as a router for messages between other node pairs. Two possible approaches can be taken for message routing: (i) the *flat* approach, or (ii) the *hierarchical* approach. In the flat approach each source node would be responsible for discovering a route to the destination of its messages. In the hierarchical approach the mobile nodes may *elect* some nodes from amongst themselves, and form a virtual backbone among the elected nodes. The virtual backbone may be used for routing messages, as in the cellular and the virtual cellular models. Alternatively, the backbone may only store routing information between node pairs. A source node can query the nearest backbone node to retrieve the route to a destination. Remember that with the passage of time the distribution of nodes may change. So, the set of nodes forming the backbone as well as the configuration of the backbone may change with time. Some important issues to be addressed are: (i) What are the criteria for the selection of nodes belonging to the backbone? (ii) How to propagate information about the nearest backbone node to other nodes? (iii) When some old backbone nodes are replaced by new backbone nodes how to efficiently disseminate the routing tables from the former to the latter?

1.3.4 Mobile IP

Each computer has its own IP address that is used by the network layer protocol to route messages to it. The IP address identifies the network class, network number and host number. Routers in the network maintain tables containing routing information. When a packet arrives at a router a table look-up is performed using the destination address in the packet to determine the link along which the packet should be forwarded.

In a mobile environment a node may move from one network to another. So, how do we route packets meant for that node? Assigning a different IP address to the node each time it enters a new network is not an acceptable solution because propagating this new address to all the routers, potentially all over the world, is too expensive and

time consuming.

Mobile IP was developed as a solution to this problem. Each node is associated with a *home agent* which resides in the node's home network. Every network that is willing to let foreign nodes connect to it has to have a *foreign agent*. A foreign agent advertises its presence to the visiting nodes at regular intervals.

When a mobile node enters the foreign network it hears the foreign agent's advertisement and registers with that agent. The foreign agent then conveys information about the presence of the mobile node to that node's home agent. All packets intended for the mobile node are first routed to its home network. When the packet reaches a router in the home network the router tries to find the Ethernet address of the mobile node. If the mobile node is present in its home network it responds with its Ethernet address, and the router delivers the packet to it. If the mobile node is not in its home network its home agent responds on its behalf. So, the packet gets routed to the home agent which subsequently encapsulates it in another IP packet and sends to the foreign agent. The foreign agent then delivers the packet to the mobile node.

There are various security and efficiency related issues involved in Mobile IP. How do we ensure that an unauthorized node does not masquerade as a foreign agent and forces the home agent to route packets for some mobile nodes to it? What do we do about mobile nodes that do not inform the home agent before leaving the network? What if a mobile node is rapidly moving among foreign networks with encapsulated packets chasing it from one foreign agent to another?

1.3.5 Transport Layer Protocols for Mobile Systems

Several applications require reliable end-to-end delivery of packets. TCP is a protocol that provides such reliable data communication. Several implementations of TCP have been optimized for wireline networks. The links in such networks are very reliable. Delays in the delivery of packets and packet losses are usually due to congestion in the network and hardly due to link failure/degradation. Hence, the source node in a TCP connection starts a count-down timer as soon as it sends a packet. If an acknowledgment from the destination does not arrive before a timeout the source goes into a congestion control mode, reduces its transmission window size, and increases its timeout period. The source also retransmits the unacknowledged packets. The congestion control scheme works quite well for wireline networks.

However, in mobile networks which frequently use wireless links congestion is not the only cause for packet losses and delays in their delivery. In a cellular environment, when a mobile host is being handed-off from one base station to another, communication may be temporarily suspended. This may lead to delays in the receipt of acknowledgment causing the source to erroneously enter into a congestion control mode. Moreover, the single wireless hop between a base station and a mobile node is the real communication bottleneck as it has a much lower bandwidth than the wireline links and is also prone to errors. Therefore, packet losses and/or delays along a single wireless link may trigger retransmissions and congestion control.

What we need are strategies to shield the wireline portion of the source–destination path from the performance limitations of the wireless link(s). Several solutions have

been proposed for this purpose. These solutions can be classified into the following categories:

1. *Split connection approach:* The TCP connection between a fixed node and a mobile node is split into two connections: a fully wireline TCP connection between the source and the base station of the mobile node, and a single wireless hop connection between the base station and the mobile node.

2. *Fast-retransmit approach:* Often the delays associated with retransmission of packets lost during handoffs are due to the coarse granularity of the TCP timer (about 250 ms) in most implementations. This problem can be alleviated by employing fast-retransmit. During handoffs some packets may be lost. Subsequent packets are received by the destination node through the new base station. Thus, there is a break in the sequence of packets received. The mobile node keeps sending the same acknowledgment containing the identity of the next packet in the sequence. When the source receives multiple identical acknowledgments it immediately retransmits the first unacknowledged packets rather than waiting until the timer to expire.

3. *Link-level retransmissions:* A link-level retransmission protocol with forward error correction is implemented for the wireless link. The retransmission timer at the link-level has a finer granularity than the transport level timer. Hence, multiple transmission attempts along the wireless link are made before the transport level timer expires. With multiple attempts the chances of packet loss over the wireless link, and TCP's entrance into the congestion control mode are considerably reduced.

1.3.6 Network Aware Computing

So far we have discussed three network models. We have also enumerated some of the resource constraints under which a mobile computing system has to operate. It is time to ask ourselves the following question: *Do mobile nodes always operate under the same environment with the same restrictions on resource availability?*

The answer to the question is *NO*. A mobile node's environment can change drastically over time. Let us consider the following scenario: A pedestrian is carrying a mobile node with a wireless link to a base station. As the node moves at a slow speed it is possible for the base station to track it very precisely and provide a high bandwidth downlink from the base station to the mobile node. However, the mobile node has a limited power supply. So, its uplink bandwidth and range is rather low. Also, the node has a small and possibly monochrome display. After a while the user gets into a car, docks the mobile node into a port in the car and starts driving along a highway. Now, it is much more difficult for the base stations to accurately track the mobile node. So, the downlink bandwidth may be diminished. However, the mobile node may now draw energy from the car's battery and be capable of sending more powerful signals. Also, the mobile node may have access to additional secondary memory and a better I/O interface in the car.

In the scenario described above, the mobile node traveling at highway speed may try to compensate for the reduction in downlink bandwidth by employing efficient disk caching strategies. Moreover, while the mobile node is restricted to a monochrome display it may inform the servers from where it is downloading images about the restriction. The servers can then employ suitable data filters and rather than sending color images only send monochrome images. This will result in significant savings in bandwidth without any perceived deterioration in quality at the mobile node.

Let a mobile node operating in a connected mode try to modify a shared data item. Tests can be immediately performed to determine whether the changes can be committed or have to be aborted. On the other hand, when the same mobile node operating in a disconnected mode tries to make the same change to the same data item it is possible that the modification can only be *tentatively* committed at the time. Final decisions about committing the change to the master copy of the data item may have to be deferred until the time that the node reconnects and synchronizes its state.

Therefore, it is important that a mobile node be aware of its surroundings and adapt to them for optimal performance. Adaptation to changing network environment may require negotiations between nodes for dynamic resource allocation and quality of service (QoS) guarantees. Extensive research and development of *mobile agents* is being performed towards this goal.

Chapter 2

Channel Allocation

Mobile computing systems that use wireless channels for communication are faced with the availability of a limited number of wireless communication channels. With increasing popularity of mobile computation and communication systems the demand for these channels is also increasing. Therefore, efficient utilization of channels is of paramount importance.

In this chapter we will concentrate on channel allocation algorithms for cellular networks as the problem is fairly well understood and several solutions exist. In later sections we will discuss issues in channel allocation for virtual cellular networks.

2.1 Definitions

Two nodes, for example a cellular base station and a mobile host can communicate using a wireless channel provided the *signal-to-noise ratio (SNR)* is above a certain threshold value, $ASNR_{min}$. Let the base station send messages to the mobile host on channel c. Other pairs of nodes in the system may also be using the same channel for communication. Signals for those sessions may also propagate to the mobile host causing interference. Such interference between two communication sessions using the same channel is referred to as *co-channel interference*. Moreover, there is some environmental noise that can interfere with the incoming signals from the base station. So, SNR at the mobile host is defined as follows:

$$SNR = \frac{strength\ of\ signals\ received\ from\ base\ station}{\sum(strength\ of\ signals\ from\ other\ sessions\ using\ same\ channel) + environmental\ noise}$$

The farther a receiver is from the source of a signal, the weaker is the received signal. If S_t is the strength of the transmitted signal, and d is the distance between the transmitter and the receiver, then S_r, the strength of the received signal, can be expressed as:

$$S_r = S_t \times d^{-\alpha}$$

where α can vary in the range $3 - 5$. This signal fading characteristic has two implications:

1. The mobile host and the base station have to be within a certain distance of each other to be able to communicate. Otherwise, the strength of the received signal will be very low. Even if the SNR is low, a very weak signal from the intended source is not acceptable as the receiver would require a large antenna to pick it up. Therefore, the size of a cell is dependent on the power level of transmissions.

2. Due to the rapid decline in signal strength with distance, co-channel interference between two sessions using the same channel in widely separated cells is negligible and can be ignored.

Let a channel be used to support a communication session in a cell. The same channel cannot be concurrently used for another session within a certain distance d' of that cell. This distance is the *co-channel interference range* of the cell.

A cell together with all the cells within its co-channel interference range forms a *mutual interference set* or a *cluster* of cells. For interference free communication a channel should be used to support only one session at a time in a cluster.

Figure 2.1 shows a *7-cell cluster*. All cells marked with the same number are referred to as *co-channel cells*. Co-channel cells can use the same channel concurrently without interfering with each other's communication. The minimum distance between two co-channel cells is referred to as the *reuse distance*. The lower the reuse distance, the more sessions a channel can support concurrently in a system.

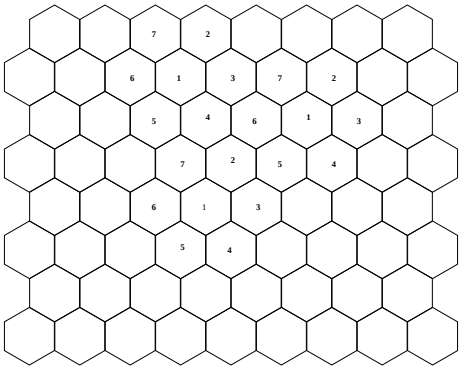


Figure 2.1: A 7-cell cluster (co-channel cells have identical numeric labels)

So far we have only considered co-channel interference. There is yet another type of interference, the *adjacent channel interference*. As the name implies, adjacent channel interference occurs between two communication sessions if they are using adjacent channels. In the context of TDMA systems two consecutive time slots would be referred to as adjacent channels. Adjacent channel interference can result in crosstalk or disruptions in communication sessions. For the rest of the chapter we will only consider co-channel interference.

Channel allocation needs to be performed under two circumstances: (i) at the beginning of a new communication session, (ii) during *handoff* of the communication session. *Handoff* means switching an ongoing communication session from one channel

to another. There are two types of handoffs: *inter-cell handoff* and *intra-cell handoff*. Inter-cell handoff takes place when a mobile host is moving out of one cell and into a neighboring cell. It's link with the old base station is broken and a new link is established with the new base station. Intra-cell handoff needs to be performed when the SNR value of a link goes below the acceptable threshold, possibly due to interference on the channel being used. The communication session is switched to another channel.

The *break-before-make* handoffs, in which the existing link is broken before a new link is established, are also referred to as *hard handoffs*. On the other hand, in CDMA systems when the mobile host is near the boundary of its cell and its signal can be received by multiple neighboring base stations, the base station controller evaluates the quality of the signal received from different base stations and picks the best signal. There is no frequency switch and as the mobile host moves into the new cell the base station controller favors the signals being received by the new base station, and ignores the signals received by the old base station. This is referred to as *soft handoff*.

As most channel allocation algorithms were developed in the context of cellular telephony, they use the term *call* instead of communication session. However, both the terms can be used interchangeably. A call is said to be *dropped* if a channel cannot be allocated for it. In the following discussion we will sometimes use the term *load* to denote the demand for channels in a cell. High load will imply a high demand for channels in a cell to support newly arriving calls or handoffs.

2.2 Classification of Algorithms

Existing channel allocation algorithms can be classified into three categories on the basis of channel partitioning or the lack thereof:

1. *Fixed Channel Allocation (FCA) algorithms:* In FCA algorithms each cell is assigned a fixed set of *nominal* channels. The sets of nominal channels assigned to two cells within co-channel interference range are disjoint. Such algorithms are unable to adapt to situations when there is a non-uniform distribution of channel demand across cells. Their lack of flexibility can lead to several dropped calls.

   Let us consider a very highly loaded cell surrounded by lightly loaded cells. All the nominal channels in the heavily loaded cell are allocated. However, the lightly loaded cells have some idle channels. If a new call arrives in the heavily loaded cell no channel can be allocated to it.

2. *Dynamic Channel Allocation (DCA) algorithms:* In DCA algorithms all the channels are available to all the cells and are allocated on a need basis. This flexibility comes with added complexity: communicating nodes need to gather information about the current channel usage within the co-channel interference range so as to avoid interference. In situations of high load some DCA algorithms can experience a performance degradation.

   Let us illustrate the situation with the following example. There are three cells A, B and C such that both A and C are within B's co-channel interference range.

However, A and C are sufficiently far apart so that they can use the same channel concurrently. Let us assume that only two channels, 1 and 2, are available. What would happen if channel 1 was being used in cell A, channel 2 in cell C and a call arrived in cell B? Obviously, the call would have to be dropped. Had both A and C been using channel 1, cell B could have used channel 2 without causing any interference.

3. *Hybrid Channel Allocation (HCA) algorithms:* In HCA algorithms the spectrum is divided into a *fixed set* and a *dynamic set*. Channels in the fixed set are divided into subsets of nominal channels associated with each cell and are allocated using FCA algorithms. Channels in the dynamic set are available to all cells and are allocated using DCA algorithms when the nominal channels of a cell are all in use.

   HCA algorithms are a compromise between the FCA and DCA algorithms. Their goal is to avoid the complexity of DCA algorithms as much as possible and also to alleviate the performance degradation experienced by DCA algorithms at high loads. At the same time they try to avoid call drops experienced by FCA algorithms in situations of non-uniform load distribution.

Channel allocation algorithms can also be classified on the basis of who makes the channel selections:

1. *Centralized algorithms:* In such algorithms all the channel allocation decisions are made by a central *Mobile Telecommunications Switching Office (MTSO).* The MTSO has information about channel usage in all the cells. All channel requests arriving in the cells are forwarded to the MTSO. Similarly, whenever a communication session ends and a channel is freed, the MTSO is informed about it.

2. *Distributed algorithms:* Here the control is distributed among multiple nodes in the system. For example, each base station can make decisions about channel allocation for calls in its cell. For this purpose, the base station may have to communicate with base stations of neighboring cells within co-channel interference range to gather their channel usage information. In some distributed algorithms the mobile hosts may also be involved. They may monitor the strength of signals received along various channels and select the one that does not cause interference.

2.3 Fixed Channel Allocation

The simplest fixed channel allocation scheme assigns an equal number of channels to every cell such that no two cells in a mutual interference set get the same set of channels. This is referred to as the *uniform channel allocation* scheme. However, channel demand is not uniformly distributed. There may be cells with high channel demand right next to cells with low channel demand. In such a situation calls will be blocked in some cells even though there idle channels in neighboring cells.

This led to the development of *non-uniform channel allocation* schemes. Zhang and Yum [55] have proposed one such scheme. They argue that allocation of nominal channels to cells should be based on the actual traffic distribution in the system. Cells with high channel demand are allocated a larger number of nominal channels than cells with lower channel demand. This leads to an increase in the number of calls that can be supported concurrently, and a reduction in the percentage of dropped calls.

Anderson [3] proposed a *fixed borrowing* scheme to handle the same non-uniform distribution of load. Every possible mutual interference set in the system is analyzed, and the set that requires the most number of channels is identified. Channel assignments are made to meet the needs of cells in this mutual interference set. The same pattern is repeated for the rest of the system based on the minimum reuse distance. As a result there are some cells that need more channels than are allocated to them. There are other cells that have more channels than they need. These extra channels can be reassigned to cells at distances that are greater than the minimum reuse distance. This *fixed borrowing*, which is performed right in the beginning, reduces the number of channels that are required to satisfy the channel demands of the system.

The lack of flexibility and the resultant performance degradation of FCA schemes was quickly recognized by several people. So, various channel borrowing schemes were suggested. These borrowing schemes are different from Anderson's fixed borrowing scheme. They operate on the basis of *borrow-use-return* [19]. Let a new channel request arrive at a heavily loaded cell when all its nominal channels are already being used to support ongoing calls. The heavily loaded cell borrows an idle channel from a lightly loaded neighbor provided the borrowing does not lead to interference. This borrowed channel is used to support the call. Once the call is over, the channel is returned to the cell from which it was borrowed.

2.3.1 Channel Locking

Note that a cell that has lent one of its nominal channels to a neighbor cannot use that channel until it is returned. During this period some of the co-channel cells of the lender cannot use that channel either. For an explanation refer to Figure 2.1. Let the cell marked with the highlighted label "1" borrow a channel from the cell marked "7" to its left. While "1" is using the channel the two nearby co-channel cells of the lender, labeled with the highlighted "7", cannot use that channel as they are within co-channel interference range of "1". This phenomenon is referred to as *channel locking*. It is as if a dummy call is in progress in these two cells for the duration the channel is borrowed by "1".

However, channel utilization can be increased by a smarter locking strategy referred to as *directional locking*. Once again, consider Figure 2.1. Even though the cell marked with the highlighted "7" to the right and above cannot use the channel, it can lend the channel to its neighbors "5" or "1" without causing any interference. Similarly, the highlighted cell marked "7" to the lower right can lend the channel to one of the following three neighbors: "1", "3" or "2" without causing interference.

Channel borrowing strategies lead to a reduction in the probability of calls being blocked in low to moderate system load situations. However, their performance degrades under high system load situations. Can you guess why?

Well, here is a simple explanation:

- When the system load is low calls get blocked when they arrive in a heavily loaded cell surrounded by lightly loaded neighbors. Borrowing an idle channel from a lightly loaded neighbor enables the highly loaded cell to avoid the blocking.
- In Figure 2.1, when "1" borrows a channel from its neighbor to the left to support one call, this borrowing reduces the number of calls that can be supported in each of the highlighted "7" cells by one. The lender cannot use the channel either. So, to support one call in "1" the potential to support three concurrent calls in cells marked "7" is being sacrificed: a net loss of two. In high load situations there is a good likelihood that these cells that experience a reduction in their capacity end up blocking some calls. There is a lower likelihood of such a blocking when the overall channel demand in the system is low.

2.3.2 Channel Reallocation

Yet another optimization for the channel borrowing strategies is *channel reallocation*. It requires that all the nominal channels be ordered *a priori*. Zhang and Yum [54] describe channel reallocation as follows:

1. Let a call using a nominal channel terminate in a cell. If there is another call in progress in the same cell using a higher order nominal channel then that call is switched to the newly released lower order nominal channel.
2. Let a call using a nominal channel terminate. If there is another call in the same cell using a borrowed channel that call is switched to the newly released nominal channel.
3. Let there be two calls in progress in a cell using two borrowed channels. If the call using the higher order borrowed channel terminates the other call is switched to it.
4. When a nominal channel is unblocked in all directions due to the termination of call(s) in neighboring cells, any call using a borrowed channel or a higher order nominal channel is switched to the newly unblocked channel.

Channel allocation favors using nominal channels over borrowed channels. This leads to fewer channel lockings in high load situation resulting in higher channel utilization and fewer blocked calls.

2.3.3 Zhang and Yum's Algorithm

Now we describe a channel allocation algorithm that incorporates the optimizations mentioned above. It is a centralized algorithm in which the control rests with the MTSO. The MTSO maintains the following information about each cell:

FC: ordered list of free nominal channels.

SC: ordered list of nominal channels being used to support local calls.

LC: ordered list of channels being locked, their locking directions and the cells responsible for their locking.

When a channel allocation request arrives at the base station of cell P it is forwarded to the MTSO. The MTSO performs the following actions:

1. If $FC(P)$ is not vacant, the first channel in $FC(P)$ is assigned to the call. The channel is moved from $FC(P)$ to $SC(P)$.
2. If all the nominal channels are busy, the MTSO searches the neighbors of P for all free channels as well as locked channels that are not locked in P's direction. If no such channel is found the call is blocked.
3. Otherwise, of the channels found in the previous step select those that are either: (a) free in their two nearby co-channel cells, or (b) being locked but the minimum distance between cell P and the locking cells is greater than the interference range. If no such channel can be found the call is blocked.
4. If such channel(s) are found, the MTSO selects the one with the highest order from the cell with the maximum number of free channels. This channel is assigned to the call.
5. The nearby three co-channel cells lock the assigned channel in the appropriate directions. This channel is also moved from the FC list to the LC list of the three cells. The identity of P is also associated with these entries in the LC lists.

When a call using a certain channel terminates in cell P first channel reallocation is performed. Then, the following actions are performed:

1. If the released channel is not a borrowed channel it is moved from $SC(P)$ to $FC(P)$.
2. If the released channel was borrowed from a neighbor Q, the locks on this channel from the LC lists of Q and its two nearby co-channel cells Q' and Q'' are removed. Once Q gets the channel it performs channel reallocation in its cell. If the released channel is completely unlocked in all directions the corresponding channel reallocation steps are performed.

2.4 Dynamic Channel Allocation

As stated earlier, in DCA algorithms all channels are usually available to all cells. There is no notion of nominal channels associated with cells. When a channel is to be allocated to a call in a cell the cost of using each available channel is determined. The channel with the least cost is allocated.

Some candidate cost functions [52] are:

- future blocking probability in the vicinity of the cell,
- average blocking probability of the system,
- the reuse distance of channels: shorter the reuse distance, greater the utilization of channels.

2.4.1 Channel Segregation: Akaiwa and Andoh's Algorithm

Channel segregation is a dynamic channel allocation algorithm proposed for FDMA and TDMA systems [2]. It is assumed that each cell can access only a subset of channels in the entire spectrum. It is argued that making the entire spectrum accessible to each cell is not necessary from the viewpoint of channel utilization. Each cell prioritizes channels through experience about how various channels were used by that cell and its neighbors. Channels with higher priority are favored over those with lower priority.

The following actions are performed when a channel allocation request arrives in a cell:

1. The priority associated with a channel is expressed as the following fraction:

$$Priority = \frac{N_s}{N_t}$$

   N_s is the number of successful uses of the channel added to the number of accesses to the idle unaccessible channel. N_t is the total number of trials for the channel.

2. The base station of the cell selects a channel with the highest priority from the channels that are not being used in the cell.

3. The received signal level of the selected channel is measured. If the level is above or below a threshold value the channel is determined to be busy or idle, respectively.

4. If the channel is found to be busy the channel's priority is decreased by incrementing the value of its N_t by one, and the channel with the next highest priority is tried.

5. If the channel is found to be idle the channel is checked to determine if it is an unaccessible channel. If the channel is determined to be unaccessible its priority is increased by increasing both N_s and N_t by one. Also, the next highest priority channel is tried.

6. If the channel is determined to be an accessible channel it is used for the call and its priority is increased.

Channel segregation is a dynamic, distributed channel allocation algorithm. It adapts to changes in distribution of channel demand.

2.4.2 Prakash, Shivaratri and Singhal's Algorithm

This is also a distributed dynamic channel allocation algorithm that requires neighboring base stations to communicate with each other along the wireline backbone [45]. It employs the principles of distributed mutual exclusion. Highly loaded cells borrow channels from their lightly loaded neighbors to support calls. However, at the end of the call the borrowed channel is not returned. This enables each cell to acquire its preferred set of channels. A cell first tries to allocate channels from its preferred set. Only when such allocations are not possible it tries to transfer channel(s) from its neighbors.

Before we actually study the algorithm, let us first discuss the theoretical basis and the main ideas of this algorithm.

Channel Allocation vs. Mutual Exclusion: In the context of a cell and its neighbors, the use of a particular channel to support a communication session is equivalent to a critical section execution by the cell in which the channel is being used. Several neighboring cells may be concurrently trying to choose channels to support sessions in their region. This can lead to conflicts because the number of communication channels is limited. The resolution of such conflicts is similar to the mutual exclusion problem [11, 50].

However, the channel allocation problem is more general than the mutual exclusion problem. Firstly, a cell may be supporting multiple communication sessions, from different mobile hosts, in its region, each session using a different communication channel. This is equivalent to a cell being in multiple, distinct critical sections concurrently. Secondly, existing mutual exclusion algorithms for distributed systems [11, 34, 36, 47, 50] assume that a node specifies the identity of the resource it wants to access in a critical section. Depending on the availability of that resource, appropriate decisions can be made. However, in distributed channel allocation, a cell asks for *any* channel as long as there is no co-channel interference. Due to the non-specificity of the request and because neighboring mobile service stations make channel allocation decisions independently based on locally available information, the decision process becomes more difficult.

Moreover, existing distributed mutual exclusion algorithms do not impose any upper bound on the time from the instant a node issues a request for the resource to the instant the node is granted that resource. These algorithms are not suitable for the channel allocation problem that requires the decisions to be made quickly, in real-time. So, a conservative approach that makes the channel allocation decisions quickly needs to be adopted. Such an approach may drop calls/communication requests that a more general but time consuming approach would have supported. This is a trade-off that has to be accepted.

Basic Idea: Requests timestamped with Lamport's clock [34] are sent by a base station to neighboring base stations to determine the channel to be assigned for a communication session. Sometimes a channel needs to be deleted from a cell's set of allocated channels and transferred to another cell's set of allocated channels to support communication sessions in the latter. The distributed nature of the algorithm, and the finite but non-deterministic propagation delays of messages between base stations can lead to

co-channel interference if a naive channel transfer strategy is employed: multiple cells in each other's interference range may concurrently and independently decide to transfer the same channel from a mutually adjacent cell. Such a possibility is prevented as follows: having selected a communication channel for transfer, based on a round of message exchange with its neighbors, the mobile service station sends the channel identity to the neighboring mobile service stations. Only if all the neighboring mobile service stations approve of the selection is the channel transferred, otherwise not.

The set of channels allocated to a cell varies with time. A newly acquired channel is not relinquished by a cell on completion of the communication session it was supporting in the cell. Instead, the channel remains allocated to the same cell until it has to be transferred to a neighboring cell. This enables the algorithm to adapt to temporal and spatial changes in load distribution. It also helps reduce the traffic due to channel allocation requests in the fixed wire network.

Data Structures: All the communication channels in the system are collectively represented by a set $Spectrum$. All the channels are ordered. In an FDMA system the channel with the lowest frequency band is considered to be the first channel and the channel with the highest frequency band is the n^{th} channel, where n is the total number of channels available.

The set of channels allocated to cell C_i is represented by $Allocate_i$. Initially, $Allocate_i$ is an empty set for every cell C_i. A subset of $Allocate_i$, known as $Busy_i$, represents the set of channels being used by C_i to support communication sessions at a particular instant of time. When a new communication request originates in C_i, one of the non-busy channels in $Allocate_i$ is assigned to support the communication session. If there is no such channel, then after a round of message exchange with the neighbors, a channel that is in the $Spectrum$, but not in the *Allocate* set of the cell or any of its neighbors is added to $Allocate_i$ as well as $Busy_i$. This channel is used to support the session. If such an attempt fails, C_i tries to transfer a non-busy channel from the *Allocate* set of its neighbors to $Allocate_i$. If such a transfer is not possible, the communication request is dropped. Otherwise, the communication is successfully completed. The set $Transfer_i$ at C_i consists of the channels earmarked for transfer from C_i to one of its neighbors. *Transfer* sets are initially empty at all the cells. T_i is the clock value maintained at cell C_i as per Lamport's logical clock [34]. Initially, T_i is zero at every cell. RT_i is the timestamp of the current channel request. If RT_i is equal to zero, then cell C_i is not requesting a channel. Initially RT_i is equal to zero. All these data structures are maintained by the corresponding mobile service stations.

Several new communication requests may originate in a cell concurrently. These new requests, originating in the same cell, may be ordered according to a policy decided *a priori*. Only after the mobile service station has made a channel allocation decision about one locally originating request, does it process the next locally originating communication request in the sequence.

The channel allocation algorithm works as follows:
(A) When a communication session is to be set-up in cell C_i, the following actions are taken by its base station (MSS_i):

1. $T_i \leftarrow T_i + 1$;

2. $RT_i \leftarrow T_i$ /* RT_i has the timestamp of this channel request. */

3. If $Available_i \leftarrow Allocate_i - Busy_i - Transfer_i \neq \Phi$, then

   A highest order channel k from $Available_i$ is selected to set-up the session;
   $Busy_i \leftarrow Busy_i \cup \{k\}$;
   Go to step 10;

   else /* $Available_i = \Phi$ */

   Send timestamped REQUEST messages to the BS of each neighbor C_j.

4. When C_i's BS has received REPLY messages from each of its neighbors, containing their *Allocate, Busy and Transfer* sets, it takes the union of $Allocate_i$ and the $Allocate$ sets received in the REPLY messages, and stores the result in $Interfere_i$.

5. If $Free_i \leftarrow Spectrum - Interfere_i \neq \Phi$, then a channel of the highest order is selected from $Free_i$ and added to $Allocate_i$. This channel is used to support the communication session. So, it is added to $Busy_i$ as well. Then go to step 10.

6. If $Free_i = \Phi$, it does not mean that no channel is available for allocation. Perhaps, the communication session can be supported by transferring a channel. C_i's BS takes the union of $Busy_i$, $Transfer_i$, and $Busy$ and $Transfer$ sets received in the REPLY messages in step 4, and stores the result in $Interfere_i$.

7. If $Free_i \leftarrow Spectrum - Interfere_i = \Phi$, then the communication request is dropped. Otherwise, the channel of the lowest order in $Free_i$ is chosen for the transfer.

8. Let the channel selected for transfer be k.

   (a) $Busy_i \leftarrow Busy_i \cup \{k\}$;

   (b) $Allocate_i \leftarrow Allocate_i \cup \{k\}$;

   (c) C_i's BS sends TRANSFER(k) messages to all the neighbors whose $Allocate$ sets have k as a member and waits for replies. Let S denote the set of these neighbors.

9. If all the cells in S reply AGREED:

   (a) Channel k is used to support the communication session.

   (b) C_i's BS sends RELEASE(k) messages to the BSs of all the cells in S.

   (c) Go to Step 10.

   Otherwise: /* Some cells have sent REFUSE message. */

(a) $Allocate_i \leftarrow Allocate_i - \{k\}$;

(b) $Busy_i \leftarrow Busy_i - \{k\}$;

(c) C_i's BS sends KEEP(k) messages to the BSs of all the cells in S.

(d) C_i's BS selects the next channel from $Free_i$, with order greater than that of k, and steps 8 and 9 are repeated. [1] To avoid excessive channel transfer overheads under heavy load situations, the number of transfer attempts can be limited to the minimum of a THRESHOLD value (parameter of the algorithm) and the cardinality of $Free_i$. If all attempts to transfer a channel fail, the communication request is dropped.

10. Once a cell has decided to drop a request or to use a channel to support the corresponding communication session:

    (a) it sends all the deferred REPLYs to its neighbors.

    (b) $RT_i \leftarrow 0$;

11. When a communication session terminates in C_i, the corresponding channel is deleted from the set $Busy_i$.

(B) When a cell C_j's BS receives a REQUEST message from C_i's BS with timestamp T_i:

1. $T_j \leftarrow T_j + 1$;

2. $T_j \leftarrow \max(T_j, T_i + 1)$;

3. C_j's BS sends a REPLY message to C_i if C_j is not requesting a channel ($i.e.$, $RT_j = 0$), or if C_j is requesting a channel and C_i's request's timestamp is smaller than C_j's request's timestamp (i.e., $T_i < RT_j$ or $T_i = RT_j$ and $i < j$). Otherwise, the REPLY is deferred. As C_i only uses the union of the $Busy_j$ and $Transfer_j$ sets received in the REPLYs, in Step $(A).6$, and never uses the two sets separately, the communication overheads can be reduced by taking their union at C_j and sending the result, rather than both the sets, in the REPLY message. Therefore, the REPLY message contains $Allocate_j$, and the union of $Busy_j$, and $Transfer_j$.

(C) When a cell C_j's BS receives TRANSFER(k) message from C_i:

If ($k \in Busy_j$) OR ($k \in Transfer_j$) then send REFUSE(k) message to C_i. Otherwise $Transfer_j \leftarrow Transfer_j \cup \{k\}$; Send AGREED($k$) message to C_i.

(D) When C_j's BS receives a RELEASE(k) message, the following actions take place.

1. $Allocate_j \leftarrow Allocate_j - \{k\}$;

2. $Transfer_j \leftarrow Transfer_j - \{k\}$;

[1] The KEEP messages can be piggybacked on TRANSFER messages, if they are going to the same cell.

(E) When C_j's BS receives KEEP(k) message, the following actions take place.

$Transfer_j \leftarrow Transfer_j - \{k\}$;

Lemma 1 *The channel allocation algorithm ensures that neighboring cells do not use the same channel concurrently.*

Proof: Let Nbr_i denote the set of neighboring cells of C_i such that concurrent use of a channel in C_i and a cell in Nbr_i will lead to co-channel interference. We have to prove the following assertion: $Busy_i \cap Busy_j = \Phi, \forall C_j \in Nbr_i$.
Initially, the assertion is trivially true as the sets are empty. Also, $Busy_i \subseteq Allocate_i$ under all circumstances. $Busy_i$ can change under three situations:

1. *In step (A).3, when* $Available_i \neq \Phi$: Let cell C_i select channel k (an element of $Allocate_i$) to support a new communication session. Assuming $Busy_i \cap Busy_j = \Phi$ and $Allocate_i \cap Allocate_j = \Phi$ prior to the addition of k to $Busy_i$, $(Busy_i \cup \{k\}) \cap Busy_j = \Phi$. So, the assertion holds after k is selected to support a call in cell C_i.

2. $Available_i = \Phi$ *in step (A).3 and* $Free_i \neq \Phi$ *in step (A).5*: Channel $k \in Spectrum$ - $(Allocate_i \bigcup_{C_j \in Nbr_i} Allocate_j)$ is added to $Busy_i$ and $Allocate_i$. The assertion is proved by contradiction. Let us assume that cell C_i, and its neighbor C_j, are using channel k concurrently. Cell C_j does not transfer channel k to its neighbor C_i as long as $k \in Busy_j$. This implies that the co-channel interference mentioned above can arise only if the $Allocate$ sets in the REPLYs received by the base stations from each other in step (A).4 did not contain k. Based on the pattern of REQUEST and REPLY messages exchanged between the two nodes, the following three situations arise:

   (a) C_i sends a REPLY to C_j before sending its own REQUEST. So, C_i's REQUEST has a higher timestamp than C_j's REQUEST. When C_j receives this REQUEST, it defers the REPLY until it has decided to use k. Then C_j sends its $Allocate$ set, containing k, in the REPLY to C_i. So, C_i cannot select channel k.

   (b) C_j sends a REPLY to C_i before sending its own REQUEST. This is the similar to the previous case. So, C_i selects channel k, while C_j does not.

   (c) Both C_i and C_j receive each other's REQUEST after sending their own REQUESTs. Both the cells compare their own channel request timestamp with that received in the REQUEST message from the other. As the timestamps are fully ordered by the Lamport's clock system, the cell whose request happens to have the lower timestamp among the two requests, will defer its REPLY until it has made its own decision. The other cell will send a REPLY. Let C_i be the cell that deferred the REPLY. If C_i decides to use k, then C_j receives this information (Allocate set) in the REPLY it receives from C_i. So, C_j will not use channel k.

Thus, two neighboring cells will not be allocated the same channel concurrently.

3. $Free_i \neq \Phi$ *in step (A).7 and AGREED messages received from all cells in S in step (A).9*: Channel $k \in Spectrum - (Busy_i \bigcup Transfer_i \bigcup_{C_j \in Nbr_i} Busy_j \bigcup_{C_j \in Nbr_i} Transfer_j)$ is added to $Allocate_i$ and $Busy_i$. TRANSFER(k) is sent to the neighbors. If any neighboring cell is using k, it sends a REFUSE message. So, channel k is not used in cell C_i. From steps (C), (D), and (E) it can be inferred that in response to a TRANSFER(k), a cell C_j sends AGREED to at most one neighbor at any time. All other TRANSFER(k) messages received by C_j, after k is added to $Transfer_j$ and before RELEASE(k) or KEEP(k) are received, are responded to with a REFUSE message. Therefore, two neighboring cells cannot simultaneously acquire channel k as a result of a transfer attempt.

∎

Lemma 2 *The channel allocation algorithm is deadlock free.*

Proof: New channel requests originating concurrently in different cells get totally ordered by their timestamps. A base station with REPLYs pending to its own REQUESTs, sends REPLYs to all REQUESTs with a lower timestamp and defers other REPLYs. As the same ordering of channel requests is seen by all the nodes, there is no circular deferring of REPLYs among the base stations.

During the interval between sending a TRANSFER(k) message to the neighbors, and receiving either a REFUSE or an AGREED message from each neighbor, a cell does not suspend replying to TRANSFER(k) messages it may itself receive from the neighbors. Instead, it responds to such transfer attempts with a REFUSE message during this interval. This conservative policy may lead to some requests, that could have otherwise been supported, being dropped. However, it avoids any circular wait during the channel transfer attempts, thus preventing deadlocks. ∎

Discussion: The channel transfer feature of the algorithm ensures that unused channels are moved from lightly loaded cells to the heavily loaded cells. Therefore, most of the channel requests that originate in heavily loaded cells can be satisfied locally by selecting a free channel from the *Available* sets. Moreover, if a mobile host, containing frequently accessed data, moves from a cell to a neighboring cell, channels are transferred from the *Allocate* set of the former to the latter, over a period of time. Thus the size of the *Allocate* sets of cells can adapt with time to support the locality of data reference. The algorithm can adapt to non-uniform load distribution.

The distributed nature of this algorithms makes the cellular network scalable. Channel allocation decisions are made by each base station locally. All the messages needed to set up a communication session in a cell are restricted to that cell and its immediate neighbors. So, the traffic on the wired network between adjacent base stations does not increase with increasing number of cells; it only increases with increasing load in the cell and its neighbors. For the centralized algorithms, the traffic on the communication paths leading to the central network switch increases with increasing network size. So,

with the centralized algorithm, as the network expands, existing links will have to be replaced with those with a higher bandwidth.

The use of logical clocks to timestamp channel requests ensures fairness. If cell C_i's channel request causally precedes a neighboring cell C_j's channel request, C_i's request is processed before C_j's request.

Simulation of the algorithm shows that the percentage of direct connections declines with increasing channel request load. This reduction occurs because as load increases, the probability of finding an idle channel in the cell decreases. However, even for high channel load, a majority of the channel requests lead to direct connections without any message exchange in a system with a small $Spectrum$. Also, for the same channel request load, as the number of channels in the $Spectrum$ increases, the percentage of channel requests that lead to direct connections also increases.

It is to be noted that the size of the messages in the algorithm increases with the number of communication channels in the $Spectrum$. However, the number of messages needed per channel request decreases with increase in the number of communication channels in the $Spectrum$, for the same load. As a result, traffic in the wireline network per channel request shows little change regardless of the total number of channels in the $Spectrum$.

With load remaining the same, an increase in the number of BSs will not change the number of inter-BS messages per channel request, and the traffic on each inter-BS link will remain the same. This is because an BS only needs to communicate with its neighbors, regardless of the total number of cells in the system. On the other hand, in a centralized channel allocation scheme, the network traffic on the link(s) incident on the $MTSO$ increases linearly with an increase in the number of cells. Thus, the links incident on the $MTSO$, and the $MTSO$ itself, can become bottlenecks. The non-dependence of link traffic on network size, in the proposed algorithm, makes the algorithm scalable.

2.5 Handoffs

As described earlier, an inter-cell handoff may be required if a mobile host moves from one cell to another while communicating. The wireless link between the mobile host and the base station of the old cell has to be broken. At the same time a new wireless link has to be established between the mobile host and the base station of the new cell. The call in progress may be disrupted if no channel is available in the new cell to support the call. Such a disruption is referred to as handoff failure.

A disruption of an ongoing call usually causes more annoyance to the user than the initial blocking of a call. Hence, it is important that priority be assigned to channel requests for handoff. Two priority based handoff strategies have been proposed in [23]. The two strategies are:

1. *Channel Reservation:* a certain number of channels (H) are reserved exclusively for hand-offs. Any available channel in the new cell can be used to support handed-off communication sessions. However, if a channel is needed for a new

communication session arising in the cell and the number of available channels is less than H, the channel request is denied.

2. *Handoff Queue:* channel requests for handoffs are queued up on an FCFS basis in the target cell. If a channel becomes available in the target cell, and its handoff queue is non-empty, the channel is used to satisfy the handoff request at the head of the queue. An upper bound is imposed on the length of the queue and the duration for which a handoff request can be queued up. This is to ensure that handoffs, if possible, are done fast enough so that the disruption in the communication session during hand-off is not noticeable.

Simulation results in [23] show that both the strategies reduce the handoff failure probability significantly. However, the first strategy leads to a corresponding increase in the probability of new channel requests being blocked. Hence, the handoff queuing strategy appears to be the better of the two strategies.

2.6 Channel Allocation in Virtual Cellular Networks

In a virtual cellular network the fixed base stations are replaced by mobile base stations (MBSs). The wireline links between the fixed BSs are replaced by wireless links between MBSs. So, the entire network is wireless. The inter-MBS links will, henceforth, be referred to as *backbone links* while the MBS-MH links will be referred to as *short-hop links*. Figure 2.2 presents a logical view of the virtual cellular network.

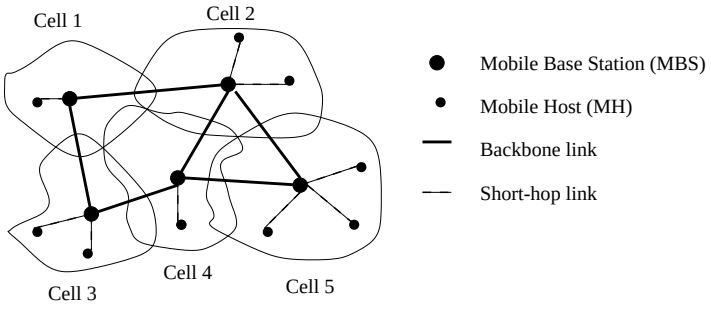


Figure 2.2: A fully wireless cellular network.

The relative position of cells changes with time. As the relative positions of MBSs cannot be determined *a priori*, it does not make sense to pre-allocate fixed sets of channels to each MBS for short-hop links. Dynamic channel allocation is needed. Also, in the interest of scalability and robustness a central controller should be replaced by a distributed channel allocation mechanism. Remember that now we have to two different types of wireless channel allocations: channel allocation for backbone links and channel allocations for short-hop links.

Two different approaches can be taken for channel allocation. In the first approach, the set of wireless channels can be partitioned into two disjoint subsets: one subset used

exclusively for backbone links and another subset used exclusively for short-hop links. Channel partitioning simplifies the task of channel allocation at the cost of utilization. In the second approach all channels can be used for backbone as well as short-hop links.

An MH communicates only through a neighboring MBS using a wireless short-hop link. An MH-MBS pair can establish a bidirectional short-hop link provided their separation is less than a threshold value d. This is equivalent to an MBS having a cell of radius d. If a wireless channel is being used to support a short-hop link between an MH and an MBS, the same channel cannot be concurrently used to support any kind of communication within a radius $\alpha \times d$ around the involved MBS, where $\alpha > 1$. Thus, the short-hop channel reuse distance is $\alpha \times d$.

Two MBSs can establish a bidirectional wireless link provided their separation is no more than another threshold value D. If two MBSs are using a channel for a backbone link, that channel cannot be simultaneously used to support any other communication in a region consisting of the union of two circles, each of radii $\beta \times D$ centered at the respective MBSs. Thus, $\beta \times D$ is the backbone channel reuse distance.

We assume that $D > d$. This is consistent with the earlier assumption that MBSs have abundant energy supply enabling them to transmit at greater power levels than MHs. The variables D, d, α and β are system parameters that depend on the networking hardware, power level of transmissions, fading characteristics, etc.

In order to allocate channels for short-hop links a dynamic channel allocation similar to the ones described earlier can be employed. However, remember that due to the mobility of MBSs, the short-hop mutual interference set changes with time. So, for each channel allocation a given MBS may have to communicate with different sets of MBSs.

For backbone channel allocation both the MBSs participating in the backbone link should co-operate. Each MBS should gather information about channel usage in its *backbone interference range*. The channel selected to establish the backbone link should not be in use to support another link in the backbone interference range of either of the MBSs. You must have realized by now that the backbone interference range is much greater than the short-hop interference range. So, backbone channel allocations can be more expensive than short-hop channel allocations.

2.6.1 Channel Reconfiguration During Connection Lifetime

There are two situations in which a backbone or a short-hop link between two nodes may have to switch from one channel to another:

1. If an MH moves out of one cell into another while involved in a communication session, inter-cell handoff has to take place. The short-hop link between the MH and the old MBS has to be terminated and a new short-hop link has to be established between the MH and the new MBS.

2. There is a possibility that a short-hop link between an MBS-MH pair or a backbone link between a pairs of MBSs may have to switch to a different channel during the lifetime of the corresponding session. This is similar to intra-cell

handoff as the pair of communicating nodes does not change. However, there are some differences between this situation and intra-cell handoff. The switch from one channel to another is necessitated primarily by the mobility of MBSs as described below.

Let there be a short-hop link between MBS_i and MH_i using a short-hop channel i. Concurrently, let there be another short-hop link in the network between MBS_j and MH_j also using channel i. Let the initial separation between MBS_i and MBS_j be greater than $\alpha \times d$, the short-hop co-channel interference range. At a later time, while the two short-hop sessions are still in progress, let one or both of MBS_i and MBS_j start moving towards the other. Let the MHs move with the MBS to which they are connected. When the separation between MBS_i and MBS_j becomes smaller than $\alpha \times d$ the two short-hop links MBS_i-MH_i and MBS_j-MH_j using the same channel start interfering with each other. At least one of these links has to switch to another short-hop channel to avoid any further interference. Without loss of generality, let us assume that the MBS_i-MH_i link has to switch to another short-hop channel. The procedure followed is equivalent to the termination of the old short-hop session between the node pair immediately followed by a new short-hop channel allocation between them.

Similarly, let us consider two backbone links: between mobile base station pairs MBS_i-MBS_j and MBS_k-MBS_l. Both MBS_i and MBS_j are more than $\beta \times D$ distance away from MBS_k and MBS_l. So, the two mobile base station pairs are not within backbone co-channel interference range of each other. Both backbone links are using the same backbone channel without interfering with each other. Subsequently, if these nodes move so as to be within the backbone co-channel interference range of each other, at least one link will have to switch to another channel.

The responsibility for channel reconfiguration lies with the mobile base stations. When the hitherto far apart MBS_i and MBS_j detect that they have moved within distance $\beta \times D$ of each other, they exchange information about their backbone channel usage and make the appropriate channel reconfiguration decisions. When MBS_i and MBS_j detect that they have moved to within $\alpha \times d$ of each other, they exchange information about their short-hop channel usage and determine if any short-hop links need to be switched to other channels. We assume that the underlying MAC sub-layer and network layer protocols for node beacons along with timestamps and location stamps enable MBSs to determine their distance from each other.

Chapter 3

IEEE 802.11 and IEEE 802.15.4 Protocols for Channel Access

In the previous chapter we discussed protocols that enable base stations to reserve channels in cellular networks. In this chapter we will focus on an entirely different kind of network, namely the *Wireless LAN (WLAN)*, and the protocol employed by nodes to access the wireless channel in such a network.

Unlike voice communications that tend to be long-lived (possibly several minutes in duration), data communication between computers takes place in short bursts. Hence, the overheads of channel allocation may be difficult to justify for such short-lived communications. A low-overhead channel-access protocol is needed. This protocol should be (i) able to adapt to changes in channel demand, and (ii) offer some *Quality of Service (QoS)* guarantees to applications that require bounded-delay delivery of data being exchanged between communicating entities. The IEEE 802.11 protocol is one such protocol that has been widely deployed. The IEEE 802.11 WLAN specification standardizes physical layer (PHY) and medium access control (MAC) sublayer implementations. We will focus on the MAC sublayer issues.

The basic building block of the IEEE 802.11 WLAN is referred to as the *Basic Service Set (BSS)*. It is a set of nodes within communication range of each other that follow a common set of rules to access a shared wireless channel. An *Independent BSS (IBSS)* is formed without any prior planning and is also referred to as an *ad hoc network*. Nodes in an IBSS are able to directly communicate with each other. An *Infrastructure BSS* has one of the participating nodes acting as an *Access Point (AP)* that can coordinate the activities of other nodes in the BSS.

Multiple Infrastructure BSSs can together form a *Distribution System (DS)*. The APs of the Infrastructure BSSs can communicate with each other using the *Distribution System Medium (DSM)*. A node in one BSS can communicate with a node in another BSS via their respective APs and the DSM. Note that the wireless medium used by a BSS and the DSM are two separate entities. The standard does not place any restriction on the nature of the DSM. For example, the DS may be an Ethernet or a wireless network of APs. Moreover, an IEEE 802.11 WLAN may be connected to

non-IEEE 802.11 networks, say an Ethernet, through a *portal*. The portal enables an IEEE 802.11 network to be integrated with a non-IEEE 802.11 network. Figure 3.1 shows the various components of the IEEE 802.11 WLAN. As you can see, multiple BSSs may overlap, i.e., occupy the same coverage area.

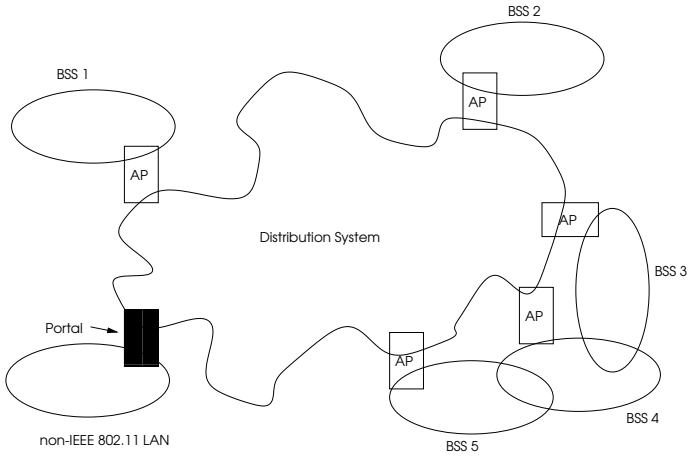


Figure 3.1: An IEEE 802.11 Distribution System

3.1 IEEE 802.11 Services

The IEEE 802.11 architecture does not describe how the distribution system should be implemented. Instead, it only lists the services that the architecture should provide. The nine services are:

1. **Distribution**: This is the service to send a frame from one node to another through the distribution system. The frames have to be forwarded from the source node to its AP, then through the distribution system to the AP of the destination BSS, and finally to the destination node.

2. **Integration**: This service is similar to the distribution service, except for one important difference: the recipient of the frame belongs to a non-IEEE 802.11 network and can be reached via the portal.

3. **Association**: This service maintains a mapping between a node in a BSS and its corresponding AP. This mapping is used by the distribution system to forward frames towards their destination BSS. A node must establish an association with an AP before it can send data into the distribution system through that AP.

4. **Reassociation**: When a node moves out of one BSS and into another BSS this service changes its association from the AP of the old BSS to the AP of the new BSS. Thus a node to AP mapping is dynamic. This service is also invoked when a node changes the attributes of the association it has with its AP.

5. **Disassociation**: This service is invoked to terminate an association.

6. **Authentication**: This service allows a node to authenticate itself to other nodes before it can start communicating with those nodes. A node cannot establish an association with an AP before it has established its identity with that AP. Authentication can be time consuming. This may delay the reassociation of a node as it moves out of own BSS into another. To reduce such delays a node can choose to preauthenticate itself with other APs after it has already established an association with the AP of its BSS.

7. **Deauthentication**: This is the service used to terminate an existing authentication. If a node is deauthenticated with respect to an AP then the corresponding association is also broken.

8. **Privacy**: This service provides the ability to encrypt messages sent between communicating nodes.

9. **MAC Service Data Unit (MSDU) delivery**: As its name implies, this is the service to deliver frames between nodes of a BSS, or between a node and its AP.

To ensure MSDU delivery the standardized MAC sublayer specification can operate in either *Distributed Coordination Function (DCF)* or *Point Coordination Function (PCF)* mode. DCF is the fundamental channel access scheme to support distributed data transfer. It is based on the *Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA)* channel access mechanism and employs binary exponential backoff mechanism. Nodes contend for the medium for each data frame transmitted. In PCF mode, an Access Point controls access to the medium and is referred to as the *Point Coordinator (PC)* in this context.

3.2 Contention-Free and Contention-Based Transmission

In an Infrastructure BSS the operation of the BSS is divided into roughly equal periods of time referred to as the *Contention-Free Repetition Interval (CFRI)*, as shown in Figure 3.2. The AP determines the duration of the CFRI. Each CFRI period starts with a beacon transmitted by the AP, followed by a contention-free period and then a contention period. The beacon contains the remaining duration of the contention-free period. During the contention-free period nodes operate in the PCF mode. In the contention period nodes operate in the DCF mode. The maximum duration of the contention-free period should be such that at least one data frame can be sent during the intervening contention period. AS DCF is the fundamental channel access scheme, it is possible that nodes in a BSS may only operate in the DCF mode, and never employ the PCF mode. This is definitely the case in an independent BSS with no AP to act as a Point Coordinator.

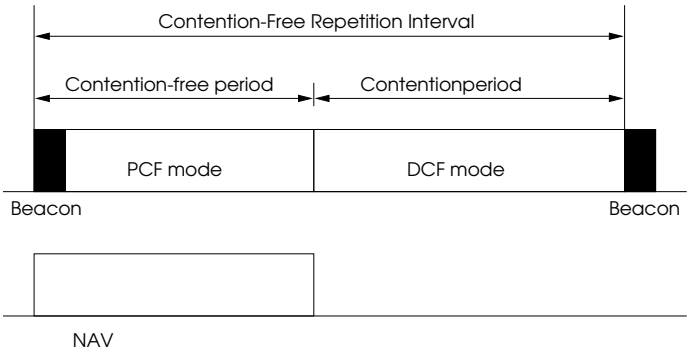


Figure 3.2: Contention-Free and Contention-Based Transmission Periods Alternate

3.3 Carrier Sensing

Using a combination of *physical* and *virtual carrier sensing* a node determines when the channel is available and when it is busy. Each node maintains a *Network Allocation Vector (NAV)* to reflect its knowledge of channel availability based on virtual carrier sensing.

The MAC sub-layer performs virtual carrier sensing. Several IEEE 802.11 frames carry information about future usage of the channel by nodes. This information is used to update the NAV. For example, when a node receives a beacon announcing the remaining duration of the contention-free period, the node updates its NAV to indicate that the channel will be busy during this period. Consequently, the node does not try to transmit during this period unless explicitly asked by the PC to do so. In the following sections we will see more examples of how virtual carrier sensing is performed to update the NAV.

The physical layer provides support for physical carrier sensing. When the transceiver senses activity on the channel it indicates that the channel is busy. This information, coupled with the NAV, tells the node when it should not try to transmit on the channel. At all other times the channel is assumed to be idle and the node can choose to transmit. However, more than one node may independently decide to transmit at the same time resulting in collisions.

The problem of collisions has been extensively studied in the context of wired LANs. A variety of protocols based on *Carrier Sense Multiple Access with Collision Detection (CSMA/CD)* have been proposed for such LANs. Hence, the following question: *Can CSMA/CD-based protocols be employed for wireless LANs?* The answer, unfortunately, is that CSMA/CD protocols can fail in WLANs. This is due to the *hidden terminal problem* in WLANs.

3.4 Hidden and Exposed Terminal Problems

In CSMA/CD-based protocols a transmitter monitors the channel for collision following its transmission. This is because all transmissions are heard by all nodes on the

LAN. However, such is not the case for wireless networks. For example, consider the situation shown in Figure 3.3. The circles with A, B and C as their centers represent the transmission ranges of A, B and C, respectively. Node A wishes to send a frame

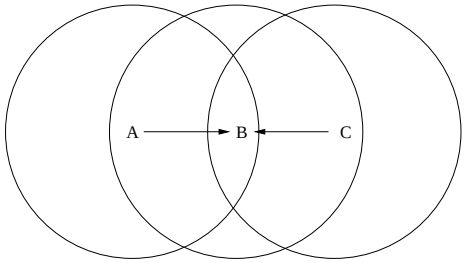


Figure 3.3: The hidden terminal problem.

to node B which is within transmission range of both A and C. Both A and C decide to transmit their frames concurrently. While A is transmitting its frame to B, C is also transmitting a frame to some other node. As a consequence there is collision at B due to which B is unable to receive A's transmission. As A and C are beyond each others communication range, neither detects the collision. A erroneously concludes that it has successfully transmitted its frame to B. In effect, the impact of C is hidden from A. So, to indicate successful reception of A's transmission, B should send an acknowledgment to A. The absence of an acknowledgment should indicate a collision.

While the hidden terminal problem could lead nodes to erroneously conclude successful transmissions, the *exposed terminal problem* has exactly the opposite consequence. It prevents nodes from transmitting when they could safely do so, without fear of collision at the intended recipient(s). Consider, for example, the situation shown in Figure 3.4 Node B is transmitting a frame to node A. Concurrently, node C wishes to

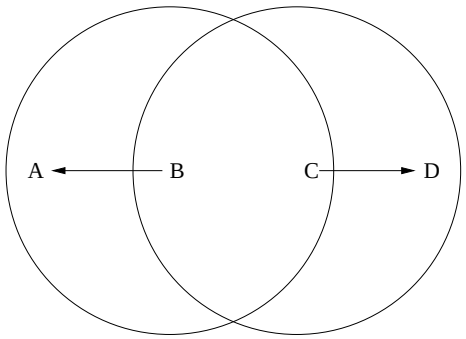


Figure 3.4: The exposed terminal problem.

transmit a frame to node D. However, on sensing the channel prior to its transmission, node C finds the channel to be busy. So, it refrains from transmitting until B has finished its transmission. Had C gone ahead with its transmission to D there would have been no collision and both transmissions would have succeeded.

3.5 DCF Mode of Operation

In the DCF mode of operation nodes that have frames to send use the CSMA/CA mechanism for channel access. In CSMA/CA, if a node finds a channel to be idle it can transmit its frame and wait for an acknowledgment to indicate success. If the channel is busy, or if the acknowledgment is not received, the node initiates backoff, followed by a transmission attempt. If a node is unable to successfully transmit a frame after a predetermined number of attempts, the node discards the frame. Nodes can optionally use the *Request To Send/Clear To Send (RTS/CTS)* handshake to avoid collisions during the subsequent exchange of data and acknowledgment frames.

Before discussing the DCF mode in detail let us first consider the four inter-frame time intervals specified by the MAC sublayer protocol. These intervals, in increasing order of duration, are:

1. SIFS: short interframe space.
2. PIFS: PCF interframe space.
3. DIFS: DCF interframe space.
4. EIFS: extended interframe space.

It is worth noting that only unicast frames are acknowledged by the receiver. Broadcast and multicast frames are not acknowledged by their receiver(s), and the sender does not try to retransmit them. Hence, broadcast and multicast transmissions are less reliable than unicast transmissions.

3.5.1 Initiating Channel Access

A node, having data to send, invokes the carrier-sense mechanism (physical or virtual carrier-sense) to determine the state of the medium. If the medium is sensed to be idle for a DIFS duration the node can transmit its frame. If the medium is sensed to be busy, the node shall defer until the medium is determined to be idle without interruption for a DIFS duration (when the last frame detected on the medium was received correctly), or an EIFS duration (when the last frame detected on the medium was not received correctly). Once the node finds the medium to be idle, without interruption, for this DIFS or EIFS time, it generates a random backoff duration (unless the backoff timer already contains a non-zero value) for an additional deferral time before transmitting.

3.5.2 Backoff Procedure

The backoff procedure always starts after the channel has been sensed to be continuously idle for DIFS or EIFS time (as stated above). The backoff duration is expressed in multiples of a slot time. This number of slots is selected randomly from the range [0,CW], where CW is the contention window, and lies in the range $CWmin \leq CW \leq CWmax$. The standard sets CWmin to 7 and CWmax to 255. For the first attempt at transmitting a frame CW is set to CWmin and a retry counter is set to zero. During the backoff procedure, nodes use the carrier-sense mechanism to determine the state of the

medium during each backoff slot. If the medium is idle for the duration of a particular backoff slot, then the backoff time is decremented by one time slot. If the medium is sensed busy at any time during a slot, then backoff timer is not decremented for that slot, and backoff procedure is suspended. In order to resume the backoff procedure the medium should be determined to be idle for the duration of a DIFS or EIFS period, as appropriate. A node transmits its frame once its backoff timer reaches zero.

3.5.3 Basic Access: Frame Transmission and Acknowledgment

As described above, the transmitting node cannot detect collision. So, following transmission the node awaits an acknowledgment to determine if its transmission was successful. The receiver, on successfully receiving the frame waits for a SIFS period and then sends an acknowledgment to the sender. If the sender does not receive an acknowledgment then it assumes that the transmission failed and increments the retry counter. If, as a result, the retry counter reaches the retry limit the frame is discarded. Otherwise, the node once again invokes the backoff procedure with CW set to twice the CW value during the previous attempt plus one, subject to a maximum of CWmax. In effect, if the previous value of CW was $2^i - 1$, then new value of CW is $2^{i+1} - 1$.

3.5.4 RTS/CTS Handshake

Assume a node transmits its frame after sensing the channel to be idle. Let there be collision on the channel preventing the receiver from correctly receiving the frame. There is no way for the sender to realize this until after it has sent the frame in its entirety. If the frame happens to be large the energy wasted by the transmitter and the bandwidth wasted due to such collisions can be significant. To avoid such wastage of resources, and to increase the reliability of unicast transmissions, the IEEE 802.11 MAC provides for an optional use to RTS/CTS handshake before frame transmission.

Each node can select its own `RTSthreshold`. Unicast frames of size greater than this threshold will be transmitted by the node following an exchange of RTS and CTS messages. Frames of size less than this threshold will be transmitted using the basic access mechanism described earlier. A node can choose to set its `RTSthreshold` to as low as zero, in which case all unicast data frame transmissions will be preceded by an RTS/CTS threshold. Conversely, it can choose to set its `RTSthreshold` to be higher than the largest permissible fragment size meaning that no frame will be sent using the RTS/CTS handshake.

To initiate a RTS/CTS handshake the sender sends a *Request To Send (RTS)* message to the receiver. This message has a duration field set to the time it will take to send the data frame, receive the CTS and acknowledgment frames, plus three SIFS times. This is essentially the time it will take, following the RTS transmission, to successfully complete the frame transmission. All nodes that receive the RTS message, except the intended receiver, update their NAV using the duration value. Consequently, they will refrain from transmitting for the duration specified in the RTS message.

On receiving the RTS, the receiver waits for one SIFS time and then responds with a *Clear To Send (CTS)* message directed towards the sender provided the NAV at the receiver indicates that the medium is idle. Otherwise, the receiver does not respond to

the RTS. The duration field of the CTS is equal to the received RTS duration minus the CTS time and one SIFS time. All nodes that receive the CTS message, except the sender, update their NAV using this duration value. The timeline of RTS/CTS exchange is shown in Figure 3.5.

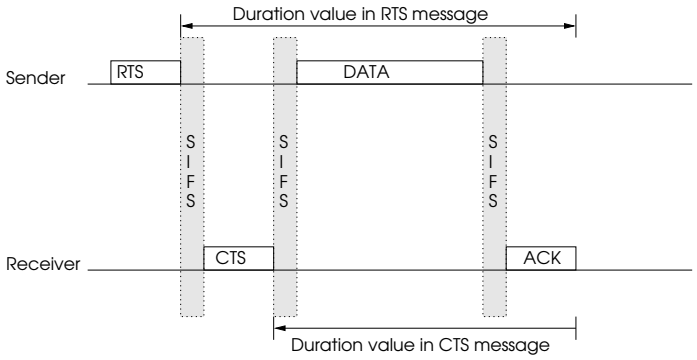


Figure 3.5: Durations specified in RTS and CTS messages.

As a result of a successful RTS/CTS handshake all nodes that could potentially interfere with reception of frames/acknowledgment at the sender or the receiver refrain from transmitting until the sender has sent the data frame and received the acknowledgment. This is illustrated in Figure 3.6. Nodes $C,\ D,\ E$ and F update their NAVs on receiving the RTS message from A. Nodes $J,\ K,\ L,\ M$ and N update their NAVs on receiving the CTS message from B. Nodes $G,\ H$ and I receive both RTS and CTS messages and also update their NAVs.

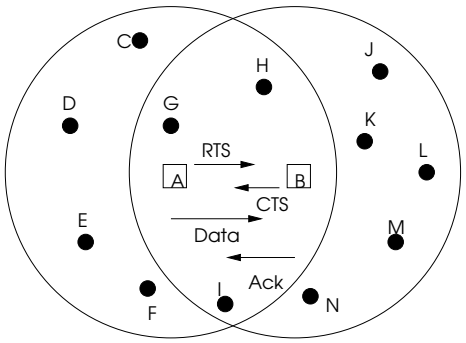


Figure 3.6: RTS/CTS Handshake

The sender, on receiving the CTS, waits one SIFS time before sending the data frame. On receiving the data frame, the receiver waits for one SIFS time before sending its acknowledgment.

Even if the RTS and/or CTS frames collide, their small size (20 bytes) means that the bandwidth and energy resources lost as a consequence are minimal.

This would be a good place to further elaborate on the retry counter mentioned in Section 3.5.3. Each node actually maintains two retry counters: the *short retry counter* and the *long retry counter*. Correspondingly, the node also maintains the short and long retry limits. Following an RTS transmission, if the sender does not receive a CTS before the CTS timeout period expires, it increments the short retry counter and invokes the backoff procedure prior to another RTS transmission, provided the short retry limit has not been exceeded. If the sender does receive the CTS it resets the short retry counter to zero. If the short retry limit is exceeded the sender discards the frame.

Each time the transmission of the data frame fails the node increments the long retry counter as the frame size is greater than the RTSthreshold. If, as a result, the long retry limit is exceeded, the node discards the frame. Otherwise, the node invokes the backoff procedure. If the node successfully receives an acknowledgment, it resets the long retry counter to zero.

The net effect is that a sender can successively send up to short retry limit number of RTS messages, waiting for a CTS, before it transmits the data frame.

3.5.5 Fragmentation of Frames

We know that wireless channels can be noisy. So, the probability that a node will find a received frame to be corrupted (due to interference) can be high. This probability increases with an increase in frame size. After all, the longer it takes to transmit/receive a frame, the greater the chance that there will be some interference during this time. Hence, to increase the reliability of transmission over the wireless channel, IEEE 802.11 MAC allows for long frames to be broken into multiple smaller frames, referred to as fragments.

Each fragment is transmitted independent of the other fragments, and each fragment is acknowledged separately. Also, all except the last fragment carry information to the effect that more fragment(s) are to follow. In the DCF mode, usually all the fragments of a frame are sent as a burst: fragments from the sender to the receiver alternating with corresponding acknowledgments from the receiver to the sender. All the fragments of a frame have the same value in a 12-bit sequence number field, and different values in a 4-bit fragment number field. Based on these information the receiver can recombine all the fragments in the correct order to reconstruct the original frame.

If RTS/CTS handshake is used in combination with fragmentation, the duration fields of the RTS and CTS messages indicate the time until the acknowledgment of the first fragment. The i^{th} fragment and the corresponding acknowledgment (except when i corresponds to the last fragment) have their duration fields indicating the time until the acknowledgment of the $i + 1^{st}$ fragment. Other nodes, on receiving the fragments and/or the acknowledgments use the duration values to update their NAVs, as described earlier. Thus, each fragment and its acknowledgment act like RTS and CTS, respectively for the next fragment's transmission. Once the sender receives the acknowledgment for the i^{th} fragment it waits for SIFS time before sending the $i + 1^{st}$ fragment. Likewise, on receiving a fragment the receiver waits for SIFS time before sending the corresponding acknowledgment to the sender. An example is shown in Figure 3.7 where a frame is fragmented into three frames. The first two have the same

size, while the third is smaller. The duration values in the third fragment and the corresponding acknowledgment are zero.

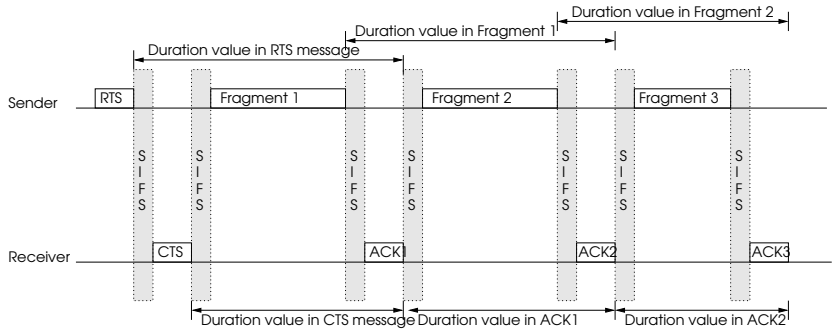


Figure 3.7: Duration values in frames when RTS/CTS is used in conjunction with fragmentation

There is a possibility that a burst of fragment transmissions gets interrupted. For example, the sender may not receive the acknowledgment for a fragment until the expiry of the acknowledgment timeout period. In that case, the sender should invoke the backoff procedure and then resume the burst starting from the first unacknowledged fragment.

Note that once a node starts transmitting a burst of fragments it has priority over other nodes that are also trying to transmit frames on the same channel. This is due to the following reason. Following the reception of the acknowledgment, the sender of the burst is required to wait for only SIFS time before sending the next fragment. Other nodes are required to wait for at least DIFS time before they can attempt to transmit on the same channel. As SIFS is less than DIFS, the other nodes are unable to find the channel idle for DIFS time until the burst ends. So, following a successful transmission of a burst of fragments, the sender must invoke the backoff procedure. This gives other nodes a chance to acquire the channel.

Finally, only unicast frames are fragmented. Multicast and broadcast frames receive a lower quality of service and no effort is made to increase the reliability of their transmission. So, it does not seem reasonable to go the extra distance of fragmentation and reassembly at the sender and receiver, respectively. Moreover, as broadcast and multicast frames are not acknowledged, the loss of even one fragment would negate the successful delivery of all other fragments at the receiver.

3.5.6 Handling Duplicate Frames

A node may receive multiple copies of the same frame. For example, let the receiver send an acknowledgment for a frame. But, the acknowledgment is lost, prompting the sender to resend the frame with a retry bit set in the frame header. Now, the receiver has more than one copy of the frame. All of them have the same source address and the same sequence number and fragment number. A node can maintain a cache of 3-tuples: <source address, sequence number, fragment number> corresponding to the

frames received in the recent past. If frames are received with header values corresponding to any of the 3-tuples in the cache they are deemed to be duplicates and are discarded. Furthermore, note that fragments of a frame are sent in sequence. So, for any source address and sequence number pair, the node should cache only the 3-tuple corresponding to the highest fragment number received thus far. There is no chance of receiving a lower numbered fragment for the same frame in the future.

3.6 PCF Mode of Operation

The PCF mode of operation yields contention-free access to the shared wireless channel. It can only be used in BSSs that have an access point (AP). The AP acts as the point coordinator (PC) for the BSS. The PC transmits the time until the end of the contention-free period (CFP) in a beacon issued at the start of the CFP Repetition Interval. It is noteworthy that the duration of the contention-free period may be greater than the beacon interval. In that case the PC transmits beacons at their scheduled times during the contention-free period. Each such beacon also indicates the time until the end of the contention-free period. The information is used by the other nodes to update their NAV. At the end of the contention-free period the PC transmits a CF-End frame.

The PC can send data and acknowledgments and poll nodes in its poll list. This poll list is created using information provided by nodes when they establish an association with the AP. Nodes in the poll list are referred to as CF-pollable nodes. If a node wishes to change its polling status (pollable or not pollable) then it has to perform a reassociation. During the PCF mode of operation, normally the following sequence repeats: PC's transmission, SIFS idle time, a non-PC node's transmission, SIFS idle time.

The PC knows the CFP Repetition Interval. So, it can determine the expected time for the start of the contention-free period. At this time the PC senses the medium. If it finds the medium to be idle for PIFS amount of time it transmits a beacon. Then, SIFS time after the beacon, the PC makes its first transmission. The following eight possibilities exist as far as the contents of transmissions by the PC are concerned (of which only 1, 3, 5 and CF-End can be sent as the first transmission):

1. `Data`: contains a data frame sent by the PC to a non-PC node. The receiving node is expected to transmit an acknowledgment, but no data frame, after SIFS time.

2. `CF-ACK`: contains an acknowledgment for a data frame received from a CF-pollable node SIFS time before sending this acknowledgment.

3. `CF-Poll`: contains a poll for the intended recipient. The recipient, is expected to send a data frame SIFS time after receiving the poll.

4. `Data + CF-ACK`: contains a data frame for the intended recipient, and an acknowledgment for a data frame received from a CF-pollable node SIFS time before sending this message. As the intended recipient is not being polled, the recipient cannot send any data frame. Instead, after SIFS time, the recipient is expected to send an acknowledgment to the PC.

5. `Data + CF-Poll`: contains a data frame for the intended recipient along with a poll for it. Consequently, after SIFS time the intended recipient is expected to transmit a data frame.

6. `CF-ACK + CF-Poll`: contains an acknowledgment for a data frame received from a CF-pollable node SIFS time in the past and a poll for the intended recipient. On receiving this message, the intended recipient will wait for SIFS time and then transmit its data frame.

7. `Data + CF-ACK + CF-Poll`: As with the last message types, this contains an acknowledgment for a previously received frame and a poll for the intended recipient. It also contains a data frame for the intended recipient. So, SIFS time after receiving the message, the intended recipient will transmit a data frame.

8. `Management frame`: contains any appropriate management frame that the AP is allowed to send.

A node that has received a data frame from the PC along with a poll message can piggyback the received frame's acknowledgment on its own transmitted data frame. If the polled node has no data frame to send, but did receive a data frame from the PC, then the node simply sends an acknowledgment to the PC. If the polled node has neither a data frame nor an acknowledgment to send, it sends a NULL message. It is worth noting that a node's transmission, in response to a poll, may not necessarily be to the PC. It can be destined for any node. However, if the node does not receive an acknowledgment for this frame, it cannot retransmit during the contention-free period unless it is polled again by the PC.

It is possible that the PC sends data and/or poll to a node and there is no response from that node for PIFS time, where PIFS>SIFS. In such situations the PC immediately transmits its next frame. As PIFS<DIFS, the PC is able to maintain control over the channel during the contention-free period. Had the PC waited for DIFS or longer, it is possible that some node from an overlapping BSS operating in the DCF mode would have seized control of the channel. The same explanation holds for why the PC transmits its first beacon after sensing the channel to be idle for PIFS duration following the expected start time of the contention-free period. Basically, the fact that PIFS<DIFS gives priority to PCF traffic over DCF traffic in overlapping BSSs.

Figure 3.8 presents an example of PCF mode operation.

3.7 Power Management

Nodes belonging to a wireless LAN are often powered by batteries. Extending the duration a node can operate on a single charge is a desirable goal. Also, there may be extended periods of time during which a node has no data to exchange with other nodes in its BSS. So, a simple approach to extending battery life would be for the node to power down its transceiver during periods of no data communication. Every once in a while the node would *wake up* its transceiver and exchange frames with other nodes. But, if every node followed its own schedule for sleeping and waking up, there

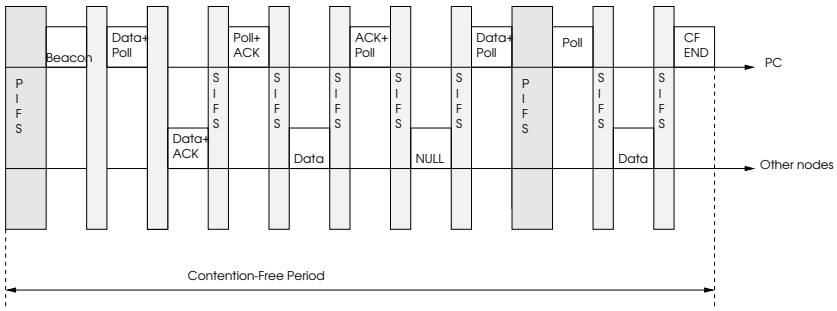


Figure 3.8: Exchange of frames between Point Controller and other nodes in PCF mode.

could be utter confusion: node A wishes to send a frame to node B, but when A is awake B is asleep and vice versa. To prevent such scenarios from arising, nodes in a BSS are synchronized with each other. They wake up periodically at around the same time. Then, they announce if they have frames to send to others in the BSS. Nodes that neither have any frame to send nor receive can promptly go back to sleep. Others stay awake until they have successfully transmitted or received the frame(s). In the following sections we will first discuss the synchronization mechanism of the IEEE 802.11 standard. Then, we will discuss the protocol for exchanging frames between nodes that are involved in power management. Note that synchronization between nodes is needed for a variety of reasons, not just for power management.

3.7.1 Synchronization in a BSS

In an Infrastructure BSS the access point provides the synchronization. The AP's beacon contains timing information. Nodes in the BSS use this information to synchronize their clocks to within $4\mu s + physical\ layer\ propagation\ delay$ of the AP's clock.

In an Independent BSS there is no AP. So, the nodes employs a distributed algorithm to synchronize their clocks. The node that initiates the BSS gets to specify the beacon interval. As other nodes join the BSS they acquire this information and the time until the next beacon. When the next beacon is due all the nodes start sensing the medium, and wait until the medium becomes idle. At this time each node selects a backoff value that is randomly distributed in the range $0 - 2 \times CWmin \times slot\ duration$, initiates the backoff procedure, and starts listening for a beacon corresponding to its BSS. On receiving a beacon from another node in its BSS, the node: (i) cancels its backoff timer, and (ii) uses the clock information in the received beacon to synchronize its own clock provided the received clock value is *greater than its own clock value*.[1] This ensures that the local clock is monotonically increasing. If the back-

[1]What if the received clock value is smaller than the node's own clock value? Obviously, the receiver's clock is ahead of the clock of the node that transmitted the beacon, and possibly ticking at a faster rate. Does this mean that over a period of time the clocks will lose synchronization? The answer is *NO*. The node with faster clock will conclude that the scheduled beacon time has been reached well before the other nodes do. So, this node will initiate its backoff timer well before others, and with a high probability count down to zero

off timer reaches zero before the node receives a beacon, the node transmits a beacon with its own clock information.

Thus, after each inter-beacon interval all the nodes in the BSS synchronize their clocks with the clock of the node that is the first to transmit a beacon.

3.7.2 Announcing Available Frames and Frame Exchange

Each node can select how aggressively it wishes to conserve its energy by choosing the time between successive wake-ups. For example, a node would wake up just before its expects the next beacon to be transmitted and listens for the beacon. If it realizes that other nodes have data for it, the node stays awake until it receives the data. Otherwise, the node can go back to sleep. Another node that wishes to save more energy could wake up just before every n^{th} beacon, where n is an integer greater than one, and try to figure out if others have data for it. So, how does a node come to know if others are trying to communicate with it? The answer depends on whether the node is part of an Infrastructure BSS or an Independent BSS.

In an Infrastructure BSS the AP buffers all unicast frames meant for nodes operating in the power-save mode, and transmits a *Traffic Indication Map (TIM)* with each beacon. The TIM contains the list of nodes, operating in the power save mode, for whom the AP has buffered frames. When a node finds itself listed in the TIM its stays awake. Otherwise, the node goes back to sleep until its next scheduled wake-up time. The way the waking nodes receive buffered frames from the AP depends on whether the BSS is operating in the DCF or PCF mode. In the PCF mode, if the power-saving node is registered as CF-pollable, the AP sends directed frame(s) to the node. With each transmitted frame, the AP also indicates if it is has more buffered frames for that node. If so, the node stays awake until it has received all the buffered frame. Otherwise, the node goes back to sleep. Now, let us consider the following scenarios: (i) the contention-free period has ended, but the CF-pollable node in the power-saving mode has still not received all its frames buffered by the AP, or (ii) the BSS operates only in the DCF mode and a node in the power-saving mode has not received all its frames buffered by the AP. Both scenarios are handled in the same manner. The node, on realizing that the AP has buffered frames for it, contends for the medium and sends a poll to the AP. The AP can respond to the poll either by sending a buffered frame to the node, or by sending an acknowledgment with a promise to send the buffered frame at a later time. Here, too, with each transmitted frame the AP indicates if it has additional buffered frames for the target node.

All this is fine for unicast frames. What if the AP has to send broadcast or multicast frames to nodes in its BSS, and a subset of recipients are in the power-save mode? If the AP were to transmit them as soon as they arrive in its buffer there is a good chance that nodes in the power-save mode may not receive them. This problem is tackled by the AP buffering such frames and employing a special type of TIM, referred to as the *Delivery TIM (DTIM)*. Every x inter-beacon intervals (where x is a value chosen by the AP) the AP transmits, with its beacon, a DTIM instead of an ordinary

before the others. So, its beacon will then be used by other nodes to synchronize their clocks: in effect help the other nodes' clocks to catch up.

TIM. Immediately following a DTIM transmission, the AP first transmits the buffered broadcast and multicast frames, and then the buffered unicast frames.

Now, let us consider the following situation. The AP has decided that every fourth beacon will be accompanies by a DTIM. So, beacons $\ldots,\ i,\ i+4,\ i+8, \ldots$, etc. will have DTIMs. A node in the power-save mode has chosen to wake up once every three inter-beacon intervals, and wakes up in time to receive beacon number $i+2$. If this node sticks to its schedule it will be asleep during the DTIM transmitted with beacon number $i+4$, and the earliest it can receive a DTIM would be with beacon number $i+8$. If the AP goes ahead and transmits the broadcast and multicast frames after beacon number $i+4$ the sleeping node will obviously miss them. A limited buffer size may also prevent the AP from buffering the multicast and broadcast frames for too long. The AP employs two measures to alleviate this problem. First, with each beacon it transmits the inter-DTIM interval as well as the number of beacon intervals until the next beacon. So, on receiving a beacon every node in the power-save mode immediately realizes when the DTIM are to be transmitted in the future. If this node does not wish to miss out on the broadcast and multicast frames it better wake up in time to receive the DTIMs. Second, the AP has a maximum duration for which it buffers multicast and broadcast frames. On the expiry of this period the AP discards these frames.

As discussed earlier, in an Independent BSS one node transmits a beacon every inter-beacon interval. Beacon transmission is followed by a period of time referred to as the ATIM Window. During the ATIM Window all the nodes stay awake. Let node A have unicast frames to send to nodes B, C and D of which B and C are known to be operating in the power-save mode. During the ATIM window node A contends for the channel, employing the backoff procedure, and sends a directed *Ad Hoc Traffic Indication Map (ATIM)* to B and another directed ATIM to C. Nodes B and C acknowledge receiving the ATIMs. If A does not receive an ATIM's acknowledgment, it employs backoff and retransmission of the ATIM during the ATIM Window. If A has broadcast and multicast frames to send it advertises them in an ATIM, too, but does not expect any acknowledgment. If, by the end of the ATIM Window, a node in the power-save mode has not received any ATIM and does not have any frame of its own to transmit, it goes back to sleep. Nodes that have received ATIMs stay awake. Following the ATIM Window, nodes that have received acknowledgment(s) for their directed ATIMs contend for the channel and transmit their frames using the DCF mode. Similarly, nodes that have broadcast/multicast frames to send to nodes in the power-save mode contend for the channel. When a node in the power-save mode has received all the frames it is expecting to receive it goes back to sleep until the next scheduled beacon time.

3.8 ZigBee and IEEE 802.11 WLAN

The IEEE 802.15.4 protocol, when operating in the 2.45 GHz ISM band, has 16 channels and supports data rates of 250 kbps. Fifteen out of these sixteen channels (channels 11-25) overlap with one of the three orthogonal channels of the IEEE 802.11 protocol as shown in Figure 3.9. Hence, it is possible that a ZigBee network, operating in channels 11-25, and an IEEE 802.11-based wireless LAN may interfere with each other.

A ZigBee network operating in channel 26 will not interfere with a colocated WLAN. However, if multiple colocated ZigBee networks were to operate in this channel they would interfere with each other.

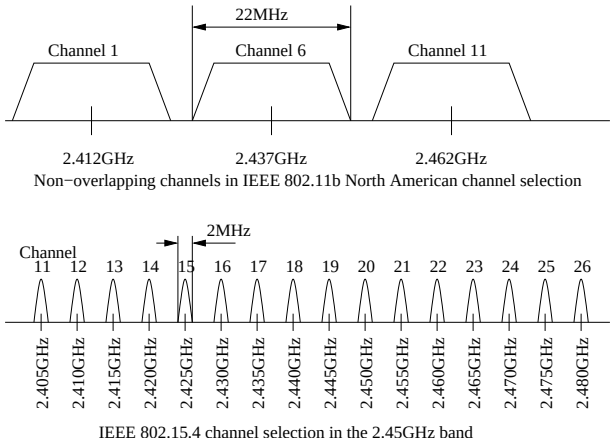


Figure 3.9: Wireless communication channels of IEEE 802.11 and IEEE 802.15.4 networks.(Source: Figure E.1, page 65 of [27]

The IEEE 802.11 protocol is well known. However, as noted in [17], the same is not true for the IEEE 802.15.4 and ZigBee protocols as the specifications of these protocols were officially released only recently. So, in this section we provide an overview of these protocols. This will provide the context in which we will explain the results obtained from the experiments run on the testbed, and also justify the choices we make in the channel selection heuristic.

3.8.1 PAN Topologies

There can be two kinds of devices in an IEEE 802.15.4 network: full-function devices (FFDs) and reduced-function devices (RFDs). An FFD can act as a Personal Area Network (PAN) coordinator, a coordinator, or an end device. An RFD can only act as an end device. Each device has a unique 64-bit extended address.

The PAN coordinator initiates the formation of the network, supplies the PAN identifier and several other network parameters, can communicate with neighboring FFDs and RFDs, and act as a router. A coordinator can communicate with neighboring FFDs and RFDs and act as a router. An end device can only communicate with one FFD. A PAN can have one of two possible topologies: star or peer-to-peer, as shown in Figure 3.10. In the star topology the PAN coordinator is the central controller. All other devices communicate only with the PAN coordinator. As is evident from Figure 3.10, a multi-hop wireless mesh network is an instance of the peer-to-peer network topology. Two devices that are within range of each other can communicate directly with each other. Upper layer protocols, for example the ZigBee protocol stack, can determine the

routing protocol that runs on such a network and how messages are forwarded between two devices that are not neighbors of each other. Each PAN has a unique 16-bit PAN identifier selected by the PAN coordinator. On joining a PAN a device obtains a 16-bit short address. Subsequently, the device can use this short address, instead of the 64-bit extended address, to identify itself.

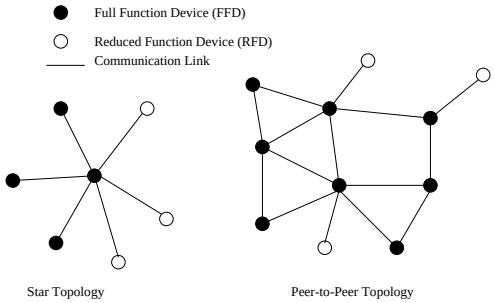


Figure 3.10: PAN topologies in an IEEE 802.15.4 network.

3.8.2 Cluster-Tree Network

A cluster-tree network, shown in Figure 3.11, is an instance of a peer-to-peer PAN. All the internal nodes are FFDs, while RFDs can only join as leaves of the tree. A cluster-tree network is initiated by a PAN coordinator which can permit a certain number of devices to join as its children. Each child device that joins as a coordinator (by definition an FFD) can permit other devices to join the PAN as its children. In this fashion a multi-hop PAN can be formed. The PAN coordinator determines the maximum depth of a cluster-tree PAN, the maximum number of children each coordinator can have, and of these children up to how many can be coordinators. The number next to a node in Figure 3.11 denotes the depth of the node (number of hops from the PAN coordinator).

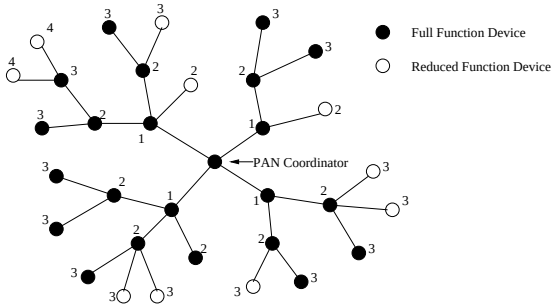


Figure 3.11: Cluster-tree PAN topology.

A major difference between a cluster-tree network and other mesh-based peer-to-

peer networks is the requirement that nodes in a cluster-tree network use *beacons*. A beacon is a message that the PAN coordinator and all other coordinators in the tree transmit periodically. A device's *superframe* starts with the transmission of a beacon and lasts until the transmission of its next beacon. A beacon contains the PAN id, macBeaconOrder (BO), and macSuperframeOrder (SO). The value of BO determines the beacon interval (BI), the time between successive beacons transmitted by a device. If $0 \leq BO \leq 14$, $BI = \texttt{aBaseSuperframeDuration} \times 2^{BO}$ symbols. If $BO = 15$, a device does not transmit a beacon periodically.[2] Instead, it transmits a beacon only on receiving a beacon request command. If $0 \leq SO \leq BO \leq 14$, the Superframe Duration (SD) is equal to $\texttt{aBaseSuperframeDuration} \times 2^{SO}$ symbols. Starting from the transmission of the beacon, a device stays active for SD amount of time during its superframe. During the rest of its superframe the device has the option of conserving its energy by going into the sleep mode. The active portion of the superframe duration is divided into the contention access period (CAP) and the contention-free period (CFP), as shown in Figure 3.12. Communication during the CAP employs carrier sense multiple access with collision avoidance (CSMA/CA).

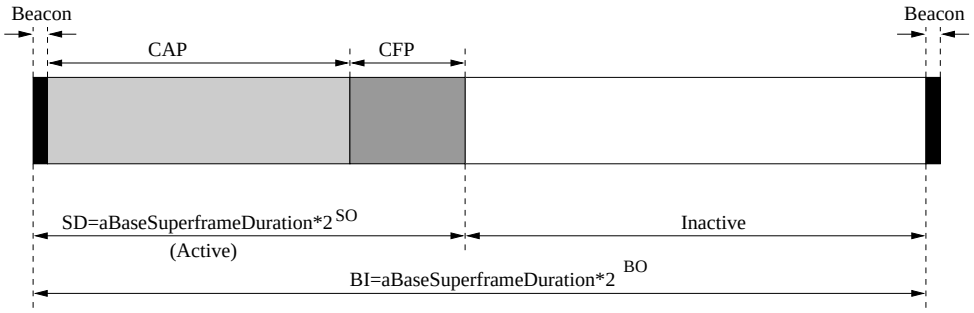


Figure 3.12: An example superframe structure (reproduced from [27]).

A tree-based ZigBee network can be built using a cluster-tree IEEE 802.15.4 PAN. A node in a tree-based ZigBee network can communicate with its parent during the active portion of its parent's superframe. The node can communicate with its children during the active portion of its own superframe. Moreover, if the parent of a node X has no messages to exchange with X then node X can go to sleep immediately after receiving the beacon from its parent. This is because in a tree-based network the parent communicates with its children *indirectly* in the following fashion. In its beacon, the parent node advertises the identities of up to seven children for which it has data to send. If a child's identity is not listed that child may revert to idle status for the rest of the active portion of the parent's superframe. If a child's identity is listed in the parent's beacon it is the child's responsibility to obtain the data from the parent at a time of its choice during the CAP of the parent's superframe. If the child is unable to obtain the message from the parent within `macTransactionPersistenceTime`, the parent discards the message.

[2]Henceforth, constants associated with the protocol will be written in `teletype` font.

3.8.3 Beacon and Superframe Scheduling

Every node tracks its parent's beacon. The beacon transmitted by a node contains the time by which this beacon is offset from its parent's beacon. Once a newly-joined node receives the beacons of all its neighbors the node knows when the neighbors and their respective parents are in the active portions of their respective superframes. This enables the node to choose the time for its beacon transmission such that the active portion of its superframe does not overlap with that of any neighbor or parent of the neighbor. If it is not possible to select such a non-overlapping duration for the active portion of the node's superframe the node can join the network only as an end device. This ensures that if a node X is trying to communicate with a child Y then the only other sources of interference from within the ZigBee network could be the other children of X.

This brings us to the issue of *duty cycling* in a ZigBee network. A coordinator node in a tree-based ZigBee network is in the active portion of its superframe for $1/2^{BO-SO}$ fraction of time. Assuming the worst case scenario from the point of energy consumption, a node has to stay awake during the active period of its own superframe as well as during all of the active period of its parent's superframe. So, the fraction of time for which a node in a tree-based ZigBee network can be in the sleep mode, $Sleep_{frac}$, can be expressed as: $Sleep_{frac} \geq 1 - \frac{2}{2^{BO-SO}}$ For example, if $BO = 14$ and $SO = 7$, $Sleep_{frac} \geq 1 - \frac{1}{2^6} = \frac{63}{64}$, meaning the node can be in the sleep mode for at least 98.44% of the time. Thus, there is a great potential for energy savings in a tree-based ZigBee network.

In contrast, a mesh-based ZigBee network does not employ beacons. So, it is not possible to schedule communication between neighboring nodes the way it can be done in a tree-based network. As a consequence, in a mesh-based ZigBee network when a node is not transmitting it has to stay awake in the receive mode, listening for transmissions from neighbors. *Therefore, for future sensor networking applications of ZigBee technology the tree-based topology appears to be more suited than the mesh topology.* The tree topology is also suitable for several sensor networking applications for yet another reason – most of these applications involve the sensor nodes sending their observations to a single sink or a subset of nodes acting as sinks. A tree-based network topology allows sensor readings to be convergecast to the sink(s) along with the ability to aggregate multiple readings along the way.

3.8.4 PAN Formation and Channel Selection

Prior to starting a PAN, the prospective PAN coordinator performs an *energy detection* scan on a specified set of channels. This scan yields the peak energy level detected on each of the channels. As a consequence of this scan, the prospective PAN coordinator can eliminate some of the busy channels from consideration. Then the prospective PAN coordinator performs an *active scan* of the channels still under consideration.

The active scan is performed one channel at a time in the following manner: (i) the prospective PAN coordinator tunes to the channel and transmits a scan request, (ii) then the prospective PAN coordinator listens for beacons for a specified period of time on that channel. At the end of this duration the prospective PAN coordinator moves on

to the next channel in its active scan list and repeats this procedure.

Devices that are coordinators of beacon-enabled PANs (like cluster-tree PANs) ignore the scan request and transmit their beacons at their regularly scheduled time. Devices belonging to non beacon-enabled PANs (like mesh networks) respond to the scan request with a beacon containing their PAN identifier.

Following the active channel scan, the prospective PAN coordinator comes to know of all the PANs in its vicinity. It selects the channel with the minimum number of existing networks, and starts its PAN on that channel with the PAN identifier value distinct from all the PAN ids received during the active channel scan.

A device wishing to join an existing PAN performs an active or *passive* channel scan. The difference between active and passive channel scans is that no scan request is issued during a passive channel scan. So, it is quite possible that a passive channel scan will only return information about beacon-enabled PANs. Following a channel scan, the device selects the PAN it wishes to join. It then sends a join request to a node of the selected PAN provided that node is willing to let other devices join the PAN through it. The join request also indicates whether the joining device wishes to join as a router or as an end device.

The newly joined device inherits the PAN id, logical channel, Beacon Order, and Superframe Order values from its parent. Therefore, all the nodes of a PAN communicate on the same channel – the channel that the PAN coordinator had selected following the its active channel scan at the time of initiating the PAN. As the PAN grows due to the joining of new devices the geographical span of the network increases.

3.8.5 Interference Scenario in ZigBee Networks

It is possible that a channel that was suitable for the PAN in the vicinity of the PAN coordinator is not so suitable for it a few hops away from the PAN coordinator. This can severely limit the ability of an IEEE 802.15.4-based PAN to grow in size. Also, while a PAN is operational an IEEE 802.11-based communication session may start on an overlapping channel. This may cause severe interference on some of the links of the PAN. Hence, there is a need to consider *modifications to the IEEE 802.15.4/ZigBee protocol stack such that different links of a PAN can possibly operate on different channels.*

It would be too expensive to have more than one transceiver on a single sensor node. So, we will limit our study to devices with a single transceiver. Using such devices it is extremely difficult to form a mesh-based PAN that operates on more than one channel. In mesh-based PANs all communication is asynchronous. So, if a device is tuned to channel X to communicate with a neighbor n_X, and another neighbor n_Y sends a message to the device on channel Y, the latter message will surely be lost.

However, in tree-based ZigBee networks this is not a problem. The time during which a coordinator node communicates with its children does not overlap with the time during which it communicates with its parent. So, using just one transceiver per device it is possible to form a tree-based ZigBee network operating on multiple channels in the following manner – *The channel on which a coordinator communicates with all its children may be different from the channel on which it communicates with its parent.* When a node wakes up to listen to its parent's beacon the node tunes to the

channel on which the parent communicates during the active period of its superframe. Subsequently, when the node wakes up to communicate with its children during the active period of its superframe it tunes its transceiver to a channel on which it can communicate with all its children.

3.9 Wired Equivalent Privacy

The IEEE 802.11 wireless LAN (WLAN) protocol employs the Wired Equivalent Privacy (WEP) protocol for security. The goal is to ensure secrecy of communication between computers with IEEE 802.11 network interface cards and corresponding Access Points (APs). The three major security goals of WEP are:

1. Confidentiality: A third party should not be able to listen to and decipher encrypted messages being exchanged between two computers with IEEE 802.11-compliant NICs.

2. Access Control: Unauthorized access to the network infrastructure should be prevented. Packets that are not encrypted as per the WEP protocol should be discarded.

3. Data Integrity: Prevent tampering of encrypted data in transit from one computer to another. If the encrypted message is modified by a third party with the intent of modifying the actual message, the receiver should be able to detect the tampering.

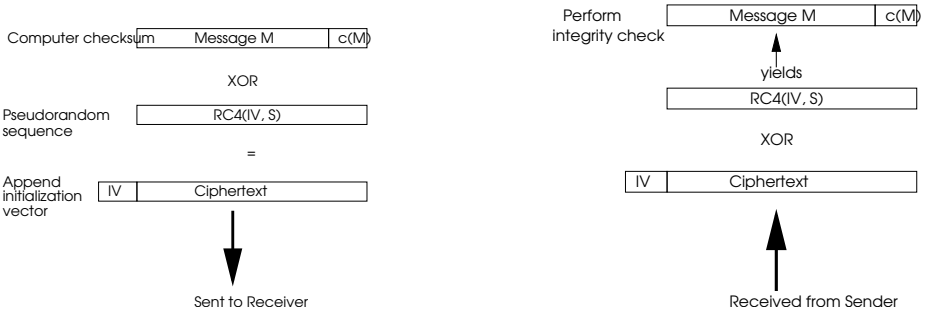
Research over the last few years has shown that WEP fails to meet any of the stated security goals in the face of attacks that may be mounted by users with modest resources [4, 9, 38, 53].

In the following description we first discuss the WEP approach to support the security goals of confidentiality, access control and data integrity. Then we describe the vulnerabilities of WEP and the resultant inability to support the stated security goals. Subsequently, we discuss some approaches being considered to overcome the shortcomings of WEP.

WEP security employs the RC4 algorithm [48] for encryption and decryption. The communicating nodes share a 40-bit secret, $\mathcal{S}$. In addition, there is the notion of a 24-bit initialization vector, $\mathcal{IV}$. Let there be a message $\mathcal{M}$ to be transmitted from node A to node B. First, A computes an integrity checksum of the message, $c(\mathcal{M})$. Subsequently, A concatenates $\mathcal{M}$ and $c(\mathcal{M})$ to create the plaintext message, $\mathcal{P}$, for transmission. Node A generates a pseudorandom sequence of bytes, $RC4(\mathcal{IV}, \mathcal{S})$ which is XORed with the concatenation of $\mathcal{M}$and $c(\mathcal{M})$ to generate the ciphertext, $\mathcal{C} = \mathcal{P}\ XOR\ RC4(\mathcal{IV}, \mathcal{S})$. Finally, A transmits the integrity vector $\mathcal{IV}$ followed by the ciphertext $\mathcal{C}$.

When B receives the message, it extracts the corresponding integrity vector, $\mathcal{IV}$, from the first forty bits of the message and retrieves the plaintext $\mathcal{P}' = \mathcal{C}\ XOR\ RC4(\mathcal{IV}, \mathcal{S})$. The plaintext message $\mathcal{P}'$ can be interpreted as the concatenation of a message $\mathcal{M}'$ followed by its checksum field. If the checksum computed by B on $\mathcal{M}'$ matches the checksum received in $\mathcal{P}'$, B accepts the message. Figure 3.13 shows the steps for encrypting and decrypting messages at the sender and receiver, respectively.

It is claimed that due to the presence of the shared secret, $\mathcal{S}$, and a 24-bit per-frame initialization vector, $\mathcal{IV}$ (which permits 2^{24} possible pseudorandom byte sequences) it is possible to ensure secrecy of communication. Also, by communicating the message



(a) WEP Send (b) WEP Receive

Figure 3.13: WEP send and receive

and the checksum in an encrypted form it is possible to ensure integrity of message communication.

3.9.1 Vulnerabilities of WEP

3.9.1.1 Weak Data Encryption

It is possible to extract the plaintext message from an encrypted message with a moderate amount of effort. As mentioned earlier, only 2^{24} possible pseudorandom byte sequences are possible. Furthermore, there is a one-to-one mapping between an initialization vector, $\mathcal{IV}$ and a pseudorandom byte sequence with which a message is XORed. An attacker may inject known plaintext into the wireless network and record the corresponding ciphertext along with the initialization vector used to produce the ciphertext. For example, the attacker may send messages from a compromised node on the wired network to a node on the wireless LAN. By XORing the known plaintext with the corresponding ciphertext, the attacker can extract the pseudorandom byte sequence corresponding to an initialization vector. Over a period of time, the attacker can build a dictionary of initialization vectors and pseudorandom byte sequence pairs. Subsequently, any message encrypted using an initialization vector present in the dictionary can be easily decrypted.

Even if it is not possible to inject known plaintext into the network, the attacker can keep listening until it detects two frames encrypted using the same initialization vector. Let the ciphertexts in the two frames be $\mathcal{C}$ and $\mathcal{C}'$, and the corresponding plaintexts (unknown to the attacker) be $\mathcal{P}$ and $\mathcal{P}'$, respectively. Now: $\mathcal{C}\ XOR\ \mathcal{C}' = \mathcal{P}\ XOR\ RC4(\mathcal{IV}, \mathcal{S})\ XOR\ \mathcal{P}'\ XOR\ RC4(\mathcal{IV}, \mathcal{S}) = \mathcal{P}\ XOR\ \mathcal{P}'$.

By employing various pattern-matching techniques, along with knowledge of the frequency of use of characters in natural languages it may be possible to extract both $\mathcal{P}$ and $\mathcal{P}'$ from the ciphertexts. Due to the 24-bit size of initialization vectors, it is quite likely that the same initialization vector will be reused within a day or two. Also, by the Birthday Paradox, the probability that two frames will share the same initialization

vector after n frames can be expressed as :

$$P_n = P_{n-1} + (n-1)(1 - P_{n-1})/2^{24}$$

From this expression, it can be inferred that with a very high probability the same initialization vector will be reused within a few hours.

Moreover, implementations of WEP by various vendors make it even easier to mount such attacks. For example, in [9] it is reported that some network cards set their initialization vector to 0 each time they are reinitialized (for example, on being inserted into a laptop computer) and increment it by one with each frame. Thus, it is quite likely that several ciphertexts using the same initialization vector (towards the lower end of the sequence) will be available to an attacker.

3.9.1.2 Data Integrity Violation

It is possible for an active attacker to modify encrypted messages without the intended receiver detecting the modification. This is because the CRC-32 checksum that is used to maintain message integrity is a linear function of the message [9], i.e., the checksum calculated on the XOR of two bit sequences is the same as the XOR of the checksums of the those two bit sequences.

Let the original message be $\mathcal{M}$, with checksum $c(\mathcal{M})$, and the corresponding ciphertext $\mathcal{C}$. Let the attacker intercept the ciphertext, XOR it with another string $(m,\ c(m))$, and send the resultant string to the receiver. The consequence of this operation is that the receiver obtains the following message:

$$\begin{aligned}\mathcal{C}\ XOR\ (m, c(m)) &= RC4(\mathcal{IV}, \mathcal{S})\ XOR\ (\mathcal{M},\ c(\mathcal{M}))\ XOR\ (m, c(m)) \\ &= RC4(\mathcal{IV}, \mathcal{S})\ XOR\ ((\mathcal{M}\ XOR\ m), (c(\mathcal{M})\ XOR\ c(m)) \\ &= RC4(\mathcal{IV}, \mathcal{S})\ XOR\ (\mathcal{M}',\ c(\mathcal{M}')),\end{aligned}$$

where $\mathcal{M}' = \mathcal{M}\ XOR\ m$.

When the receiver performs an integrity check on the message it concludes that the sender intended to send $\mathcal{M}'$, and fails to detect the message modification.

3.9.1.3 Access Control Violation

It is possible to circumvent WEP such that unauthorized nodes can be allowed to authenticate themselves with the network and inject messages into the network. WEP does not disallow the reuse of an initialization vector. As stated earlier, it is possible to build a dictionary of initialization vector and corresponding pseudorandom byte sequence. An unauthorized user can generate a message, encrypt it using a known pseudorandom byte sequence and send it with the corresponding initialization vector. The recipient will never suspect that the sender is unauthorized.

Also, it is possible for an unauthorized node to get authenticated with an access point by exploiting WEP's tolerance of repeated use of the same initialization vector. When a mobile node wishes to get authenticated with the access point, the access point sends it a 128-byte plaintext challenge. A genuine mobile node knows the shared

secret $\mathcal{S}$, and generates a 128-byte pseudorandom sequence using $\mathcal{S}$ and a 24-bit initialization vector chosen by it, $\mathcal{IV}$. The mobile node XORs the challenge plaintext with the pseudorandom sequence and sends it to the access point which accepts the encrypted response. An intruder can easily extract the 128-byte pseudorandom sequence by XORing the plaintext and encrypted text. Subsequently, the intruder can get authenticated with the access point by using the same pseudorandom sequence to successfully encrypt a challenge plaintext.

3.9.2 Suggested Solution Approaches

Following the identification of security loopholes in WEP, IEEE is considering a new security architecture, referred to as the Robust Security Network (RSN) as a security solution for the IEEE 802.11 protocol. RSN employs the IEEE 802.1x standard. However, in this protocol the emphasis is on authentication, integrity and key management, and not on secrecy of communication.

The security approach can be described as *port-based network access control*. When an AP receives a request from a node it opens a logical port for that node. However, this port is initially restricted in the sense that it only permits authentication traffic to pass through to the DS. Once authentication is done, the port is opened to allow other traffic to pass through.

A distinguishing feature of RSN is that the AP is not required to make all the security decisions. It is possible to have a dedicated *Authentication Server* in the network. The mobile nodes, also referred to as *clients*, and the authentication server perform mutual authentication with the AP acting merely as a *pass-through* for authentication traffic. This has two beneficial outcomes:

1. The AP becomes simpler and cheaper to implement.

2. The authentication server and clients can change their security protocols without the need for any modifications at the AP.

The IEEE 802.1x standard has evolved over time. Initially, the *Extensible Authentication Protocol (EAP)* [1] was used for authentication. Only the client was authenticated and messages were sent in the clear. It was assumed that all network access points were reliable and their authentication was not required. As a result, an attacker could masquerade as an access point and send an authentication acceptance message to a mobile node. Subsequently, all traffic from the mobile node, meant for the access point, could flow through the attacker. This is an example of the man-in-the-middle attack.

Also, as session control messages were not checked for integrity, it would have been possible for an attacker to hijack a session by making the mobile node terminate the session, and then communicating with the access point while pretending to be the disconnected mobile node. Both session hijacking and the man-in-the-middle attack are described in detail in [38].

A mobile node (client) can move from one BSS to another. Having the client perform authentication from scratch each time it moves between BSSs is expensive and

can disrupt ongoing network applications. So, there is a need to minimize the length of such disruptions.

The *Protected EAP (PEAP)* [40] offers the desired solutions. PEAP has two parts:

Part 1: The authentication server authenticates itself to the client. Optionally, the client can authenticate to the server. This *Transport Layer Security (TLS)* protocol [16] is employed to create a TLS tunnel between the client and the authentication server.

Part 2: The client authenticates itself with the authentication server using EAP. All the EAP messages pass through the TLS tunnel created in Part 1. Hence, session hijacking and man-in-the-middle attacks are thwarted.

PEAP exploits the TLS session resumption facility to quickly re-authenticate a client moving from one BSS to another.

In the following sections we discuss the two parts of PEAP in greater detail. For the purpose of simplicity some of the details have been omitted, and multiple messages have sometimes been combined into a single message. Readers who are interested in all the gory details would relish going through the EAP [1], PEAP [40] and TLS [16] documents.

3.9.3 PEAP Part 1

The message exchange for PEAP Part 1 is shown in Figure 3.14. Initially, the access point sends a message to the client requesting its identity. The client responds with its identity. Based on the client's identity the access point forwards the response to the appropriate authentication server. These two messages (identity request and response) are optional. If the client's identity is already known this message exchange can be skipped and the protocol can directly proceed to the next message, which is a `PEAP start` message sent by the authentication server to the client. This message carries the identity of the authentication server. It also carries the version of the PEAP protocol that the authentication server is capable of supporting.

On receiving the PEAP start message the client sends a `Client Hello` message to the authentication server. This message contains the highest protocol version that the client supports, a random number, and a set of ciphersuites listed in descending order of client preference. The version numbers exchanged between the client and the server enable them to agree on a common protocol version. For the protocol to proceed any further the version numbers must be TLS version 1.0 or later.

The server responds to the `Client Hello` with a `Server Hello` message. This message contains a random number proposed by the server, one of the ciphersuites proposed by the client in its hello message, and a series of certificates. In this series, the first certificate is that of the server, and every certificate certifies the one before it. Assuming that the client has the public key to verify the authenticity of a certificate in the series, it can transitively verify all the certificates preceding it including the server's certificate. Thus, the server manages to authenticate itself with the client and the client obtains the server's public key.

TLS allows for either the RSA or the Diffie-Hellman key exchange mechanism to be employed for the ultimate generation of the master secret. If the RSA mechanism is to be employed then no additional information needs to be sent to the client. If the Diffie-Hellman key exchange mechanism is to be employed then the server also sends the two Diffie-Hellman parameters and its corresponding public key.

Diffie-Hellman Key Exchange: This protocol allows two nodes, A and B, to create a new secret. All communication can be over an insecure channel. Also, the participating nodes do not need to have any prior secrets. The protocol requires two integer parameters p and g. The first parameter, p, is a prime number. $0 < g < p$, such that for every value n in the range $[1, p-1]$ there exists a k such that $n = g^k \ mod \ p$. The parameters p and g may either be advertised by a server, or may be proposed by either A or B.
Node A selects a private key a and computes a public key $\alpha = g^a \ mod \ p$. Node B independently selects a private key b and computes a public key $\beta = g^b \ mod \ p$. A and B send their public keys to each other. Armed with their own private key and the other party's public key, both A and B can generate the same secret key $K = (g^a \ mod \ p)^b = (g^b \ mod \ p)^a$. Specifically, A computes K to be $\beta^a \ mod \ p$ and B computes K to be $\alpha^b \ mod \ p$.
Diffie-Hellman key exchange is susceptible to man-in-the-middle attacks. Let there be an intruder C that can intercept messages sent between A and B. C knows the parameters p and g. It intercepts α and sends $\gamma = g^c \ mod \ p$ to B on A's behalf, where c is the private key selected by C. Likewise, C intercepts β and sends γ to A on B's behalf. Now, the secret computed by A is $x = \gamma^a \ mod \ p$, while the secret computed by B is $y = \gamma^b \ mod \ p$. Node C can compute both these secrets. Messages sent from A to B will be encrypted using x, which can be decrypted by C using the same secret. Subsequently, C can encrypt the same message or a modified message using y and forward it to B. Messages sent from B to A can similarly be read and modified by C.
In essence, a secure channel is established between A and C and another between B and C. Hence, that the intruder C is not required to select the same secret key c for both the channels.

On receiving the `Server Hello` message the client responds with a `Client Key Exchange` message. If RSA is being used then the client sends a 48-byte premaster secret encrypted using the public key of the server. If Diffie-Hellman is being used: (i) the client sends its public key corresponding to the Diffie-Hellman parameters sent by the server, and (ii) uses its own and the server's Diffie-Hellman public keys to generate the shared premaster secret. Subsequently, the client computes the master secret on the basis of the premaster secret, its own random number (sent in the client hello message) and the random number received in the server hello message. The client also asks the server to switch over to the ciphersuite agreed upon by both of them. The client then sends a `Finish` message to the server. This message is the first *protected* message sent by the client using the master secret and ciphersuite agreed upon between the client and server.

On receiving the premaster secret, the server also computes the master secret us-

ing the premaster secret, and the client and server random numbers, and then switches over to the ciphersuite proposed by the client. Subsequently, it is able to verify the correctness of the Finish message received from the client, and responds with its own encrypted Finish message. When the client receives and verifies the server's `Finish` message the TLS tunnel formation is complete. From this point onwards the client and server can communicate with each other in privacy using the TLS tunnel. The authentication and encryption algorithms and the algorithm for the generation of message authentication codes (one way hash function used to determine message integrity) are determined by the ciphersuite agreed upon by the client and server.

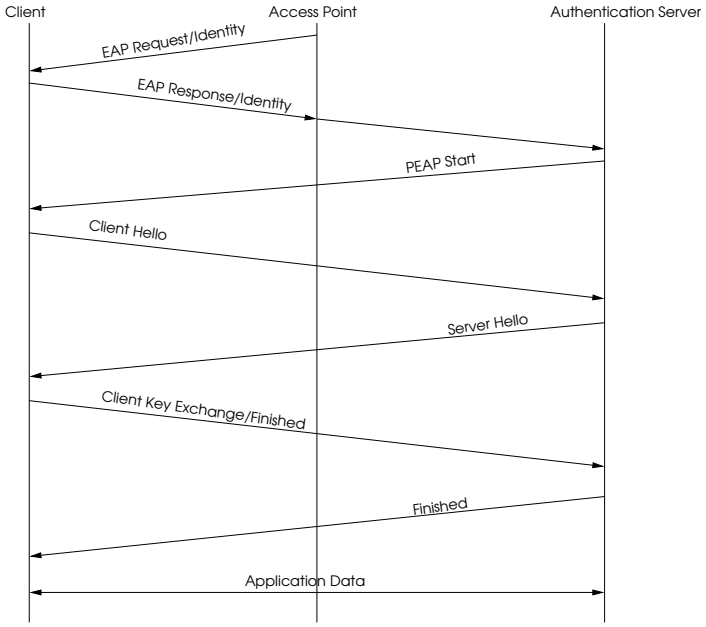


Figure 3.14: PEAP Part 1 message exchange to establish TLS tunnel.

Fast Session Resumption: A mobile client that moves from one BSS to another would like to maintain the same TLS session while it moves. The information exchanged in the initial stages of PEAP Part 1 can help achieve this. The `PEAP Start` message sent by the authentication server contains the server's identity. If this is the same server with which the client has established a session, the client sends the *session identifier* for that session in its `Client Hello` message. If the server finds this session identifier in its cache and is willing to continue that session it responds with the same session identifier in its `Server Hello` message. Following this both the client and server can move to the Finish messages and resume the session.

3.9.4 PEAP Part 2

At the conclusion of PEAP Part 1 the authentication server has authenticated itself to the client and a TLS session has been established between them for secure communication. Through this secure tunnel the client and the authentication server exchange messages to authenticate the client with the authentication server using one or more EAP methods. The corresponding message exchange is shown in Figure 3.15.

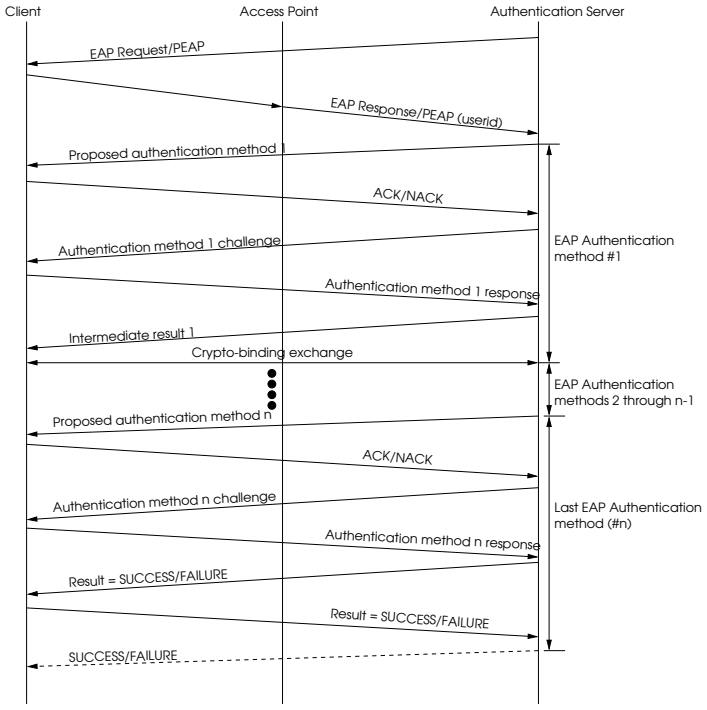


Figure 3.15: PEAP Part 2 message exchange for client authentication.

Initially, the authentication server sends an EAP Request message of type PEAP to the client. The client responds with its `userid`. After this a sequence of authentication methods are employed to authenticate the client. Each authentication method involves the following message exchange:

1. The authentication server proposes an authentication method.
2. The client can either accept it by sending an `ACK`, or reject it by sending a `NACK` and propose another method.
3. On receiving an `ACK`, or if the client's proposed method is acceptable to the server, the server sends a corresponding challenge message. The client sends its response to the challenge.

4. The server indicates its acceptance or rejection of the client's response through an intermediate result sent to the client. It can indicate either success or failure of that method. If this is the last EAP method then the server responds with a final SUCCESS or FAILURE response for the series of EAP methods used to authenticate the client.

5. Following the communication of the intermediate or final result the server sends a *cryptographic binding* message to the client, and the client responds with a similar method. The cryptographic method sent by the server (client) contains the PEAP version number received from the client (server, respectively) while negotiating the PEAP version in the beginning of PEAP Part 1. This enables the client and server to determine if the other party received the same version number that they proposed, and that the EAP method is negotiated between the same pair of nodes that negotiated the TLS session.

6. On receiving the *protected* SUCCESS or FAILURE message from the server following the last EAP method, the client responds with its own SUCCESS or FAILURE message. A client cannot respond with a SUCCESS if it receives a FAILURE message from the authentication server. Finally, the server responds with a SUCCESS or FAILURE message in the clear (outside the TLS tunnel) indicating the conclusion of PEAP Part 2. As the SUCCESS/FAILURE messages are exchanged as part of the TLS session, a third party cannot inject such messages, nor is it aware of the contents of these messages.

The use of cryptographic binding safeguards against version downgrade attacks. You may remember that the version negotiation messages in PEAP Part 1 are sent in the clear before the TLS session is established. So, there is a possibility that a malicious node may spoof the client or authentication server identity, and send a lower version number of the protocol on their behalf. This will result in a lower level of security being negotiated between the client and the authentication server. Such spoofing will become known to them when they receive a version number that is lower than what they proposed to the other.

Also, note that failure of any subset of the EAP methods does not necessarily mean that client authentication fails. The server may decide the subset of EAP methods whose success is critical for the overall success of client authentication.

At the conclusion of PEAP Part 2, the server and the client have authenticated each other. They also have a TLS session between them using which they may negotiate the cryptographic key and ciphersuite to be used for link-level encryption of communication between the client and the access point. If the access point and the authentication server are not colocated then it is the responsibility of the authentication server to communicate this information to the access point. Assuming that the access point and the authentication server already trust each other, it should be possible to perform this information transfer.

Chapter 4

Location Management

Location tracking in mobile computing systems deals with the issues of process and resource migration. The goal is to determine the node where a process is executing or a resource is located so that messages can be communicated and/or service requests can be sent. Recently, with the increasing popularity of wireless communication devices, location tracking has become a major area of interest in the mobile computing field. This article will concentrate on location tracking for wireless networks. However, several issues and solutions for wireless networks are also relevant for traditional distributed systems.

Wireless networks typically consist of several mobile and static nodes. The location of mobile nodes can change with time. If a network entity wishes to communicate with a mobile node, the former has to determine the current location of the latter to route messages. Therefore, mechanisms for mobility management and location tracking are needed.

Three important aspects of any location tracking mechanism are:

1. location update,
2. location query, and
3. paging.

Location update, also called *location registration*, refers to the actions performed by a mobile node to inform some designated network node(s) about its current location. These designated nodes are responsible for maintaining the database of location information for the mobile nodes. *Location query* is the act of retrieving the latest location information about a mobile node from the location database. This information may indicate the general vicinity of the mobile node. In such situations, a *page message* is multicast to determine the exact location of the mobile node.

Any location tracking mechanism should address the following issues:

1. *Placement of location information:* Should we store location information about all the mobile nodes in a centralized database, or distribute the information

among multiple location databases? The centralized approach is simple to specify, understand and implement. However, it is also prone to failures. Failure of a single database can paralyze the entire network because location information all MHs can neither be updated nor queried. Moreover, centralized solutions are not scalable. As the system grows more memory will be needed at the server to store location information. Also, the network in the vicinity of this server may be congested due to location update and query traffic.

Distributed approaches on the other hand are more robust and scalable. If location information is spread over a large number of servers, failure of a few servers will most probably affect only a subset of mobile nodes. However, distributing information efficiently requires some thought. Also, if we need to locate a mobile node which location server(s) should we query?

2. *Location update frequency:* Should the updates be performed periodically, or as a function of the mobility profile of the mobile node? If periodic location update is performed, the choice of the time between successive updates is of utmost importance. Infrequent updates will mean that most of the time *stale* location information is present at the server(s). Frequent updates will generate to much network traffic.

   Before we start worrying ourselves silly about the update period, we should be asking ourselves a more fundamental question: *Should the frequency of location updates be influenced by the mobility characteristics of the mobile nodes?*

   For a moment assume that I have a mobile computer/phone that I always carry with me.[1] In the morning I drive a few kilometers from home to school. Then, I spend the next several hours on campus sending emails, surfing the web, brewing and drinking coffee, teaching, and pretending to do research. Late evening I go back home where, too, I do the same things. This process is repeated almost every day. So, my location changes only twice a day: once in the morning and once late in the evening. Why should my mobile computer send its location information more than twice a day? On the other hand, my friend is always on the move, driving around the Dallas - Fort Worth area. Perhaps, his mobile node should update its location several times every hour. So, it makes sense to link location update frequency to the mobility characteristics of the node. But, the mobility characteristics of a node can change from time to time.

3. *Targets of location updates and queries:* Which components of a distributed location database should receive the updates and/or queries for a mobile node? The cost (in terms of response time and communication overheads) of location updates and queries will depend on the number of location servers in the system, their placement in the network, and how messages are routed to them. Also, the variance in the cost of location update and query operations will depend on the association between the mobile nodes and the location servers. If one location server is permanently associated with a mobile node, the cost of updating the

[1] Actually, I have been resisting the enticements to acquire a mobile phone. I shudder to think that if I succumb to their charms I will be left with very few excuses for not being reachable over the phone.

mobile node's location may vary drastically depending on how far the node is from its server. A dynamic association between a mobile node and the server that stores its location information may succeed in reducing: (i) the cost of location updates, (ii) the variance in the cost of location updates. However, if I want to contact that mobile node which location server do I ask for its location information?

4. *Accuracy of location information:* Is there a possibility of a location query returning erroneous location information about a mobile node? If so, what are the scenarios when such errors can take place? Also, what is the impact of such an error? For example, if the query returns no location information the querying node immediately knows that an error has occurred and may expand its region of search. If location query returns stale information, perhaps, maintaining forwarding addresses in regions recently visited may ultimately lead to the correct location. If the query returns an arbitrary location the problem may be more difficult to handle. How can such errors be avoided or rectified?

5. *Overheads:* What is the cost of location updates and queries in terms of storage requirements, number and size of messages, and the time required to complete the operation? It is obvious that any location management strategy would strive to incur minimum communication overheads, fastest response, and lowest possible storage requirements. However, achieving all these goals simultaneously is not possible. Usually, there is a tradeoff between response time, communication overheads and storage requirements. Various optimizations may be applicable to improve the overall performance of the system.

Note that maintenance of coarse grain location information about a mobile node does not require location updates to be performed very frequently. On the other hand, fine grain location information maintenance saves paging costs at query time while requiring frequent location updates.

Similarly, maintaining multiple replicas of location information in a distributed database may speed up location queries and provide fault tolerance. However, this is at the cost of increased effort during location updates and greater storage requirements.

4.1 System Model

Let us consider a system composed of several cellular networks. The wireline part of the network contains nodes such as the *home location register (HLR)* and the *visitor location register (VLR)* that maintain the location database. *Switching centers* in a network perform connection switching when calls are placed, and manage the wireless channel resources. The mobile nodes can move from one cell to another within the same network, or from one network to another.

The coverage area of a network is divided into several *registration areas (RAs)*. A cluster of neighboring cells constitute an *RA*. Each mobile node is associated with an *HLR*. The *HLR* maintains information about the location, types of subscribed services, billing, etc. of all its subscriber mobile nodes. There may be several *VLRs* in the

network. A *VLR* may serve multiple *RAs*. It stores location information about mobile nodes that are currently in its *RAs*.

The performance of various mobility management solutions depends on the degree of mobility of the mobile nodes and the frequency with which their location is queried. The relation between mobility and query rate can be expressed as the *call to mobility ratio (CMR)* defined as follows: the ratio between the number of location queries made for a mobile node and the number of times that node moves from one registration area (RA) to another during a fixed period of time.

4.2 Location Tracking Standards

Mobile computing systems based on cellular networks can use the location management solutions developed for cellular telephony. So, first let us briefly state the cellular telephony standards for location management.

In North America, the Interim Standard-41 [18] of the Electronic Industry Association/ Telecommunications Industry Association is followed for the Advanced Mobile Phone System (AMPS) and the Personal Access Communication System (PACS) networks. The GSM-MAP standard [39] followed in Europe for GSM and Digital Cellular Systems is similar to IS-41. Both the standards assume the existence of *HLRs* and *VLRs* in the network.

4.2.1 Location Update Operation

A mobile node initiates a location update when it moves from one *RA* to another. If the new and old *RAs* are served by the same *VLR*, the *VLR*'s database is updated to reflect the mobile node's presence in the new *RA*. If the new and old *RAs* are served by different *VLRs* then the following steps are executed:

1. The mobile node registers with the new VLR.
2. The new VLR sends a message to the mobile node's HLR.
3. The HLR replaces the old *VLR* with the new *VLR* in its record of the mobile node.
4. The HLR sends a *deregistration* message to the old VLR and a *registration* message to the new VLR.
5. The mobile node's information is deleted from the old *VLR* (*deregistration*) and added to the new *VLR* (*registration*). Acknowledgments are sent to the HLR.

4.2.2 Location Query Operation

Each mobile node has a unique identifier. Let a query be made for a mobile node. The switching center serving the querying node uses the unique identifier of the mobile node as the key to perform a table look-up in a *global title translation table*. This

operation yields the address of the mobile node's *HLR*. Then the mobile node is located through the following sequence of operations:

1. A request is sent to the mobile node's HLR.
2. The HLR forwards the request to the VLR where it knows the mobile node to be registered.
3. The VLR forwards the request to the switching center serving the *RA* where the mobile node is present.
4. However, the mobile node's switching center does not know the exact cell in which the mobile node is present. Therefore, a *paging* message is broadcast by the base stations of all the cells in the mobile node's *RA*.
5. Only the target mobile node sends its base station a response to the page.
6. The response is forwarded by the base station to the switching center of the mobile node's cell. This enables the *RA*'s switching center to determine the exact cell of the mobile node.

4.3 Performance Enhancement Strategies

Location tracking costs are dependent on factors like the distance of the mobile node from its *HLR*, and the *call to mobility ratio (CMR)* of the mobile node. These costs are progressively increasing with the increasing market penetration of wireless communication devices. Hence, a need has been felt for the development of low cost scalable location tracking mechanisms. Performance improvement can be achieved in the following ways: (i) making better decisions about when to update location information, (ii) improved location update and query strategies, and (iii) developing entirely new alternatives to the HLR/VLR based location management solution.

4.3.1 Efficient Update Decisions

In the current standards, fixed size *RAs* are used for all the mobile nodes regardless of their mobility characteristics. It reminds me of the "one size fits all" approach taken by all the airlines when it comes to designing their seats - and am I wrong in feeling that the seats do not fit anybody at all? Anyway, the "one size fits all" approach to location management can result in excessive location updates or long intervals between updates. Bar-Noy and Kessler [8] analyzed the following alternatives:

1. *Time-based:* Perform location updates every T time units.
2. *Movement-based:* Perform a location update when a mobile node has crossed a certain number, say M, of cell boundaries.
3. *Distance-based:* A mobile node tracks the distance (in terms of number of cells) it has moved since its last location update. When this distance exceeds a predetermined value, D, a location update is performed.

The time-based strategy is the simplest to implement. The distance-based strategy is the most difficult to implement as it requires knowledge of the network topology on the part of the mobile node. When the movement of mobile nodes is assumed to be identical and independently distributed (memoryless) the distance-based strategy has the best performance and the time-based strategy has the worst performance.

4.3.2 Improved Location Update and Query Strategies

4.3.2.1 Location Information Caching

The location information about a mobile node, retrieved by a query operation, is cached at or near the switching centers of the querying node [28]. Subsequent location queries for the same mobile node can be satisfied by accessing the cached data first. If an entry for the mobile node exists in the cache, the *VLR* specified in that entry is queried directly. If the mobile node is still served by the same *VLR* (a cache hit), the location query is accomplished by a single cache look-up. If the mobile node is no longer served by that *VLR*, or if no cache entry for the mobile node exists, the location query operation, described earlier, is performed.

Therefore, caching reduces location query cost on a cache hit, while increasing it if the cached entry is stale. The greater the mobility of the node, the shorter the period of time for which it stays registered at a *VLR*. If the number of location queries made between two successive location updates is above a threshold, the traffic savings due to cache hits more than offset the penalties incurred on cache misses. Moreover, a lifetime is associated with each cache entry. The entry is purged once it has spent its lifetime in the cache. This helps keep the number of cache misses under check.

4.3.2.2 Pointer Forwarding

Let us assume that we make very smart decisions about when to update the location information of a certain mobile node at its HLR. However, what is the point of making such updates if other nodes hardly ever query that mobile node's location? Isn't the problem exacerbated if this particular mobile node also happens to be quite far from its HLR? All this effort maintaining location information, and what for? People realized that if the cost of location updates could be minimized, performance could be improved even if the *call-to-mobility (CMR)* ratio was low. Pointer forwarding [29] is one approach to achieving this goal.

When a mobile node moves out of an old *VLR*'s area into a new *VLR*'s area the HLR is not informed about the move. Instead, the old *VLR* sets a pointer to the new *VLR*. The cost of informing the *HLR* is avoided. The farther the mobile node is from its HLR, the greater the savings.

We may end up with a chain of *VLRs* linked by forwarding pointers. A location query results in the *HLR* forwarding the query to the first *VLR* in the chain. This is followed by a traversal of the chain until the *VLR* currently serving the mobile node is reached.

Imagine a situation where a mobile node changes *VLRs* infrequently and is queried much more often (high *CMR*). If the chain of forwarding pointers is long, the occasional

savings in location update cost are more than offset by the increased location query costs that are incurred more often. Therefore, a limit is placed on the maximum length of the forwarding pointer chain. When this length is exceeded, the entire chain is broken and the mobile node's *HLR* is informed about the *VLR* with which the mobile node is currently registered. Subsequent moves of the mobile node will result in the formation of a new chain of forwarding pointers starting at this VLR.

4.3.2.3 Static and Dynamic Local Anchoring

Like pointer forwarding, local anchoring [26] is also an approach to reduce location update costs. A *VLR* close to the mobile node is designated as the local anchor for that node. The *HLR* records information about the current local anchor of the mobile node. In the case of static local anchoring, between successive location queries for a mobile node, all location updates are sent to the local anchor instead of the *HLR*. When a location query for the mobile node originates the following actions are performed:

1. The mobile node's HLR sends the query to the local anchor of the mobile node.
2. The local anchor knows the VLR with which the mobile node is currently registered and forwards the query to that VLR.
3. The VLR serving the mobile node's *RA* becomes the new local anchor.

Thus, local anchors can change only due to location queries made during connection set-up. In dynamic anchoring, the local anchor may change between two successive location queries as well, as a function of the mobile node's *CMR*. In both the anchoring schemes, the mobile node is usually closer to the local anchor than the *HLR*. Hence, the communication overheads of location updates are reduced.

4.4 Alternatives to HLR/VLR Scheme

The strategies described above are enhancements of the *HLR/VLR* based scheme described by the standards. Let us now consider two alternatives that do not employ the *HLR/VLR* based scheme. They maintain a distributed database of location information.

4.4.1 Awerbuch and Peleg's Solution: Hierarchical Database

The goal of this location management scheme [5] is to design a *partial-information* strategy that will perform well for any combination of mobility and query patterns. The cost of location updates and queries are sought to be reduced.

There are multiple location servers in the system, configured as a tree. The mobile nodes correspond to leaves of the tree while the internal nodes are the regional location servers. Location information at a regional server is recursively defined as the union of the location information at its children nodes.

For a given mobile node it is cheaper to update its location information at location servers that are closer to it, than to perform updates at servers that are far away.

Consequently, more recent and accurate location information is maintained at location servers close to the mobile node. As the distance from the mobile node increases, location information becomes less accurate. Thus the hierarchy of location servers reflects the accuracy of location information. Awerbuch and Peleg associated a level value with the nodes of the tree. The level increases monotonically from the leaves to the root. If there are N location servers in the system there are $O(\log N)$ levels in the tree. The purpose of the regional location server at level i of the hierarchy is to enable a potential querying node to track any mobile node currently within distance $O(2^i)$ from it.

Whenever a mobile node moves a distance d only the regional databases corresponding to the lowest *log d* levels on the path from the mobile node to the root are updated to reflect the new location. The higher level databases continue to store outdated information.

A location query starts at the lowest level database nearest to the querying node. If the queried database does not have the location information about the mobile node, the search is widened and the database's parent in the tree is queried. This process is repeated until the location information is found.

Queries originating close to the mobile node access lower level databases and are able to retrieve current location information quickly. Queries originating at a greater distance have to access higher level directories and may obtain outdated information. This problem can be handled by using a technique similar to forwarding pointers.

The time complexity of the query operation is upper bound by the height of the tree because the root has location information about every mobile node. In practise, very few queries go all the way to the root as most calls to a mobile node originate in that node's vicinity.

4.4.2 Prakash, Haas and Singhal's Solution: Quorum Based Approach

This solution [44, 46] is motivated by the desire to balance the load among the various location servers. In the HLR/VLR scheme if a large number of mobile nodes are concentrated in the area covered by a particular VLR that VLR will be swamped by location updates sent by the mobile nodes and location queries forwarded to it by the HLR(s) of the nodes. Similarly, in the hierarchical approach, if a large number of mobile nodes are concentrated in a region the location servers in the tree serving that area will be overloaded. Even if the mobile nodes are evenly distributed, if some mobile nodes are queried very often the location servers handling them will be overloaded.

The quorum based approach employs dynamic along with location update and query quorums to provide load balancing among the location servers. Moreover, unlike the hierarchical scheme in which a query may have to ascend multiple levels of the tree, location information can be queried and updated with a single round of messages.

The location databases are grouped into several overlapping subsets referred to as *quorums*. Each quorum is identified by its quorum number. The quorum of location servers storing the location information of a mobile node is a function of the identities of the mobile node as well as the cell in which the mobile node is present. Such a function can be represented as follows: $h : BS \times MH \rightarrow S_{LS}$, where BS denotes

the base station of the cell in which the mobile node MH is present, and S_{LS} denotes a quorum of location servers. The location servers corresponding to an MH will change as the MH moves in the network from one *registration area* (RA) to another. A timestamped location update is sent as a single multicast message to the quorum of databases so determined.

Function $h()$, described above, is also employed to determine the quorum of location servers that should be queried when an MH is to be located. The quorum corresponding to $h(BS, MH)$ represents the set of location servers that a node, in the cell represented by BS, should query when it wishes to locate a mobile host MH. As every pair of quorums intersect, at least one of the queried databases will return the latest location information. Thus, location update and query can be performed in a single round of messages.

If the location of a particular mobile node is queried from two different cells, it is very likely that the two queries will be sent to different quorums of location servers. Similarly, when two mobile nodes in the same cell send location updates their updates are likely to be sent to different quorums. This even distribution of load among quorums can either be achieved by using a hash function or by randomly selecting a quorum for location updates and queries. If dynamic hashing is employed in quorum selection, location servers can be added and removed from the system as the overall load increases and decreases, respectively.

The quorum based approach provides user profile replication in a distributed fashion. The degree of replication depends on the quorum construction scheme. Some simple quorum construction schemes can generate quorums of size $O(\sqrt{N})$, where N is the total number of location servers [36].

Regardless of the solution strategy adopted, an acceptable solution should have the following properties: (i) scalability, (ii) adaptability in the face of widely changing update and query patterns, (iii) robustness under unfavorable operating conditions, and (iv) load balancing among location databases.

Chapter 5

Routing in Mobile Systems

In Chapter 4 we discussed various methods of locating mobile nodes in a cellular environment. In our daily lives we usually try to locate somebody if we are interested in communicating with them.[1] In order to communicate with another entity we need a means to route our messages (packets) to them. Several approaches towards routing are possible. One possibility is that the sender of the message specifies the exact path that the message should take on its way from the sender to the receiver. Of course, this assumes that the sender knows the topology of the network and the current traffic situation. Another alternative is to forward the message to a neighbor in the general direction of the destination. The neighbor then makes a similar decision regarding how to route the message.

Several routing algorithms for fixed wireline networks exist. However, in this chapter we will consider a new kind of network, which we will refer to as the Mobile Ad Hoc Network (MANET). A MANET is an autonomous group of mobile nodes. Each node has one or more wireless interfaces of limited communication range. A MANET node can directly communicate with other nodes in its communication range. Those MANET nodes that are not within direct communication range of each other exchange messages along a multi-hop path comprising other MANET nodes. Thus, each MANET node acts as a host as well as a router. Due to the mobility of MANET nodes, network topology is dynamic.

Do you think routing algorithms for fixed wireline networks can be employed for routing in MANETs? The answer, as you may have guessed, is *no*? The culprit is the ever changing network topology. Several routing algorithms are based on the determination of the shortest path between the source and the destination nodes of an undirected graph where the edges correspond to links between adjacent nodes. In MANETs the graph is always in a state of flux: neighboring nodes may move apart disrupting old links, and previously separated nodes may come sufficiently close to each other forming wireless links.

Changes in graph topology in one region of the network may alter the shortest paths

[1] How many people do you know who strive to memorize addresses of total strangers whom they never expect to meet? I have some friends who do that. But, I hope you move amongst a more "normal" and "better adjusted" people that I do!

between several pairs of nodes. Unfortunately, this information takes time to propagate to other nodes in the system. In the interim they may continue to route messages based on outdated information. This may lead to inefficient routing. Sometimes this may lead to situations where the message keeps going in a loop without ever reaching the destination.

Looks like we are in real trouble...time to cancel the course, sell the book to the next available sucker, and go home!

Fear not for I bring thee good news. A number of smart people set about solving the problem. You, the smart reader, know that the first step towards developing a solution is to have a better understanding of the problem. So, let us first discuss the constraints on MANET route discovery, as described in [14].

1. *Dynamic topologies:* As mentioned earlier, due to the potentially arbitrary movement of nodes, the network topology may change in unpredictable ways. This change in topology is not limited to the creation or deletion of bidirectional wireless links. Some bidirectional links may become unidirectional, or vice versa. Similarly, some unidirectional links may get created.

2. *Bandwidth-constrained, variable capacity links:* Wireless links support a significantly lower bit-rate, when compared to their wired counterparts. This disparity in link capacity is likely to persist in the foreseeable future. Moreover, the capacity of wireless links is also affected by factors like fading, background noise, collision between concurrent transmissions by two nodes that are within range of each other, etc. Hence, the bit-rate that can be supported between two neighboring nodes may fluctuate wildly over time.

3. *Energy-constrained operation:* Most MANET nodes do not have the luxury of being plugged into the electrical supply network. They have to make do with whatever energy is stored in their battery packs between recharges or battery replacement. Hence, every attempt should be made to minimize energy consumption.

4. *Limited physical security:* Wireless links are prone to eavesdropping, jamming and spoofing. While eavesdropping can be minimized by employing the security measures discussed in Chapter 3, jamming of the communication medium can prevent nodes from exchanging routing information. In addition, unauthorized nodes may be able to inject false topology information into the MANET and mount denial-of-service attacks.

As you may realize, any routing protocol for MANETs should be respectful of the constraints mentioned above. A protocol that cannot handle topology dynamism may end up generating a number of messages. These messages will consume a significant amount of network bandwidth and drain the energy supply of MANET nodes. A protocol that is not secure may, likewise, consume a significant amount of network resources. This has a direct impact on *scalability*. After all, it is desirable to having routing protocols that can operate efficiently in a large MANET for an extended period of time.

There are some desirable properties of routing algorithms for MANETs [41, 14]:

1. *Distributed control:* A centralized routing scheme, in which one node has all the topological information and makes all routing decisions, is neither robust, nor scalable. The central router is a single point of failure. Also, the network in the vicinity of the central router may get congested with routing queries and responses.

2. *Loop-free routing:* Presence of loops in the path from the source to the destination result in inefficient routing. In the worst case situation the packets may keep traversing the loop indefinitely and never reach their destination. Of course, the performance penalty due to loops can be limited through a judicious setting of the TTL value when the packet is created. Still, the unnecessary forwarding of packets until the TTL value reaches zero results in wastage of network resources.

3. *Reduced overheads with reactive (on-demand) route discovery:* It is imaginable that only a subset of MANET nodes would communicate with each other at a given time. So, why should all nodes incur the communication and energy overheads of discovering routes to all other nodes? Instead, a node, X, should try to find a route to another node, Y, only when X has packets to send to Y. Thus, route discovery is driven by the need for routing information.

4. *Fast routing with proactive route discovery:* The problem with reactive route discovery is that until node X finds a route to Y, packets meant for Y are buffered at X. This results in an increase in packet forwarding latency. Also, if packets for Y arrive in a burst, the buffer at X may fill up fast, and even overflow. Therefore, the quicker routing decisions are made the sooner the packets can be routed towards the destination. This can result in reduced latency and avoid buffer overflows. A proactive route discovery can achieve such goals.

5. *Localized reaction to topological changes:* Topological changes in one part of the network should lead to minimal changes in routing strategy in other distant parts of the network. This will keep the routing update overheads in check and make the algorithm scalable.

6. *Multiplicity of routes:* Even if node mobility results in disruption of some routes, other routes are available for packet delivery. This is especially important for tactical mobile ad hoc networks where reliability and low latency are extremely important.

7. *Security:* As mentioned earlier, wireless networks are susceptible to eavesdropping, jamming and snooping. It is important that MANET routing protocols be able to withstand attacks by malicious adversaries.

8. *Sleep period operation:* A MANET node may not have messages to exchange with other nodes, and may decide to conserve its energy by going into a *sleep* state. While in the sleep state, the node does not participate in route discovery, nor does it act as a router. However, sleep period operation can be a mixed blessing. As far as other nodes are concerned, this results in a change in network topology and disruption of some existing routes. They may have to incur additional route discovery overheads. Is the energy savings of one node offset by the

additional overheads incurred by other node(s)? Moreover, when a node *wakes up* it may find that all its routing information is stale and may have to discover routes afresh. The net performance gain due to sleep period operation would depend on the number of nodes that sleep, the frequency of transition to the sleep state, and the duration of the sleep period.

9. *Unidirectional link support:* Most routing protocols implicitly assume that all links are bidirectional. This is usually the case for wired networks, and is at the heart of *reverse path forwarding*. However, in wireless networks some links may be unidirectional.[2] If only a small fraction of links are unidirectional, it may be possible to ignore such links and find routes between pairs of nodes that consist only of bidirectional links. However, there may be situations where the underlying network topology is *strongly connected*, but it is impossible to find a path from some node A to another node B composed only of bidirectional links. In such situations it may become necessary to use unidirectional links to forward packets from A to B.

Having discussed some of the desirable properties of routing protocols, a logical question that arises is how do they get reflected in the performance of applications running on MANETs. Specifically, what metrics to use to measure the effectiveness of various MANET routing protocols? Throughput and latency are the obvious metrics for any network protocol's performance. These and some additional metrics that provide insights into the performance of MANET routing protocols are discussed next.

1. *End-to-end throughput and delay:* Throughput of data transfer from a source node to a destination node is the number of bits delivered from the source to the destination per unit time. Delay is the time taken for a packet to travel from the source to destination. Specifically, delay is the elapsed time between a packet appearing in the network layer queue at the source and that packet being offered by the network layer to the transport layer at the destination. It is important that throughput and delay of any routing protocol be considered in tandem. Consider a scenario in which a significant number of packets forwarded along routes discovered by a given routing protocol are dropped. The remaining packets are delivered to the destination with low latency. If one were to look only at latency, the routing protocol may appear to be quite acceptable.

2. *Route acquisition time:* This is the time for a node to discover a route to another node. This metric is especially important for reactive route discovery protocols. As mentioned earlier, reactive protocols initiate route discovery for a node only when they have packet(s) to send to that node. Route acquisition time indicates the time for which packets have to be buffered at the node doing the route discovery. If such packets arrive at the route discoverer in quick succession, while it is trying to find a route, they may not be space in the buffer to accommodate them.

[2]Let node X be able to receive transmissions of a neighboring node Y, but not vice versa due to high interference in Y's vicinity. Or let node X, in its attempt to reduce its energy consumption, transmit at a lower power than neighbor Y. Once again, X is able to receive Y's transmissions, but not vice versa. These situations result in a unidirectional link from Y to X.

So, some packets will be discarded. This will adversely affect the throughput. If a long lived flow of packets have to be forwarded from source to destination, the route acquisition time for reactive protocols is reflected in the latency for only the packets in the beginning of the flow. Later packets do not experience this latency as the source already has a route to the destination by the time they appear at the source. Hence, the average latency may appear to be low. However, for short flows, where the route acquisition time is not amortized over a large number of packets, the latency of the same path may appear to be high!

3. *Percentage of out-of-order delivery:* If packets are forwarded from a node X to node Y along multiple routes, it is likely that they may arrive out of order at node Y. IP forwarding makes no guarantees about in-order delivery of packets to the destination. However, out-of-order delivery of packets has a significant impact on transport layer performance. Readers familiar with the Transport Control Protocol (TCP) will recall that TCP acknowledgments inform the sender about the next byte expected by the destination. Out-of-order delivery of packets results in the sender receiving duplicate acknowledgments. If the sender receives three duplicate acknowledgments, it may trigger *fast retransmission and fast recovery*. This results in unnecessary retransmission of some packets and also reduces throughput.

4. *Efficiency:* This indicates the overheads of the routing protocol, and may not necessarily impact the performance of applications that use the discovered route to forward their data. Efficiency of a routing protocols measures the volume of control traffic, for example, the number and size of route discovery, route error and route notification messages. If control and data traffic share the same low data rate channel, a high volume of control traffic is likely to adversely impact throughput and latency of data traffic. Some measures of routing efficiency are:

   (a) *Ratio of average number of data bits transmitted and delivered:* Each time a packet is transmitted by a node along a multi-hop path to the destination, the count of the number of bits transmitted is increased accordingly. Only when a packet is delivered to the destination is the count of the number of bits delivered increased. Hence, this ratio indirectly denotes the number of hops on the path from source to destination. Note that this metric measures network layer performance. So, even though a frame may have to be transmitted multiple times at the data link layer to be forwarded along a link, it is counted as one transmission of the corresponding network layer packet.

   (b) *Ratio of average number of control bits transmitted to data bits delivered:* This is indicative of the control overheads. The control bits include the messages of the routing protocol as well as the headers of the data packets.

   (c) *Ratio of the average number of control and data packets transmitted to data packets delivered:* This metric denotes the routing protocol's channel access efficiency at the MAC sub-layer.

Given the set of metrics discussed above, it is possible to construct an experimental scenario in which a given routing protocol outperforms all other routing protocols. Un-

fortunately, the MANET literature has its own share of papers where the experimental set-up has been *rigged* to favor the protocol proposed by the authors of those papers. A credible evaluation of a routing protocol's performance should consider a variety of scenarios. Some of the parameters that can be varied to construct different scenarios are:

1. *Network size:* measured by the number of nodes in the network. A protocol that performs well for small networks, but not when the sizes grows is definitely not a scalable protocol.

2. *Network connectivity:* indicates the average number of neighbors that nodes have in the MANET. Networks with high connectivity may have multiple routes between pairs of nodes. Protocols that can discover these routes may perform better than those that only discover one route at a time.

3. *Mobility pattern and topological rate of change:* indicates the direction and speed of movement of nodes, duration of such movement, and intervening periods of no movement. This influences the rate at which new links are formed and existing ones are broken. It is indicative of the longevity of routes. Highly dynamic networks may have short-lived routes, and may incur high route discovery overheads. Note that it is the relative movement of nodes (in addition to the terrain in which the MANET is operating) that impacts topology, and not just the absolute speed of nodes.

4. *Link capacity:* indicates the effective bit-rate that can be supported on links, after subtracting the overheads due to MAC sub-layer collisions, packets headers, etc. Protocols with high volume of control traffic will perform especially badly when running in a network with low-capacity links. On the other hand, if all links have high capacity and the offered traffic is well below link capacity, control traffic overheads may not a noticeable impact on performance.

5. *Fraction of unidirectional links:* A protocol that implicitly assumes all links to be bidirectional may yield wrong routes in the presence of unidirectional links. A protocol that ignores all unidirectional links during route discovery may yield routes that have a higher hop count than one that does use these links, especially as the fraction of unidirectional links increases. Moreover, two routing protocols that are both capable of using unidirectional links may not show different performance trends as the fraction of such links is varied.

6. *Traffic patterns:* denotes the distribution of data packets over time. Proactive protocols incur the same route discovery overheads, regardless of the traffic pattern. However, reactive protocols' overheads are impacted by the nature and duration of data bursts. For example, if only a small fraction of MANETs have long-lived flows between them, reactive protocols are likely to incur significantly lower routing overheads. However, everything else staying unchanged, if the data traffic is composed of several short bursts between different pairs of nodes, the routing overheads of reactive protocols may be higher.

7. *Fraction and frequency of sleeping nodes:* impacts the topological rate of change and the resultant longevity of routes. As discussed earlier, each time a node goes to sleep it stops acting as a router to forward other nodes' packets. Whenever a node wakes up, a new node becomes available to the rest of the network for forwarding packets.

There are solutions to the routing problem. The solutions can be classified into two broad categories:

1. *Flooding:* The sender sends a copy of the message to every neighbor. The neighbors then propagate copies of the message to their all their neighbors except the one from which they received the message. This process is repeated until the network is entirely flooded with the message. If the destination node is in the same network partition as the source the message is sure to reach the destination.

   Flooding incurs high communication overheads. However, its primary advantage is its simplicity. Nodes do not have to maintain any information about network topology.

2. *Topology aware routing:* Nodes use their knowledge of the network topology to route messages. There can be several ways of implementing this policy. One possibility is that each node determines the optimal path to every other node in the system and stores this information. Each time a stream of packets have to be sent from a source to a destination a connection is established between two endpoints and all the packets follow this path. However, with changing topology nodes will have to update their routing information, and reestablish paths that were broken during communication. If the network topology does not change very often it is likely that the path establishment costs are incurred once in the beginning and every subsequent packet is routed without additional overheads.

   The second possibility is connectionless routing where a route is determined on the fly for every packet as it moves from one node to another. It is hoped that this will require nodes to store less information about network topology. However, every packet incurs the routing overheads.

5.1 Issues in Routing

As mentioned above distributed routing is desirable due to its robustness and scalability. However, there can be several approaches towards distributed routing, each with its advantages and disadvantages. Moreover, distributed control does not mandate that every node should maintain topological information and make routing decisions. Perhaps, the responsibility for selecting routes can be shared among a set of distinguished nodes resulting in a hierarchical routing strategy.

5.1.1 Distributed Routing

Several distributed routing algorithms for ad-hoc networks have been proposed. In every algorithm a node makes routing decisions based on its local knowledge. Typically,

every node sends its local connectivity information to its neighbors. The neighbors propagate this information to their neighbors. Thus, the connectivity information diffuses through the network. In some cases besides connectivity, information about the traffic on links, buffer sizes at nodes, available bandwidth, etc. are also propagated.

On receiving topology information from the neighbors, a node updates its routing tables. In the *distance vector routing* algorithm the routing table at a node has an entry for every other node. The entry for a destination node contains the cost of routing a packet to that node and the neighboring node to which the packet should be forwarded on way to the destination. This algorithm was initially adopted for routing in the Arpanet. But, it was later discarded due to the *count to infinity* problem and the high cost of local routing information broadcast. However, in ad-hoc networks where local broadcasts over a wireless channel are no more expensive than a unicast the latter disadvantage disappears.

5.1.2 Hierarchical vs. Flat Routing

In hierarchical routing schemes the set of nodes is divided into several clusters. Every cluster has one node which is designated as the *cluster-head*. So, every node is either a cluster-head or one wireless hop away from a cluster-head. A node that is not a cluster-head, but adjacent to more than one cluster-head is referred to as a *gateway*. Packets between cluster-heads are routed through gateways. Nodes that are neither cluster-heads, nor gateways are referred to as *ordinary* nodes. The subnet comprising the cluster-heads and gateways is referred to as the *backbone network*. Each cluster-head maintains information about other nodes in its cluster.

From time-to-time, this information is exchanged between cluster-heads over the backbone network. Thus, the cluster-heads gather network topology information. A node that has a packet to send to another node can obtain routing information from its cluster-head. It is not necessary for a packet to be routed through the backbone network. Data packets may be routed along other more efficient routes in the network. By restricting the number of nodes that participate in making routing decisions the overheads of maintaining routing tables are reduced. However, remember that cluster-heads may become ordinary nodes/gateways, and vice-versa due to changes in network topology. In such a situation the newly anointed cluster-head(s) have to obtain topology information and the older cluster-heads have to purge their information.

How cluster-heads are elected can significantly impact the performance of the routing algorithm. During the transition between ordinary node and cluster-head status care has to be taken not to propagate inconsistent routing information. Thus, the reduction in the overheads of maintaining routing information is at the cost of increased complexity.

As opposed to hierarchical routing, in flat routing schemes all nodes are treated equally and they all participate in routing packets.

5.2 Hierarchical Routing Algorithms

Having discussed the various issues in routing let us now look at some routing algorithms that have been proposed. We will first look at some hierarchical routing algorithms, and then discuss some flat routing algorithms.

5.2.1 Linked Cluster Algorithm

This algorithm was proposed by Baker and Ephremides [7] and later modified by Ephremides, Wieselthier and Baker[20]. This algorithm triggered considerable research in the area of routing in packet radio networks.

The *Linked Cluster Algorithm* (henceforth referred to as LCA) is a distributed algorithm in which each node's actions are based on its local information. Nodes are classified as *cluster-heads*, *gateway nodes* and *ordinary nodes*.

LCA is a two-step process:

1. *Formation of clusters:* Nodes exchange information with their neighbors in multiple rounds. At the end of this exchange each node is either a cluster-head or one hop away from a cluster-head.

2. *Linking of clusters:* Links interconnecting cluster-heads are identified and the backbone network is formed. Gateways, which are non cluster-head nodes and are part of the backbone network, are identified.

5.2.1.1 Data Structures

Each node maintains the following data structures:

Connectivity: This is an $n \times n$ binary matrix, where n is the number of nodes in the system. If the element in the (i, j) location is 1 it means that there exists a wireless link between nodes i and j. If the value of the element is 0 it indicates the absence of a wireless link between i and j. It is quite likely that at any given time different nodes may have different values in the matrix. This is due to two reasons: (i) the network may be partitioned, (ii) connectivity information takes time to propagate through the network.

Nodestatus: It can assume one of three values: *head*, *gateway*, or *ordinary*.

Ownhead: This stores the identity of the node's cluster-head.

Nodesheard: A list of all neighboring nodes that this node can hear.

Headsonehopaway: A list of cluster-heads that are neighbors of the node.

Headstwohopsaway: A list of cluster-heads that are not directly connected to the node, but are connected to the node's neighbors.

5.2.1.2 Cluster Formation

The following description is based on the clustering algorithm described in [7]. The cluster formation step of LCA lasts two TDMA frames. There are n slots in each frame: one per node. It is assumed that the clocks of all the nodes are synchronized with a certain degree of accuracy.

In the first frame node i broadcasts in the i^{th} slot. The contents of the broadcast are its Nodesheard list, $i.e.$, the nodes it has heard from during the earlier slots of this frame. In the later slots of the frame node i receives the broadcasts of some of the nodes with higher identities. The information received in the broadcasts is used to partially build the Connectivity matrix. Let there be a node j such that $i < j$. If node i heard from node j in the j^{th} slot and node i is part of the Nodesheard list of j, then it sets $Connectivity_i(i, j)$ to 1. As you can see, entries in the Connectivity matrix indicate bidirectional connectivity. This property is noteworthy because wireless links are not necessarily bidirectional.

In the second frame node i once again broadcasts in the i^{th} slot. However, this time i broadcasts the i^{th} row of its Connectivity matrix. By this time node i has complete information about all its bidirectional connectivities. As explained in the previous paragraph, by the end of the first frame i knows about its bidirectional connectivity with all nodes with greater node identities than its own. Let there be a node j such that $i > j$. In the j^{th} slot of the second frame node j transmits the j^{th} row of its Connectivity matrix. If the i^{th} element of this row, as heard by i, is equal to 1 then i sets $Connectivity_i(i, j)$ to 1.

Therefore, by the end of the second frame each node is aware of all its bidirectional connectivities as well as the bidirectional connectivities of all its neighbors.

Nodestatus determination: Node i also transmits its Nodestatus information in the i^{th} slot of the second frame. A node with the highest identity in a neighborhood becomes a cluster-head for that neighborhood. Determination of status is performed according to the following rules:

1. Node i checks its connectivity row in the matrix. If all neighbors have lower node identities then i declares itself to be a cluster-head.

2. If there exist other nodes with higher identities then the highest numbered node will become a cluster-head.

3. However, if there exists a node $j < i$ such that i is the highest identity neighbor of j then i must become the cluster-head for j. This determination is done based on the contents of the row broadcast by neighbor j in its second frame.

If the cluster-head selection rule were changed so as to select the lowest identity nodes as cluster-heads then additional frames would be needed to determine Nodestatus.

At the end of the second frame each node constructs its Headsonehopaway and Headstwohopsaway lists. Headonehopaway list is constructed on the basis of the Nodestatus values heard from the neighbors during the second frame. As stated earlier,

a node is aware of the bidirectional connectivities of all its neighbors at the end of the second frame. Based on this information it identifies the cluster-head of its neighbor. If the cluster-head of the neighbor is not in the Headsonehopaway list of the node then that cluster-head is placed in the Headstwohopsaway list.

5.2.1.3 Linking of Clusters

Having identified all cluster-heads and their clusters, the next step is the identification of gateways to construct the backbone network. Every node that is not a cluster-head is a candidate for becoming a gateway. Three possible scenarios of connecting cluster-heads exist:

1. *Two cluster-heads are neighbors of each other.* There is no need for a gateway between them as they are connected directly.

2. *Two cluster-heads are not neighbors of each other. However, there exist (possibly multiple) non-cluster-head nodes that are neighbors of both the cluster-heads.* Each node i tests all node pairs in its Headsonehopaway list. For each pair i determines if these nodes are connected to each other through a third cluster-head. This determination can be performed as node i is aware of the neighbors of its neighbors. If the pair of cluster-heads is not connected through a third cluster-head then i is a candidate for becoming a gateway. Multiple nodes can be candidates for becoming gateways. The candidate with the highest identity is chosen as the gateway to connect the two cluster-heads. There is no ambiguity in the selection of gateways as all the candidates are within two wireless hops and are aware of each other.

3. *There is no non-cluster-head node that is the neighbor of two cluster-heads.* This is referred to as the case of non-overlapping clusters. One node from each cluster has to become a gateway and the two gateways have to be connected. In order to accomplish this task each node i examines all possible node pairs in which the first element is its cluster-head and the second element is a node in its Headstwohopsaway list. If the cluster-head nodes in a pair are not already connected through a third cluster (as described in the previous case) then i becomes a candidate gateway. There may be several pairs of candidate gateways to connect cluster-heads of two non-overlapping clusters. Any arbitrary rule can be used to select one pair of gateways from the several pairs of candidate gateways. One possibility could be to select the pair with the largest sum of node identities. Ties could be broken by selecting the pair which has the node with the highest identity.

Figure 5.1 shows the result of executing LCA on a network of twenty nodes. Nodes 20, 19, 18, 16, 15, 11 and 8 become clusterheads because they have the highest identities in their respective neighborhoods. Node 17 becomes a clusterhead even though it does not have the highest identity in its neighborhood. This node becomes a clusterhead because it has the highest identity in node 13's neighborhood.

As clusterheads 17 and 19 are adjacent to each other there is no need for a gateway between them. Clusterheads 19 and 15 have one common neighbors, 10, that acts as a gateway between them. For the pair of clusterheads 15 and 16 there are two options for selecting gateways: either 6 or 1. Based on the preceding description, node 6 becomes the gateway connecting them. Node 3 becomes a gateway connecting clusterheads 11 and 16, and node 1 becomes a gateway between clusterheads 15 and 18. Node 7 becomes the gateway between clusterheads 8, 11 and 18. The clusters corresponding to clusterheads 15 and 20 are non-overlapping. Nodes 5 and 9 together act as gateways connecting these two clusters. Likewise, nodes 2 and 12 are the gateways between the disjoint clusters headed by 19 and 20. Two alternatives exist to connect the non-overlapping clusters headed by nodes 17 and 20. One is to choose nodes 4 and 12 to act as gateways. The other is to have nodes 4 and 14 act as gateways. If the policy is to select the pair that has the largest sum of node identities, nodes 4 and 14 would be selected as the gateways.

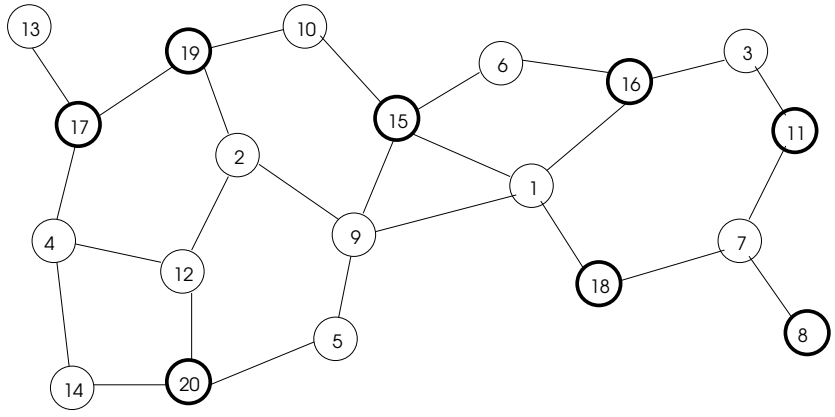


Figure 5.1: Selection of clusterheads using LCA

LCA does not make any claims about the quality of its solutions. For some topologies a significant number of nodes may become clusterheads, with some clusterheads being adjacent to each other. For example, in Figure 5.1 clusterheads 17 and 19 are neighbors. Let us consider a pathological case. The nodes happen to be linearly arranged with monotonically increasing or decreasing node identities. Except for the node with the lowest identity, all others become clusterheads. The clustering solution described in [20] avoids such redundancies by changing the rules for clusterhead selection. Prior to the beginning of the second TDMA frame, none of the nodes are considered to be *covered* by any clusterhead. A node changes its status to covered when it hears a lower identity neighbor declaring itself to be a clusterhead. A node declares itself to be a clusterhead during its slot in the second frame if it has not yet been covered. Hence, the lowest identity node always declares itself to be a clusterhead in the first slot of the second TDMA frame. For the network topology shown in Figure 5.1, the set of clusters obtained as a result of this modification are shown in Figure 5.2. As you can see, no two clusterheads are adjacent to each other. However, once

again, optimal clustering is not guaranteed. In fact, now we have nine clusterheads (one more than before): 1, 2, 3, 4, 5, 6, 7, 10, and 13.

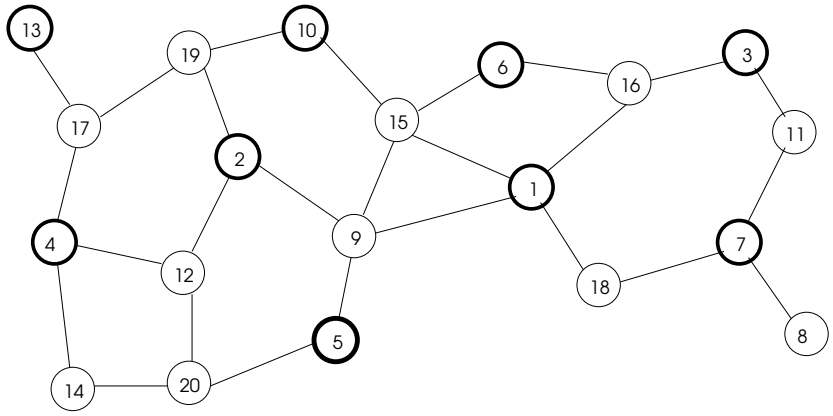


Figure 5.2: Alternative clustering with LCA to prevent adjacent clusterheads.

Once all cluster-heads and gateways have been identified, the links connecting these nodes form the backbone network. Each cluster-head then propagates information about nodes in its cluster to the other cluster-heads through the backbone network. In this fashion each cluster-head becomes aware of the network topology and the cluster to which each node belongs. This information can then be used to either (i) compute shortest paths between node pairs to route messages between them, or (ii) route messages between nodes through the backbone network. The shortest path between node pairs may not necessarily be along the backbone network.

To route messages between nodes in the same cluster the source could send the message to the cluster-head, which in turn could forward it to the destination.

5.2.1.4 Discussion

The *Link Cluster Algorithm* requires knowledge of the number of nodes *a priori*. In several instances it may not be possible to ascertain this number. Moreover, the number of nodes in the system may change with time. So, an upper bound N on the number of nodes has to be estimated and each frame has to have N slots. This is extremely inefficient. An alternative could be to adjust the frame length from time-to-time based on the current knowledge of the number of nodes in the system. However, lack of synchronization among the nodes while changing frame lengths and slot assignments can lead to serious errors in cluster formation, gateway identification, and backbone network formation. Development of correct and efficient algorithms for this purpose is critical.

Another aspect of LCA is the simultaneous presence of several cluster structures. The assumption is that the frequency band for communication is divided into sub-bands. Each sub-band has distinct propagation characteristics. Lower frequency transmissions have a greater range and are less prone to absorption. High frequency signals

are more prone to absorption and propagate along line of sight. Therefore, the connectivity characteristics for each sub-band are different. The Link Cluster Algorithm is run for each sub-band at different times resulting in a different cluster structure for each sub-band.

It has been claimed that due to node mobility the topology of the network for some sub-bands might change leaving it unchanged for other sub-bands. Between the time the topology changes and this information propagates to the entire network nodes may have inconsistent information for routing. During this time, routing may be performed along the cluster structure that does not change.

5.2.2 Highest-Connectivity Cluster Algorithm

As you would have noticed, LCA favors nodes with higher identity values in choosing cluster-heads. It does not take into account network topology. So, it can be argued that LCA may not be able to optimize the number and placement of cluster-heads.

In [25] LCA is compared with another algorithm referred to as the *Highest-Connectivity Cluster Algorithm* (HCCA). In HCCA all the nodes are initially *uncovered*. A node that has elected its cluster-head is said to be covered. A node is selected to be a cluster-head if, between all its uncovered neighbors and itself, it is connected to the most nodes. If there is a tie between multiple nodes, it could be broken using the node identity. A node that has already identified another node to be its cluster-head does not continue as a cluster-head.

The performance of LCA and HCCA were compared through simulation experiments. Even when the network connectivity changes due to mobility nodes tend to re-elect their cluster-heads. The simulation experiments considered two mobility patterns: (i) random movement of nodes, and (ii) predefined trajectories. The number of nodes that switch between being cluster-heads and non-cluster-heads was measured. Another performance measure was the number of nodes that switch clusters, $i.e.$, move form the neighborhood of one cluster-head to that of another cluster-head.

It was observed that the number of switches between node status, and the number of switches between clusters were lower for the LCA algorithm. The LCA is more stable than HCCA. In HCCA if a high-connectivity node acting as a cluster-head loses even one node it may cease to be a cluster-head. However, in LCA such small changes in degree of connectivity may not have any impact on cluster formation.

5.2.3 Backbone of Minimum Connected Dominating Set

In LCA and HCCA no cluster-heads are directly connected to each other. Also, in LCA no attempt is made to minimize the number of cluster-heads. The formation of a virtual backbone using a *Minimum Connected Dominating Set* (MCDS) [15] is an attempt in that direction. Even though MCDS does not explicitly mention clusters, it is appropriate to discuss it with the clustering algorithms due to the following reason: in MCDS, just as in HCCA and LCA, a node is either a *distinguished node* (equivalent to a cluster-head) or one hop away from a distinguished node.

Before we discuss the algorithm let us define some terms:

Dominating set: A dominating set S of a graph G(V, E) is a subset of V such that every vertex is either in S or adjacent to some element of S.

Connected dominating set: A connected dominating set C of G(V, E) is a dominating set such that a path can be constructed between every pair of vertices in C that only passes through vertices in C.

Constructing an MCDS in a general graph is an NP-complete problem. So, an algorithm to construct an approximation of an MCDS is employed. First a small dominating set S is constructed. If node u is dominated by node $dom(u)$, then the edges $(u, dom(u))$ form a spanning forest. Subsequently, the forests are connected using a distributed minimum spanning tree algorithm. The approximated MCDS C consists of the interior nodes of the spanning tree so constructed and the tree edges.

Nodes not in C transmit their local topology information to the node in C that dominates them. Thus, each node in C obtains topology information of its neighborhood. This information is then broadcast to all the other nodes in C along the minimum spanning tree of nodes in C. At the end of all the broadcasts each node in C has complete information of the network topology. This local information is used by nodes in C to compute all the pair-wise shortest paths. The non-MCDS nodes are informed about these paths by their respective dominators. Periodically, the MCDS nodes rebroadcast their topology information to update the routing information.

When nodes move, first the MCDS is updated. Then the routing tables at non-MCDS nodes are updated. Note that the shortest path between a pair of nodes may not have any edge in common with the backbone network connecting the MCDS nodes. The backbone network is used to propagate topology information. However, the backbone can always be used as a backup for routing packets if the shortest path is disrupted.

5.3 Flat Routing Algorithms

In flat routing algorithms all nodes act as routers, and share the responsibility of forwarding packets destined for other nodes. Thus, there is no need to elect clusterheads, and periodically reorganize the network.

Most flat routing algorithms try to implement a distributed version of the shortest path algorithm. In this section we will discuss two such algorithms: (i) Destination-Sequenced Distance-Vector (DSDV) routing algorithm, and (ii) Algorithms based on link reversals. Temporally-Ordered Routing Algorithm (TORA), to be discussed later, is an example of the latter.

5.3.1 Destination-Sequenced Distance-Vector Algorithm

DSDV[43] is a variation of the distance-vector routing approach that was initially used in ARPANET.

The distance-vector algorithm suffers from the count-to-infinity problem when there are link failures. This can lead to messages traversing along loops. The split-horizon hack [51] does not entirely solve the problem. This drawback of the distance-vector

scheme becomes pronounced in mobile systems where wireless links are prone to failure.

The problem with the distance-vector approach is that it responds very slowly to link breaks: nodes may continue to think they can still reach the destination on the other side of the broken link, albeit through a longer path. The reason for this misconception is that nodes make their routing updates based on the path lengths received from other nodes, and not based on any knowledge of how recent or old this information is. So, recent knowledge of the disappearance of a path ($i.e.$ path length equal to infinity) gets overshadowed by outdated information about a path of finite length to the destination.

DSDV addresses the problem by associating a sequence number with routing information for each destination. When a node has to choose between different sources of routing information, each timestamped with a sequence number, the node always favors the more recent information. DSDV is similar to the distance-vector approach in the sense that nodes exchange routing information with their neighbors at regular intervals. In addition, in DSDV, nodes also send routing updates when there are link breaks, or new links are established.

In a highly mobile network, where links get established and disrupted fairly rapidly, updates sent as a result of addition/deletion of links will propagate over the entire network and generate considerable traffic. Also, if multiple paths exist between the originator of an update and another node A, A may receive multiple updates, one along each path. Thus, A's estimate of its distance to the originator will fluctuate for a while before stabilizing. Propagating distance estimates before they have stabilized it not a wise idea. So, DSDV tries to dampen the flow of updates, by inducing waits, to keep the communication overheads in check.

5.3.1.1 Data Structures

Each node maintains a table with one entry for each destination.[3] Each entry has the following entries: (i) destination name, (ii) the node's estimate of its shortest route, in number of hops, to the destination, (iii) the neighbor through which packets for the destination should be sent so as to take the shortest route, (iv) a sequence number associated with the destination. The sequence number acts as a timestamp.

Each node maintains its sequence number which is monotonically increasing. Each node also maintains a history of the arrival times of routing updates about each destination. This history is used to delay route advertisements, and keep communication overheads in control.

A node may send its entire routing table in a message. This is call a *full dump*. For large systems this may impose very high overheads. An alternative is to do an *incremental dump* in which only information that has changed since the last full dump is sent.

[3] In fact two tables are maintained by each node. The significance of the second table will be described shortly.

5.3.1.2 Algorithm

Each node transmits its updates periodically, and also when its receives information about topological change. Changes in topology include establishment of a new link and deletion of an existing link. The periodic update message transmitted by a node contains the table as already mentioned, along with the new sequence number of the node. Thus, the latest periodic update sent by a node will have a higher sequence number than all previous updates sent by the node.

When a link to a neighbor is broken any route through that neighbor is assigned a distance (hop count) equal to ∞. Also, a sequence number one greater than the last known finite sequence number of the corresponding destination is assigned. The updates are transmitted to the neighbors.

When a node receives an update message the node evaluates each entry in the following manner: if the sequence number for the destination in the received message is lower than the sequence number for the same destination in the local routing table then the received entry is ignored. If the sequence number of the received entry is greater than the entry for the destination in the local routing table, the local entry is replaced by the received entry with one modification: the hop count of the local entry is set to one more than the hop count in the received entry. If the sequence numbers of the local routing table entry and the received entry for the destination are the same, and the hop count of the received entry is lower than the local entry then the distance estimate is suitably updated.

Let a node receive an ∞ hop count to a destination in a routing update message. If in the node's own routing table the entry for the destination has a finite value and an equal or higher sequence number, the received entry is ignored and a routing update is immediately sent.

Broadcast of routing updates by different nodes are not synchronized. Also a routing update initiated by node A may reach B along different paths at different times. The order of arrival of these updates can have a significant impact on the stability of the routing algorithm. Suppose A initiates a routing update with a certain sequence number. This reaches B along five different paths. Let the first update to reach B be along the path with the greatest hop count, the second update along a shorter path, and so on. The last update to reach B was routed along the path with the least hop count. If B triggered routing update each time its estimate of the distance to A decreased five updates would be triggered. This would impose excessive communication overheads and degrade overall system performance. If B is more patient and waits a while before triggering updates the excessive communication overheads can be avoided. The general policy is to delay the advertisement of route updates when the node feels that information about a better route is likely to arrive soon.

So, on receiving an update how long should a node wait before propagating it to its neighbors? The node maintains a history of the duration for which the route to a destination fluctuates before information about the best route is received. A weighted average of these durations is the time for which route update advertisements should be delayed. In computing the weighted average a greater weight is associated with the recent values. Route updates with ∞ hop counts are not used to compute the average. The two exceptions when propagation of routing updates is not delayed are

the following:

1. the update carries routing information, with a finite hop count, about a destination that was previously unreachable,
2. routes with ∞ hop counts indicating link breaks.

Let us say an update message arrives at a node that contains information about shorter routes to some destinations. At this point the node is not sure if the routing information has stabilized. Until the node is sure of stability, it should not send this received information in update messages. However, the node can forward packets on their way to the destination using the latest update. Thus, at times a node may maintain two routing tables: one used for forwarding packets, and another, older table to be sent to neighbors in update messages.

5.3.1.3 Discussion

The example in Figure 5.3 demonstrates how DSDV avoids the count-to-infinity problem. On the left hand side you can see the performance of the distance-vector algorithm while on the right hand side you can see the performance of DSDV. Initially, nodes P, Q, R, S, and T are 1, 2, 3, 4, and 5 hops away from the destination, respectively. At this time the link between P and the destination breaks.

In the distance-vector approach, in the first exchange of routing information, P knows that it has do direct link to the destination while there is a path of two hops from Q to the destination. So, P erroneously thinks that it can reach the destination in three hops through Q. Every other node in the system also makes similar mistakes. It takes a long time for the nodes to realize that the destination is unreachable: when their hop counts to the destination become too big to be stored in the hop counter.

In the case of DSDV P immediately sets its hop count to the destination to ∞ and sends an update with a higher sequence number to its neighbor Q. When Q receives the update with value ∞ it relays it immediately to R and so on. Very soon all the nodes whose only path to the destination is through P realize that the destination is unreachable and set their hop count to the destination to ∞.

What if a node disappears from the network and the neighbors do not detect any link failures? Such a situation would not arise often as the medium-access control sublayer is usually able to detect the disappearance of neighbors and link breaks. Anyway, to handle such an eventuality a lifetime is associated with the routing table entries. Each neighbor is expected to send updates at regular intervals. If nothing is heard from a neighbor for a long time (some multiple of that neighbor's update period) it is assumed that the node is no longer a neighbor. So, all routes that pass through that neighbor are assigned a hop count of ∞ and the information is propagated. Thus, stale entries are purged from the routing tables.

5.3.2 Optimized Link State Routing (OLSR)

The Optimized Link State Routing (OLSR) protocol [12, 13] is also a proactive routing protocol in the sense that nodes periodically exchange topology information to maintain their routing tables. Unlike DSDV, OLSR is a link state routing protocol, $i.e.$,

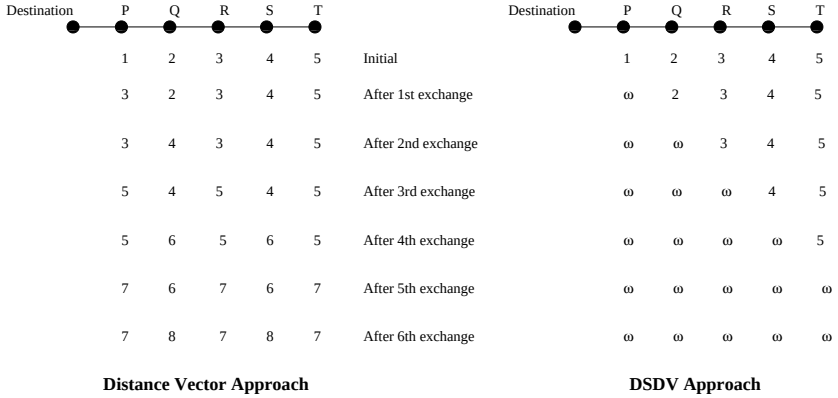


Figure 5.3: Avoidance of the count-to-infinity problem by DSDV routing scheme.

nodes propagate and maintain topology information enabling them to locally compute routes to destinations.

Propagating information about all links in a MANET can incur high communication overheads. So, only a subset of information that is enough to determine optimal routes between nodes is shared among nodes. OLSR is supposed to yield good routes while reducing communication overheads in dense MANETs.

5.3.2.1 Information Maintained at Nodes

Each node periodically transmits *Hello* messages that can reach its neighbors. The Hello message contains the node's knowledge about the links incident on it. If a node X has received a Hello message sent by node Y then X knows that there is a link from Y to itself. In addition, if Y's Hello message also contains information that it can hear X then X can conclude that there exists a bidirectional link between itself and Y. A subsequent Hello message transmission by X will contain information about this link. Let another Z, which is a neighbor of X through a bidirectional link but not a neighbor of Y, receive this information from X's Hello message. Then, Z can conclude that it is two hops away from Y along a path that consists solely of bidirectional links. Thus, each node locally maintains information about its one-hop and two-hop neighbors. This is similar to LCA's mode of local topology discovery.

In link state routing topology information needs to be propagated throughout the network. The simplest approach would be to flood the information throughout the network with each node transmitting a message when it receives the first copy of the message. For example, in Figure 5.4, let node 1 wish to propagate its topology information. Its local transmission will reach all its neighbors $2,\ 3,\ 5,\ 7,\ 8$ and 9. If each of these six neighbors were to transmit the information received from 1 then all the two-hop neighbors of 1 would receive 1's information. However, this would be a terrible waste of bandwidth. If only a subset of neighbors of 1, namely $3,\ 5,\ 7$ and 9, were to transmit the information received from 1, all the two-hop neighbors of 1 would still receive the information. Node 1 can determine this based solely on its two-

hop neighborhood information. OLSR exploits this property to reduce the number of transmissions. Nodes 3, 5, 7 and 9 are collectively referred to as the *Multipoint relays (MPRs)* of node 1. Each node transmits its MPR set in its Hello message. A sequence number is associated with the MPR set and is incremented each time the node's MPR set changes.

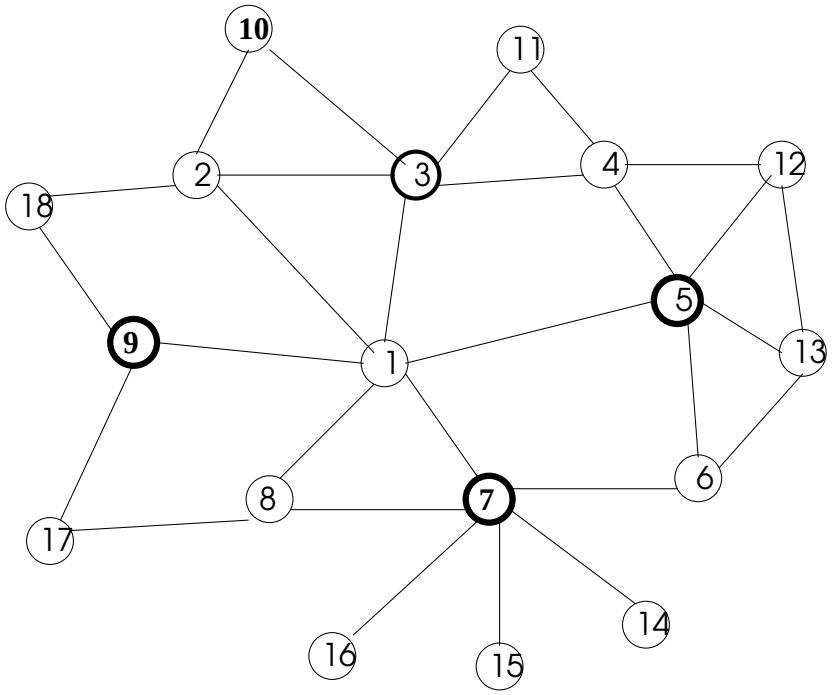


Figure 5.4: Two-hop neighborhood with MPR selection.

When node 7 receives node 1's Hello message, node 7 realizes that it is an MPR for 1. Node 7 maintains this information by adding node 1 to its *MPR Selector* set. A monotonically increasing sequence number associated with a node's MPR Selected set is increased by the node each time its MPR Selector set changes.

The smaller the MPR set, the greater the savings in communication bandwidth. Computing the optimum MPR set for each node so that communication overhead is minimized is quite challenging. Hence, several heuristics have been proposed. In Section 5.3.2.4 we describe the heuristic presented in [12].

5.3.2.2 Propagating Topology Information

Each node, X, periodically generates a *Topology Control (TC)* message to its neighbors. This message contains the node's MPR Selector set and the associated sequence number. Each TC message carries the identity of its originator, $i.e.$, the node that generated this message. When a node Y receives a TC message from a neighbor X, Y

forwards the message only if X belongs to Y's MPR Selector set. This ensures that information about every link $\overline{XY}$, such that X has selected Y as its MPR, is propagated to all nodes in the MANET. If a node has not been selected as an MPR by any of its neighbors then it need not generate a TC message.

It is possible that a node's MPR Selector set changes well before its next scheduled TC message generation. Waiting until the scheduled transmission time would delay the propagation of topology changes to other nodes in the network. Hence, a node can generate a new TC message following a change in its MPR Selector set provided at least a minimum amount of time has elapsed since its last TC message generation. While this quickly propagates topology change information, it limits the rate at which TC messages are injected into the network.

Once a node receives a TC message it needs to update its local topology table which contains information about network topology. The sequence number associated with the MPR Selector set is useful in this regard. Let there be an entry in the topology table of node X corresponding to topology information generated by node Y. Let X receive a TC message originated by Y. If the sequence number of this message is less than the sequence number of the table entry corresponding to Y then the received message is discarded. This is because X has more recent topology information pertaining to Y than is carried in the TC message. Otherwise, the received message carries more recent information about Y and is used to update X's topology table about Y.

5.3.2.3 Formation of Routing Tables

Each node can use its local topology table to determine routes to other nodes in the network. In effect, the topology table contains information about links $\overline{PQ}$, generated by node Q, such that P can be reached in one hop from Q. So, if there exists a route to Q of h hops, then this route followed by the link from Q to P yields a route to P of length $h + 1$ hops. Let us explain the formation of this table at some arbitrary node X.

Initially, the routing table at X is cleared. Then an entry is added for each node Y that is a neighbor of X indicating that Y is one hop away from X, and the next hop towards Y is Y itself. Following this, the routing table is populated iteratively, one-hop at a time.

Let the routing table of X contain entries for all nodes that are at most h hops away from X. Let there be an entry in X's topology table indicating that P has selected Q as its MPR such that: (i) there is no entry for P in X's routing table, and (ii) Q is h hops away from X. Then, a new entry is added to X's routing table for P such that: (i) P is $h + 1$ hops away from X, and (ii) the next hop on the route to P is the node that is the next hop on the route to Q.

5.3.2.4 MPR Set Computation

The goal of MPR set computation is to find a subset of one-hop neighbors of a node that collectively *cover* all the two-hop neighbors of the node. If node X selects a neighbor Y as an MPR node then all the two hop neighbors of X that are neighbors of Y are said to be covered by Y.

OLSR allows nodes some control over the amount of traffic flowing through them by indicating their *willingness*. The five levels of willingness, ranging from lowest to highest are: *WILL_NEVER, WILL_LOW, WILL_DEFAULT, WILL_HIGH, WILL_ALWAYS*. As can be inferred from Section 5.3.2.3, only the MPRs of a node can forward messages to that node. Hence, to begin with, all neighbors of a node X that have their willingness value equal to WILL_ALWAYS are selected as MPRs of X. Following this, the MPR set of X is enlarged as per the following rules:

1. Initially, none of the two-hop neighbors of X are covered.
2. For each neighbor of X, determine the number of bidirectional links it has with the two-hop neighbors of X.
3. For each node Z that is two hops away from X, and only reachable from X through a single one-hop neighbor Y, add Y to the MPR set of X. Remove from consideration all the two-hop neighbors of X that are neighbors of nodes in the MPR set. For example, in Figure 5.4, let us try to construct the MPR of node 1. Nodes $14,\ 15$ and 16 can be reached only through node 7. So, 7 is added to the MPR of 1. Also, because 6 can be reached through 7, 6 is removed from the set of 2-hop neighbors that need to be covered, along with $14,\ 15$ and 16.
4. While some two-hop neighbors of X are not covered repeat the following steps:
   (a) For each non-MPR neighbor, determine the number of uncovered two-hop neighbors of X that can be reached through it. This number is referred to as the *reachability* of the neighboring node.
   (b) Among the neighbors with non-zero reachability, select the node with the highest willingness. Add this node to the MPR set of X. Break any tie by choosing the node with the highest reachability. If there is still a tie, select the neighbor with the highest degree. Remove from consideration all the two-hop neighbors of X that are neighbors of the selected node.

5.3.2.5 Discussion

OLSR enables all nodes to locally compute their routing tables based on topology information propagated through the TC messages. Only bidirectional links are considered for the purpose of routing. Also, there are two optimizations to reduce the communication overheads. First, information about every link is not propagated in the network. A node only generates information about its links with its MPR Selectors. Second, a node does not forward all the protocol messages its receives. Instead, a node forwards only the topology information received from its MPR Selectors.

In sparse networks most of the neighbors of a node may end up becoming its MPRs. However, in dense networks it is possible that only a small subset of neighbors of a node become its MPRs. Hence, OLSR's performance gains, as compared to simple flooding, may be more pronounced for dense networks.

5.3.3 Reactive Routing

One criticism of DSDV has been that it incurs route maintenance overheads even when there is no data traffic in the MANET. There may be intervals of time during which a node, X, does not have any packet to be forwarded to some destination node, Y. Still X needs to process distance estimates for Y received from neighbors, and forward its own distance estimate for Y to its neighbors. Thus, nodes may incur communication overheads to discover routes to a destination when those routes never get used.

To address this problem a number of *reactive* or *on-demand* routing protocols have been proposed. In these protocols a node, X, attempts to find a route to Y only if X has packets to send to Y. Having found a route to Y, X caches the route information. Packets destined for Y in the future are forwarded by X using locally cached routing information. At a later time, due to topology changes, the cache route will no longer be correct. If X does not have packets to send to Y, X will not have to find a new route to Y. However, if X tries to send a packet to Y using stale routing information in its cache, X receives a *Route Error* message. This message prompts X to find a new route to Y and update its routing cache.

In the following sections we will discuss some reactive routing protocols for MANETs, like Dynamic Source Routing [30, 31] and Ad hoc On-demand Distance Vector Routing [42].

5.3.4 Dynamic Source Routing (DSR)

Dynamic Source Routing (DSR) employs two mechanisms that work in tandem:

1. *Route Discovery*: This involves a node X trying to find a route to another node Y.

2. *Route Maintenance*: While forwarding packets to Y, let X realize that its route to Y is no longer valid. As part of route maintenance, X tries to use any alternative route that it knows of to forward packets to Y. If no such alternative exists, X invokes route discovery for Y.

DSR is a *source routing* protocol, meaning that when the source injects a packet into the network the packet carries in it the full path from the source to the destination. Each intermediate node on the path uses this information to: (i) forward the packet to the next node on its way to the destination, and (ii) update its route cache. A node may maintain multiple paths to a destination in its route cache.

Adding the IP address of all the intermediate nodes to the packet header (32 bits per node in IPv4) increases the share of network bandwidth used to carry the IP header. We know that wireless links used in MANETs have low bandwidth to begin with. There is a concern that the increase in header size will result in overall reduction of application level throughput. While this reduction may be acceptable for low diameter MANETs, the performance degradation may be significant for high diameter MANETs. To alleviate this problem DSR supports the option of flow state extension (discussed later in the chapter).

The wireless links in a MANET can be sometimes be unidirectional due to a variety of reasons, including differences in transmission power, interference, etc. DSR can route packets over both bidirectional and unidirectional links. If all the links are known to be bidirectional, the performance of the protocol can be optimized. Also, as a route contains a list of all nodes to be visited in the specified sequence, packets never visit the same node twice. This yields loop-free routing.

Nodes implementing DSR have the option to operate in the *promiscuous* mode. Normally, when a node receives a frame at the data-link layer it inspects the destination MAC address in the frame. If the destination MAC address is its own MAC address or a broadcast MAC address, the data-link layer presents it to the network layer. Otherwise, the frame is ignored. Promiscuous mode of operation means that regardless of the destination MAC address the frame is presented to the network layer which determines the action to be taken for the corresponding packet. By operating in the promiscuous mode, a MANET node can obtain routing information from packets sent by its neighbors, even though the packet is neither destined for that node, nor is passing through the node on its way to the destination. However, promiscuous mode of operation has its price. Frames that would have been dropped at the data-link layer now have to be processed at the network layer, thus increasing the overall computing demands at the nodes. As long as this extra computation overhead is worth the gain in routing information, promiscuous mode of operation can be justified.

5.3.4.1 Data Structures

Each node maintains a *route cache* in which it stores routes to various destination nodes. Unlike DSDV, the route cache need not have an entry for each node in the MANET. Also, a node may have multiple routes for a given destination. A route cache entry contains the sequence of nodes along which a packet should be forwarded to reach the destination.

For each potential destination node, a node maintains an identification number which the node uses to distinguish between different route requests to the destination. Other nodes maintain $<$*indentification number, source-destination pair*$>$ pair for route requests heard by them in the recent past.

Each node maintains a *Send Buffer* of packets that it is not able to send due to lack of routing information. For each packet in the send buffer there is a timer to track the duration for which the packet has resided in the buffer.

Also, for each outgoing link XY, the node X maintains the source nodes of packets sent along XY that have yet to be acknowledged by Y.

5.3.4.2 Protocol

As mentioned earlier, DSR employs *Route Discovery* and *Route Maintenance*. In the following paragraphs we will discuss each of these mechanisms in some detail.

Route Discovery : Let node X wish to find a route to another node Y. X locally broadcasts a *Route Request* message with Y as the destination address, and a locally chosen identification number. The message has a route record field, which is initially

empty. Depending on the underlying MAC protocol and channel contention, some subset of neighbors (i.e., nodes within communication range of X) will receive the route request.

If the node receiving the route request is the destination node, Y, it sends a *Route Reply* message to X. This reply message contains the sequence of nodes visited by the request to reach Y. If the receiver is not the destination, the receiver examines the request. If the receiver has already received a route request with the same source, destination, and identification number then the receiver discards the request message. Also, if the receiver's own address is already listed in the route record then the message is discarded. This is done to avoid loops.

If the receiver is not the destination, nor is the request to be discarded, then the receiver appends its own address in the route record of the request message, and propagates it as a local broadcast.

So, how does the destination node, Y, send its route reply to X? If Y has a route to X in its route cache then Y sends the reply along that route. Otherwise, Y initiates a *Route Discovery* with itself as the source and X as the destination. The route from X to Y is piggybacked on the corresponding route request messages. X, on receiving the route request message from Y, can extract the route from itself to Y. Consider the consequence of not piggybacking the *X-to-Y* route information on the route discovery message: X knows of a route from Y to X, but cannot communicate this information to Y because it is still trying to find a route from itself to Y. This would trigger another route discovery from X to Y, triggering another route discovery from Y to X, and so on.

Let all the links on the route from X to Y be known to be bidirectional. Then Y should send its reply to X along the route obtained by reversing the route from X to Y.

As DSR is a reactive protocol, route discovery by X must have been initiated because X had a packet to send to Y. What should X do with this packet (and subsequent packets for Y) while route discovery is in progress? Such packets should be kept in the send buffer and a countdown timer associated with the packet should be started. If a route to Y is not found before the timer expires, X purges the packet from its send buffer. A timeout is also associated with a route discovery attempt for a given destination. The route discovery timeout is smaller than initial value of the packet timer to permit a few route discovery attempts to be made before the packet timer expires. Also, to avoid an explosion of route discovery attempts for a given destination, the corresponding timer should employ binary exponential backoff.

Route Maintenance : Let node A send packets to a neighbor B on link AB. At the link level, A expects an acknowledgment from B: either a MAC-based ACK as in the IEEE 802.11 protocol, or a passive acknowledgment.[4] If A does not receive an acknowledgment from B, A assumes that the link AB is broken and removes it from

[4]If node BN is not the destination of the packet, B would transmit the packet to the next node on the route to the destination. Node A is said to have received a passive acknowledgment for a packet from B when it hears B's transmission of the same packet. A word of caution about passive acknowledgments: the MAC sub-layer protocol used by B to transmit the packet may not necessarily ensure that A also hears the transmission. So, passive acknowledgment should be employed carefully.

its route cache. A also sends a *Route Error* message to all nodes that originated packets transmitted by A over link AB since the reception of the last acknowledgment from B. This route error message indicates that the link AB is faulty. When nodes receive this route error message they invalidate routes in their cache that pass through AB and try to resend their packets using alternative routes. If no such alternative exists then these nodes initiate Route Discovery.

Additional Route Discovery Features : DSR proposes several optional features meant to improve its performance with respect to route discovery overheads, latency, quality of routes, etc. Some of these features are described below:

Caching Overheard Routes: A DSR node operating in the promiscuous mode can overhear packets sent by its neighbors, even though the node is not the link-level target of such transmissions. The node can populate its routing cache using information contained in these overheard packets. Let node X overhear a neighbor W's transmission that contains information about a route $U - V - W - Y - Z$. If all links are known to be bidirectional then X can use this information to add the following routes to its cache (i) in the forward direction: $X - W - Y - Z$ and all its prefixes, and (ii) in the reverse direction: $X - W - V - U$ and all its prefixes. However, if some links are likely to be unidirectional then X can only cache routes in the forward direction, provided X knows that it can directly transmit to W.

Route Replies Not from Target: Let node X initiate a route discovery for a destination node Z. Let node Y be an intermediate node that receives the route discovery message and has a path from itself to Z in its route cache. Then, Y sends a route reply to X containing the sequence of nodes on the path from X to Y (contained in the route discovery message received by Y), concatenated with the sequence of nodes on the path from Y to Z. This speeds up route discovery and reduces the associated communication overheads.

To avoid loops, Y should ensure that no node appears more than once in the resultant path. Consider the situation depicted in Figure 5.5. Node G receives the route discovery message initiated by A, along the path $A - B - C - G$, for destination F. G knows of the path $G - C - D - E - F$. Concatenation of the paths yields a path $A - B - C - G - C - D - E - F$ which has the loop $C - G - C$. Removal of the loop from the path eliminates G from the path. DSR stipulates that if a node gets eliminated from a path while loop removal that node does not send a route reply. So, in this example, node G would not send a route reply. However, if node C knows of the path $C - D - E - F$ to F then it can respond to A with the path $A - B - C - D - E - F$.

Preventing Route Reply Storms: Let node X transmit a Route Request (either originated or forwarded by it). As shown in Figure 5.6, several neighbors of X may have routes to the destination. If each neighbor responds with the concatenated route (as explained above), the responses will collide resulting in wasted

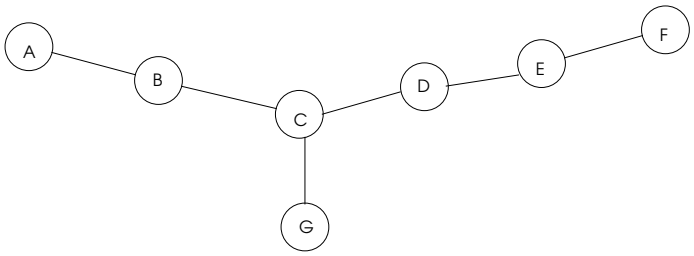


Figure 5.5: Route replies not from target: elimination of loops.

bandwidth. To prevent such reply storms, before responding to X's Route Request each neighbor, Y, sets a local timer proportional to the length of the path from itself to the destination. While the timer has not expired Y listens in the promiscuous mode for data packets from the source of the route discovery to the destination. Let Y hear the transmission of such a data packet, and the path carried in that packet is shorter than what Y has to offer. Then, Y cancels its timer and does not send a Route Reply to the source. However, if Y's timer expires before it receives such a data packet then Y sends a Route Reply to the source. In the example shown in Figure 5.6, node B responds first while A and C are still listening for data packets in the promiscuous mode.

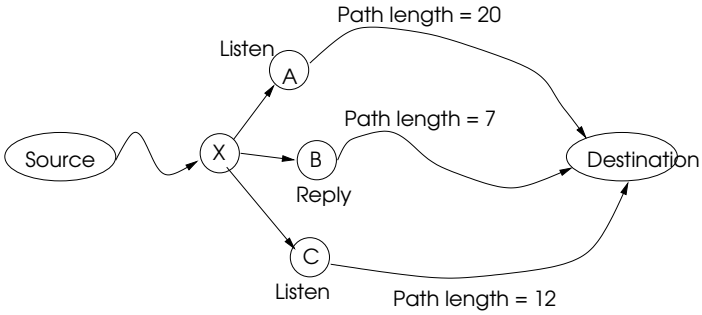


Figure 5.6: Preventing route reply storms.

Route Request Hop Limits: A source node can use the Time-To-Live (TTL) field of its Route Request packets to limit their propagation. The first Route Request is *non-propagating*, i.e., sent with a TTL of 1. If no neighbor sends a Route Reply before the expiry of a timer then the source initiates *expanding ring search*. As part of expanding ring search, the source doubles the TTL value of its Route Request packets each time the timer expires before receiving a reply. Expanding ring search has the potential to increase the time to discover a route when compared with an approach where the very first attempt is made with a big enough TTL value. However, expanding ring search helps in reducing the route discovery overheads, especially if some of the nearby nodes know of a path to the

destination.[5]

Additional Route Maintenance Features: DSR also proposes three optional features for route maintenance. They are:

Packet Salvaging: When an intermediate node detects that the next link on the path of a packet towards its destination is broken the node sends a Route Error message to the source. Instead of silently discarding the packet, the intermediate node tries to *salvage* the packet if it knows of an alternative route to the destination. Packet salvaging works in the following manner. The intermediate node increments the salvage counter of the packet (initialized to zero). If the counter exceeds a threshold the packet is discarded. Otherwise, the intermediate node replaces the original route in the packet (which is known to be broken) with a route in its own cache, and forwards the packet to the next node on the new path. The salvage counter is used to place a limit on the number of times nodes in the network try to salvage a packet.

A number of packets may be queued at the intermediate node with their next hop being a broken link. The intermediate node identifies the source nodes of these packets and sends one Route Error message per source node. The intermediate node tries to salvage all the packets in its queue.

Automatic Route Shortening: Let node Y overhear the transmission of a neighboring node X while Y is operating in the promiscuous mode. Refer to Figure 5.7. In the overheard packet, Y is not the next hop. Instead, it appears later in the source route, $Source - A - X - B - C - Y - D - E - Dest$, carried by the packet. It is obvious to Y that the source of the packet is unaware that X and Y are neighbors, and there exists a potential to use a shorter route to the destination. So, Y sends a *gratuitous Route Reply* to the source of the packet. The Route Reply contains the following path from the source to the destination: the path from the source node to X, followed by Y, followed by the path from Y to the destination.

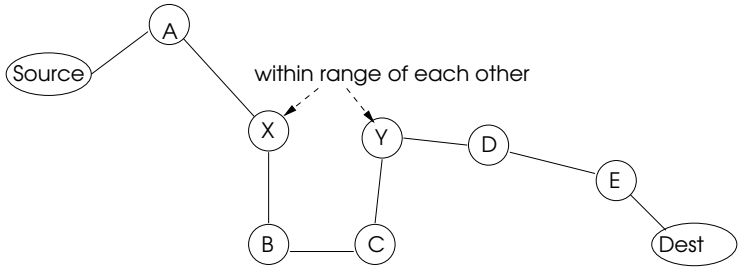


Figure 5.7: Automatic route shortening in DSR.

[5]Why does expanding ring search double the TTL value in each iteration, as opposed to increasing it by 1? Is there any advantage to be gained by this?

If a stream of packets sent by the same source to the same destination are overheard by Y, it would not be wise for Y to send a gratuitous Route Reply in response to every such packet. Hence, each node maintains a *Gratuitous Route Reply Table* to limit the rate at which it sends such replies for a given route.

Increased Spreading of Route Error Messages: Let a source node receive a Route Error message for a destination. If the source does not know of any alternative route to the destination it generates a new Route Request. Some subset of neighbors receive the route request. While the source is aware of the broken link that triggered the Route Error message, the neighbors may not have the same knowledge. So, there is a likelihood that the neighbors respond with a cached route that contains the broken link. This would definitely not be of any help to the source node. So, to preempt such messages, the source node piggybacks the Route Error message with the new Route Request message. The neighbors, on receiving this message, purge all routes containing the broken link from their route cache and do not send their replies containing the invalid route(s).

Flow State Extension : As stated earlier, sending the entire path with every packet can place significant demands on network bandwidth. The longer the path, the more severe the problem. So, there are definitely scalability issues involved.

To ameliorate this problem, DSR has the option of flow-based routing. A *flow* is a stream of packets traveling from a source node to a destination node along a path over a certain period of time. In addition to the source route, the source node can add a header option to a packet (the first packet of the header) to carry its flow identifier. As the packet is forwarded along the source route, every node adds an entry into its flow table corresponding to this flow. The entry corresponds to the IP addresses of the source and destination nodes, the IP address of the next hop neighbor, and the flow identifier. This process is referred to as *flow establishment*. Subsequent packets of the flow do not carry the source route. They only carry the flow identifier. When an intermediate node on the route receives a packet with the flow identifer it uses the source and destination addresses and the flow identifier to search its flow table entry. If there is a match, the node forwards the packet to the corresponding next hop neighbor. Otherwise, the node sends a Route Error message to the source node.

Each flow has a lifetime associated with it. This lifetime value is sent with the first packet of the flow during flow establishment. Each intermediate node records the lifetime value in its flow table entry. At the expiry of the lifetime, a node purges the flow information from its flow table. This *soft state* maintenance of flow information prevents outdated flow information from being present at nodes. If topology changes disrupt the route along which the packets were flowing the source or destination nodes do not have to do anything special. Soft state maintenance of flow information ensures that all the nodes on the flow path independently delete the flow information when the flow lifetime expires.

It is possible to have multiple flows between a pair of nodes. Each flow the source to destination is identified by a unique flow identifier. An odd numbered flow is the default flow between a source and destination. Some of the route maintenance options

(described earlier) can interoperate with flow-based extensions. Interested readers can find the details in [31].

5.3.5 Ad hoc On-Demand Distance Vector (AODV) Routing

Just like DSDV, AODV uses sequence numbers to determine the *freshness* of routing information. For a given destination node, routing information with a highest destination sequence number overrides that with a lower sequence number. However, unlike DSDV, nodes do not periodically exchange their distance vectors with neighbors. Instead, nodes try to discover routes to other nodes only when they need to forward packets to them.

In the following sections we will discuss how AODV maintains data structures to track active routes, discovers new routes, generates gratuitous replies for destinations, handles link breaks and resultant route failures along with local repair of routes. We will also discuss AODV's approach towards unidirectional and/or unreliable links.

5.3.5.1 Data Structures

Each node, X maintains a routing table containing entries for a subset of nodes in the MANET. An entry corresponding to node Y contains Y's IP address, the hop-count to Y, the next-hop neighbor on the path to Y, and a sequence number associated with Y. A countdown timer, referred to as the *lifetime*, is also associated with the routing table entry for Y. While X is forwarding packets destined for Y along this route, the route is said to be *active*. Each time X forwards such a packet is forwarded, it resets the lifetime to a positive constant. If the lifetime expires without X forwarding any packet for Y along this route, the route becomes inactive. However, X continues to maintain such inactive or expired routing table entries for an additional amount of time. The rationale for this will become clear in the following description.

In addition to the packets for Y that originate locally, X may be receiving packets destined for Y from some of its neighbors. X maintains a list of these neighbors in its *precursor list* corresponding to each active route.

Each node also maintains a monotonically increasing integer, *Route Request Identifier (RREQ ID)*. This identifier is used to distinguish between route discovery requests issued by a node.

5.3.5.2 Protocol

Let node X have a packet destined for Y. If X has an active route for Y then the packet can be forwarded to the next hop neighbor on the route to Y. Otherwise, X buffers the packet, starts a route discovery countdown timer, and initiates route discovery for Y.

Route Discovery: X generates a route request message that is uniquely identified by its own IP address and its RREQ ID. The message contains Y's IP address as the destination address, the last known sequence number of Y, and X's own sequence number. Each time X generates a new route request it increments its own sequence

number. The route request message also carries a hop count which is initialized to zero by X.

X sends this route request message to all its neighbors. This can be accomplished either by a MAC sub-layer broadcast, or by multiple unicasts to the neighbors. A node Z may receive multiple copies of the same route request along different paths. Each of them has the same $<$ source IP address, RREQ ID $>$ pair. Only the first such message is processed by Z, the rest are silently discarded. When the first copy of the route request is received by Z, it establishes a path to the neighbor from which the request was received. This is needed to send the resultant route reply back towards X. Z also increments the hop count of the route request by one to reflect the number of hops that the route request has traveled from X. Z also creates a routing table entry for X with the neighbor as the next hop, hop count equal to the received hop count plus one, and the sequence number equal to the received source sequence number. Thus, X, while trying to find a route to Y, enables intermediate nodes like Z to update their routes to X.

If there exists a routing table entry for Y (whether active or inactive), then Z sets the destination sequence number in the route request message to be the maximum of the received value and the destination sequence number for Y that it has in its own routing table. Consequently, as the route request propagates it carries with it the latest known sequence number of the destination.

If $Z \neq Y$ and Z does not have an active route to Y then Z forwards the route request to its neighbors. Otherwise, Z generates a route reply. The possibilities are:

- *Z is the destination node:* Let the destinations own sequence number be α while the received destination sequence number be β. If $\beta = \alpha$ (β can never exceed $alpha$) then the destination increments its own sequence number to be greater than β and sends this updated value in the route reply.

- *Z has an active route to destination Y:* Z sends its hop count towards Y in the route reply message. The neighbor from which Z received the route request is added to the precursor list for the route to Y. This is because in the future Z expects to receive packets meant for Y from this neighbor.

Note that the route reply is unicast to the node from which Z received the route request. Each node, on receiving the route reply for destination Y performs the following actions: (i) increments the received hop count for Y by one and updates its routing table entry for Y as per the received information, and (ii) forwards the route reply towards X by unicasting it to the next node on the route to X. It is worth remembering that this *next node* was the node from which Y had received its first copy of the route request message generated by X. Hence, the route reply retraces the path traversed by the route request message.

If X receives the route reply for Y before the corresponding route discovery countdown timer expires, it can start sending packets destined for Y. However, if the timer expires before a route reply is received, X employs binary exponential backoff in route discovery, $i.e.$, it generates a new route request for Y with a higher RREQ ID and doubles the route discovery countdown timer. If X is unable to find a route to Y after a finite number of attempts it aborts the route discovery.

Expanding Ring Search: Route discovery, as described above, has the potential to flood the entire MANET with route request messages. This is not a very good idea, especially if X has some idea about its distance to Y based on an expired routing table entry for Y. Hence, expanding ring search can be used as an optimization to reduce the route discovery overhead.

In expanding ring search, when X starts route discovery for Y the TTL value of the route request message is set to the last known distance to Y plus a small constant value. Otherwise, the TTL value is set to another constant value referred to as TTL_START. The value of the route discovery countdown timer is a function of the TTL value of the route request message injected into the network by X.

Each time the route discovery countdown timer expires before a route reply is received, X injects a new route request message for Y with the TTL value incremented by a small constant. This is continued until the TTL value reaches or exceeds the upper bound on the expected diameter of the MANET.

While AODV's expanding ring search is similar to DSR's, there is one significant difference. While DSR doubles the TTL in each attempt, AODV employs constant increments.

Gratuitous Route Replies: There is a possibility that none of the route requests generated by X for Y ever propagate all the way to Y. Instead, intermediate nodes that know of routes to Y generate the route replies and discard the route request. So, while X knows the route to Y, Y does not know the route to X. If X sends a packet to Y that requires Y to respond, a route discovery will have to be initiated by Y. This will increase the time it takes for X to receive the reply. To eliminate the reverse route discovery delay (from Y to X), AODV provides for the generation of gratuitous route replies.

As per this optional feature of AODV, when the intermediate node Z sends a route reply towards X, it also generates a route reply directed towards Y. This route reply carries information about the route to X (hop count to X, sequence number for X, etc.) and is sent to the next node on the path towards Y. So, this route reply is similar to the one that would have been generated had Y initiated a route discovery for X.

Packet Forwarding: Each time an intermediate node Z forwards packets sent by X to Y along an active route, Z resets the active route lifetime fields not only for X and Y, but also for the next hop towards Y and the precursor node (from which it received the packet). This is with the expectation that most communication between two nodes is duplex and the routes between X and Y are symmetric. So, Y is likely to send packets to X in the near future and these packets will be forwarded by Z to the precursor node for Y.

Propagating Route Error messages: Let a node Z detect that its link with the next hop neighbor for an active route from X to Y no longer exists. Then Z identifies all active routes passing through itself that use this link. The sources of all these active routes need to be informed about the disruption of their paths to the corresponding destinations. So, Z generates the appropriate route error message and propagates them

towards the source nodes through the corresponding precursor nodes. The corresponding routes are also invalidated.

Another possibility is that Z receives a packet generated by X and destined for Y, for which Z does not have an active route. In that case Z generates a route error message and forwards it to the precursor node from which it received the packet.

For both these cases, Z increments the destination sequence numbers for the unreachable destinations and places this incremented sequence number in the corresponding route error messages.

Let a neighbor of Z receive the route error message corresponding to destination Y. If the neighbor has an active route to Y that uses Z as the next hop then the neighbor performs the following actions: (i) invalidates its route to Y, (ii) increments the corresponding destination sequence number for Y based on the received sequence number, (iii) forwards the route error message to all the precursor nodes for route to Y.

Local Repair: Sometimes, on the break of a link along an active route, the propagation of the route error message towards the source of packets can be avoided. Let the intermediate node Z detect that its link to the next hop neighbor on the active path from X to Y has broken. If, prior to the link break, the destination Y was no more than a certain constant number of hops from Z then Z initiates local path repair as follows. Z initiates a route discovery for Y while buffering the received packets destined from X to Y. The TTL value of the route request message is small enough that X will most likely be unaware of this route discovery. Let Z succeed in finding a new route to Y. If the hop count of the newly discovered path from Z to Y is less than or equal to the hop count of the previous path no further action is necessary. However, if the hop count of the new path is greater than the previous path from Z to Y then Z generates a route error message which is propagated towards the originator. However, this route error message carries a flag advising the receiver(s) of this message that they must not invalidate their routing table entries for Y. The receiver(s), including the source node X, may choose to initiate a new route discovery for Y.

The rationale behind this approach is the following. The path from Z to Y, discovered as a result of local repair may be the shortest path between the two nodes. However, the path from X to Z, concatenated with the path from Z to Y may not necessarily be the shortest path from X to Y. So, the choice lies with the originator X whether it wishes to incur the overheads of a new route discovery. If X knows that it is almost done sending packets to Y, X may choose not to initiate a new route discovery.

Dealing with Unidirectional Links: AODV assumes that the path from a node X to another node Y is exactly the reverse of the path from Y to X. This assumption is valid as long as all links are bidirectional. However, consider the following scenario with unidirectional links. Node X wishes to find a route to node Y. The route request message reaches a node P which forwards the request along a unidirectional link to a neighbor Q. Let this be the first route request message received by Q corresponding to this route discovery. So, Q forwards the request to its neighbor(s) and sets P as the precursor on the path from X to Y. Subsequently, Q receives another copy of the same route request message from another neighbor R along a bidirectional link. As per the

protocol, Q silently discards this message because it is a duplicate. Ultimately, a route is discovered and a route reply reaches Q. All attempts by Q to propagate the reply to the precursor node P fail because the link is not bidirectional. So, X does not get to know of the discovered route. Ultimately, X times out and attempts route discovery again. Once again, the first copy of the request reaches Q from P, and not from R. So, this route discovery also fails. Had Q propagated the request received from R, it would have been successful in sending the route reply back towards X through R.

So, it appears that propagating only the first copy of the route request, and discarding subsequent copies is not a good idea, especially in the presence of unidirectional links. AODV's solution to the problem is to blacklist the unidirectional links, $i.e.$, to not use them for routing and packet forwarding. But, how is a node to know that the link along which it received a route request is unidirectional from a neighbor to itself? This information can be obtained in one of two ways. Let us illustrate the two options in the context of the unidirectional link from P to Q. When Q sends a route reply to P, let the MAC sub-layer protocol require an acknowledgment from P. As the link is unidirectional, P never receives the route reply, and does not generate the acknowledgment. So, Q concludes that it does not have a bidirectional link with P. If the MAC sub-layer protocol cannot be relied upon for acknowledgments then an explicit network layer acknowledgment can be requested from P. Once again, Q will receive no such acknowledgment, and reach the same conclusion about unidirectionality.

Consequently, node Q adds node P to its *blacklist* for a period of time that is long enough to permit the source node to exhaust all its attempts at route discovery. While node P is in node Q's blacklist, Q ignores all route request messages it receives from P.

On Rebooting: When a node X is rebooted its routing table is wiped clean. As a result, X does not know the route to any other node in the MANET and is unable to forward any data packets. However, other nodes may still have entries in their routing tables based on the old network topology prior to the rebooting of X, and X end up receiving data packets to be forwarded to their destinations.

To avoid misrouting of packets one must ensure that: (i) all stale routing table entries, based on old network topology, are eliminated, (ii) the rebooted node X can gather as much topology information as possible. Towards these goals, AODV requires that the rebooted node X does not send any route discovery message for a period of time long enough to allow neighbors, with invalid routes containing X as the next hop, to purge those routes from their routing tables. This period is referred to as the *delete period*. Also, during this period if X receives route request, route reply or route error messages, X uses the information carried in these messages to build its routing table. But, X does not forward these messages. Also, if X receives a data packet then it should broadcast a route error message, and once again resume waiting for the delete period before initiating a route discovery.

5.3.6 Link Reversal Based Algorithms

Gafni and Bertsekas [24] proposed using link reversals to (i) create a new route, or (ii) to establish a new route to the destination in a finite time after topological changes.

Link reversal algorithms model the network as a directed graph G: (V,E). If an edge is directed from node i to node j then i is said to be the *upstream* node and j is the *downstream* node. One instance of the algorithm is run for each destination. Given an arbitrary initial orientation of edges, the algorithm performs multiple iterations of changes in edge orientations. At the end, there is a directed path from every node to the destination node. The resultant graph is said to be *destination oriented*. A graph in which at least one node does not have a directed path to the destination is said to be *destination disoriented*.

5.3.6.1 Basic Idea

There are two types of link reversal algorithms:

1. *Full Reversal Algorithm:* In each iteration nodes that are not the destination and have no outgoing edges are identified. The orientation of all the incident edges on such nodes is reversed.

   Figure 5.8, first presented in [24], illustrates the operation of the algorithm. In Figure 5.8.(i) the node labeled R loses its direct link to the destination D. As a result there is no directed path from this node to the destination. In fact, all the edges incident on this node are directed towards it. Full edge reversal at this node results in two other nodes, marked R, that have all incident edges directed towards them. So, full edge reversal is performed at these nodes. This process is continued until the destination oriented configuration is reached in four steps, as shown in Figure 5.8.(v). At this point there is no node which has all its edges directed towards it. So, the algorithm terminates.

2. *Partial Reversal Algorithm:* Every non-destination node keeps a list of incident edges whose orientation has been reversed by the nodes at the other end of those edges. In each iteration nodes that have no outgoing edges are identified. All the incident edges that do not belong to the list are reversed and the list is emptied. If all the incident edges belong to the list then all of them are reversed and the list is emptied. The algorithm terminates leaving a destination oriented graph.

   Figure 5.9, which has the same initial configuration as the earlier example for the full reversal algorithm, shows the operation of the partial reversal algorithm. The nodes where edge reversals are performed are marked with R and the destination is marked D. Once again, in finite time a destination oriented graph is generated.

You may have already noticed that both the link reversal algorithms implicitly assume that the clocks at all the nodes are synchronized. All the nodes agree on when an iteration starts and finishes. In order to achieve clock synchronization an underlying distributed computation to keep all the clocks in step has to be executed. Alternatively, all the nodes could get their time off a global source like the Global Positioning System (GPS) satellites.

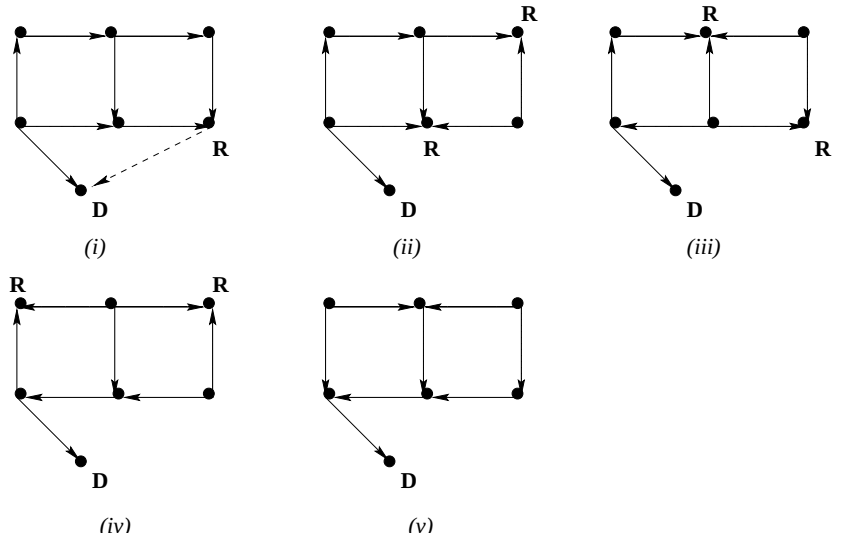


Figure 5.8: Sequence of full edge reversals transforming a destination disoriented graph into a destination oriented graph.

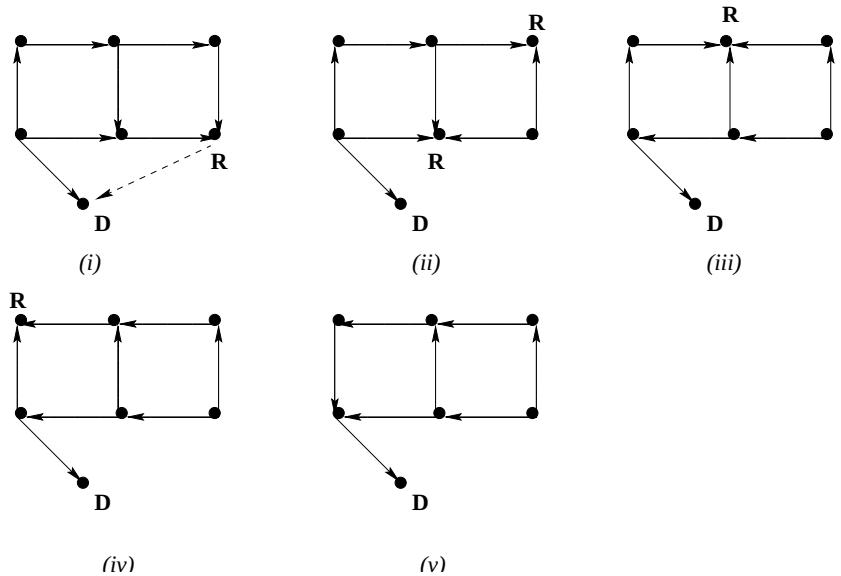


Figure 5.9: Sequence of partial edge reversals transforming a destination disoriented graph into a destination oriented graph.

5.3.6.2 Implementation Issues

In order to implement the full reversal algorithm each node i is assigned a height $h_i = (\alpha_i, i)$. While i is the node's identity, α_i is an integer variable associated with the node. We say that $h_i > h_j$ if:

1. $\alpha_i > \alpha_j$, or
2. $\alpha_i = \alpha_j$ and $i > j$.

If $h_i > h_j$ for neighbors i and j then node i is upstream from node j and the edge is directed from i to j. Performing an edge reversal involves a readjustment of the height(s) of the node(s) on which the edge is incident.

In a destination disoriented graph there exists at least one non-destination node i such that $h_i < h_j$ for all neighbors j. To perform a full edge reversal, node i assigns a new value to α_i that is greater than α_j for all neighbors j. As a result i's height is greater than the height of all its neighbors. To perform a partial edge reversal a new height has to be selected that is greater than the height of some of the neighbors while being less than the height of the other neighbors. This is a non-trivial process. If you are interested in the method of height adjustment for partial reversal the original paper by Gafni and Bertsekas [24] is a good starting point.

5.3.6.3 Network Partitioning

Let us assume that a node loses its path to a destination due to network partitioning: the node and the destination belong to different partitions. No amount of edge reversal, full or partial, is ever going to find a route to the destination. This will result in a non-terminating execution of the algorithm incurring high communication and computation overheads.

Park and Corson [41] identified this problem with the link reversal algorithms, and proposed a solution. We will discuss their solution very briefly. Once again, if you are interested in the gory details, what better source than the original paper! Basically, a node that has no downstream link, due to link failure, selects a height that is the global maximum. This is achieved by defining a new *reference level* for the node that is higher than any earlier reference level in the system. This triggers link reversals associated with the new reference level. The new reference level is diffused outward from the point of failure. Now, let there be a node p that only had downstream links before this process started. Let p experience link reversals, associated with the new reference level, from all its neighbors. Then p has to select a new height greater than the heights of all its neighbors. This is achieved by p selecting a new *sub-level* associated with this reference level. The resultant link reversals in effect *reflect the reference level* back towards the node that initiated the process. When the initiator receives the reflected reference level from all its neighbors the initiator infers that the destination belongs to a different network partition. It then initiates a process to erase all the invalid routes.

5.4 Geographical Routing

The MANET routing protocols we have studied so far require nodes to maintain and exchange topology information, either periodically or on an on-demand basis. Some researchers have argued that this results in high communication overheads, making the routing protocols non-scalable. To address the scalability problem, they have proposed *geographical routing* [33, 32].

5.4.1 Greedy Geographical Routing

Ko and Vaidya [33] proposed one of the earliest geographical routing protocols for MANETs. The basic idea is quite simple. Each node knows its own location, and employs a beaconing protocol to advertise its location information to its neighbors. In addition, there exists a location service with which nodes register their location information.

Let node *X* wish to send a packet to node *Y*. Node *X* queries the location service to obtain *Y*'s location. Then, *X* forwards the message to a neighbor, *Z*, that is closest to *Y*. Node *Z* follows the same procedure by forwarding the packet to its neighbor that is closest to *Y*. The idea is that with each forwarding, the packet gets closer to its destination. Ultimately, the packet reaches its destination. As you may have realized, this is a greedy approach. Each node makes a forwarding decision that is locally optimal. Figure 5.10 illustrates the routes resulting from this approach. Packets sent from node *1* to node *23* take the route indicated by the dashed arrows. Packets send by node *16* to node *4* take the route indicated by the solid arrows.

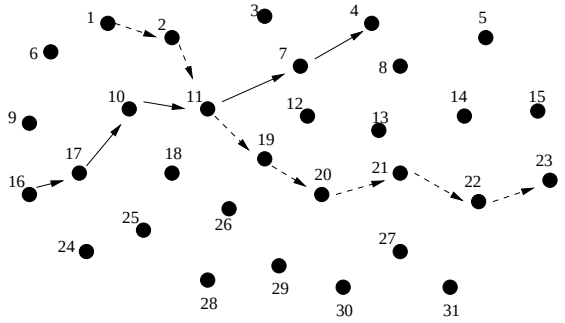


Figure 5.10: Routes resulting from greedy geographical routing.

5.4.2 Greedy Perimeter Stateless Routing

The greedy geographical routing approach works fine if the network is sufficiently dense and nodes are uniformly distributed. However, if there are regions with no nodes in them, the greedy approach may fail. Figure 5.11 presents an example topology where the greedy geographical approach does not work. Node *A* is trying to send a packet to node *D*. Based on greedy forwarding, the packet travels three hops and reaches a node

such that all neighbors of that node are farther from *D* than the node itself. So, the node does not know where to forward the packet.

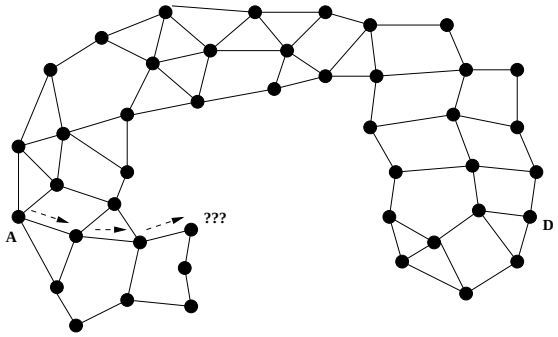


Figure 5.11: Failure scenario for greedy geographical routing.

The *Greedy Perimeter Stateless Routing (GPSR)* protocol, proposed by Karp and Kung [32] provides a solution to the problem. The solution combines two approaches:

- Greedy forwarding when such forwarding leads the packet closer to its destination.
- *Perimeter forwarding* on the *planarized graph* when greedy forwarding is not possible.

To understand the second component of GPSR, let us first discuss what it means to planarize a graph. Then, we will look at perimeter forwarding.

Planarizing a graph: The topology graph representing a MANET may not be planar. There may be edges crossing each other. A graph from which such edge crossings are eliminated, is a planar graph. Karp and Kung discuss two well known types of planar graphs: (i) *Relative Neighborhood Graph (RNG)*, and (ii) *Gabriel Graph (GG)*.

Let there be three nodes, *x, y, z* and let *d(x,y)* denote the length of the edge (distance) between *x* and *y*. Then, an RNG is defined as follows. An edge between nodes *x* and *y* exists in the RNG if:

$$\forall z \neq x, y: \ d(x,y) \leq max[d(x,z),\ d(y,z)]$$

Figure 5.12 illustrates what it means to obtain an RNG from a non-planar graph. In the original graph shown in the upper left-hand corner, no such node z exists. Hence, the link between *x* and *y* stays in the RNG. In the original graph shown in the lower left-hand corner, there does exist a node z with edges to both *x* and *y*, such that $d(x,y)$ is greater than $both d(x,z)$ and $d(y,z)$. Hence, the edge *(x,y)* is deleted from the resultant planar graph.

In a Gabriel Graph an edge *(x,y)* exists between a pair of nodes if:

$$\forall z \neq x, y \ d^2(x,y) < [d^2(x,z) + d^2(y,z)]$$

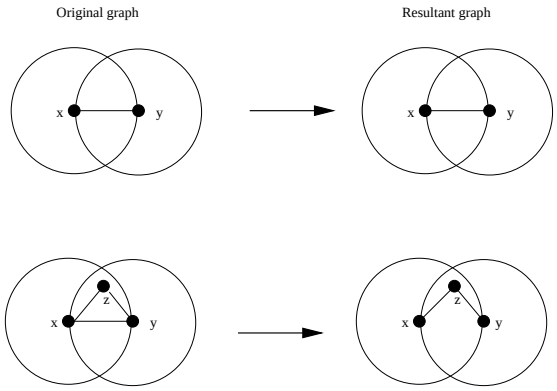


Figure 5.12: Obtaining a Relative Neighborhood Graph from a general graph.

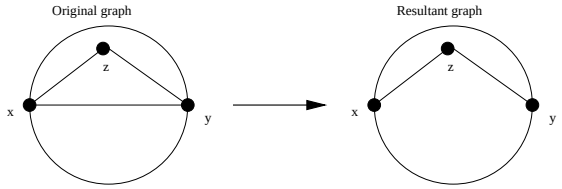


Figure 5.13: Obtaining a Gabriel Graph from a general graph.

In figure 5.13, the edge *(x,y)* fails to satisfy the condition. Hence, that edge is removed from the resultant Gabriel Graph.

Note that for a given network topology, the corresponding *Relative Neighborhood Graph* has a subset of the edges present in the corresponding *Gabriel Graph*. The proof is left as an exercise.

Before we discuss perimeter forwarding, it is important to introduce a couple of terms. An *interior face* of a planarized graph is a closed polygon enclosed by the edges of the planarized graph. There could be multiple interior faces in a graph. There exists only one *exterior face* of the graph, corresponding to the unbounded region beyond the boundaries of the graph.

Perimeter Forwarding: As stated earlier, GPSR first tries to forward packets using the greedy approach. However, if a node, *x*, is unable to find a neighbor closer to the destination, *D*, than itself, forwarding switches to the perimeter mode.

A key component of the perimeter mode is the *right-hand rule* for graph traversal. The idea behind the right-hand rule is the following: Let a packet arrive at node *x* from a neighbor *y*. Node *x* sweeps counterclockwise with respect to the edge *(x,y)*, using itself as the pivot, until an edge *(x,z)* is encountered. Node *x* forwards the packet to *z*.

When node *x* decides to switch forwarding of a packet to the perimeter mode, it inserts its location, L_p, into the packet. This indicates the location where the perimeter mode was entered. Node *x* initiates the right-hand rule traversal by sweeping counter-

clockwise with respect to the edge *(x,D)*.

When a node, *y*, receives a packet that was forwarded to it using the perimeter mode, *y* evaluates if its own distance to destination *D* is less than the distance from L_p to *D*. If so, packet forwarding reverts to the greedy mode. Otherwise, *y* employs the right-hand rule for packet forwarding.

If the edge, *(y,z)*, chosen by node *y* to forward the packet intersects with the line *(L_p,D)* the the packet is not forwarded to *z*. Instead *y* performs the following actions:

1. Switches to the other face delimited by the edge *(y,z)*. That is, *y* sweeps counterclockwise about itself with respect to the edge *(y,z)* until it encounters an edge *(y,a)*.

2. Records the first edge of the new face, *(y,a)*, in a field, e_0, of the packet.

3. Forwards the packet to *a*.

Packets are forwarded in the perimeter mode in this manner until either the destination is reached, or forwarding switches to the greedy mode. Figure 5.14 illustrates packet forwarding when only the perimeter mode is employed. Sweeping counterclockwise with respect to *(x,D)*, node *x* encounters edge *(x,a)*. So, *x* forwards the packet to *a*. Similarly, *a* sweeps counterclockwise with respect to *(x,a)* and identifies the edge *(a,b)*. However, edge *(a,b)* intersects with *(x,D)*. So, *a* switches to the interior face corresponding to the triangle *(a,b,c)*. As part of the face switch, *a* sweeps counterclockwise with respect to the edge *(a,b)* until it hits edge *(a,c)*. So, *a* forwards the packet to *c*. When *c* sweeps counterclockwise about *(a,c)*, it chooses edge *(c,b)* which intersects with *(x,D)*. So, *c* switches to the face represented by the triangle *(c,b,e)*. However, while sweeping counterclockwise about edge *(c,b)*, the chosen edge *(c,e)* also intersects with *(x,D)*. Hence, yet another face switch is performed, this time to the exterior face. When *c* sweeps counterclockwise with respect to edge *c,e)*, it hits edge *(c,f)*. So, *c* forwards the packet to *f*. Finally, node *f* performs a face switch to the interior face *(f,g,D)* and identifies edge *(f,D)* to forward the packet to the destination.

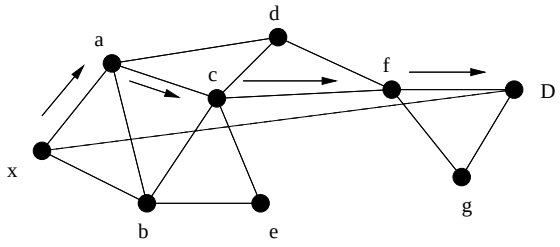


Figure 5.14: Packet forwarding in the perimeter mode as per the right hand rule

Figure 5.15 recalls the example of greedy approach failure presented in Figure 5.11, and illustrates the working of GPSR on the same topology. The packet travels from the source to the destination node using a combination of the greedy and the perimeter-based approaches. After three hops, when the packet reaches *x*, it can no longer be forwarded using the greedy approach. So, the protocol switches to the perimeter-based

approach. Sweeping counterclockwise with respect to *(x,D)* about node x, the chosen edge is *(x,a)*. So, *x* forwards the packet to *a*. Continuing with the perimeter-based approach, *a* forwards the packet to *b*, which then forwards it to *c*. After three further hops, using the perimeter-based approach, the packet reaches *f*, after passing through *d* and *e*. The sequence of solid arrows denote the path taken by the packet in the perimter mode.

You will notice that node *f* lies within the arc centered at *D* of radius *(x,D)*. So, when *f* receives the packet it realizes that it is closer to *D* than *x* is to *D*. Therefore, at *f* packet forwarding reverts to the greedy approach. From that point on, the packet is forwarded in the greedy mode through *g, h, i, j, k* and *l* to reach *D*.

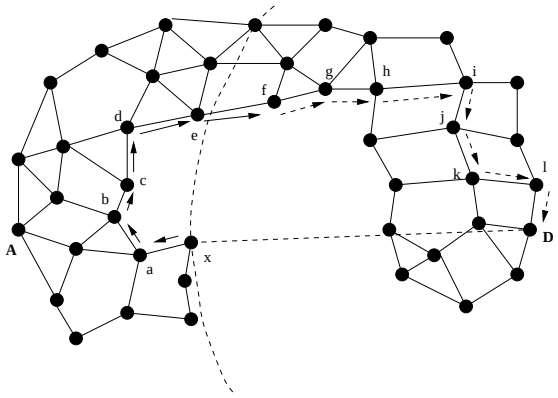


Figure 5.15: Forwarding of packets in the perimeter mode.

So far, we have implicitly assumed that the destination is reachable from the source. What if this is not the case? Do nodes keep forwarding the packet unnecessarily until the TTL field (or *hop count* for IPv6 packets) becomes zero? GPSR addresses this issue in the following manner. The unreachable destination either lies within an interior face, or outside the exterior face. In either case, when the packet reaches this face, it goes around the face and ends up at the node where it entered the face. When this node selects the next edge to forward the packet, it realizes that the selected edge has already been visited: it is listed in the e_0 field of the packet. So, the node concludes that the destination is unreachable. For example, in Figure 5.15, had the destination been somewhere outside the exterior face to the right of *x*, the packet would have been forwarded along the periphery of the network, starting with edge *(x,a)*, and ended up at *x*. On its second arrival at *x*, assuming edge *(x,a)* is still present and chosen as the next selected edge for forwarding, unreachability would be detected.

5.4.3 Discussion

All through our discussion of geographical routing we have assumed that there exists a location service. Source nodes use this service to determine the destination node's location. The critical question is how do the overheads of supporting the location service compare with the route discovery and maintenance overheads of topology-based

routing protocols? If the overheads are comparable then the perceived advantage of geographical routing will disappear. On the other hand, if the cost of providing the location service is significantly lower the cost of maintaining topology information, geographical routing can become a viable alternative.

A location service will need to support two operations: (i) location update, and (ii) location query. Location update operations allow nodes to register their new location with nodes that maintain location information. Location queries will enable source nodes to obtain the destination nodes' location information. Intuitively, a highly dynamic MANET will trigger a high volume of location updates. Location query costs will also depend on the dynamism of the network, in addition to the relative locations of the packet source and destination. If the two nodes are close to each other, an efficient location service may be able to limit the query costs by limiting the scope of the query messages. Similarly, if the volume of location queries for a node x originating from a region R exceeds a threshold, the location service may try to place information about x's location close to R.

If these issues intrigue you, the paper by Li et al. [35], that describes a location service to aid geographical routing in MANETs, is a good starting point.

Chapter 6

Mobile IP

By now we have some familiarity with routing issues in wireless ad hoc networks. In Chapter 5 we discussed various protocols to route packets in such a network. In this chapter we will focus on routing packets in a mobile network that consists of a fixed part and a mobile part. The fixed part could be anything from a local area network to the entire Internet. Examples of mobile nodes could be laptops equipped with a wireless communication device. The mobile nodes can move from one part of the network to another while maintaining connectivity. The goal is to be able to deliver packets destined for mobile nodes in a *timely* and *secure* fashion. Mobile IP has been developed for this purpose: to add mobility support to IP, the predominant Internet routing protocol.

Before we delve into the details of Mobile IP, let us briefly consider the mobility support we have in our lives. Suppose an extended trip takes me away for a month from my house in Dallas, Texas to New York city. A few days before my trip I go to the post office near my Dallas house. There I fill a form listing my current and future temporary addresses. I also specify the date on which I will be ready to start receiving mail at my temporary address in New York, and the duration of my stay there. My trip has been planned in a hurry and I have not had an opportunity to inform others about it. Moreover, even if I did inform all my friends about my new address there is no way I can predict all the people and organizations that will be writing to me during my absence from Dallas. So, letters may still get sent to me at my Dallas address.

When such a letter arrives at my Dallas post office, the post office generates an address label with my New York address. This label is affixed to the envelope, and the letter is forwarded to the post office in New York that serves my apartment. The post office in New York delivers it to my apartment. This obviously increases the time it takes for letters to reach me. I can write to people, informing them of my new temporary address. Subsequently, they can write to me directly at my New York address for the duration of my stay there.

The post office in Dallas will discontinue the forwarding at the end of the period specified by me in the form. From that time onwards it will resume delivering the letters to my Dallas address. If due to some reason my stay in New York gets extended it is my responsibility to contact the Dallas post office before the expiry of the period

and request an extension of the service. If I fail to contact the Dallas post office before the expiry of the period, the forwarding service will be discontinued.

If I return from New York earlier than planned I have to inform my Dallas post office. On receiving an intimation from me they will discontinue the forwarding service and resume delivering the mail to my permanent address in Dallas. Certain types of mail, like third class mail, are normally not forwarded. If I wish these letters to be forwarded to me I have to make special arrangements with my Dallas post office.

Two important requirements for the correct operation of this service are: (i) knowledge of the identity of post offices in Dallas and New York that will serve me, (ii) prevention of frauds where someone can assume my identity and request the Dallas post office to forward my mail to an address that he specifies. Usually, we can use some kind of directory service, or ask a postal employee for the address of the post offices. My experience has been that nobody ever asks me for any identification when I request the post office to start or stop forwarding my mail. There is an assumption that people will refrain from *hijacking* other people's mail. Also, if they are caught in such an act, there are fairly severe punishments.

If everything works fine I can rest assured that all the letters addressed to me will get delivered to me, even if I keep changing my address. Mobile IP implements mobility support in computer networks along similar lines, with special attention to security issues.

A node X has an IP address, say X_{home}, that is used by other nodes to identify X. In fixed networks this address, X_{home}, also reflects the location of X. X_{home} can be split into two parts: (i) the network prefix, and (ii) host number. The network prefix identifies the subnet in which X resides while the host number is used to distinguish X from other nodes in the same subnet. In fixed networks a node's IP address is also used to identify the node. If node Y wishes to send a packet to X it uses X_{home} to identify the packet destination. Intermediate routers use X_{home} to make forwarding decisions.

However, in mobile networks a node's identity and location are not always the same. What if X moves away from its home subnet into some other network? X cannot continue to receive packets with destination address X_{home}. Y may not be aware of a change in X's location and address, and may continue to send packets for X at the address X_{home}. In order to prevent these packets from getting lost, a level of indirection is supported. When X moves away from its *home network* into a *foreign network*, X acquires a care-of address, X_{away}. The home network is informed about the care-of address. When packets meant for X arrive in the home network they are forwarded to the care-of address. As X moves from one foreign network to another it relinquishes the outdated care-of address and acquires a new care-of address. So, (i) are there special nodes in the home and foreign networks that support this level of indirection, and if so, (ii) how do they operate? The answers to these questions vary slightly depending on whether IP version 4 (IPv4) or IP version 6 (IPv6) is the IP protocol being used. However, the mobility support functions for both IPv4 and IPv6 can be classified into the following three categories:

Agent discovery consists of operations performed to determine the identity of special routers accessible to a mobile node, referred to as the home and/or foreign agents. The primary function of the home agent is to act as a proxy for the mobile node

in its home network as long as the mobile node is in some foreign network. A foreign agent provides a visiting mobile node with a care-of address in Mobile IPv4. Mobile IPv6 does not require foreign agents. Agent discovery can be accomplished by the agents advertising their presence within the subnet they serve. Alternatively, a mobile node may solicit service from available agent(s) in the subnet.

Registration involves a mobile node informing its home agent about its care-of address. Registration is performed when a mobile node enters a foreign network, and is valid for a finite duration of time. If a mobile node intends to stay in a foreign network for a duration that exceeds the registration duration, it is the responsibility of the mobile node to renew its registration. When the mobile node moves out of a foreign network it has to perform the deregistration operation to remove the outdated care-of address from the home agent's records. This happens when the mobile node either returns to its home network, or moves out of one foreign network into another foreign network.

Packet Forwarding is the operation of a home agent forwarding the packet meant for a mobile node to its care-of address.

Before going into the details of the three operations listed above, let us consider some of the requirements for a protocol that supports mobility for the IP protocol:

1. *Transparency:* A mobile node that disconnects from one part of a network and connects at another part of the network should be able to continue communicating using the same home IP address. Thus, node mobility should be hidden from applications running at the transport layer and above, both at the mobile node and other nodes that communicate with the mobile node.

2. *Interoperability:* A mobile node that implements Mobile-IP should be able to communicate with other nodes, mobile as well as stationary, that do not implement Mobile IP.

3. *Security:* The identity of nodes should be authenticated to ensure that unauthorized nodes cannot request a home agent to forward packets meant for a mobile node to a wrong address.

4. *Efficiency:* The number and size of control messages required to implement Mobile IP should be as small as possible. This is required to conserve the scarce bandwidth of the wireless channels used to communicate with the mobile nodes. Lower communication overheads also reduce energy consumption by the mobile nodes.

5. *Unconstrained addressing:* Mobile IP should not place any additional constraint on the assignment of IP addresses to nodes.

Both Mobile IPv4 (MIPv4) and Mobile IPv6 (MIPv6) employ similar strategies to support mobility. However, there are some significant differences in implementation. These differences are due to the following reasons:

1. IP address are scarce in IPv4. So, in MIPv4 several mobile nodes visiting a foreign network may share the same care-of address. MIPv6, with its much larger 128-bit address space has no paucity of addresses. So, all the mobile nodes visiting a foreign network acquire unique care-of addresses. Some mobile nodes may have more than one care-of address: one primary care-of address, and one or more seconday care-of addresses.

2. MIPv6 does not need foreign agents to support the communication of mobile nodes that are currently residing in foreign networks.

3. MIPv4 uses packet encapsulation (described later in the chapter) to redirect packets from the home network of a mobile node to its foreign network. MIPv6 uses an IPv6 routing header for this purpose. This reduces the communication overheads.

4. MIPv4 and MIPv6 use directed broadcast and anycast, respectively, to discover home agents. As a result, mobile nodes may receive multiple replies (one from each home agent) in MIPv4. A mobile node receives a reply from only one home agent in MIPv6.

5. MIPv4 employs ARP protocol to find the link-layer address of nodes, while MIPv6 uses IPv6 Neighbor Discovery which includes functionality provided by IPv4 protocols like ARP, ICMP Router Discovery and ICMP Redirect.

6. Route optimization is an option in MIPv4. It is an integral part of the protocol in MIPv6.

7. Unlike MIPv4, MIPv4 does not provide explicit support for broadcasts. Broadcasts are treated as a special case of multicasting.

6.1 Agent Discovery

As mentioned earlier, mobility support in networks requires the presence of home agents and foreign agents. Each mobile node belongs to a home network. In the home network reside special routers referred to as the home agents. A mobile node selects one of them at *its* home agent. When the mobile node moves out of its home network, into a foreign network, it has to obtain a foreign network care-of address. In MIPv4 the care-of address is usually obtained from a foreign agent. Sometimes the care-of address can be obtained using a protocol like DHCP. In the case of MIPv6, there is no foreign agent. So, the care-of address is always obtained using IPv6 mechanism for stateless or stateful auto-configuration, as described later.

Agent discovery can be performed in the following ways:

1. *Agent advertisement:* This involves an agent periodically announcing its presence to all nodes to its network. When a mobile node is in its home network and hears such an advertisement it knows that the announcer is its home agent. When

a mobile node outside its home network hears such an announcement it identifies the announcer as the foreign agent for the network in which it is currently residing.

2. *Agent solicitation:* Sometimes a mobile node may not be able to hear any agent advertisement for an extended period of time. The node gets *impatient* and starts broadcasting messages in its current network, looking for an agent. When such a solicitation message is received by an agent it sends a response to the mobile node. Sometimes, a mobile node residing in a foreign network may need to discover the identity of its home agent. This may happen if its previous home agent is no longer operation due to node failure, or due to reorganization in the home network. In such a situation agent solicitation messages have to be sent to the home network, and corresponding replies have to be awaited.

As home and foreign agents are routers providing mobility support, the agent advertisement and solicitation messages are extensions of the ICMP Router Advertisement and ICMP Router Solicitation messages, respectively.

6.1.1 Agent Advertisement

A router broadcasts an ICMP Router Advertisement message at regular intervals on each of the links to which it is connected. The advertisement sent by a router on a link may contain multiple IP addresses associated with the router interface from which the message is sent. If the *home agent bit* in advertisement is set to one, it indicates that the advertising agent can act as the home agent for nodes in the network it serves. In MIPv4, if the *foreign agent bit* is set, it indicates that the advertiser offers services as a foreign agent. A preference level is associated with each advertised address. The higher the preference level, the greater the willingness of the router to serve other nodes in the network using this address as the home or foreign agent. Each advertisement has a lifetime, in seconds, associated with it. This is the maximum duration for which the advertised addresses can be considered valid.

Each home agent maintains a *Home Agents List* for each link on which it is acting as a home agent. This list is constructed from information contained in Router Advertisements heard by the home agent on the link. The following actions are performed by a home agent to maintain and update its Home Agents List corresponding to a particular incident link:

1. Let the Home Agent List contain an entry for a router. If the router advertisement sent by that router does not have the home agent bit set, or if the home agent bit is set but the advertised lifetime is zero, the router's entry is deleted from the Home Agent List.

2. If the home agent bit is set in the advertisement received from a router that is not in the Home Agent List, and the advertised lifetime is non-zero, the router is added to the Home Agent List with its preference and lifetime values extracted from the advertisement.

3. If the home agent bit is set in an advertisement received from a router already in the list, and the advertised lifetime is non-zero, the home agent resets the lifetime and preference values for that router using the valued contained in the advertisement.

4. An entry is deleted from the Home Agent List when its lifetime expires.

The entries in the Home Agent List are maintained in decreasing order of their preference.

6.1.1.1 MIPv4 Specific Actions

It is suggested that if agent advertisement messages are to be sent at regular intervals, the interval between successive advertisements should be approximately one third of the advertised lifetime, with the actual interval randomly perturbed to avoid collisions. Thus a mobile node can miss up to two successive advertisement messages without deleting the advertiser from it list of available agents. Besides the router address(es) and lifetime, the agent advertisement message contains the following information:

1. *Sequence number:* This is a 16-bit integer whose value varies between 0 and $65,535$. When an agent is booted its first advertisement message has a sequence number 0. Successive advertisements increase the sequence number by one. However, when the sequence number reaches the maximum permissible value it does not roll over to zero. Instead, it rolls over to sequence number 256. Let a mobile node observe a decline in the sequence number of consecutive messages heard from a foreign agent. If the latter advertisement had a sequence number in the range $0 - 255$ it is an indication that the foreign agent must have crashed and has been rebooted. This would then trigger a new registration by the mobile node. However, if the earlier advertisement's sequence number was closer to $65,535$ and the latter advertisement's sequence number was 256 or higher the mobile node need not reregister with its home agent. After-all, what are the chances of a mobile node missing at least 256 consecutive advertisements by an agent in its network?

2. *Registration lifetime:* This is the longest period for which the a mobile node can be registered with the advertising agent for mobility support services. The maximum permitted registration duration is $65,534$ seconds.[1] If this value is set to $65,535$ it indicates an infinite duration. It is important to distinguish between registration lifetime and advertisement lifetime. If an advertisement is not heard for the advertisement lifetime, the advertising agent is considered to be lost.

3. *Busy bit:* This bit, in a foreign agent's advertisement, indicates the agent's willingness to provide service to additional mobile nodes. If the bit is set the foreign agent is not prepared to provide mobility support to any more nodes. Note that a home agent can never claim to be too busy to serve the mobile nodes in its network, $i.e.$, a home agent's advertisements should have the busy bit set to zero.

[1] This is like the six month upper limit on letter forwarding provided by post offices.

4. *Care-of addresses:* This specifies the care-of addresses of the foreign agent that the mobile nodes can use. If the advertisement has the foreign agent bit set, then this field should have at least one care-of address.

5. Registration required bit: If this bit is set (only permitted in a foreign agent's advertisements), it indicates that visiting mobile nodes *must* register with some foreign agent. You may be wondering about the utility of this field. After-all, does the visiting mobile node have any other option? In fact, there does exist another option: that of the visiting mobile node using a *colocated care-of address* instead of a foreign agent's care-of address. The meaning of a colocated care-of address will be described later, in Section 6.2.

6.1.1.2 MIPv6 Specific Actions

MIPv6 also uses sequence numbers to distinguish between fresh and stale advertisements. However, unlike MIPv4, when the sequence number reaches the maximum permissible value it does rollover to zero. A newly received sequence number, x, is considered to be less than the previously received sequence number, y, if $y - 2^{15} \leq x < y$, modulo 2^{16}.

The advertisement may carry a 16-bit integer denoting the home agent lifetime in units of seconds. The maximum lifetime is 18.2 hours. Also, this field is prohibited from having a value of 0.

As stated earlier, MIPv6 employs the Neighbor Discovery protocol for agent advertisements. The Neighbor Discovery protocols stipulates that a router cannot send Router Advertisement messages more than once every three seconds. This means that mobile nodes that move into new networks may take as long as three seconds to detect the presence of new routers. Similarly, mobile nodes may delay concluding that they have moved out of their old network by several second. To alleviate this problem, MIPv6 allows routers that may be serving mobile nodes or may be acting as home agents to send their router advertisements with a higher frequency (smaller inter-advertisement interval) on the corresponding links. However, there is a tradeoff: smaller the inter-advertisement interval, faster the detection of mobility and associated mobility bindings, but greater the communication overheads.

6.1.2 Agent Solicitation

The agent solicitation message is an extension of the ICMP router solicitation message. In MIPv4 agent solicitation may be performed by a mobile node to determine the identity of home agents while in the home network, and foreign agents while away from the home network. In MIPv6 there are no foreign agents. So, agent solicitation is performed to identify the home agents. A mobile node can do so on returning to a home network, or while away from the home network.

MIPv4 specifies that the maximum rate at which a node can send agent solicitation messages is one per second. Initially, a mobile node can broadcast up to three agent solicitation messages in the network in which it is present at the maximum rate. If some agent responds to an agent solicitation message the mobile node is successful in

finding an agent. The response is a unicast message sent by the agent to the soliciting node, and is similar to an agent advertisement message.

However, if no response is received to these messages the mobile node continues to send agent solicitation messages, albeit at a lower rate. This decrease in agent solicitations is done with the purpose of limiting the solicitation overheads. The interval between successive solicitation messages is doubled each time a response to the latest solicitation is not received. This exponential backoff is performed until the maximum interval is reached. The value of the maximum interval is an implementation issue and should be determined based on the characteristics of the network.

Once a mobile node has found an agent the node registers itself with that agent. The next time the node has to find an agent to register with, it resumes sending the agent solicitation messages at the maximum rate, and performs exponential backoff when replies are not received.

In the context of MIPv6, a mobile node that is currently in the foreign network may not know the identity of any home agent on its home link. This may be so because the router that it chose a its home agent is no longer acting as a home agent. So, to find a home agent the mobile node performs dynamic home agent address discovery. The mobile node sends an ICMP Home Agent Address Discovery Request message with its care-of address as the source address, and the MIPv6 Home Agent Anycast address corresponding to its home network as the destination IP address.

This address discovery request message, being an anycast message, is delivered to one of the home agents that responds with a Home Agent Address Discovery Reply message. This reply message contains the replying home agent's Home Agent List with the home agents: (i) arranged in descending order of preference, and (ii) home agents with the same preference randomly ordered. On receiving the reply, the mobile node tries to register with one of the home agents listed in the reply.

Falling back to our post office example, we have performed the equivalent of identifying the Dallas and New York post offices. Now, we need to perform *registration* which is equivalent to setting up the forwarding service at these post offices.

6.2 Registration

When a mobile node enters a foreign network it initiates the registration process. This involves acquiring a care-of address and informing the home agent of this address. If the registration is successful, the mobile node's home agent maintains information about it and can start *tunneling* packets meant for the mobile node to its care-of address. The home address to care-of address mapping and associated lifetime information maintained by the home agent about a mobile node is referred to as the *mobility binding* for that node.

The registration has a finite lifetime. Before the registration expires, the mobile node has to extend it by performing another registration, also referred to as a binding update. When a mobile node returns to its home network it has to deregister with its home agent. Deregistration is an indication to the home agent that it no longer needs to forward the packets meant for the mobile node to that node's care-of address.

When a mobile node is away from its home network it can register with the home agent in one of the following two ways:

1. through a foreign agent using the foreign agent's IP address as its care-of address, or
2. by acquiring a colocated care-of address and directly communicating with the home agent.

As MIPv6 does not use foreign agents, it does not use the first approach.

The latter approach was added to Mobile IP keeping in mind that, in the future, powerful mobile nodes can directly communicate with the home agent rather than going through the foreign agent.

6.2.1 Registration Procedure

The registration procedure basically consists of: (i) the mobile node originating a *registration request* message whose ultimate destination is the mobile node's home agent, and (i) the home agent sending a reply message either granting or denying the request. If colocated care-of addressing is being used the mobile node directly sends the request to the home agent who, in turn, can directly send the reply to the mobile agent. Otherwise, the request and the reply are relayed through the foreign agent.

User Datagram Protocol (UDP) messages are used for registration purposes. UDP provides connectionless, unreliable communication at the transport layer. It's connection-oriented, reliable counterpart, TCP, is not used for sending registration messages because Mobile IP registration does not require windowing, renumbering of messages, congestion control or flow control. Furthermore, Mobile IP itself handles dropped messages through retransmissions. So, using TCP would be an overkill.

6.2.1.1 Registration Request

A mobile node needs to send a registration request either to refresh an existing registration before it expires, or when the node detects that it has moved from one subnet to a different subnet.

Care has to be taken when a mobile node is trying to determine if it has moved out of one network into another. Neighbor Unreachability Detection can be employed for this purpose in the following manner. Protocols at the upper layer can provide hints to the network layer if IP packets sent to other nodes in the recent past are being correctly delivered. If such hints are not received, the mobile node sends Neighbor Solicitation messages to nodes that are the next hop on the route to those destinations and expects to receive Neighbor Advertisements. The absence of the corresponding Neighbor Advertisements indicates that the mobile node has moved to a different network.

However, Neighbor Unreachability Detection is useful in detecting moves only if the mobile node has packets to send. Hence, this move detection method should be augmented with other methods. One approach is *lazy cell switching*. You may recall that each router advertises its presence from time to time. The router advertisement contains the maximum time between successive advertisements. If the mobile node

does not hear another advertisement from the same agent before the expiry of the previous advertisement's lifetime it concludes that the agent is no longer accessible, or the advertisement was lost. After a certain number of advertisements from that router are not heard in succession, the mobile node can conclude that the router is not accessible. If the node has heard some other agent's advertisement, whose lifetime has not expired, it tries to register with that agent. If no such agent exists, the mobile node has to initiate an agent solicitation.

Sometimes, Layer 2 (data link layer) information can be used to determine if the mobile node has moved from one network to another. However, each change in Layer 2 association does not imply a move between two networks.

Once a mobile node has concluded that it has moved out of one network, and into another it needs to acquire a new care-of address. The mobile node prepares the request message to register this information with its home agent.

6.2.1.2 MIPv4 Specific Actions

The *source address* field contains the interface address from which the message is sent. So, if this message is sent by the mobile node to the foreign agent or directly to the home agent, the source address field contains the former's IP address. If the message is sent by the foreign agent to the home agent the source address field has the foreign agent's IP address. The *destination address* field contains either the address of the foreign agent or the home agent, depending on who is sending the message and whether colocated care-of addressing is being employed.

There are two UDP fields. The *source port* is variable while the *destination port* is set to 434.

There are multiple Mobile IP fields. Some of the fields of interest are:

1. *Type:* This field is set to 1 indicating that this is a registration request message.

2. *Simultaneous binding bit:* A mobile node can have multiple care-of address bindings at the home agent. If this bit is set in the request message the home agent adds a new binding while maintaining the earlier bindings. If this bit is not set, the home agent deletes all the old bindings. Simultaneous binding comes in handy if a mobile node frequently moves between multiple foreign networks. It enables the mobile node to receive the packet in any of those networks.

3. *Broadcast bit:* Normally a home agent will only forward packets explicitly sent to the mobile node, and not forward those messages that were broadcast to all nodes in the home network. However, if the broadcast bit is set in the mobility binding, the home agent also forwards to the mobile node all messages broadcast to nodes in the home network.

4. *Decapsulation bit:* By setting this bit the mobile node informs the home agent that it will decapsulate the packets sent to the care-of address. The decapsulation bit is set if the mobile node is using a colocated care-of address.

5. *Lifetime:* This is the duration in seconds for which the registration is requested. If the registration is being performed through a foreign agent, the lifetime should

be less than or equal to the registration lifetime value in the agent advertisement message from the foreign agent.

6. *Home address:* This field contains the IP address of the mobile node.
7. *Home agent:* This field contains the IP address of the mobile node's home agent.
8. *Care-of address:* If colocated care-of addressing is being employed, this field contains the care-of address acquired by the mobile node in the foreign network. Otherwise, it contains the address of the foreign agent.
9. *Identification:* This is a 64-bit number generated by the mobile node for security purposes and is used to match the registration request with its reply.

A mobile node should generate at most one registration request per second. If this registration request is being generated to refresh an existing registration it should be generated well in advance of the expiration. This is necessary to account for propagation delays, message losses, and possible retransmissions.

6.2.1.3 MIPv6 Specific Actions

Though there are some significant differences in message formats, registration (referred to as binding update) in MIPv6 is similar to that in MIPv4. As there are no foreign agents, there is no need to convey information about who will decapsulate packets sent by the home agent to the mobile node. The mobile node's care-of address should be used as the source IP address in the binding update message. The proposed lifetime should not exceed the remaining lifetime of the home address and the care-of address.

A mobile node is supposed to remember the sequence number used by it to send its last binding update to its home agent. A new binding update should have a sequence number one greater than the previous update's sequence number. A mobile node that does not remember the last sequence number used by it generates a binding update with an arbitrarily selected sequence number. The home agent can reject such update messages and, in the in the corresponding error message, send the last accepted sequence number. The next binding update message from the mobile node should have a sequence number one higher than that received in the acknowledgement.

6.2.1.4 Registration Reply

The reply message informs the mobile node whether its registration request has been granted or denied. There is a *status code* field indicating whether the registration has been successful or denied. There are different codes to indicate the reason for denial of the request. Additionally, in MIPv4 the code also indicates whether the registration request was denied by the home agent or the foreign agent.

The lifetime field indicates the duration for which the registration request has been granted. Once a request has been issued no node can increase the value in the lifetime field. However, a home agent can grant a registration request for a duration that is less than the requested lifetime.

Actually, soon after issuing a registration request the mobile node starts counting down the duration of the requested registration lifetime. When a positive reply is received, the value of the downward counter is equal to the remaining lifetime of the requested registration. If the lifetime value received from the home agent is equal to the lifetime originally requested, the remaining duration of registration is set to be equal to the value of the downward counter when the reply was received. If the lifetime granted by the home agent is less than the requested lifetime, the remaining duration of the registration is modified to be equal to the value of the downward counter minus the difference between the requested and granted lifetimes.

6.3 Packet Delivery

Now that we have discovered the home and foreign agents, and performed the appropriate registrations, the task at hand is to correctly deliver packets to mobile nodes. One approach for the delivery of packets to mobile nodes is through a process called *tunneling*.

In a fixed network the source specifies the IP address of the destination in the packet header, and the routers deliver the packet to the destination. However, with nodes changing their location and moving out of their home networks, two new entities are required: the *encapsulator* and the *decapsulator*.

Packets sent to the destination get delivered to the encapsulator. The encapsulator then *encapsulates* the packet and forwards it to the decapsulator. The decapsulator *decapsulates* (what else did you expect!?!) the packet and delivers it to the destination node. The encapsulator and decapsulator are considered to be the two endpoints of a tunnel. The process of forwarding packets between the endpoints is referred to as *tunneling*.

A pertinent question at this stage is: how do packets destined for a mobile node get intercepted by the encapsulator? This is achieved by exchanging appropriate *address resolution* messages to be described shortly. It is only after address resolution has been performed that tunneling can be done.

6.3.1 Address Resolution

Let an IP packet arrive at a router, and let the router discover that the its own IP address and the destination IP address (in the packet) have the same network prefix. This leads the router to conclude that the destination node and the router belong to the same subnet. So, the router has to deliver the packet to the destination node, instead of forwarding it to some other router. In order to deliver the packet, the router needs to know the data link layer address of the destination node. The IP packet is sent by the router to the destination as a data link layer frame, with the destination's data link layer address in the frame. The destination node accepts the frame, while other nodes normally ignore it. The protocol that helps routers maintain the mapping between IP addresses and the corresponding link layer addresses of nodes in their subnet is the *Address Resolution Protocol (ARP)* in IPv4 and the *Neighbor Discovery Protocol* in IPv6. Each router maintains a cache containing the mapping.

When a mobile node is not in its home network, what is the router for the home network to do with the packets that arrive for the mobile node? It cannot deliver them to the destination node. Mobile IP specifies that these packets be delivered to the home agent which is in the same subnet as the router.

6.3.1.1 MIPv4 Specific Actions

In MIPv4, while the mobile node is absent from its home network the ARP cache of the router should map the mobile node's IP address to the data link layer address of the home agent. When the mobile node returns to its home network, the ARP cache should be modified to reflect the change. For this purpose the following ARP messages are used: *gratuitous ARP* and *proxy ARP*.

On registration from a foreign network: Let a mobile node move out of its home network and register with the home agent through a care-of address. The home agent broadcasts a gratuitous ARP packet on its subnet. The gratuitous ARP packet contains the IP address of the mobile node in its source and destination fields. It also contains the data link layer address of the home agent as the sender's hardware address. When other nodes in the subnet receive this message they update their ARP cache to map the mobile node's IP address to the home agent's data link layer address.

On return to home network: Once a mobile node returns to its home network, packets destined for it should not be intercepted the home agent. Instead, they should be delivered to the mobile node. Towards this goal, before deregistering with the home agent, the mobile node broadcasts a gratuitous ARP reply message in its home network with its own IP address as the source and destination IP addresses, and its data link layer address as the source and target hardware addresses. All other nodes in the network, including the home agent record this information in their ARP cache.

After the mobile node has deregistered with the home agent, the home agent also broadcasts a similar ARP reply message to the home network. It is possible that some nodes in the home network can only hear the home agent but not the mobile node, and vice versa. Hence, to propagate information about the mobile node's return the home agent also needs to perform a broadcast of the gratuitous ARP reply message.

On address enquiries: Sometimes nodes may not have a mobile node's IP address to data link layer address mapping in their ARP cache. When such nodes have to communicate with the mobile node they first send an ARP request message to find the mobile node's data link layer address. When the home agent encounters this message it looks in its list of nodes that originally belong to this network, but are registered through a care-of address. If the mobile node belongs to the list, the home agent sends a *proxy ARP* reply on behalf of the mobile node with its data link layer address instead of the mobile node's link layer address. On receiving the reply, the receiver updates its ARP cache to map the mobile node's IP address to the home agent's data link layer address.

Note that when a mobile node is roaming in a foreign network it is not allowed to broadcast any ARP request or reply messages. Also, if the roaming mobile node encounters an ARP request message whose destination address is the mobile node's home address, the mobile node refrains from replying. The only exception is when the request originated from the foreign agent with which the mobile node is either registered or with which the node is trying to register. In such a situation the reply should be a unicast message meant for the foreign agent, and should not be broadcast to the foreign network.

Thus, a mobile node restricts its ARP processing capabilities when it moves out of its home network, and lifts those restrictions on returning to the home network. Also, a home agent stops sending proxy ARP replies on behalf of a mobile node the moment that node returns to the home network and deregisters.

6.3.1.2 MIPv6 Specific Actions

MIPv6 employs Neighbor Advertisement messages to ensure that appropriate IP address to link layer address mappings are maintained. The home agent multicasts a Neighbor Advertisement message to all nodes on the home link. The source IP address in the IPv6 header is the home agent's IP address, while the target IP address is the mobile node's home IP address. The target link-layer address is the home agent's link-layer address. In addition, there is an Override Flag in the message. This flag must be set forcing every node on the home link to replace any previous binding it may have for the target IP address in its Neighbor Cache to the new binding which corresponds to the mobile node's home address and the home agent's link-layer address.

As in MIPv4, the home agent acts as a proxy for the mobile node. On receiving Neighbor Solicitation messages with the target address field containing the mobile node's home address, the home agent responds with a Neighbor Advertisement message as described above.

On return to home network: As in MIPv4, the goal is to stop the home agent from intercepting packets that have the mobile node's home address as the destination address. Towards this goal, the mobile node sends a binding update message to the home agent's link-layer address with: (i) the source and care-of addresses being the mobile node's home address, and (ii) lifetime set to zero. On receiving this message the home agent stops intercepting packets destined for the mobile node's home address, and sends a binding acknowledgment. On receiving the binding acknowledgement, the mobile node sends a Neighbor Advertisement message to the all-nodes multicast address on its home link with the Override flag set. The target IP address and link-layer address in this message are the mobile node's home IP address and link-layer address, respectively. This will enable other nodes on the home link to update their neighbor cache entries for the mobile node's home address: replacing the home agent's link-layer address with the mobile node's link-layer address.

If the mobile node that has just returned to the home network does not know the home agent's link-layer address, things can get interesting. Normally, one would expect the mobile node to send a Neighbor Solicitation message to find the home agent's link-layer address. However, note that the home agent is still under the impression that:

(i) the mobile node is in some foreign network, and (ii) the home agent is responsible for intercepting packets on behalf of the mobile node. So, any Neighbor Solicitation message with the mobile node's home IP address as the source address is likely to cause concern: signalling that some other node has acquired the mobile node's home IP address, violating the uniqueness of IP addresses assigned to nodes. Hence, if the mobile node needs to know the link-layer address of the home agent it should multicast the Neighbor Solicitation message with the source address set to the unspecified address (all zeroes), the target address set to the mobile node's home address, and the destination address set to the Solicited-Node multicast address. The home agent will respond to such a message with a Neighbor Advertisement containing its link-layer address.

6.3.2 Tunneling

Once registration and address resolution have been accomplished, packets meant for a roaming mobile node, but intercepted by its home agent, have to be tunneled by its home agent to the foreign network where the mobile node is present. Likewise, a mobile node an tunnel packets to its home agent. The home agent can then forward the packets towards the intended destination. A bidirectional tunnel consists of two unidirectional tunnels: (i) the forward tunnel from the home agent to the mobile node, and (ii) the reverse tunnel from the mobile node to its home agent. A tunnel makes the path between its end-points appear like a single virtual link to the packets being tunneled. This means that an IPv4 packet being tunneled will have its time-to-live (TTL) field decremented by one during its passage through the tunnel. Similarly, an IPv6 packet will have its hop limit decremented by one as it passes through the tunnel. Tunneling requires packet encapsulation and decapsulation at the tunnel endpoints, and can be achieved through IP-in-IP encapsulation.

6.3.2.1 IP-in-IP Encapsulation

In this encapsulation method an IP packet is carried as the payload of another IP packet. For example, when the home agent of a roaming mobile node has to forward tunnel a packet to the care-of address of the mobile node, it can perform IP-in-IP encapsulation. The home agent takes the IP packet meant for the mobile node and adds a new IP header, and possibly some option fields, before the packet. The source and destination addresses of the new header correspond to the IP addresses of the encapsulator (home agent) and decapsulator (the care-of address), respectively. On delivery to the mobile node at its care-of address, the mobile node strips-off the outer header and obtains the packet that was originally sent by the source node. During reverse tunneling, the original packet's source address is the IP address of the mobile node and the destination address is the IP address of the destination. The outer header has the care-of address of the mobile node and the home agent's address as its source and destination addresses.

From the point of view of the original source and destination of the packet, the path from the encapsulator to the decapsulator is *one logical hop* in length. Hence, at the time of encapsulation, if it is an IPv4 packet the TTL (time to live) field of the inner IP header is decremented by one. If as a result of this decrement, the TTL value becomes zero the packet is discarded and an ICMP message is sent to the packet source. The

ICMP message indicates that the time limit has been exceeded.[2] The decrease in the TTL value is the only modification to the inner IP header. Some optional header fields may be inserted between the inner and outer headers. Similarly, for IPv6 packets that are being tunneled, the hop-limit field is decremented by one.

None of the routers between the two endpoints of the tunnel see the original IP header. So, even if they employ *ingress filtering* the tunneled packet passes through.

6.3.2.2 Route Optimization

If tunneling is employed, whenever a correspondent node has to send a packet to a mobile node that is not in its home network, the packet is routed *indirectly:* first from the correspondent node to the mobile node's home agent, then from the home agent to the mobile node's care-of address. It is proposed that nodes maintain a binding cache that contains mobility binding about mobile nodes. Just like the binding cache at the home agent, an entry in the binding cache at a correspondent node contains the home address and the care-of address of the mobile node. If the correspondent node can somehow be informed about the mobile node's care-of address, future packets can be directly sent to this care-of address, bypassing the home agent. This will speed up packet transfer and reduce the burden on the home agent.

In MIPv4, a mobile node's home agent takes active part in updating the binding cache entries for that mobile node. Let the home agent receive a packet from its home network to be tunneled to the foreign agent. The home agent infers that the source node does not have a binding cache entry for the mobile node. So, the home agent sends a binding update entry to the source node. This update contains the current mobility binding of the mobile node. Once the binding update is correctly delivered to the source node, it can directly send all the following packets destined for the mobile node to that node's care-of address. Similarly, let some node with a binding cache entry, but not a visitor list entry, for a mobile node receive a tunneled message destined for that mobile node. The node infers that the sender of the message has an outdated binding cache entry for the mobile node. So, it sends a warning message to the mobile node's home agent. The home agent, in turn, sends a binding update message to the sender of the tunneled message. Just like node registrations, binding cache entries are assigned a lifetime. At the end of this lifetime, the entry is discarded. Moreover, as the size of the binding cache is limited some cache management policy has to be implied to purge some existing entries to make space for new entries.

In MIPv6, the mobile node is resposible for updating the binding cache of its correspondent nodes with care-of address. If the correspondent node is not careful malicious nodes can populate its binding cache with bogus entries, thus hijacking packets meant for other nodes. Therefore, a correspondent node adds an entry into its binding cache only when it is reasonably sure that the information is valid.

[2]The TTL field's purpose is to prevent packets from living in the network for unnecessarily long periods of time. The TTL value is initially set to a finite and reasonable integer value based on the source node's estimate of the number of links to be traveled along the path to the destination. The TTL value is decremented with each hop of the packet and the packet is discarded when the count becomes zero. If the packet has still not reached the destination, then most likely it has been misrouted and attempts to route it to the destination may result in loops.

MIPv6 employs six messages as part of the *Return Routability Procedure* to update the binding cache of a mobile node.

6.3.2.3 Return Routability Procedure

Let a mobile node, currently away from its home network, receive a packet sent by a correspondent node. If the packet arrived along a forward tunnel, $i.e.$, via the home agent then the mobile node concludes that the correspondent node is unaware of its care-of address. The mobile node sends two messages to the correspondent node:

(i) *Home Test Init message*: sent by the mobile node to the correspondent node via the mobile node's home agent. The source IP address is the mobile node's home address. the message contains a 64-bit home agent cookie.

(ii) *Care-of Test Init message*: sent directly by the mobile node to the correspondent node. the source IP address is the mobile node's care-of address. The message contains a 64-bit care-of init cookie.

Each correspondent node maintains a secret key, henceforth referred to as *Kcn*. Periodically, a correspondent node also generates a random number, henceforth referred to as a *nonce*. each nonce has a certain lifetime and is identified by a nonce index. The correspondent node maintains a list of nonces whose lifetime has not expired, and their corresponding indices. The correspondent node sends the following messages to the mobile node:

(iii) *Home Test message*: sent in response to the Home Test Init message. The Home Test message is sent via the mobile node's home agent. It contains the home init cookie received in the Home Test Init message, and a *home keygen token*. The home keygen token is generated by a secure hash function that takes the correspondent node's secret key, *Kcn*, the mobile node's home address and a nonce as input parameters. the Home Test message also contains the nonce index corresponding to the nonce used in generating the home keygen token.

(iv) *Care-of Test message*: sent in response to the Care-of Test Init message. This message is sent directly to the mobile node. It contains the care-of init cookie and a *care-of keygen token*. The care-of keygen token is also generated using the same secure hash function as the home keygen token. One significant difference is that the care-of address of the mobile node is used as a parameter instead of its home address. Also, the nonce used to generate the care-of keygen token need not be the same as the nonce used to generate the home keygen token. Hence, the correspondent node is required to send the corresponding nonce index in this message.

The mobile node uses the home init cookie to match a Home Test Init message sent by it with a received Home Test message. Likewise, it uses the care-of init cookie to match the corresponding Care-of Test Init message with the received Care-of Test message. having received both the Home Test and Care-of Test messages, the mobile node employs the secure hash function to generate a *binding management key, Kbm*.

The input parameters to the hash function are the home and care-of keygen tokens. Then the mobile node and the correspondent node exchange the following messages:

(v) *Binding Update message*: sent by the mobile node to the correspondent node. It contains the care-of address to be placed in the binding cache of the correspondent node, a binding update sequence number, a message authentication code and the nonce indices received from the correspondent node. The message authentication code is generated by a secure hash function with *Kbm*, correspondent node's IP address, the care-of address and the binding update message as input parameters. On receiving the Binding Update message the correspondent node has all the information it needs to determine the veracity of the message.

Using the received nonce indices the correspondent node can retrieve teh corresponding nonces, generate teh home and care-of keygen tokens, use these tokens to produce the binding management key (*Kbm*), and use this key to compute the message authentication code corresponding to the binding update message. If the locally computed message authentication code is the same as the message authentication code received in the binding update message the correspondent node: (a) concludes that the care-of address indeed belongs to the mobile node that send the binding update, (b) updates its binding cache. The binding update has a lifetime which is recorded in the binding cache.

(vi) *Binding Acknowledgement message*: sent by the correspondent node to the mobile node's care-of address. It contains the status of the binding update attempt, a sequence number, and also a message authentication code. The sequence number is copied from the binding update message received from the mobile node.

A mobile node can also send a binding update to a correspondent node to delete its binding cache entry. This could be done when the mobile node returns to its home network and is no longer able to receive packets at its care-of address. In such situations the binding management key is computed using only the home keygen token. Also, the binding lifetime is set to zero.

The sequence number and the message authentication codes differ for every pair of nodes. For the same pair of nodes they differ from one binding update to the next. Hence, the two values together safeguard against replay attacks.

6.3.2.4 IPv6 Routing Header for Route Optimization

Once a binding cache entry has been added at the correspondent node the mobile and correspondent nodes can directly communicate with each other, bypassing the home agent. However, if the ensuing packets from the mobile node contain its home IP address in the source address field they may be discarded by intermediate routers employing ingress filtering for the following reason. An intermediate router, on inspecting the packet header and its own forwarding table, may realize that the incident link along which the packet arrived is different from the link along which packets meant for its source address should be forwarded. Suspecting foul play, the router would discard the packet.

MIPv6 solves the problem in the following manner. The source address of the packet is the mobile node's care-of address, the destination address is the correspondent node's address. A Home Address option is inserted into the packet with the mobile node's home address carried in it. None of the intermediate routers look at the Home Address option while making their forwarding decision. The packet is not discarded by routers that implement ingress filtering as it arrives along an expected interface. When the packet arrives at the correspondent node, the correspondent node can replace the care-of address with the home address and deliver the packet to protocols running at higher layers of the protocol stack.

When a correspondent node has to send a packet to a mobile node for which it has a binding cache entry the correspondent node uses a type 2 routing header for the packet. The destination address is set to the mobile node's care-of address, while its home IP address is present in the routing header. The type 2 routing header is processed only by the destination node. So, when the mobile node receives the packet it replaces its care-of address with its home IP address in the destination address field and delivers the packet to protocols running at higher layers of the protocol stack.

Thus, applications running above the network layer do not have to do anything to adapt to node mobility.

6.3.3 Support for Multicasts

A roaming mobile node that wishes to receive multicast messages has to join a multicast group. If the foreign network, in which the mobile node is currently resident, has a multicast router, the mobile node can register with that router using its care-of address. Subsequently, it can receive the multicast packets in the foreign network.

If there is no multicast router in the foreign network, or if the mobile node does not wish to use such a router to receive multicast messages, it can establish a tunnel between its care-of address and its home agent. Then, the home agent can forward the multicast messages to the mobile node along this tunnel. MIPv6 limits the multicast messages that can be tunneled by a home agent to a mobile node's care-of address to only those multicast messages that have a global scope.

A mobile node can multicast a message on the foreign network in which it is present. Such messages must use the mobile node's care-of address as the source IP address. Alternatively, the mobile node can tunnel the multicast message to its home agent. The home agent can then multicast the message using the mobile node's home IP address as the source address.

Both approaches to multicast packet transmission (directly from the foreign network using the care-of address, and using bidirectional tunnel with home agent and using home address) have their utility. The multicast tree corresponding to a specific group is source-specific. If the mobile node uses a tunnel to have its multicast packets sent by the home agent, then the multicast tree that is effectively rooted at the home agent. Therefore, the tree is not influenced by the mobility of the mobile node. In contrast, when multicast packets are sent directly from the foreign network, each time the mobile node moves to a new network the multicast tree has to be reconstructed.

However, the packet forwarding time between the home agent and the foreign network may be significant. If so, the use of tunneling for multicasting can result in

increased multicast latency. By comparison, multicasting directly from the foreign netowork using the care-of address may incur lower latency.

6.3.4 Additional MIPv4 Operations

6.3.4.1 Loop Avoidance

In MIPv4 where packets meant for mobile nodes go through foreign agents there is a likelihood of routing loops. Let the home agent receive an encapsulated packet. The outer header's destination field contains the home IP address of the roaming mobile node. If so, the home agent looks at the inner destination address. If that is also equal to the mobile node's home IP address the home agent looks at the outer source address. If the outer source address is equal to the mobile node's care-of address then the home agent discards the packet. This is because the home agent detects an inconsistency: its records indicate that the mobile node is accessible through the care-of address while the router corresponding to the care-of address thinks that the mobile node is accessible through the home agent. If the packet is not discarded it may keep looping indefinitely in the network between the home agent and the router with the care-of address. However, if the inner and outer destination addresses are different the packet is encapsulated once again with destination address of the outer header set to the care-of address of the mobile node.

In MIPv6 such a problem will not arise. If the packet with the care-of address reaches the foreign network and the mobile node is not there, the packet will get dropped.

6.3.4.2 Support for IP Broadcasts

MIPv6 does not provide explicit support for broadcasts. Broadcasting is treated as a special case of multicasting. Hence, the following description applies only to MIPv4.

If the home agent receives a broadcast packet it forwards the packet to only those mobile nodes that have requested forwarding of broadcast packets during their registration process. However, ARP broadcast packets are never forwarded. Otherwise, mobile nodes outside their home networks will start sending ARP replies in the foreign networks creating all kinds of confusion.

The tunneling procedure for broadcast packets depends on the nature of the care-of address used. If a colocated care-of address is being used by the mobile node, the home agent tunnels the packet to the care-of address. The mobile node decapsulates the received packet and obtains the broadcast packet. If a foreign agent's care-of address is being used, the home agent first encapsulates the broadcast packet in a unicast packet destined for the mobile node's home address. The resultant packet is then tunneled to the mobile node's care-of address. When the packet arrives at the foreign agent it is decapsulated. On decapsulation the foreign agent realizes that it is a unicast packet meant for the mobile node and delivers the packet to that node. The mobile then decapsulates the received packet and obtains the broadcast packet, just like the colocated care-of address case.

6.3.4.3 Decapsulation by Foreign Agent

This applies only to MIPv4. Each foreign agent maintains a list of mobile nodes that have registered with their respective home agents through this foreign agent. Let a foreign agent receive an encapsulated packet in which the outer header's destination field is set to the foreign agent's advertised care-of address. The foreign agent decapsulates the packet and looks at the destination address in the inner header. If no node with that destination address is present in the foreign agent's list there is obviously a problem. Someone thinks that the node is registered through the foreign agent when that is not the case. Usually, in such situations the foreign agent discards the packet and does not generate any error message.

6.3.4.4 Smooth Handoff between Foreign Agents

Let a mobile node move from one foreign network to another. On reaching the new network the mobile agent informs its home agent to update its mobility binding. Once the binding has been updated, subsequent packets destined for the mobile node get tunneled to the new foreign network instead of the old one. However, there is a period of time during which the mobile node has moved to the new foreign network but the home agent's mobility binding for the node has not been updated. During this time the home agent will continue to tunnel packets to the old care-of address. Unless something is done to handle this situation, the old foreign agent will quietly discard the packets. To avoid such packet drops the mobile node requests the new foreign agent to notify its old foreign agent. Once the notification is received by the old foreign agent, subsequent packets tunneled to the old foreign agent are forwarded to the new foreign agent. This forwarding pointer is also useful to handle packets sent to the old foreign agent by nodes with outdated binding cache entries for the mobile node.

Chapter 7

TCP for Mobile Networks

Chapter 8

Mobile Applications Middleware

Chapter 9

Network Aware Mobile Computing

Chapter 10

Quality of Service Issues in Mobile Computing

Bibliography

[1] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz. Extensible Authentication Protocol (EAP). RFC 3748, Internet Engineering Task Force, June 2004.

[2] Y. Akaiwa and H. Andoh. Channel Segregation – A Self-Organized Dynamic Channel Allocation Method: Application to TDMA/FDMA Microcellular System. *IEEE Journal on Selected Areas in Communications*, 11(6):949–954, August 1993.

[3] L. G. Anderson. A Simulation Study of Some Dynamic Channel Assignment Algorithms in a High Capacity Mobile Telecommunications System. *IEEE Transactions on Vehicular Technology*, 22(4):210–217, November 1973.

[4] W.A. Arbaugh, N. Shankar, and Y.C.J. Wan. Your Wireless Network has No Clothes. In *Proceedings of the First IEEE International Conference on Wireless LANs and Home Networks*, December 2001.

[5] B. Awerbuch and D. Peleg. Online Tracking of Mobile Users. *Journal of the Association for Computing Machinery*, 42(5):1021–1058, September 1995.

[6] B. R. Badrinath, A. Acharya, and T. Imielinski. Structuring Distributed Algorithms for Mobile Hosts. In *Proceedings of the* 14^{th} *International Conference on Distributed Computing Systems*, pages 21–28, June 1994.

[7] D. J. Baker and A. Ephremides. The Architectural Organization of a Mobile Radio Network via a Distributed Algorithm. *IEEE Transactions on Communications*, COM-29(11):1694–1701, November 1981.

[8] A. Bar-Noy, I. Kessler, and M. Sidi. Mobile Users: To Update or not to Update? In *Proceedings of IEEE INFOCOM*, pages 570–576, 1994.

[9] N. Borisov, I. Goldberg, and D. Wagner. Intercepting Mobile Communications: The Insecurity of 802.11. In *Proceedings of the Seventh International Annual Conference on Mobile Computing and Networking*, July 2001.

[10] M. Callendar. International Standards For Personal Communications. In *Proceedings of the* 39^{th} *IEEE Vehicular Technology Conference*, pages 722–728, 1989.

[11] K. M. Chandy and J. Misra. The Drinking Philosophers Problem. *ACM Transactions on Programming Languages and Systems*, 6(4):632–646, October 1984.

[12] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol. Technical Report RFC 3626, Internet Engineering Task Force, Network Working Group, October 2003.

[13] T. Clausen, P. Jacquet, A. Laouiti, P. Muhlethaler, A. Qayyum, and L. Viennot. Optimized Link State Routing Protocol. In *Proceedings of IEEE INMIC*, Pakistan, 2001.

[14] S. Corson and J. Macker. Mobile Ad Hoc Networking(MANET) Routing Protocol Performance Issues and Evaluation Considerations. Technical report, Request for Comments : 2501, January 1999.

[15] B. Das and V. Bharghavan. Routing in Ad-Hoc Networks Using Minimum Connected Dominating Sets. In *Proceedings of ICC*, 1997.

[16] T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246, Internet Engineering Task Force, January 1999.

[17] G. Ding, Z. Sahinoglu, P. Orlik, J. Zhang, and B. Bhargava. Tree-Based Data Broadcast in IEEE 802.15.4 and ZigBee Networks. *IEEE Transactions on Mobile Computing*, 5(11):1561–1574, November 2006.

[18] EIA/TIA IS-41 Rev. C. Cellular Radio Telecommunications Intersystem Operations, PN-2991, November 1995.

[19] S. M. Elnoubi, R. Singh, and S. C. Gupta. A New Frequency Channel Assignment Algorithm in High Capacity Mobile Communication Systems. *IEEE Transactions on Vehicular Technology*, VT-31(3), 1982.

[20] A. Ephremides, J. E. Wieselthier, and D. J. Baker. A Design Concept for Reliable Mobile Radio Networks with Frequency Hopping Signaling. *Proceedings of IEEE*, 75(1):56–73, 1987.

[21] J. Fidge. Timestamps in Message-Passing Systems that Preserve the Partial Ordering. In *Proceedings of the* 11^{th} *Australian Computer Science Conference*, pages 56–66, February 1988.

[22] G. H. Forman and J. Zahorjan. The Challenges of Mobile Computing. *IEEE Computer*, 27(4):38–47, April 1994.

[23] P.-O. Gaasvik, M. Cornefjord, and V. Svensson. Different Methods of Giving Priority to Handoff Traffic in a Mobile Telephone System with Directed Retry. In *Proceedings of the* 41^{st} *Vehicular Technology Conference*, pages 549–553. IEEE, 1991.

[24] E. Gafni and D. Bertsekas. Distributed Algorithms for Generating Loop-free Routes in Networks with Frequently Changing Topology. *IEEE Transactions on Communications*, pages 11–18, January 1981.

[25] M. Gerla and J. T.-C. Tsai. Multicluster, mobile, multimedia radio network. *ACM Baltzer Journal of Wireless Networks*, 1(3):255–265, 1995.

[26] J. S. M. Ho and I. F. Akyildiz. Local Anchor Scheme for Reducing Location Tracking Costs in PCNs. In *Proceedings of Mobicom*, pages 181–193. ACM, 1995.

[27] IEEE. *IEEE Std 802, Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)*, 2006.

[28] R. Jain, Y.-B. Lin, and S. Mohan. A Caching Strategy to Reduce Network Impacts of PCS. *IEEE Journal on Selected Areas of Communication*, 12(8):1434–1444, October 1994.

[29] R. Jain, Y.-B. Lin, and S. Mohan. A Forwarding Strategy to Reduce Network Impacts of PCS. In *Proceedings of IEEE INFOCOM*, pages 481–489, 1995.

[30] D. B. Johnson and D. A. Maltz. Dynamic Source Routing in Ad-Hoc Wireless Networks. In T. Imielinski and H. Korth, editors, *Mobile Computing*. Kluwer Academic Publishers, 1996.

[31] D.B. Johnson, D.A. Maltz, and Y.-C. Hu. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR). Technical Report draft-ieft-manet-dsr-09.txt, IETF MANET Working Group, April 2003.

[32] B. Karp and H.T. Kung. GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. In *Proceedings of MobiCom*. ACM, 2000.

[33] Y.-B. Ko and N.H. Vaidya. Location-Aided Routing (LAR) in Mobile Ad Hoc Networks. In *Proceedings of ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'98)*, pages 66–75, Dallas, 1998.

[34] L. Lamport. Time, Clocks and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, July 1978.

[35] J. Li, J. Jannotti, D.S.J. De Couto, and D.R. Karger R. Morris. A scalable location service for geographical ad hoc routing. In *Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 2000)*, August 2000.

[36] M. Maekawa. A $\sqrt{N}$ Algorithm for Mutual Exclusion in Decentralized Systems. *ACM Transactions on Computer Systems*, pages 145–159, May 1985.

[37] F. Mattern. Virtual Time and Global States of Distributed Systems. In M.Cosnard et. al., editor, *Proceedings of the Workshop on Parallel and Distributed Algorithm*, pages 215–226. Elsevier Science Publishers B.V.(North-Holland), 1989.

[38] A. Mishra and W.A. Arbaugh. An Initial Security Analysis of the IEEE 802.1X Standard. Technical Report CS-TR-4328/UMIACS-TR-2002-10, Department of Computer Science, University of Maryland, February 2002.

[39] M. Mouly and M. B. Pautet. *The GSM System for Mobile Communications*. Book published by authors, 49, rue Louise Bruneau, F-91120 Palaiseau, France, 1992. ISBN 2-9507190-0-7.

[40] A. Palekar, D. Simon, J. Salowey, H. Zhou, G. Zorn, and S. Josefsson. Protected EAP Protocol (PEAP) Version 2. Internet Draft, EAP Working Group, draft-josefsson-pppext-eap-tls-eap-08.txt, July 2004.

[41] V. D. Park and M. S. Corson. A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks. In *Proceedings of IEEE INFOCOM*, April 1997.

[42] C. Perkins, E. Belding-Royer, and S. Das. *Ad hoc On-Demand Distance Vector (AODV) Routing*. IETF, RFC 3561 edition, July 2003.

[43] C.E. Perkins and P. Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. In *Proceedings of ACM SIGCOMM Conference on Communication Architectures, Protocols and Applications*, pages 234–244, August 1994.

[44] R. Prakash, Z. J. Haas, and M. Singhal. Load Balanced Location Management for Mobile Systems using Dynamic Hashing and Quorums. Technical Report UTDCS-05-97, The University of Texas at Dallas, October 1997.

[45] R. Prakash, N. Shivaratri, and M. Singhal. Distributed Dynamic Channel Allocation for Mobile Computing. In *Proceedings of the* 14^{th} *ACM Symposium on Principles of Distributed Computing*, pages 47–56, Ottawa, Canada, August 1995.

[46] R. Prakash and M. Singhal. Dynamic Hashing + Quorum = Efficient Location Management for Mobile Computing Systems. In *Proceedings of ACM Symposium on Principles of Distributed Computing (short presentation)*, Santa Barbara, August 1997.

[47] G. Ricart and A. K. Agrawala. An Optimal Algorithm for Mutual Exclusion in Computer Networks. *Communications of the ACM*, 24(1):9–17, January 1981.

[48] R.L. Rivest. *The RC4 Encryption Algorithm (Proprietary)*. RSA Data Security, Inc., March 1992.

[49] C. Shields, Jr., V. Jain, S. Ntafos, R. Prakash, and S. Venkatesan. Fault-Tolerant Mobility Planning for Rapidly Deployable Wireless Networks. In *Proceedings of the IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems*, LNCS. Springer-Verlag, April 1998.

[50] M. Singhal. A Dynamic Information-Structure Mutual Exclusion Algorithm for Distributed Systems. *IEEE Transactions on Parallel and Distributed Systems*, 3(1):121–125, January 1992.

[51] A. Tanenbaum. *Computer Networks(*3^{rd} *Edition)*. Prentice Hall, Upper Saddle River, N.J., 1996.

[52] S. Tekinay and B. Jabbari. Handover and Channel Assignment in Mobile Cellular Networks. *IEEE Communications Magazine*, 29, 1991.

[53] J.R. Walker. *Unsafe at any key size; an analysis of the WEP encapsulation*. IEEE, document 802.11-00/362 edition, October 2000.

[54] M. Zhang and T.-S. P. Yum. Comparisons of Channel-Assignment Strategies in Cellular Mobile Telephone Systems. *IEEE Transactions on Vehicular Technology*, 38(4):211–215, November 1989.

[55] M. Zhang and T.-S. P. Yum. The Nonuniform Compact Pattern Allocation Algorithm for Cellular Mobile Systems. *IEEE Transactions on Vehicular Technology*, 40(2):387–391, May 1991.