Mobility Support in Mesh Networks

By

Steven X. Xu

A thesis submitted to

the Faculty of Graduate Studies and Research

in partial fulfillment of

the requirements for the degree of

Master of Computer Science

Ottawa-Carleton Institute for Computer Science

School of Computer Science

Carleton University

Ottawa, Ontario

November 29, 2004

The undersigned hereby recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis,

Mobility Support in Mesh Networks

Submitted by

Steven X. Xu

In partial fulfillment of the requirements for
the degree of Master of Computer Science

Dr. Thomas Kunz
(Thesis Supervisor)

Dr. Doug Howe
(Director, School of Computer Science)

Carleton University

November 29, 2004

Abstract

Wireless mesh networking is a very promising technology to provide wireless broadband access. This thesis presents a novel scheme to build a wireless mesh network with mobility support in IPv6. One great feature of the scheme is that nodes in the mesh network are built from off-the-shelf hardware, and support standard configured wireless clients without modifying any software and hardware. The proposed mesh network consists of two parts, the backbone and local footprints. Each mesh node functions as a wireless router, and the Optimized Link State Routing (OLSR) is employed as the routing protocol. And each mesh node also works as an intelligent Access Point (AP), which provides access to wireless clients in the local footprint.

An implementation of the novel scheme fully integrates the mesh backbone and local footprint. Dynamic Host Configuration Protocol for IPv6 (DHCPv6) is deployed in the proposed IPv6 network, which provides zero-conf to visiting wireless clients.

To the best of our knowledge, this is the first implemented wireless mesh network that offers these salient features. A testbed is built based on this implementation. Furthermore, we are considering the deployment of such a mesh network in the real world.

Acknowledgements

I would like to thank Dr. Thomas Kunz, my supervisor, for all his help on this thesis. His guidance, advice, and comments are greatly appreciated.

I am grateful to my family for their support and understanding.

Table of Contents

List of Figures

List of Tables

List of Acronyms

AES	Advanced Encryption Standard
AID	Association ID
AODV	Ad hoc On-Demand Distance Vector Routing Protocol
AP	Access Point
CRC	Communication Research Center of Canada
CTS	Clear To Send
DAD	Duplicate Address Detection
DCF	Distributed Coordination Function
DHCPv6	Dynamic Host Configuration Protocol for IPv6
DNS	Domain Name Service
DoD	Department of Defense
DS	Distribution System
DSDV	Destination Sequenced Distance Vector
DSR	Dynamic Source Routing
DSSS	Direct Sequence Spread Spectrum
DSTM	Dual Stack Transition Method
DTI	Dynamic Tunneling Interface

DUID	DHCP Unique Identification
ESS	Extended Service Set
ESSID	Extended Service Set Identification
ETX	Expected Transmission count
FIFO	First In First Out
FSSS	Frequency Hopping Spread Spectrum
FTP	File Transfer Protocol
HNA	Host and Network Association
IAID	Identity Association Identification
IAPP	Inter-Access Point Protocol
IBSS	Independent Basic Service Set
ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical & Electronics Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPC	Inter-Process Communication
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
IV	Initialization Vector
LQSR	Link Quality Source Routing
MAC	Media Access Control
MADWiFi	Multiband Atheros Driver for WiFi
MANET	Mobile Ad Hoc Network

MCL	Mesh Connectivity Layer
MNE	Mobile Network Emulator
MPR	Multi-Point Relay
NA	Neighbor Advertisement
NLA	Next Level Aggregator
NS	Neighbor Solicitation
OLSR	Optimized Link State Routing
PCF	Point Coordination Function
PDA	Personal Digital Assistant
RA	Router Advertisement
RFC	Request for Comments
RTS	Request To Send
SLA	Subnet Local Aggregator
SSID	Service Set Identification
STA	Station
TLA	Top Level Aggregator
WEP	Wired Equivalent Privacy
Wi-Fi	Wireless Fidelity
WISP	Wireless Internet Service Provider
WLAN	Wireless Local Area Network
WMN	Wireless Mesh Network

Chapter 1

Introduction

IEEE 802.11 technology [14] based Wi-Fi networks have gained tremendous success recently. According to a report from In-Stat/MDR [15], the home Wi-Fi market rolled out 22.7 million network interface and Access Point (AP) units in 2003, a 214% increase from 2002's 7.2 million unit shipments. Revenues are expected to reach $1.7 billion, an increase of 140% from 2002 total revenues of $700 million. In today's mobile device market, Wi-Fi support is a standard feature on laptops and PDAs. With more and more public wireless local area networks (WLAN) hot spots installed in hotels, coffee shops and parks, cost-effective, ease of use wireless broadband access has become reality. However, all these public hot spots rely on wired networks as backhauls. Counting on the huge cost of wired networks, ubiquitous deployment of Wi-Fi is only possible with a wireless backhaul. Wireless mesh network (WMN) technology has emerged to become an increasingly vital part of the wireless evolution.

1.1 Wireless Mesh Networking

The topology of WMN is a mesh, which is an irregular structure as shown in Figure 2.1. Nodes are connected with a subset of other nodes, and generally there are a number of

alternative paths between each pair of nodes. Each node operates not only as a host but also as a router, forwarding packets on behalf of other nodes that may not be within direct wireless transmission range of an Internet gateway. The network is dynamically self-organizing, self-configuring and self-healing, with the nodes in the network automatically establishing and maintaining routes among themselves. You can add, remove, or relocate mesh nodes, whenever you wish, without disrupting the rest of the network or rebooting the system.

WMN has several advantages over traditional WLAN:

- Rapid deployment. Installed in hours instead of days or weeks
- Network coverage can be increased dramatically
- Robust and resilient. It automatically reroutes through an alternate path if one link fails, no longer dependent on a system with a single point of failure
- Less fixed infrastructure than other topologies, bringing great flexibility in increasing density or changing coverage
- Lower cost

Therefore, WMN is a promising solution to build wireless home networks, helping to free home users from the jumble of cables while offering the capability to support bandwidth-intensive applications. WMN is also a good candidate for enterprises. It can quickly extend an existing WLAN without having to wire up new base stations. Industrial control systems can deploy wireless mesh networks of sensors and controllers in factories and laboratories. WMN can help Wireless Internet Service Providers (WISPs) provide service in small towns or rural areas where wired infrastructure is unavailable, inadequate or

unreliable. Many developing countries will also benefit from wireless mesh networking technology.

1.2 Motivation

While this promising technology is attracting the attention of network managers, and a lot of research has been done, the real world deployment of WMN is rare. Most of the research publications are evaluated based on simulations. MeshNetworks [19] is one of the few companies that deploy a wireless mesh network in the market currently. But it requires special client and server hardware that works like Wi-Fi with extensions.
The Roofnet project [3] at MIT implemented an experimental multi-hop IEEE 802.11b/g mesh network consisting of about 50 nodes. The original routing protocol used in Roofnet is the modified Destination Sequenced Distance Vector (DSDV) protocol [23]. And now it is replaced by SrcRR, a variant of Dynamic Source Routing (DSR) [16].
There are also many community wireless networks around the world. The Southampton Open Wireless Network (SOWN) [27] aims to build a wireless mesh network in Southampton, UK. But as of the writing of this thesis, their new design is not fully mesh based. Its backbone consists of central nodes, which are located in high areas that have good line of sight, such as top of tower blocks or church spires. Clients communicate with these central nodes using directional antennas. Central nodes are carefully deployed so that no two neighboring nodes work on the same channel. In IEEE 802.11, there are 13 available channels. Channels 1, 6, and 11 do not overlap. Central nodes work in these channels to reduce interference with each other. Central nodes use dedicated radio links

to communicate with each other. OSPF is employed as the routing protocol, which is not appropriate in the wireless mobile environment.

In these WMNs, without exception, all the nodes are built from either special hardware or software. While general wireless clients are mostly off-the-shelf laptops or PDAs, it is unrealistic to require them to become mesh nodes. Most importantly, allowing everyone to be a mesh node causes big security problems.

With these concerns in mind, our approach is to build a wireless mesh network as the backbone. At the same time each mesh node serves as an Access Point (AP) to the wireless clients. This way we can have more control of the network without sacrificing the flexibility of WMN. This approach also simplifies client management and enables true nomadic operation without reliance on external client hardware or software.

1.3 Contribution

It presents some unique challenges to implement such a mesh network with mobility support. We can not rely on the commodity APs, because they lack the ability to function as routers running complex wireless ad hoc routing algorithms. This thesis presents an innovative solution to build mesh nodes from plain Linux boxes with tailored software.

The main contributions of this thesis are:

- Design an innovative scheme to provide mobility support in mesh network.
- Make modifications to OLSR to support the novel scheme.
- Implementation of the scheme.
- Build a testbed for performance evaluation of the scheme.

1.4 Thesis Structure

This thesis begins with a literature review of wireless networking and related work in Chapter 2, and describes the design and implementation in Chapter 3. Chapter 4 introduces a testbed based on the implementation and evaluates the performance of the proposed network. Finally, Chapter 5 draws the conclusion and discusses future work.

Chapter 2

Background and Related Work

Our proposed mesh network is based on the IEEE 802.11 standard. The following diagram illustrates the architecture of our wireless mesh network.

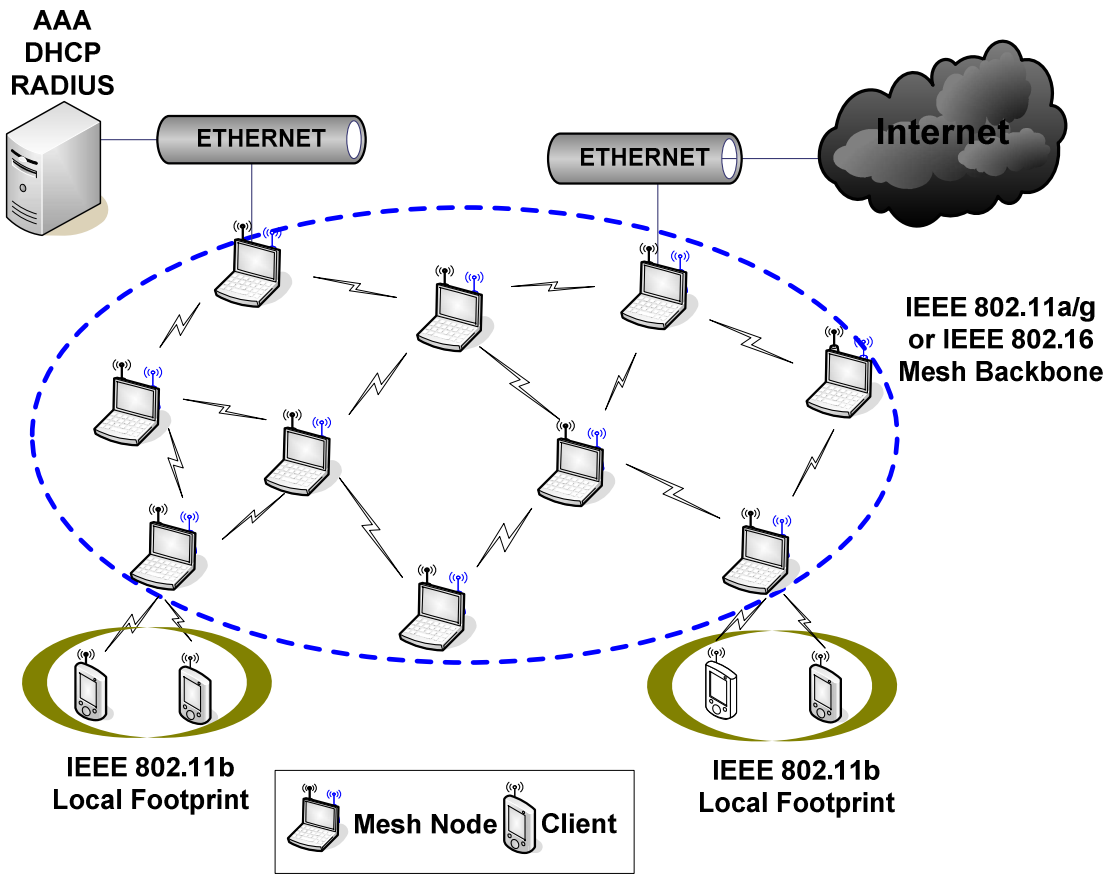

Figure 2.1 Mesh Network Architecture

In order to build such a mesh network, we face challenges involves both network layer and link layer. In this chapter, we first briefly introduce the functionalities of physical layer and link layer of the IEEE 802.11 specification. Next we discuss issues related to the network layer, mainly about routing and IPv6 [9] address allocation. In Section 2.2, we summarize some of the most commonly used ad hoc routing protocols. In Section 2.3, the IP address autoconfiguration mechanism is introduced. In Section 2.4, a number of related wireless networking projects are explored.

2.1 IEEE 802.11 Overview

The IEEE 802.11 specification defines the functionalities and procedures that the Medium Access Control (MAC) layer and Physical (PHY) layer must provide.

2.1.1 Physical Layer

In IEEE 802.11 there are three different PHY definitions, frequency hopping spread spectrum (FHSS), direct sequence spread spectrum (DSSS), and InfraRed (IR). FHSS and DSSS operate at the unlicensed 2.4 GHz band. Both FHSS and DSSS currently support 1 and 2 Mbps data rates. But DSSS can support date rates up to 11 Mbps. IEEE 802.11b, a, and g are different extensions made to the general IEEE 802.11 standard at the physical layer to support higher data rates. While IEEE 802.11b and IEEE 802.11g still operate at 2.4 GHz, IEEE 802.11a works in a new frequency spectrum, the 5GHz radio band. IEEE 802.11g and IEEE 802.11a support transfer speed up to 54 Mbps.

2.1.2 MAC Layer

The 802.11 Medium Access Control (MAC) layer defines two forms of media access, distributed coordination function (DCF) and point coordination function (PCF). DCF is the basic way to support media access control, and therefore is required to be implemented by all stations (STAs). It is based on the carrier sense multiple access with collision avoidance (CSMA/CA) protocol. When a STA needs to transmit packets, it defers transmission until it can not hear other STAs. This carrier sense mechanism is defined in the PHY layer. In addition to physical carrier sensing, IEEE 802.11 introduces a virtual carrier sense mechanism, which enables a station to hold the medium for a specified period of time through the exchange of Request To Send (RTS) frame and Clear To Send (CTS) frame. There are three types of IEEE 802.11 MAC layer frames: control, management and data frames. RTS frame and CTS frame are control frames. Figure 2.2 shows the general frame format.

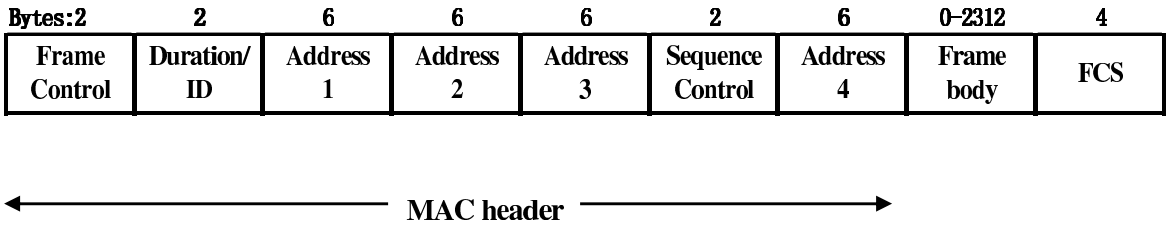

Figure 2.2 General MAC Frame Format

The source STA starts the transmission process by sending a RTS frame first. In the duration/ID field of the MAC header, the value is set to the time that is required to transmit the pending data or management frame, plus one CTS frame, plus one ACK frame, plus three Short Interframe Space (SIFS) intervals. This time value is called Network Allocation Vector (NAV), which is the time that the current transmission needs to reserve the medium. If the destination STA receives the RTS frame and the channel is idle, it sends back a CTS frame. After the source STA gets the CTS frame, it transmits the data payload. The destination STA is required to send an ACK frame back after successfully receiving data frames. All other STAs hear the CTS frame and know it is not destined for them, so they back off for NAV microseconds before starting the contention for the next transmission.

The point coordination function (PCF) is an optional access method, aiming to support the transmission of time-sensitive data streams such as voice and video data. It is usable when wireless networks are configured to work in the infrastructure mode. The point coordinator within the access point grants channel access to STAs by polling the STA during the contention free period (CFP). If a Basic Service Set (BSS) is set up with PCF enabled, time is spliced between the contention free period with PCF mode and the contention period in the DCF mode. Currently very few vendors support the PCF protocol.

2.1.3 MAC Layer Management Services

There are two basic operation modes in IEEE 802.11, the infrastructure mode, and the ad

hoc mode. An ad hoc network, or Independent Basic Service Set (IBSS) in IEEE 802.11 terminology, is a network where STAs communicate with each other directly without central control. In an infrastructure based WLAN, each Basic Service Set (BSS) consists of exactly one station that provides the functionality to relay packets between other stations in the same BSS and the Distribution System (DS). This station is called the Access Point (AP). In order to increase the coverage, multiple overlapping BSSs are connected together through a DS to form the Extended Service Set (ESS). Although the DS could be any type of network, it is typically an Ethernet LAN. All the APs in the ESS have the same ESS Identification (ESSID). STAs can seamlessly roam from one BSS to another BSS within the same ESS, provided that all the APs are in the same IP subnet. Also of note is that APs should be compliant to IEEE 802.11f specification, the Inter-Access Point Protocol (IAPP), to ensure multi-vendor AP interoperability.

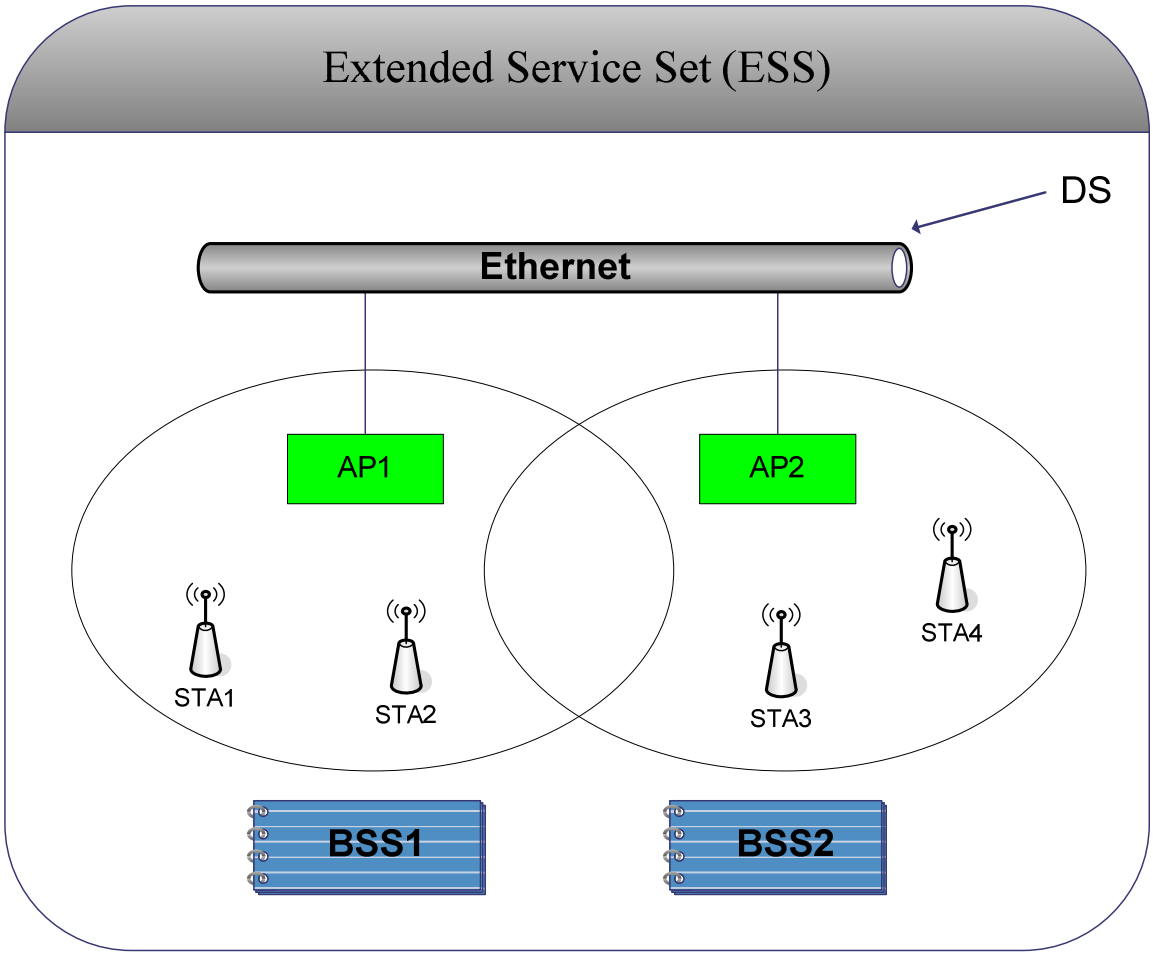

Figure 2.3 Extended Service Set

MAC layer management frames are defined to enable IEEE 802.11 STAs to establish and maintain communications. The following are common IEEE 802.11 management frame subtypes:

- Beacon frame: AP announces the existence of a network by periodically sending a beacon, which includes timestamp, SSID, and other information.
- Probe request frame: A STA sends a probe request frame to scan for the existence of an IEEE 802.11 network. It contains two fields: the SSID and the rates supported by the STA.

- Probe response frame: In infrastructure networks, the AP will respond with a probe response frame if parameters match.
- Authentication/ Deauthentication frame: STAs exchange unicast authentication frames to prevent unauthorized access.
- Association request frame: After a STA gets authenticated with the AP, it attempts to join the network by transmitting this frame to the AP. The frame carries Capability Information, SSID, Supported Rates, and other information. When the AP confirms that this information matches, it accepts the STA.
- Association response frame: An AP sends an association response frame to the STA. It contains the Status Code field, indicating an acceptance or rejection of the STA. If the AP accepts the STA, the frame would include an association ID.
- Reassociation request frame: If a STA roams between APs within the same ESSID, it needs to send a reassociation frame to the new AP. Compared with an association request frame, the reassociation request frame contains one field, the address of the current AP.
- Reassociation response frame: Similar to the association response frame.
- Disassociation frame: Used to terminate the association.

All the APs have the functionality of handling these IEEE 802.11 management frames. But with off-the-shelf APs, we have no access to these frames. Thanks to the Open Source society, they provide some device drivers that meet our needs. Host AP [12], developed by Jouni Malinen, is a Linux Wireless LAN device driver, which mainly supports IEEE 802.11b cards that are based on Intersil's Prism2/2.5/3 chipset. When configured in HostAP (Master) mode, the Linux box can function as an Access Point.

The driver handles IEEE 802.11 management frames, such as authentication, association in the kernel space. Multiband Atheros Driver for WiFi (MADWIFI) [17], another Linux driver, is designed for Atheros' multiband chip sets that support IEEE 802.11b, a, and g. It also provides AP functionality to the wireless radio.

2.2 Wireless Ad Hoc Routing Protocols

Different from an infrastructure based WLAN, an ad hoc WLAN lacks a central control station to relay traffic. Stations communicate directly in a peer-to-peer manner when they are within the transmission range. For an IEEE 802.11 device, the range is about 300 feet. To create large-scale wireless networks, routing is needed. During the past several years, much research has been done on routing protocols for mobile ad hoc networks (MANETs). There are mainly three types, reactive, proactive, and hybrid. Reactive routing protocols discover new routes only when needed by the source node. Reactive protocols have less overhead but higher latency of route discovery. An early reactive routing protocol is the Dynamic Source Routing Protocol (DSR) [16]. Another competing reactive routing protocol is Ad hoc On-Demand Distance Vector Routing Protocol (AODV) [22]. Both protocols have the functionalities of route discovery and route maintenance, but differ in neighbor detection and route storage. DSR stores source routes in packet headers. Network performance degrades as packet headers get larger. AODV maintains routing tables at the nodes instead of in each packet.

Proactive routing protocols determine routes based on periodically updated network topology information. Traditional link-state and distance-vector routing protocols all

belong to this category. Since routes are always maintained in proactive protocols, more overhead is created, but route convergence time is low.

Topology Broadcast Based on Reverse-Path Forwarding Routing Protocol (TBRPF) [21] is a proactive protocol. Every node keeps partial network topology information. When a node needs the shortest path to other nodes, a source tree is computed using a modified Dijkstra's algorithm. To reduce overhead, a node only broadcasts a partial source tree, which is called the reportable tree. And in the neighbor discovery procedure, a node uses "differential" HELLO messages, which only report the changes of neighbor status.

Optimized Link State Routing Protocol (OLSR) [4] is an optimization of the classical link state algorithm. It is a table-driven and proactive routing scheme. In the classical flooding mechanism, every node retransmits each message when it receives the first copy of the message. While in OLSR, link state information is broadcasted only by nodes elected as multi-point relays (MPRs). Thus it greatly reduces the overhead. MPRs of a node *y* are chosen according to the following heuristic algorithm:

1. For each node *x* that is a 1-hop neighbor of *y*, calculate D(*x*), the degree (the number of neighbors) of *x*.
2. Select as MPRs those nodes that are 1-hop neighbors of *y*, and they provide the "only path" to some nodes in the 2-hop neighbor set N.
3. While there exist nodes in N which are not covered:

   Select as an MPR a 1-hop neighbor of *y*, which reaches the highest number of uncovered nodes in N. If multiple nodes provide the same amount of reachability, the one with higher degree is chosen.

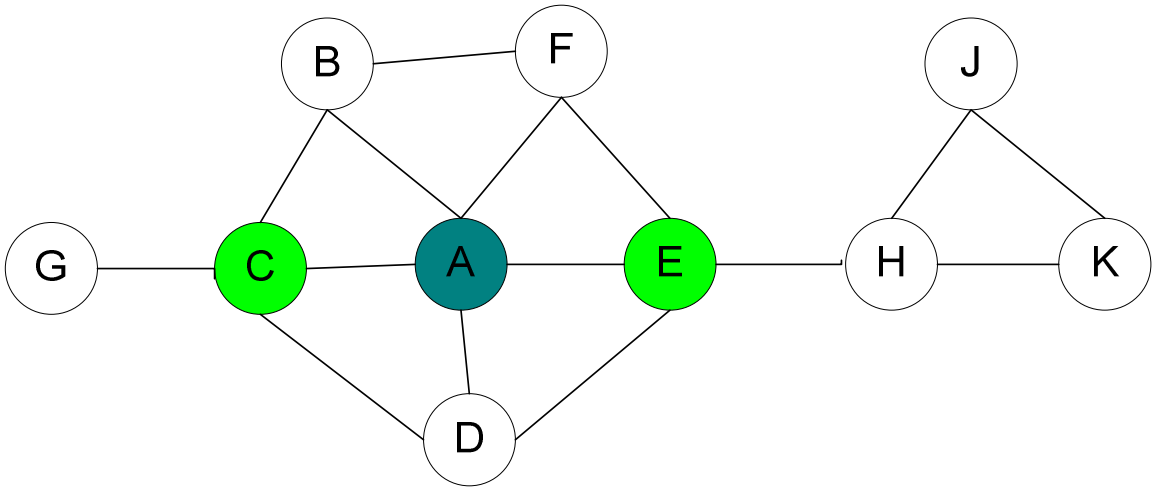

Figure 2.4 MPR Selection Algorithm

As shown in the above diagram, among A's 1-hop neighbors, C and E are chosen as MPRs, and only C and E will relay A's control messages. MPRs also help to reduce the packet size of the Topology Control (TC) messages. Nodes do not need to declare all links to their neighbors, instead, they only declare links to their MPR selector set, which are nodes that have selected this node as MPR.

For neighbor sensing, each node periodically broadcasts Hello messages. Hello messages are not forwarded. Upon receiving Hello messages, a node calculates its MPRs.

Hello messages enable each node to discover its neighbors, while TC messages enable each node to get the topological information about the network. Each node forms a picture of the network topology, and then calculates the best route to any destination. OLSR is particularly suitable for large and dense networks as the technique of MPRs works well in this context.

OLSR also has a mechanism to provide connectivity from the OLSR interface(s) to

non-OLSR interface(s). A node equipped with both of these interfaces can act as a "gateway" by injecting external route information to the OLSR MANET. It periodically issues a Host and Network Association (HNA) message, with the network address of the associated network, and the net mask.

In our proposed mesh networks, as shown in Figure 2.1, local footprints are isolated networks. To bridge the gap between the mesh backhaul and local footprints, OLSR HNA mechanism is employed. A mesh node that has associated clients emits HNA messages to advertise network information of the local footprint. Upon reception of HNA messages, other mesh nodes create corresponding route entries in their routing tables. Therefore, packets destined to clients in that local footprint can be routed by the mesh backhaul. The two independent networks are integrated.

2.3 Addressing Scheme

Due to concerns over the impending depletion of IPv4 address space, Japan and China already started to deploy IPv6. The US Department of Defense (DoD) plans to migrate to IPv6 by 2008 [31]. DoD also requires that devices acquired for its massive Global Information Grid (GIG) network must be compatible with IPv6. In the wireless networking world, as consumers and businesses use an increasing number of mobile devices, the pressure for available IP addresses will continue to increase. So our proposed mesh network is based on IPv6.

Although a vast amount of research efforts has been geared toward developing better routing protocols, much less attention has been given to autoconfiguration mechanisms,

especially the address autoconfiguration. In this section, we first introduce the IPv6 Addressing Architecture. Then we give a more detailed description of address autoconfiguration mechanisms in IPv6.

2.3.1 IPv6 Addressing Architecture (RFC 3513) [10]

There are three address types in IPv6: unicast, anycast, and multicast. The IPv4 terminology broadcast is discarded in IPv6.The broadcast function is replaced by multicast addresses. Unicast is the most commonly used address type. It also has three different types: Link-local, site-local, and global scope. The global unicast address format is shown in the following figure.

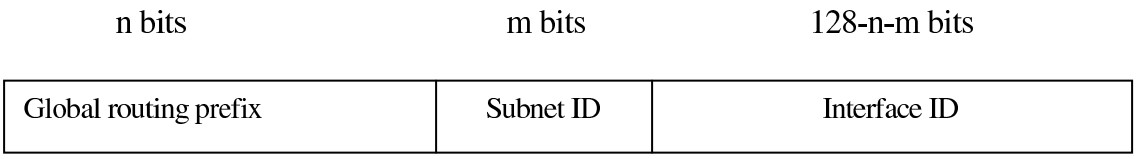

Figure 2.5 General Format for IPv6 Global Unicast Address

The address allocation structure defined in RFC 2374, An IPv6 Aggregatable Global Unicast Address Format, is obsolete. Top Level Aggregator (TLA) and Next Level Aggregator (NLA) in RFC 2374 [11] are replaced with Global routing prefix. The Subnet Local Aggregator (SLA) field in RFC 2374 is given a new name, Subnet ID.

2.3.2 IPv6 Address Autoconfiguration

IPv6 defines two ways of address autoconfiguration: Stateless Autoconfiguration (RFC 2462) [30], and Stateful Autoconfiguration, the Dynamic Host Configuration Protocol for IPv6 (DHCPv6: RFC 3315) [7]. With Stateless Autoconfiguration, an IPv6 host can configure its IPv6 address by combining locally available information such as interface identifier, and information advertised by routers such as prefixes that identify the subnet(s) associated with a link. Stateless Autoconfiguration is a very nice feature of IPv6, a zero-conf mechanism that enables plug-and-play. However, it does not provide a mechanism for DNS information to be automatically configured. Some applications function well without DNS servers, such as file and printer sharing services using the SAMBA protocol. But surfing the Internet with IPv6 addresses rather than DNS names, such as "*www.carleton.ca*", is simply unacceptable for most people.

DHCPv6 servers can provide not only IPv6 addresses, but also other network configuration parameters like DNS information, domain search path, NTP server addresses etc. by means of a rich set of options. With the relay agent mechanism, the DHCPv6 server does not necessarily have to attach to the same link as clients.

2.3.3 DHCPv6 Operation Overview

The DHCPv6 protocol is a client server model. Through the DHCPv6 message exchange, a client acquires configuration information from the server. DHCPv6 messages are carried by UDP packets. However, when the client initiates the interaction process, it

needs an IPv6 address to put into the source IP address of the UDP packet. So its link-local address is used, which is configured when the client boots up. The client sends a DHCPv6 Solicit message by multicast to a reserved link-scoped address, All_DHCP_Relay_Agents_and_Servers (FF02::1:2). Servers or relay agents sitting on the same link will receive the Solicit message, because they are configured to join this multicast group. The client keeps retransmitting the solicit message till it receives valid DHCPv6 Advertisement messages from any server, either directly or forwarded by relay agents. Then the client uses a Request message to get addresses and obtain other configuration information. After receiving Reply messages from the server, the client can configure its interfaces with the supplied information.

After a certain time, the client needs to unicast the Renew message with the addresses that it wishes to renew to the server. The client can also choose to release the addresses. A client can use the Rebind message to extend the valid and preferred lifetimes for the addresses if it fails to extent the lease in the renew phase.

Sometimes a client may move to a new link with different network prefixes, therefore, the old addresses are no longer appropriate for the link to which the client is currently attached. One example is that in our proposed mesh network, a client travels from one local footprint to another one. In this case, the client must transmit a Confirm message. The server will respond with a Reply message, indicating whether the address is still appropriate to use.

2.4 Related Work

There are a number of research projects that aim to build wireless mesh networks. In this section we present a short summary of these projects.

2.4.1 MIT Roofnet

In Chapter 1, we briefly introduced the MIT Roofnet project. Each node in the Roofnet only has one wireless interface, which runs the routing protocol, SrcRR, to form the mesh. Roofnet does not support wireless clients. End users are supported through a wired Ethernet interface. The addressing scheme of Roofnet is based on IPv4. Each node has two network interfaces, and both are configured with private IPv4 address. The wireless radio is assigned an address in the range 10.x.x.x. The wired Ethernet interface has an IPv4 address 192.168.0.1, and a DHCPv4 server is running on this interface. Home users connect to Roofnet nodes through Ethernet ports. From DHCPv4 servers running on Roofnet nodes, home computers attain IPv4 addresses in the range 192.168.0.x.
Some Roofnet nodes act as gateways to the Internet, with their Ethernet ports connecting to DSL or cable modems. Only these gateway nodes have global IPv4 addresses allocated from ISPs. To enable internet connectivity to non-gateway nodes and home users, Network Address Translation (NAT) is used. A packet from a home computer to the internet is NATed from the 192.168.0.x address on the wired interface to the 10.x.x.x address on the wireless radio. The mapping from one group to another in NAT is transparent to end users. Then the packet is routed through the wireless mesh to a

gateway node. Again the packet is NATed from the 10.x.x.x address in the wireless radio to the global IPv4 address, and transmitted to the Internet through the Ethernet port.

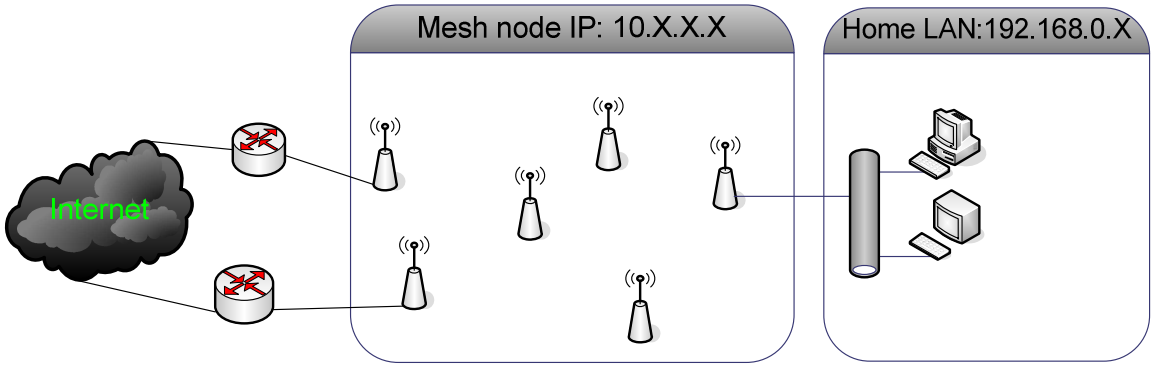

Figure 2.6 MIT Roofnet Architecture and Addressing Scheme

NAT was introduced as a short term solution for the IPv4 address exhaustion problem. But as indicated in Architectural Implications of NAT (RFC 2993), NAT is a violation of the end-to-end communication principle. It complicates applications like Voice-over-IP. NAT also makes implementation of IP level security, e.g. IPSec, impossible.

2.4.2 MONARCH Project

The Rice University MONARCH project [18] develops a wireless ad hoc network testbed. It consists of two stationary nodes and five car-mounted nodes that move around on a predefined testbed site. These nodes communicate through 900 MHz WaveLAN-I radios. DSR is employed as the routing protocol. A car-mounted roving node roams between its

home network, the central office, and the foreign network, the ad hoc network. Mobile IP is configured in all nodes to support roaming. For the roving node to connect to the ad hoc network, it also needs to start DSR to join the routing process.

Similar to MIT Roofnet, the MONARCH project is also a single radio wireless network. But it does not use a wired interface to provide support to end users. As a pure MANET network, standard configured clients are not supported.

2.4.3 Microsoft's Mesh Technology

Wireless Mesh Networking has also attracted Microsoft's attention [1]. It has developed a module called Mesh Connectivity Layer (MCL), which is a loadable Windows driver that implements a virtual network adapter. MCL works between the link layer and the network layer. MCL is mainly a routing module based on DSR. The purpose of the modification is to support link quality measurements. That is why the routing protocol is given a new name, Link Quality Source Routing (LQSR).

Microsoft's mesh network is not designed to support standard configured wireless clients. Clients need to install MCL and to initiate the routing process to join the mesh network. MCL runs on Windows XP only and needs .NET Framework 1.1 support. If clients want to install two radios, according to the installation guide of MCL, the radios must be from different manufacturers. The reason is that current wireless drivers for Windows do not support two or more identical radios in the same computer. But in Linux, the HostAP and MadWiFi drivers do not have these constrains. Compared with Windows, deploying Linux in a large mesh network can also save a lot of money.

2.4.4 Hyacinth Project

The Hyacinth project [25] in the State University of New York at Stony Brook aims to build a Multi-channel Wireless Mesh Network. The following diagram illustrates their proposed network architecture.

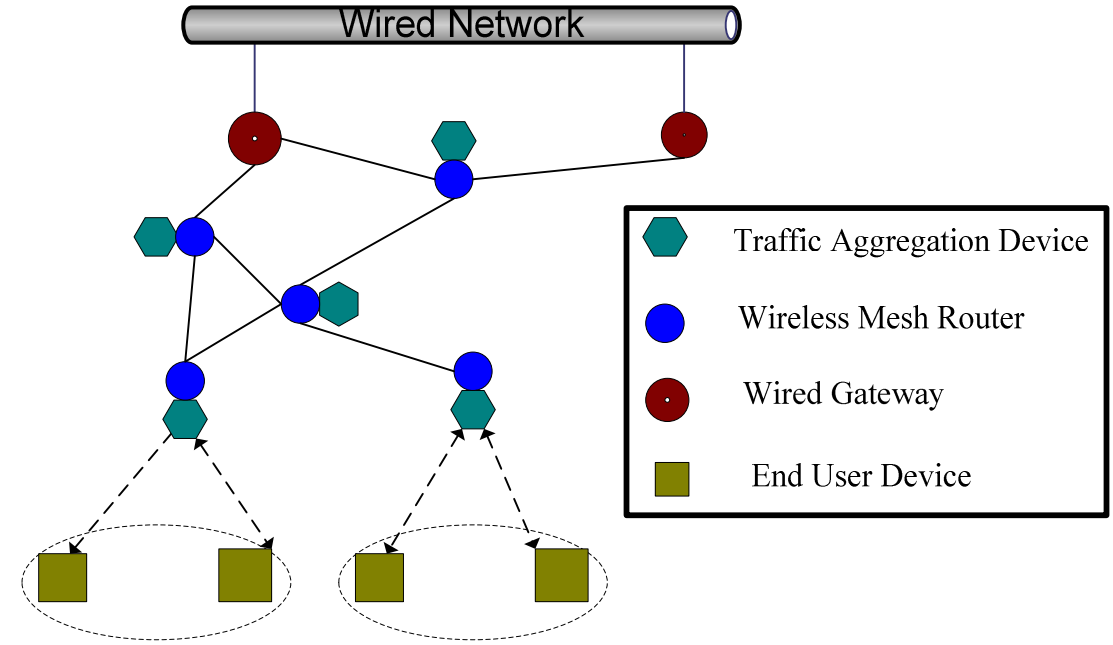

Figure 2.7 Hyacinth Network Architecture (Adapted from the Hyacinth Project)

Each node in a Hyacinth network is equipped with multiple IEEE 802.11 radios. Each radio is set to a particular channel for several minutes or hours. All these nodes together form a multi-channel wireless mesh network, and relay traffic to or from wireless clients. Among the multiple radios in a node, one is set to work in infrastructure mode. This interface is called "traffic aggregation access point", which provides connectivity to

wireless clients within its coverage area. Some nodes serve as gateways between the wireless mesh network and the wired network.

Compared with a traditional single-channel network, a Hyacinth network can increase the network bandwidth. To evaluate the performance, a nine-node Hyacinth prototype testbed has been built. According to published research papers, the Hyacinth project is still in its early stage. At present, focus is on implementing a distributed channel assignment and routing algorithm. Research efforts are also made to support autoconfiguration of IP addresses and routing table. Though end user support is one of the design goals, it is not implemented yet.

The design of a Hyacinth network shares some similarities with our proposed network.

1. Provides access to wireless clients without modifying software/hardware on the mobile devices.
2. Backbone nodes form a wireless mesh network, and relay traffic to/from wireless clients.

There are also different characteristics between the two designs:

1. Nodes in a Hyacinth network are static, while our mesh nodes are mobile. Our implementation enjoys all the benefits of wireless mesh networks discussed in Chapter 1. But a Hyacinth network requires extra administration overhead and cost.
2. Our implementation supports regular wireless clients. IP address autoconfiguration is provided. The current version of Hyacinth lacks these features.
3. Although nodes in our proposed network are configured with two radios, only one provides network bandwidth to the backbone. So it is still a single-channel multi-hop mesh network.

Chapter 3

Design and Implementation

In Chapter 2, we investigated several approaches to build wireless mesh networks. Here, we present our new scheme. In Section 3.1, we introduce the network architecture. Section 3.2 discusses the assignment of IP addresses. In Section 3.3, we investigate the challenges to build such a wireless mesh network, and our approaches to solve these problems. Section 3.4 presents our implementation details.

3.1 Network Architecture

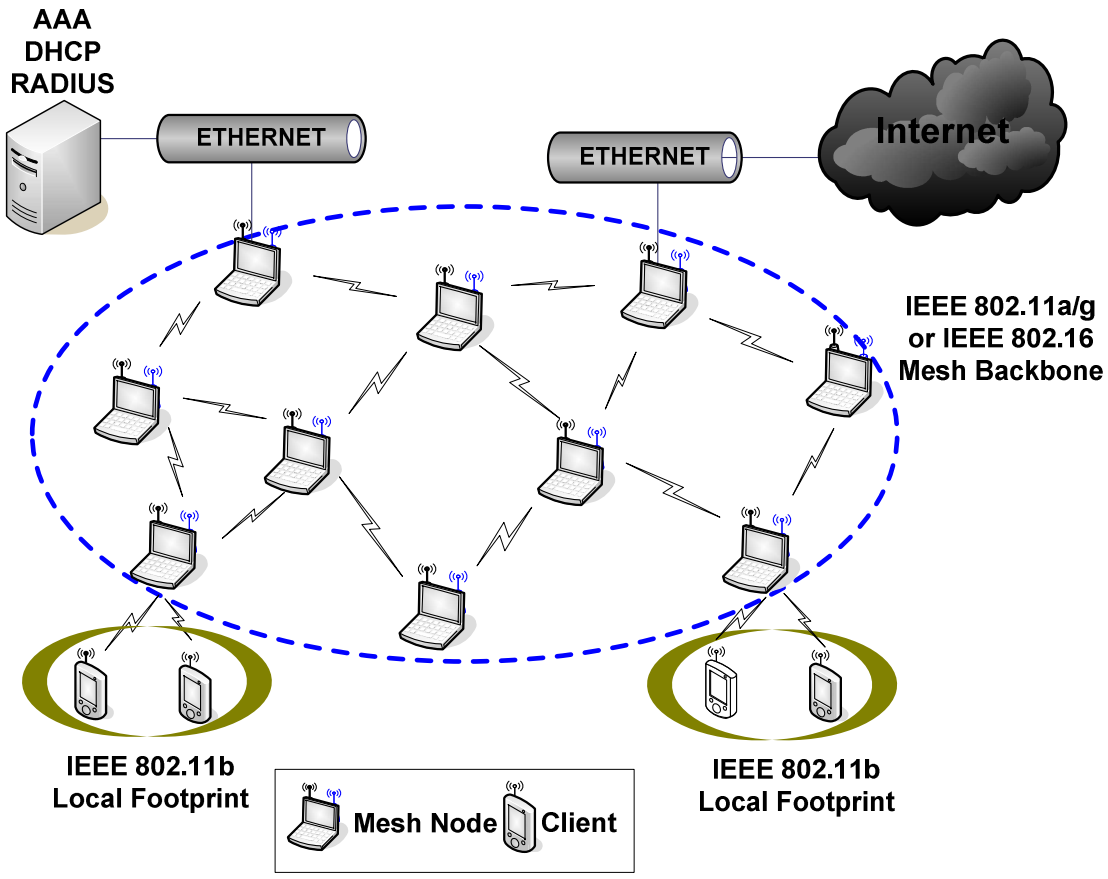

Figure 3.1 Our Mesh Network Architecture

As illustrated in Figure 3.1 (the same as Figure 2.1), the proposed network architecture in this research consists of two parts, the mesh backbone and local footprints. All the mesh modes are equipped with two wireless interfaces. One is an IEEE 802.11a/g compliant

radio, and it is the backbone traffic carrier. Another is an IEEE 802.11b radio, which provides access to wireless clients in the local footprint. One goal of our design is to support standard configured wireless clients, which usually have off-the-shelf IEEE 802.11 hardware. Wireless clients can have access to the Internet or intranet through the wireless mesh backbone. Arriving users can immediately join the network when they come into range and turn on their devices.

3.2 IP Addressing Allocation

The proposed network is IPv6 based. To assign IP addresses to the visiting wireless client, we need to make a choice between Stateless and Stateful autoconfiguration (DHCPv6). In the IPv4 world, there is no counterpart to the stateless autoconfiguration, DHCPv4 is the dominant practice. So is this new stateless autoconfiguration mechanism appropriate to our wireless mesh network?

The apparent advantage with stateless autoconfiguration is its ease of deployment. By multicasting Router Advertisements (RAs), e.g. through running the radvd application on each mesh node, a network prefix that identifying the subnet associated with a local footprint is delivered to visiting clients. Clients then can generate their own addresses by combining the network prefix with locally available information such as interface identifier that are typically EUI-64 identifiers. Stateless autoconfiguration also has some disadvantages:

1. Routers need to present on each link, in our case, radvd is required to run on each mesh node on the non-OLSR (AP) radio. We have to maintain a *radvd* configuration file on each node.
2. It does not supply a DNS server address. A host needs to configure at least IP

addresses and a recursive DNS server address in order to be used.

With stateful autoconfiguration (DHCPv6), we have these benefits:

1. DHCPv6 servers provide means for securing access control to network resources by first checking admission control policies before allocating any addresses. In stateless autoconfiguration, every host that connects to the network can get an IPv6 address assigned and can use network resources.
2. DHCPv6 allows for the assignment of multiple addresses to an interface, and configuration of an interface with multiple IPv6 addresses is a fundamental feature of IPv6. DHCPv6 also allows for the dynamic assignment of additional addresses over time. Addresses are assigned to a host with a lease, a preferred lifetime and a valid lifetime. The mechanism can support renumbering through the assignment of new addresses whose lifetimes overlap existing addresses to allow for graceful transition. It can also provide different configuration to different users.
3. A DHCPv6 server can also provide DNS server list, domain name, search path and other configuration data needed by a client.
4. Security is included in the DHCPv6 base specification.

DHCPv6's main drawback lies in its complexity. The protocol is hard to implement and deploy. At present, there are no fully RFC 3315 compliant implementations available yet. By comparing the trade-off of these two mechanisms, we decide to use DHCPv6 to allocate IP address to clients.

3.3 Challenges in Integrating the Local Footprint and the Backhaul

To enable Access Point functionality on the non-OLSR interface, HostAP and MadWiFi device drivers are two good candidates. Since HostAP is more stable, we decide to use HostAP. Firmware in the IEEE 802.11 card is responsible for time critical tasks like beacon transmission and frame acknowledgment, while HostAP or MadWiFi drivers take care of management tasks: authentication/deauthentication, association/reassociation, and disassociation, data transmission between two wireless stations, power saving (PS) mode signaling and frame buffering for PS stations.

However, simply installing the HostAP driver and the OLSR routing protocol on the Linux box will not give us a mesh node that fits into the proposed mesh network, as OLSR radios on the mesh nodes do not know the reachability of wireless clients in local footprints.

The Host and Network Association (HNA) message in OLSR is employed to provide connectivity from the OLSR interface(s) to those non-OLSR interface(s). A node equipped with both of these interfaces can inject external route information into the OLSR mesh network. A mesh node announces the existence of an associated network in the local footprint by periodically multicasting HNA messages, with the following information:

- *Network Address*: The network address of the associated network.
- *Netmask*: The netmask, corresponding to the network address immediately above.

There are two kinds of HNA message. One is the network-specific HNA, announcing the reachability of a whole subnet. Another is the host-specific HNA, advertising reachability

on a per-host basis.

Obviously we need to determine the IP addresses of arriving wireless clients in order to generate the HNA messages. Then the AP propagates HNA messages to all the other nodes in the mesh network. This will cause the other nodes to update their routing tables. The local footprint thus integrates with the mesh backbone. To achieve this goal, we need to solve these problems:

1. When to start generating and propagating HNA message?

   If there is no client associated in a local footprint, a mesh node should not propagate HNA messages into the mesh backbone. The importance of reducing routing overhead can not be overstated. On the other hand, when a client becomes associated with the local footprint, HNA messages should be transmitted without any delay. Otherwise, packets may get lost.
2. When to stop propagating HNA message?

   After wireless clients leave the local footprint, HNA message propagating should be terminated. But how can a mesh node know that a client has left?
3. How to determine the IP address of the client?

3.3.1 Start HNA Messages

When an IEEE 802.11 client comes to an AP, it will go through the process of authentication and association, by exchanging IEEE 802.11 management frames with the format showing in Figure 3.2.

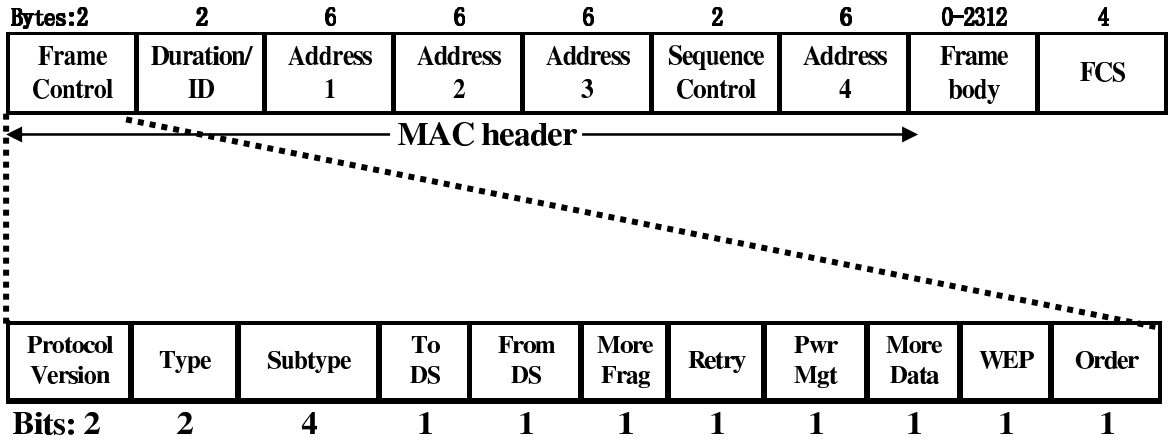

Figure 3.2 MAC Frame Format and Frame Control Details

The frame control field embedded in the MAC header provides rich information about each IEEE 802.11 frame. The 2-bit Type field identifies the frame as management, control or data frame. For a specific frame type, i.e. the management frame, the Subtype field identifies it as authentication, association or other management frames.

In order to study the association procedure, let us have a look at the frame body of the association request frame a client sent to an AP. It contains the following information shown in Table 1

Order	Information
1	Capability information
2	Listen interval
3	SSID
4	Supported rates

Table 3.1 Association Response Frame Body

When an AP receives an association request frame from a client, it checks whether the SSID, supported rates, and other parameters in the association request frame match its own. This handling process is shown in Figure 3.3. If the association request passes the check, the AP will sends an *association response* frame to the client with status code set to success. After the client acked the *association response* frame, the AP accepts the association, and creates a new association ID (AID). The MAC address of the client can be obtained from the Address 2 field in the MAC header of frames transmitted from the client. It is a good key to identify the associated clients in the local footprint.

Next, the client will initiate the procedure of acquiring an IP address, through DHCPv6. After the client successfully obtained an IP address, HNA message propagation should be started.

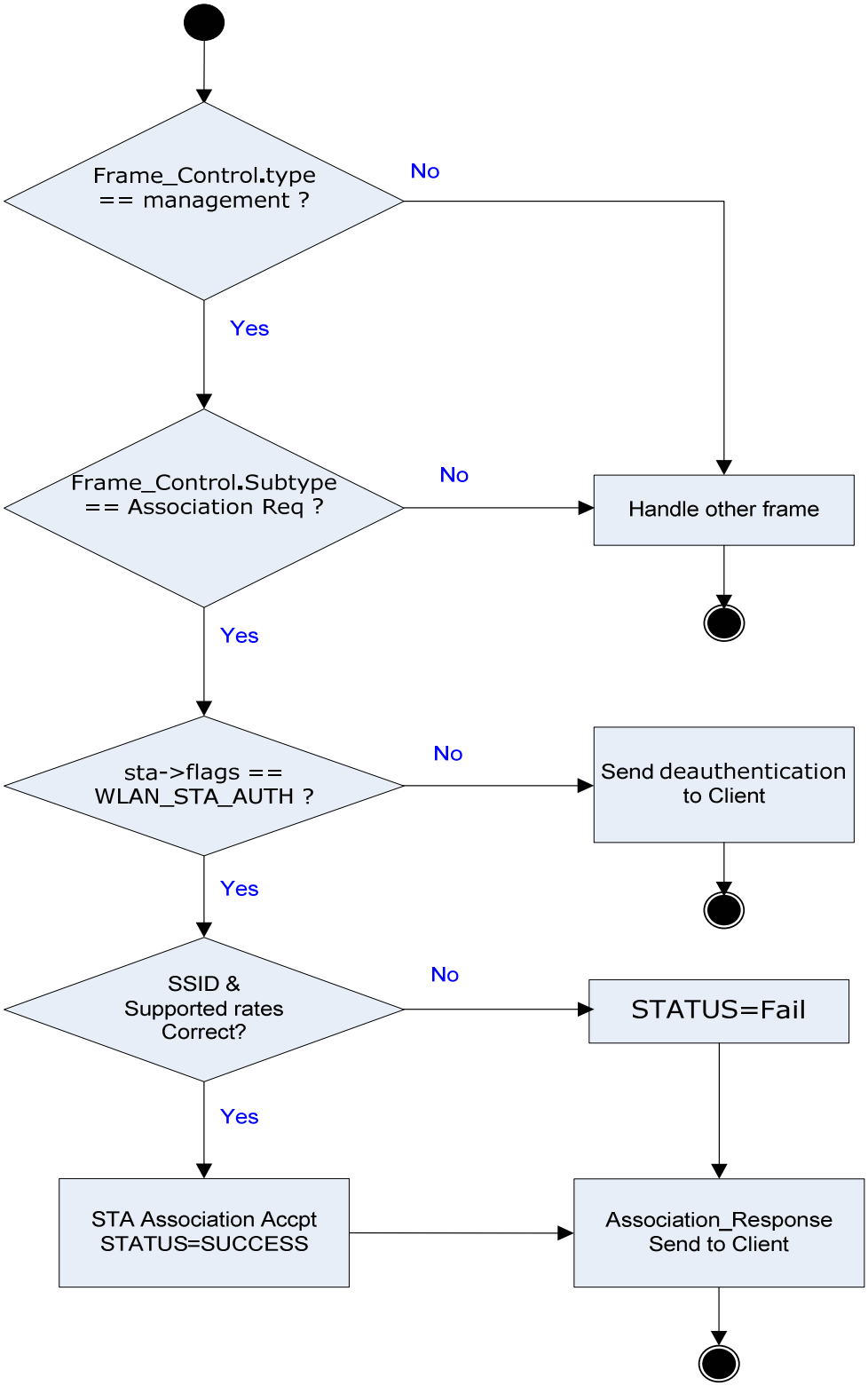

Figure 3.3 Control Flow of Association Request Processing

3.3.2 Stop HNA Messages

When the wireless client leaves the local footprint, the AP should stop propagating HNA messages. The IEEE 802.11 specifications describe the disassociation as "STAs (stations) shall attempt to disassociate whenever they leave a network. However, the MAC protocol does not depend on STAs invoking the disassociation service. (MAC management is designed to accommodate loss of an associated STA.)".

In Section 11 of the specifications, titled "MAC sublayer management entity", it does talk about association and reassociation, and then the section ends without mentioning anything about how to accommodate loss of an associated STA. So the specifications leave the implementation of this disassociation procedure up to the vendors.

If the station proactively sends a disassociation frame to the AP that it wishes to terminate the association (i.e. the client shuts down), upon receiving this frame, the AP then removes this client from the association list. But most of the IEEE 802.11 cards on the market do not provide the feature to disassociate when they go out of transmission range of an AP.

In the HostAP device driver, it is the responsibility of the AP to initiate the disassociation process. HostAP makes use of the special data frame type called *Null function*, which means the frame body is 0 octets in length. After each predefined interval, the AP sends a *Null function* data frame to poll the client. If the frame is not ACKed, the client will be disassociated. And that is the time when the AP should stop propagating HNA messages. One problem may result from this approach, especially when we can not define an appropriate timeout interval. If the interval is too long, the client may be reassociated with a new AP in another local footprint, and configured with a new IP address through

the DHCPv6 server. But the old AP is still waiting for the next timeout interval, and at the same time HNA messages are propagated to the mesh. To make things worse, the DHCPv6 server may renew or allocate the same IPv6 address to the client based on the client's DUID, the DHCP Unique IDentifier for a DHCP participant.
In this case, both the old and new AP would propagate HNA messages with the same network information! Other mesh nodes would get confused, and have no idea how to create a routing entry for the advertised network.

An alternative is to pre-configure all the APs in different subnets, and let the DHCPv6 server allocate addresses based on an AP's subnet. In our network design, the client will need the AP to relay the DHCPv6 Solicit or other messages to DHCPv6 server. The relay agent uses a Relay-forward Message, with the format shown in the following figure.

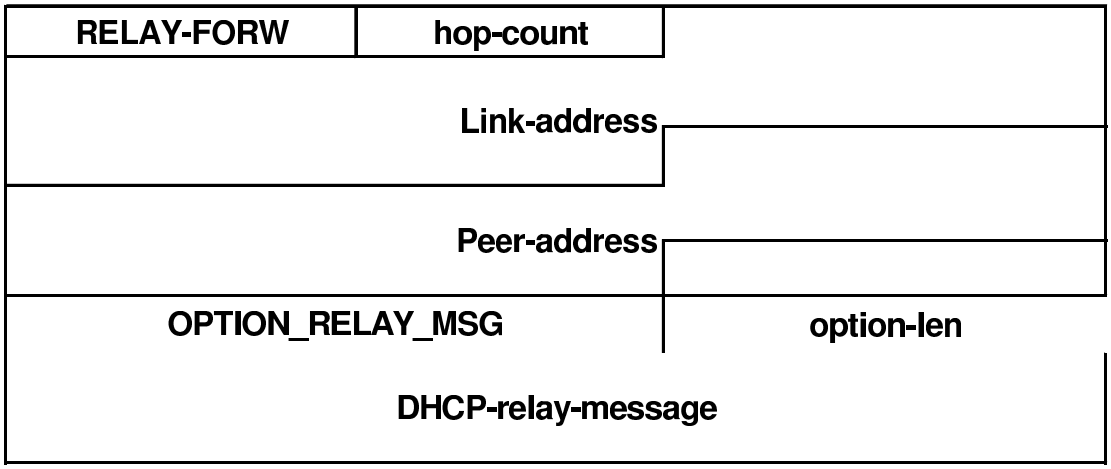

Figure 3.4 Relay-forward Message Format

The fields of the message are defined as follows:

- **Link-address:** Address used by the DHCP server to identify the link on which the client is located.
- **Peer-address:** Source IP in the IP header of the packet that the relay agent received
- **DHCP-relay-message:** Copy of the received DHCP message (excluding any IP or UDP headers)

The link-address field plays an important role. The DHCP server will allocate addresses based on the link-address. So when the client moves to a new AP, it will send a Confirm message to determine whether the addresses it was assigned are still appropriate to the link to which the client is newly associated. Because we configure each AP with different subnets, the server will return a status of *NotOnLink*. The client knows that the old IPv6 address is not suitable for the new footprint. Then it would send a DHCPv6 Solicit Message, initiating the DHCP protocol again. The result is that the client will get a different IPv6 address.

To configure each AP with different prefixes, we can use a new DHCPv6 Option, "IPv6 Prefix Options for DHCPv6", which is defined in RFC 3633. This option provides a mechanism for automated delegation of IPv6 prefixes using DHCPv6. It serves for the communication between routers; it is designed to let ISP send routing prefix to subscribers.

As shown in the following figure, a requesting router acts as a DHCP client and requests a prefix to be assigned. The delegating router acts as a DHCP server, and responds to the prefix request.

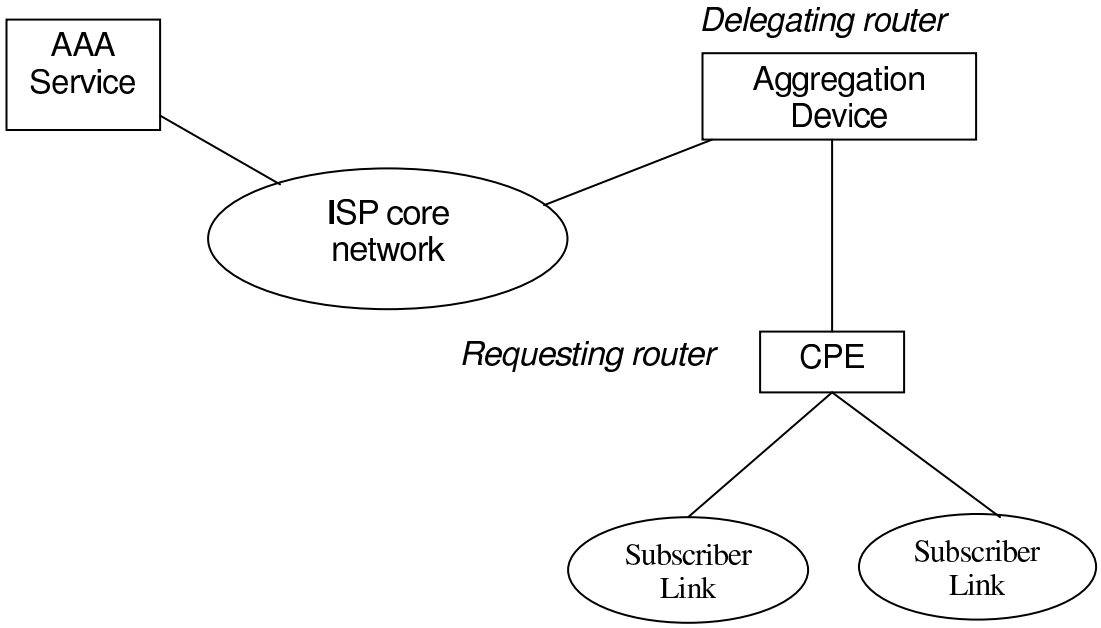

Figure 3.5 Prefix Delegation

When the Prefix Delegation is applied to our proposed network, each mesh node is a requesting router. From the DHCP server, each node will be guaranteed to obtain a different prefix assigned. Advantages of this solution are:

- Robustness. This solution guarantees that the new AP will not propagate duplicate HNA messages to the OLSR mesh.
- Less overhead. The old AP will not bother to poll the client frequently. We may set up a relatively long timeout interval.

However, to configure each local footprint with different subnets requires much more IPv6 prefixes. While this may be considered an inefficient use of IP addresses, given the size of IPv6's overall address space, it is still acceptable.

Based on the above functional requirement analysis, our design decision is to support both the network-specific and host-specific HNA message. If plenty of subnet prefixes are available, network administrators can configure mesh nodes with different prefixes. Otherwise, with the support of host-specific HNA message, mesh nodes configured with the same prefix still will not cause problems for the mesh network.

In conclusion, Null Function frame can be safely used to poll the activities of clients. Once *hostapd* daemon has detected an inactive client, it will notify *meshd*. If *meshd* is configured to support host-specific HNA messages, it will let *olsrd* stop propagating HNA messages for that inactive client. When *meshd* is configured to support network-specific HNA messages, HNA message propagation stops only if the inactive client is the last client in the local footprint.

3.3.3 Design of Dynamic HNA

The current CRC OLSR implementation *olsrd* supports only static HNA messages. It assumes the associated networks or hosts exist before *olsrd* starts, and will never change after *olsrd* executes. This assumption is acceptable considering the design of HNA message is to advertise gateway information to the mesh backbone, and the gateway is typically a wired router that provides connection to the Internet or intranet. So when *olsrd* is initiated, it first attempts to get HNA information by reading a configuration file. Then *olsrd* builds a HNA message that contains all the associated networks or hosts in the configuration file. If there is no such configuration file available, *olsrd* considers the interface that it is running on does not have any associated networks or hosts, and no HNA message will be generated.

Based on this design, OLSR specification allows removal of information in HNA messages only upon expiration. In our proposed mesh network, the number of wireless clients is unpredictable. An initially empty local footprint will have visitors later, and also a client associated with one local footprint may move to another local footprint. Therefore the HNA mechanism needs to be modified to support the proposed network architecture. The following are our modifications:

1. Instead of reading associated networks or hosts from a configuration file, we collect information dynamically at run time.
2. Instead of supporting network-specific only or host-specific only HNA messages, our implementation supports both types of HNA message.
3. To support host-specific HNA messages, removal of information in HNA messages is allowed before expiration. When a wireless client leaves a local footprint, the entry for the client should be deleted in the HNA message. Similarly, a newly arrived client in a local footprint should cause a new entry to be created in the HNA message.

3.3.4 Determining the IP Address of a Client

The IP address allocated by the DHCPv6 server is carried in the Reply message, with the format shown below:

Reply	transaction-id
OPTION_CLIENTID	
Client DUID (variable length)	
OPTION_SERVERID	
Server DUID (variable length)	
OPTION_IA_NA	option-len
IAID (4 octects)	
T1	
T2	
OPTION_IAADDR	option-len
IPv6 address (16 octects)	
preferred-lifetime	
valid-lifetime	
OPTION_STATUS_CODE	option-len
status-code	
status-message	

Figure 3.6 DHCPv6 Solicit Message Format

- Transaction ID**:** A client includes a Transaction ID in each of the messages it transmits, the server should respond with a Reply with the same Transaction ID. Any Reply Messages with different Transaction ID will be discarded.
- DHCP Unique Identifier (DUID): used to identify clients and servers. Each DHCP client and server should generate a DUID, and include the DUID in every message they transmit. When the server creates a Reply message, it copies the client DUID from the original Solicit or other messages, and includes its server DUID. Upon receiving the Reply message, the client compares its DUID and the

Client DUID option in the Reply message. Unless they are the same, the client will ignore the Reply message.

- Identity Association ID (IAID): used to identify a collection of addresses. The DHCPv6 server will assign IPv6 addresses by filling an Identity Association (IA) Address Option.

By capturing the Reply packet, we can extract the IPv6 address. And since the Netmask mechanism is discarded in IPv6, we modify the OLSR HNA message format as follows:

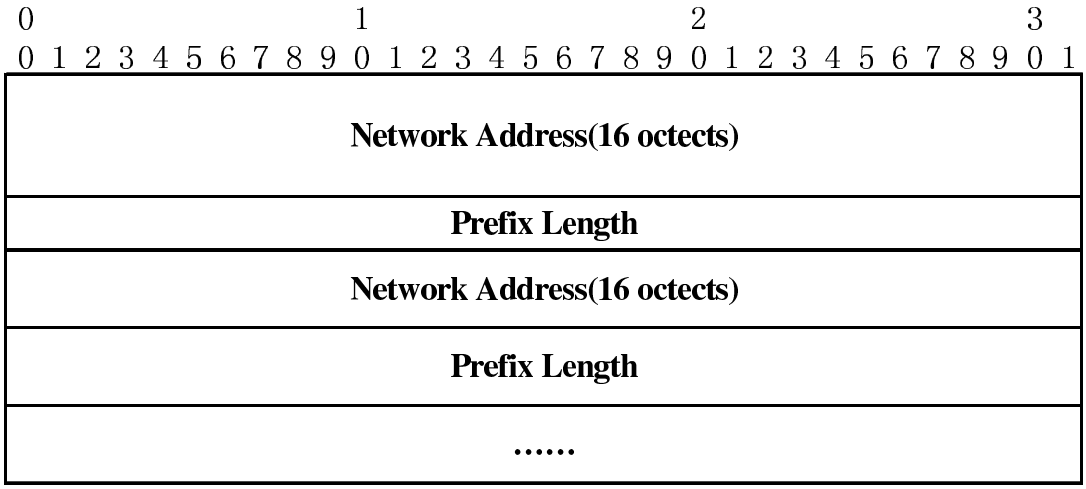

Figure 3.7 New HNA Message Format

Since the DHCP server provides prefix length in addition to the IP address, we can construct the HNA message.

3.4 Implementation Details

3.4.1 Determine Association Time

To start or terminate the HNA message propagation, we must know the time when a client joins or leaves the local footprint, and we also need to identify each client, for example, by means of its hardware address. In order to acquire all these information, we have to sniff the IEEE 802.11 association frames.

Traditionally, people rely on the *libpcap* library to capture and parse network packets. Well-known applications, such as *ethereal* [8] and *tcpdump* [29] are based on *libpcap*. But to use *libpcap*, HostAP must work in the special Monitor mode. Unfortunately Monitor mode and Master mode are mutually exclusive. And since having the functionality of an access point is a requirement of our design, we must set HostAP in Master mode.

Because we can not make use of the *libpcap* library, we create a raw Linux PF_PACKET socket, and bind it to the HostAP interface to capture the traffic between the AP and wireless clients. But when we analyze the captured packets, we find that these packets only contain Ethernet headers. This means even though packets have IEEE 802.11 MAC headers in the air, the device driver removes them before sending them to the user space. Our original idea was to modify the HostAP driver, having the driver not remove the IEEE 802.11 MAC header. But since a user space daemon program, *hostapd*, was later added to the HostAP driver package, and IEEE 802.11 management frame handling is moved to the user space instead of the kernel space. We also make our modifications

based on *hostapd*, leaving the driver intact. *Hostapd* includes extra features, such as support for IEEE 802.1X, dynamic WEP rekeying, and RADIUS accounting.

To have the kernel driver send the IEEE 802.11 management frames to user space, with IEEE 802.11 MAC frame headers, the HostAP driver needs to be configured to work in a so called "*hostapd*" mode. When *hostapd* is started on a wireless network interface, it triggers an *ioctl()* system call to the HostAP driver through a socket, configuring the device driver into *hostapd* mode. In *hostapd* mode, the driver will not process any management frames, and instead it sends them to *hostapd*. The following diagram roughly illustrates how the IEEE 802.11 management frames are sent to user space. It mainly involves these steps:

1. During the driver initialization, the HostAP driver registers a private ioctl in net_device, through the function pointer *do_ioctl*: *dev-> do_ioctl = hostap_ioctl.*
2. When the HostAP driver receives private ioctl calls from the user space daemon *hostapd*, it executes corresponding actions, such as putting itself in *hostapd* mode.
3. Whenever the driver receives an IEEE 802.11 management frame, it builds and appends the IEEE 802.11 MAC header to the frame. Then the driver calls the Linux kernel procedure *netif_rx()*. This procedure enqueues the received frame to the kernel backlog queue, and schedules a *softirq*.
4. When the *softirq* is executed, it calls *net_rx_action()*, which dequeues all the packets in the backlog queue. And for each of the packets, its packet reception handler is executed, such as *ip_rcv()* or other procedures depending on the type of the packet. In our case, a promiscuous receive procedure is triggered.

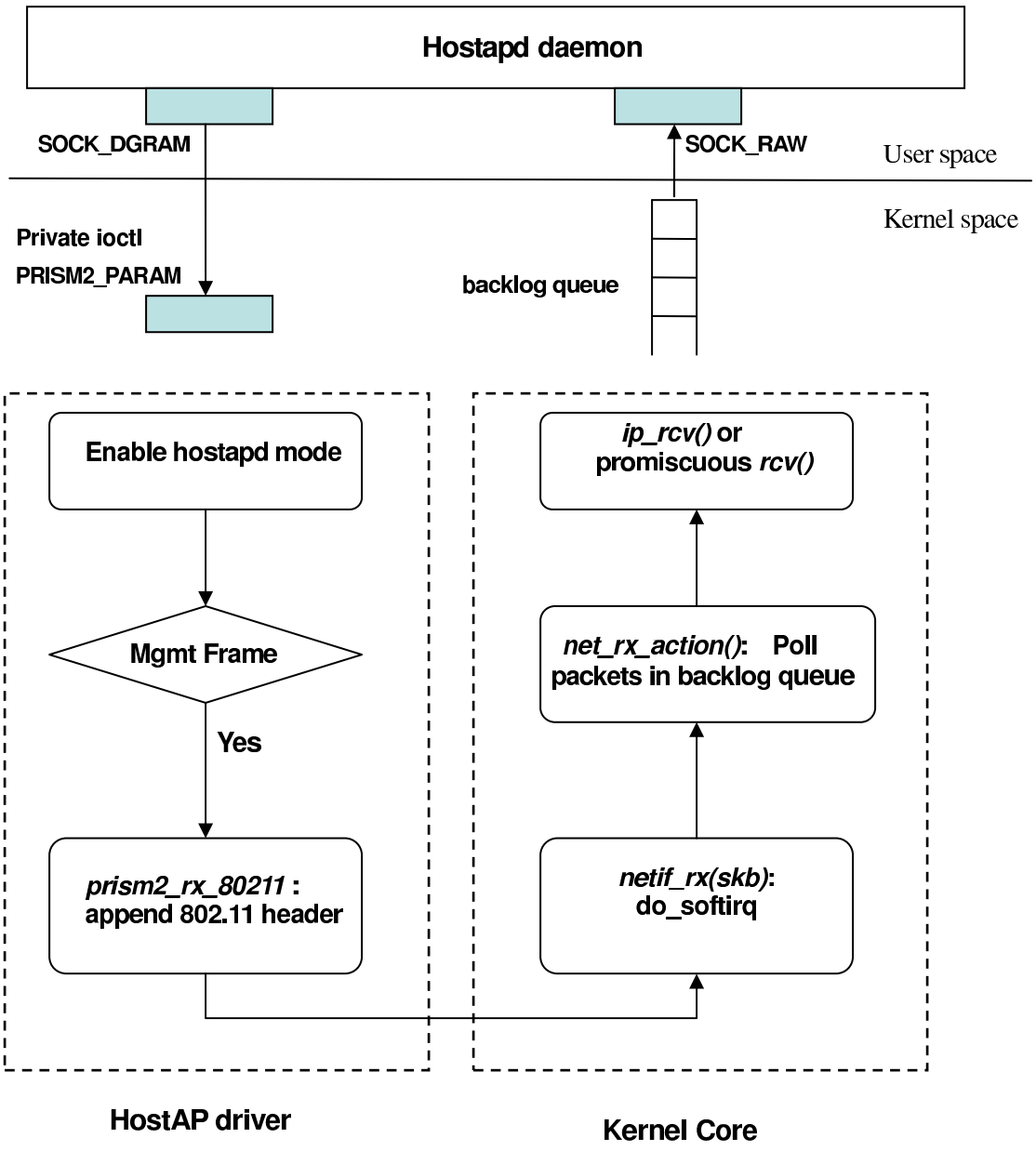

Figure 3.8 Management Frame with IEEE 802.11 MAC Header Reception

Our system, *meshd*, is the central control process that collects information about the local footprint and then propagates them into the mesh backbone. In order to fulfill this requirement, we need a mechanism to collect information from *hostapd* when wireless clients join or leave the local footprint. Linux has several mechanisms for Interprocess Communication (IPC). Pipe, message queue, and shared memory are popular ones. PF_UNIX domain socket is also a good approach to implement local communication. But it is not appropriate to change the main structure of the *hostapd* code. We would like to keep the modifications as limited as possible, since in the future we want to make use of its authentication features. Another reason is that HostAP is an actively developing project, if our approach is less flexible; we will have to adapt our code each time HostAP delivers a new release.

Taking this into account, we choose FIFO or named pipe to communicate with *hostapd*. Initially *meshd* creates a FIFO as read only and non_block, ready to receive messages from *hostapd*. If a wireless client visits the local footprint, it will go through the authentication and association process with the AP. In the time the client gets associated, *hostapd* sends a message to the FIFO, notifying *meshd* that a new client has arrived.

Figure 3.9 IPC between *hostapd* and *meshd*

How to know when a client leaves the local footprint? In HostAP, it keeps polling the wireless station in a predefined time interval, MAX_INACTIVITY. If HostAP can not detect any activity in the client, and the station does not acknowledge the polling, HostAP simply assumes the station is down or has left.

To understand how the inactivity time is detected, first let us look at the timing management in the Linux kernel. On 32-bit Intel platforms, Linux defines the time interrupt constant HZ to be 100, which means in one second the system clock ticks 100 times. Linux kernel variable jiffies are used to keep track of the current time. Every clock tick will increment jiffies once. The kernel also provides timers to schedule tasks. They are organized as a doubly-linked list. This is the timer data structure defined in <linux/timer.h>:

```
struct timer_list {
        struct timer_list *next;                /* never touch this */
        struct timer_list *prev;                /* never touch this */
        unsigned long expires;                      /* the timeout, in jiffies */
        unsigned long data;                         /* argument to the handler */
        void (*function)(unsigned long);    /* handler of the timeout */
        volatile int running;                   /* added in 2.4; don't touch */
};
```

The *expires* field in the timer structure is the timeout interval. When the timer expires, the scheduled handler function will be triggered. The *hostapd* daemon adds such a timer for

each associated client, and initializes the *expires* field with MAX_INACTIVITY. Every time the HostAP driver receives a frame destined to a client, or from the client, it timestamps the client by setting a variable *last_rx* with the current jiffes. After MAX_INACTIVITY time, the timer handler function in *hostapd* is activated to get the inactivity time of the client. This is done by making a private ioctl call to the HostAP driver. When the HostAP driver receives this request, it calculates the time elapsed since *last_rx*, and sends the result back to the *hostapd* daemon. Then *hostapd* checks whether the inactivity time is greater than MAX_INACTIVITY. A larger inactivity time indicates that there is no traffic to or from the client for MAX_INACTIVITY time. The *hostapd* daemon then sends an empty data frame, Nullfunc, to the client. If the client sends back an acknowledgment, *hostapd* resets the client's timer, and this process starts over again. Otherwise, *hostapd* will transmit a disassociation frame to the client. If after a short delay, *hostapd* still can not detect any activity from the client, *hostapd* sends a deauthentication frame to the client and removes the client from the associated clients list. The following figure depicts the process of how an AP disassociates an inactive client.

At this time, *hostapd* notifies the *meshd* process through the FIFO that a client, identified by its hardware address, has left the local footprint. Depending on whether the mesh network is configured to generate host-specific or network-specific HNA messages, *meshd* should activate the corresponding actions.

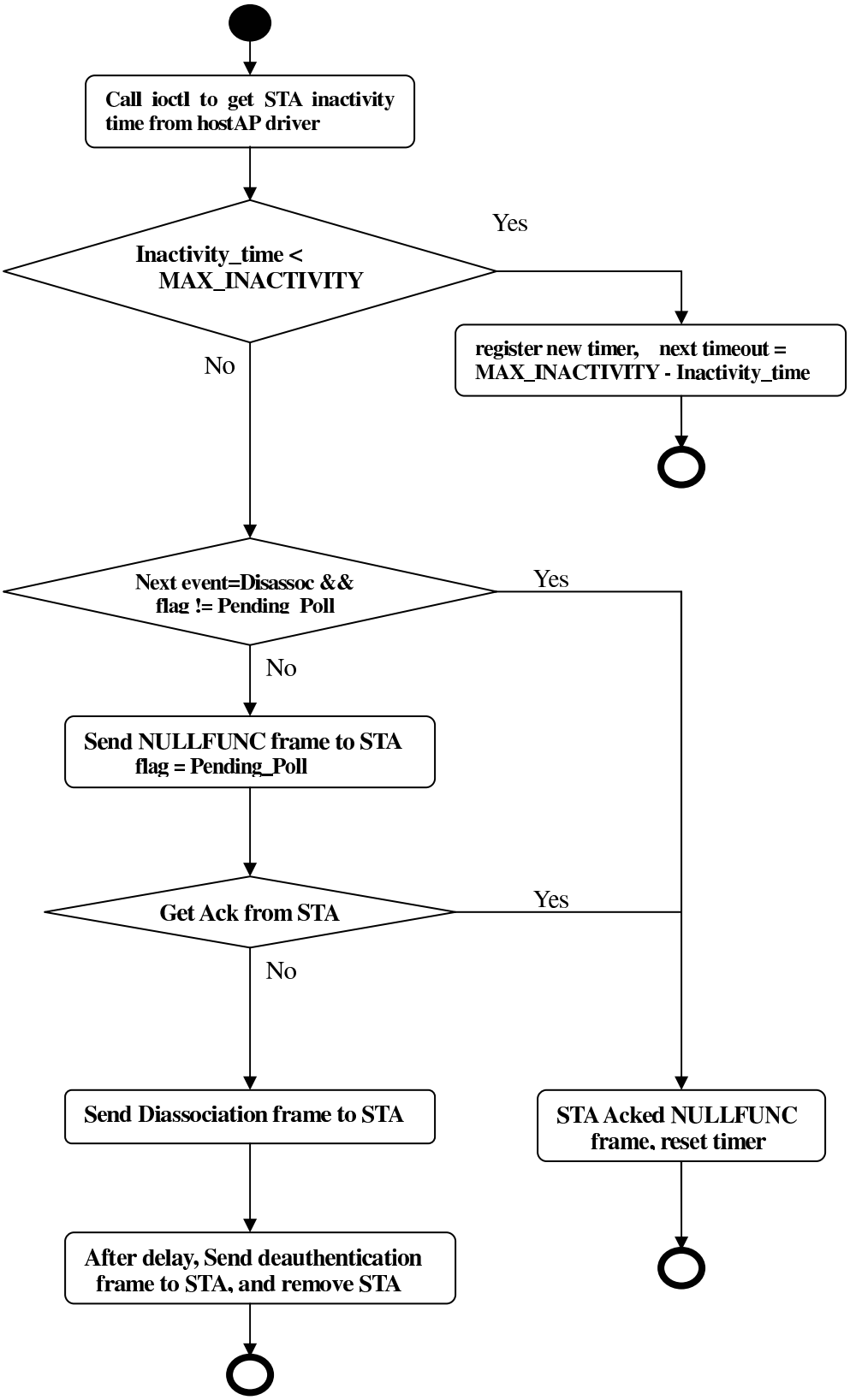

Figure 3.10 Disassociation Process

3.4.2 Extracting IPv6 Addresses

Before finalized as RFC 3315 in July 2003, DHCPv6 had gone through twenty eight drafts, with significant changes throughout the lifetime of the specification. That is why there are few available DHCPv6 implementations fully compatible with the final specifcation.

The HP DHCPv6 implementation seems to be the most complete implementation, but it only supports the HP-UX platform. Cisco IOS software has limited DHCPv6 support, mainly the Prefix Delegation support. It does not implement the entire DHCPv6 protocol.The Sourceforge DHCPv6 project intends to develop an open source DHCPv6 implementation. It is based on a test implementation from the KAME project from Japan, and ported from the original FreeBSD version to Linux. On March 15, 2004, the most recent release dhcpv6-0.10 provides the relay agent support. Our implementation is based on this release.

A DHCPv6 client initiates the message exchange with a server or servers to acquire or update IPv6 addresses and other configuration information. Typically it sends a Solicit message to discover DHCPv6 server(s). After a server or servers respond with Advertise Message(s), the client selects a server based on the server preference value extracted from the Advertise Message(s) and other factors. Then the client transmits a Request message, and waits for a Reply message from the server. Optionally, the client may transmit a Solicit message with a Rapid Commit option, asking the server to assign an address directly without any further Request/Reply exchange.

In either case, the Solicit or Request message is received by the relay agent daemon,

dhcp6r, on the AP radio. The relay agent then copies the payload part of the UDP packet into a Relay Message option in the Relay-forward message, and transmits the new message to other relay agents or DHCPv6 servers on the OLSR radio, the backbone radio. The response to the Relay-forward message is the Relay-reply message from other relay agents or servers. Upon receiving such a message on the OLSR radio, the relay agent extracts the Reply message included in the Relay Message option, and forwards the Reply message. The message exchange is shown in the following diagram.

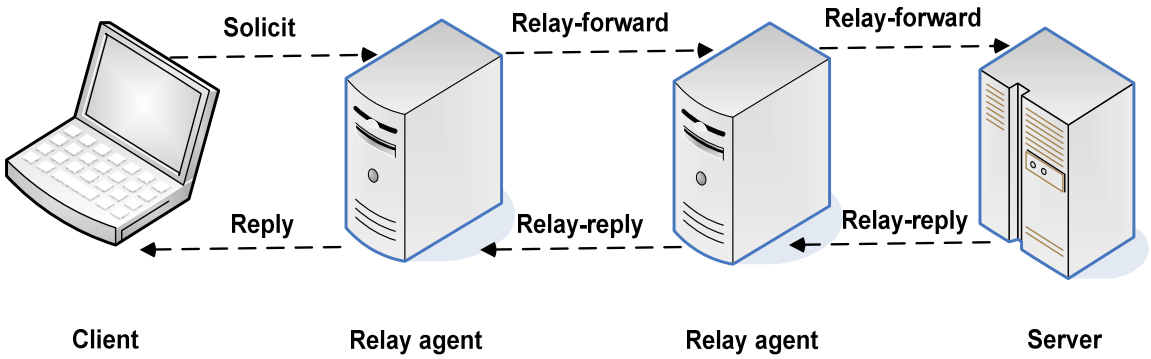

Figure 3.11 DHCPv6 Relay Agent Message Flow

One problem arises here. The relay agent should forward the Reply message to the interface on which it had received the Solicit or Request message. But how does the relay agent know the source interface information? It is achieved through the recvmsg() system call. As defined in RFC 2292 [28], "Advanced Sockets API for IPv6", network information, such as incoming/outgoing interface may be carried as ancillary data in addition to the actual payload. Ancillary data is also called control information, and it has the following header format:

```
struct cmsghdr {
        socklen_t     cmsg_len;          /* data byte count, including hdr */
        int               cmsg_level;        /* originating protocol */
        int               cmsg_type;         /* protocol-specific type */
                        /* followed by       u_char      cmsg_data[]; */
};
```

The cmsg_level is IPPROTO_IPV6 in our IPv6 based wireless mesh network. The cmsg_type is IPV6_PKTINFO, which means the cmsg_data field contains IPv6 packet information. The packet information structure is defined as following:

```
struct in6_pktinfo {
        struct in6_addr    ipi6_addr;        /* src/dst IPv6 address */
        unsigned int       ipi6_ifindex;     /* send/recv interface index */
};
```

The ipi6_ifindex field contains the interface index of the incoming packet. In our case, each time the relay agent receives a DHCPv6 message from the client on the AP interface, it records the receiving interface. When the relay agent extracts the Reply message from the Relay-forward message, it forwards the Reply to the client through the AP interface.

The above analysis tells us that we only need to capture packets on the AP interface, and still we can intercept both the Solicit/Request message from the client and the Reply message from the server. Because even though the Reply message is received on the OLSR interface, the dhcp6r daemon forwards it to the AP interface. Besides, we do not need to parse any Relay-forward or Relay-reply message.

The DHCPv6 message capturing operation mainly includes these following steps:

1. Each time a client joins the local footprint, *hostapd* notifies the *meshd* process through a named pipe or FIFO. *Meshd* creates a new entry in the associated client list identified by the client's MAC address.
2. Capture all DHCPv6 packets, which are UDP packets with destination port 546 or 547. Clients listen for DHCPv6 messages on UDP port 546. Servers and relay agents listen for DHCP v6 messages on UDP port 547.
3. If the packet has destination UDP port 547, we know these are DHCPv6 messages from the client side. But how to identify the source of a DHCPv6 message? Originally we used the PF_INET6 socket to capture the UDP packets. With the system call *recvfrom()*, we can get the payload directly, which is the DHCPv6 message. Since all the DHCPv6 messages from the clients contain the Client DUID option, and each DHCP client and server has exactly one DUID, by analyzing this option, we can obtain the information of the client. There are three types of DUID defined by the DHCPv6:
    - Link-layer address plus time
    - Vendor-assigned unique ID based on Enterprise Number
    - Link-layer address

    The Sourceforge DHCPv6 project implements the first type. It has the following format:

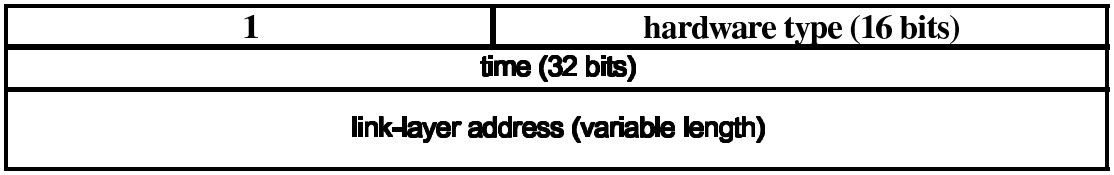

Figure 3.12 DUID Format

Because the device driver, HostAP, identifies a wireless client by its link layer address (hardware address), we need to extract the link layer address from the DUID option. Then we can determine which client transmitted the DHCPv6 message by comparing this extracted link layer address with the associated client list.

But there are problems arise from this solution. Sometimes we found the link layer address extracted from the DUID option did not match any of the associated client list. Through debugging the code, we finally found out the reason. The DUID is generated the first time the *dhcp6c* or the *dhcp6s* daemon is started, and stored on the hard disk. It does not change over time, even when the device's network hardware has been changed. Our system is developed in laptops, and wireless cards are randomly selected each time. So when we change the wireless card, the link layer address contained in the DUID does not change accordingly. Apparently, we can not use DUID to identify a client. The PF_INET6 socket is not appropriate in this case.

Instead we use a PF_PACKET socket. Both of the types, SOCK_RAW and SOCK_DGRAM, can meet our needs. Packets received through a SOCK_DGRAM packet socket are called cooked packets, which do not have the link level headers. To attain the link level address, the *recvfrom()* system call is needed. The address information is available from the returned parameter, which has the following format:

```
struct sockaddr_ll {
    unsigned short   sll_family;     /* Always AF_PACKET */
    unsigned short   sll_protocol;   /* Physical layer protocol */
    int              sll_ifindex;    /* Interface number */
    unsigned short   sll_hatype;     /* Header type */
    unsigned char    sll_pkttype;    /* Packet type */
```

```
        unsigned char     sll_halen;        /* Length of address */
        unsigned char     sll_addr[8];      /* Physical layer address */
}
```

As shown in the above structure, the field *sll_addr* is the 8-byte source hardware address. With a SOCK_DGRAM packet socket, the destination hardware address is not available. But a SOCK_RAW packet socket provides both the source and the destination hardware addresses. So we choose to use this type of socket to capture packets.

4. After obtaining the MAC address of the client from the link layer header, we can find the client from the station list. And we record Transaction ID, Client DUID and other information in the list.
5. If the packet has destination UDP port 546, it is a Reply message. The validation of the Reply message is illustrated in the following diagram.
   a) Extract the Transaction ID, and search the associated clients list to find the client who had requested IPv6 address(es) based on the Transaction ID.
   b) After finding out the corresponding message of a client, compare the Client DUID in the two messages. If the Client DUIDs are the same, proceed to the next step, otherwise discard the Reply message.
   c) If the DHCPv6 message from the client is a Request or Solicit message and with a Rapid Commit option, check the status code of the Reply message. If the DHCPv6 message from the client is other types, for example, Confirm, Rebind, Renew, or any other message, discard the Reply message.
   d) If the status code is Success, the Reply is accepted.

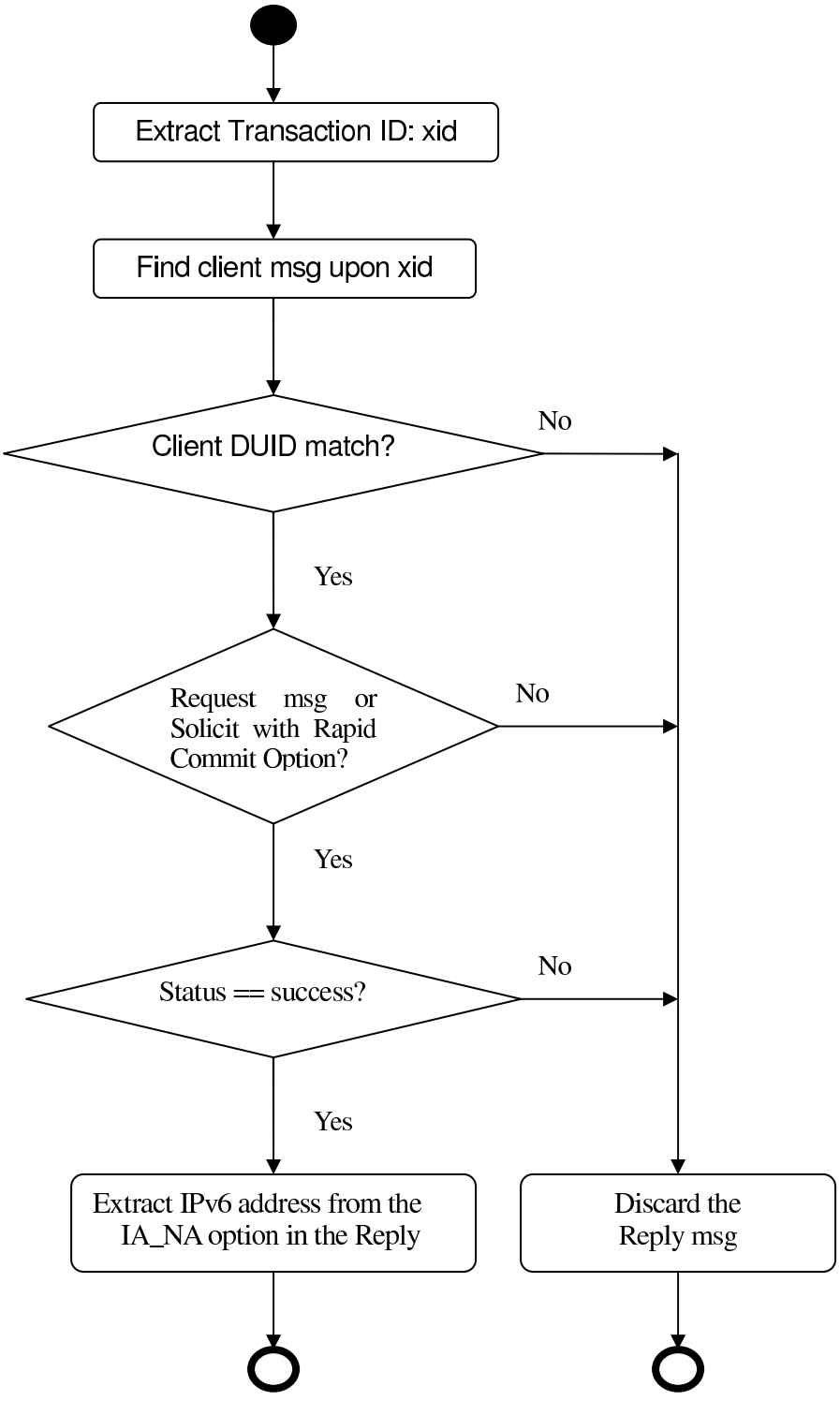

Figure 3.13 IPv6 Address Extraction

At this stage, it is straightforward to extract the IPv6 address from the IA_NA option. As for the Renew or Rebind Messages, which are used to extend the lifetime of the lease, we can simply ignore them. The Confirm Message is processed similarly.

With the extracted IPv6 address, should we start to build the HNA message? The answer is no, since we are not sure whether the client will accept the assigned IPv6 address. The client is required to make sure this new IPv6 address is unique.

3.4.3 Duplicate Address Detection

An IPv6 host must perform Duplicate Address Detection (DAD) on all unicast addresses before assigning them to an interface, no matter whether these addresses are acquired by means of DHCP or stateless autoconfiguration. Section 5.4 of the stateless autoconfiguration RFC (RFC 2462) describes the DAD procedure. It uses two ICMPv6 messages, defined in Neighbor Discovery (RFC 2461) [20]. One is Neighbor Solicitation (NS) and the other is Neighbor Advertisement (NA). NS and NA are also used to perform the Neighbor Unreachability Detection, and address resolution, which corresponds to the ARP protocols in IPv4. DAD NS differentiates with other NSs in that the source IP address is the unspecified address (::, or 128-bit zero).

In our case, after the DHCPv6 client validates the Reply message, it begins the DAD process by multicasting a DAD NS message. The interval between two NSs is *RetransTimer* milliseconds. If the client receives no response after the time the last NS message being sent plus a *RetransTimer* milliseconds delay, the address can be considered unique.

After we extract the IPv6 address, we also need to perform the Duplicate Address Detection. We can either implement DAD by parsing Neighbor Solicitation Message and

Neighbor Advertisement Message, or we can check whether the DAD procedure succeeds or fails, by parsing the file /proc/net/if_inet6. The following is a sample file of /proc/net/if_inet6, for a loopback interface.

cat /proc/net/if_inet6

1	2	3	4	5	6
00000000000000000000000000000001	01	80	10	80	lo

Figure 3.14 DAD Flag

There are six fields in each entry, representing different information about the interface. All fields are displayed in hexadecimal:

1) IPv6 address, 32 hexadecimal chars. It is ::1 in this example for the loopback interface.
2) Interface index in hexadecimal, 2 hexadecimal chars.
3) Prefix length, 2 hexadecimal chars.
4) Scope value, 2 hexadecimal chars.
5) Interface flags, 2 hexadecimal chars. It contains DAD flags.
6) Device name, the value is “lo”.

If DAD fails, it will set the Interface flag to IFA_F_TENTATIVE (defined in include/linux/rtnetlink.h), the opposite is IFA_F_PERMANENT. In the file, /proc/net/if_inet6, the kernel will set the value of 0xc0 to indicate a failed DAD, while 0x80 in the interface flag field indicates that the address is unique.

Since our implementation has a constraint that we do not want to change any client side code in order to support standard configured clients, we can not make the client send DAD information to *meshd*.

Our approach is to sniff the DAD Neighbor Solicitation (NS) and Neighbor Advertisement (NA) messages directly. After receiving the first DAD NS from a client, *meshd* creates a new thread. The main function of this thread is to check whether DAD passes or fails. Whenever a DAD NA is received in response to the DAD NS, the DAD fail flag is set for that client.

The current IPv6 implementation in Linux keeps the parameter DupAddrDetectTransmits in the file /proc/sys/net/ipv6/conf/default/dad_transmits, and the default is 1. The value for RetransTimer is defined in RFC 2461 in Section 10. It defaults to 1,000 ms.

Although these constants can be changed, most of the time clients would probably simply use the default values. But for the sake of safety, we suppose DupAddrDetectTransmits is set to 3. So if after 3 seconds, the DAD flag is not set to fail, we assume the client has passed the DAD check, and the thread can exit. At this time, the uniqueness of the allocated IPv6 address can be determined, and it is the time to generate the HNA message and propagate it to the mesh backbone.

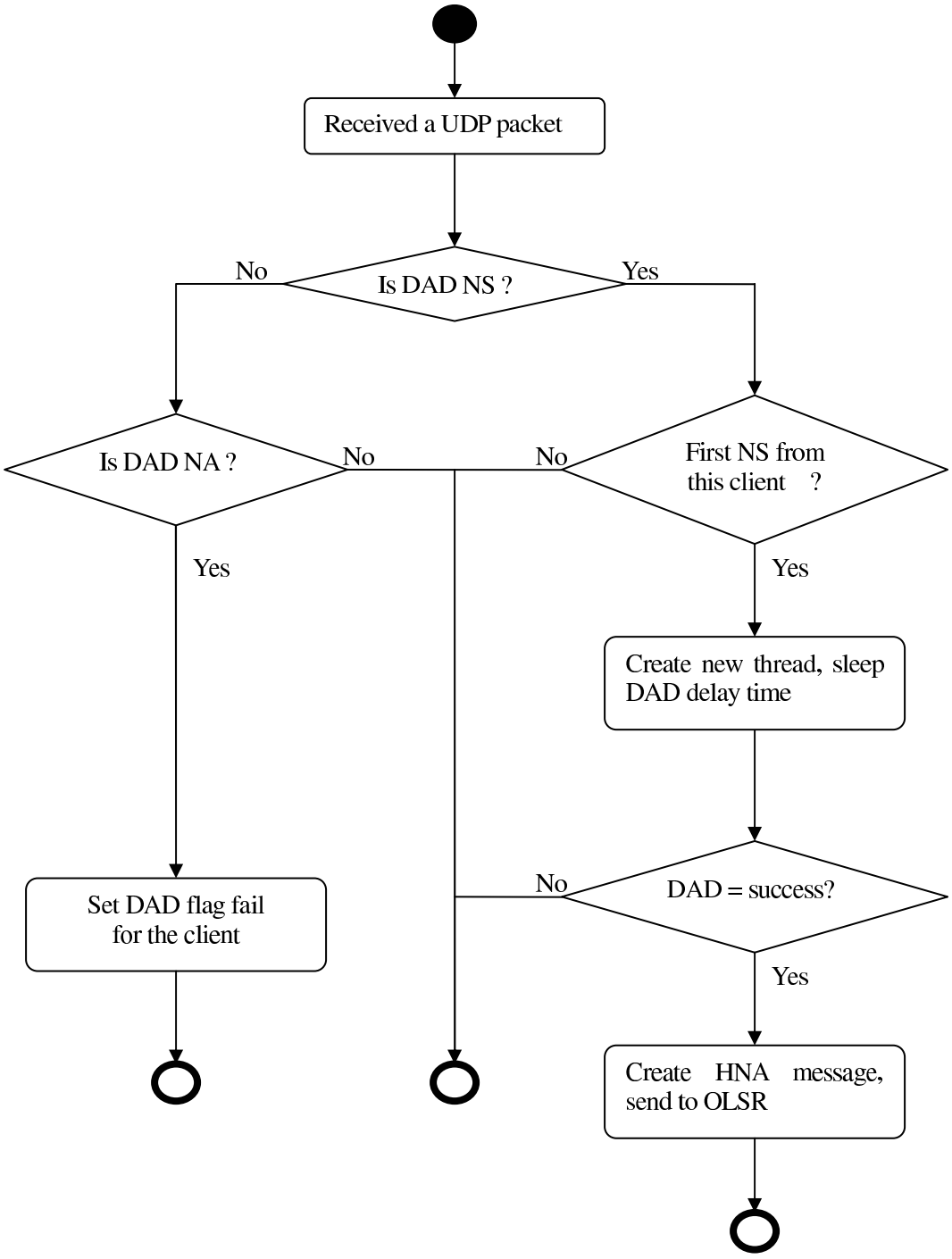

Figure 3.15 DAD Parsing

3.4.4 Implementation of Dynamic HNA

Dynamic HNA is achieved with the help of *meshd*. In *meshd*, it maintains a client list that stores various information for each associated client, such as hardware address, IP address, and etc. Each time when a client associates with a local footprint, it configures an IPv6 address through a DHCPv6 server. *meshd* acquires the IP address and prefix length information and sends the data to *olsrd* through a FIFO.

FIFO is chosen instead of local domain Unix socket for interprocess communication, because we try to make as few changes to the existing code as possible, and keep the coding style consistent.

Figure 3.16 IPC between *meshd* and *olsrd*

Besides the IP address and prefix information, the data *meshd* writes to the FIFO also contains information that instructs *olsrd* whether it should generate host-specific or network-specific HNA messages, and tells *olsrd* whether it should start advertising HNA or stop advertising HNA message for a host or a network.

Upon receiving FIFO data from *meshd*, *olsrd* manages HNA messages according to the FIFO data.

1. If the action type is stop, *olsrd* then further checks the HNA type. If it is network-specific, *olsrd* can safely conclude that the last wireless client has left the local footprint. Therefore, HNA propagation should be stopped. If it is host-specific, *olsrd* removes the entry for this host in the host list. *olsrd* then checks whether the

host list is empty. If it is empty, *olsrd* will not propagate HNA messages.

2. If the action type is start, *olsrd* creates a new entry in the host list or subnet list based on the HNA type, and builds an HNA message. The timer for HNA will trigger the transmission of an HNA message.
3. Each time an entry in the host list or subnet list is created or removed, the sequence number for the HNA message is incremented. This is just like the case of TC message broadcasts. Each time the network topology changes, the sequence number for the topology table is incremented. Upon receiving an HNA message, a mesh node will check whether there exists an entry in the HNA table whose hna_host_addr corresponds to the originator address of the HNA message, and whose hna_host_seq is smaller than the sequence number in the received HNA. If it finds such an entry, then the entry will be removed. And for each of the host or subnet IP addresses received in the HNA message, it will create a corresponding new entry in the HNA table where:
   - hna_subnet_addr is set to the host or subnet address,
   - hna_host_addr is set to the originator address of the HNA message,
   - hna_host_seq is set to the sequence number of the received HNA,
   - hna_host_timer is set to the value of HNA_HOLD_TIME.

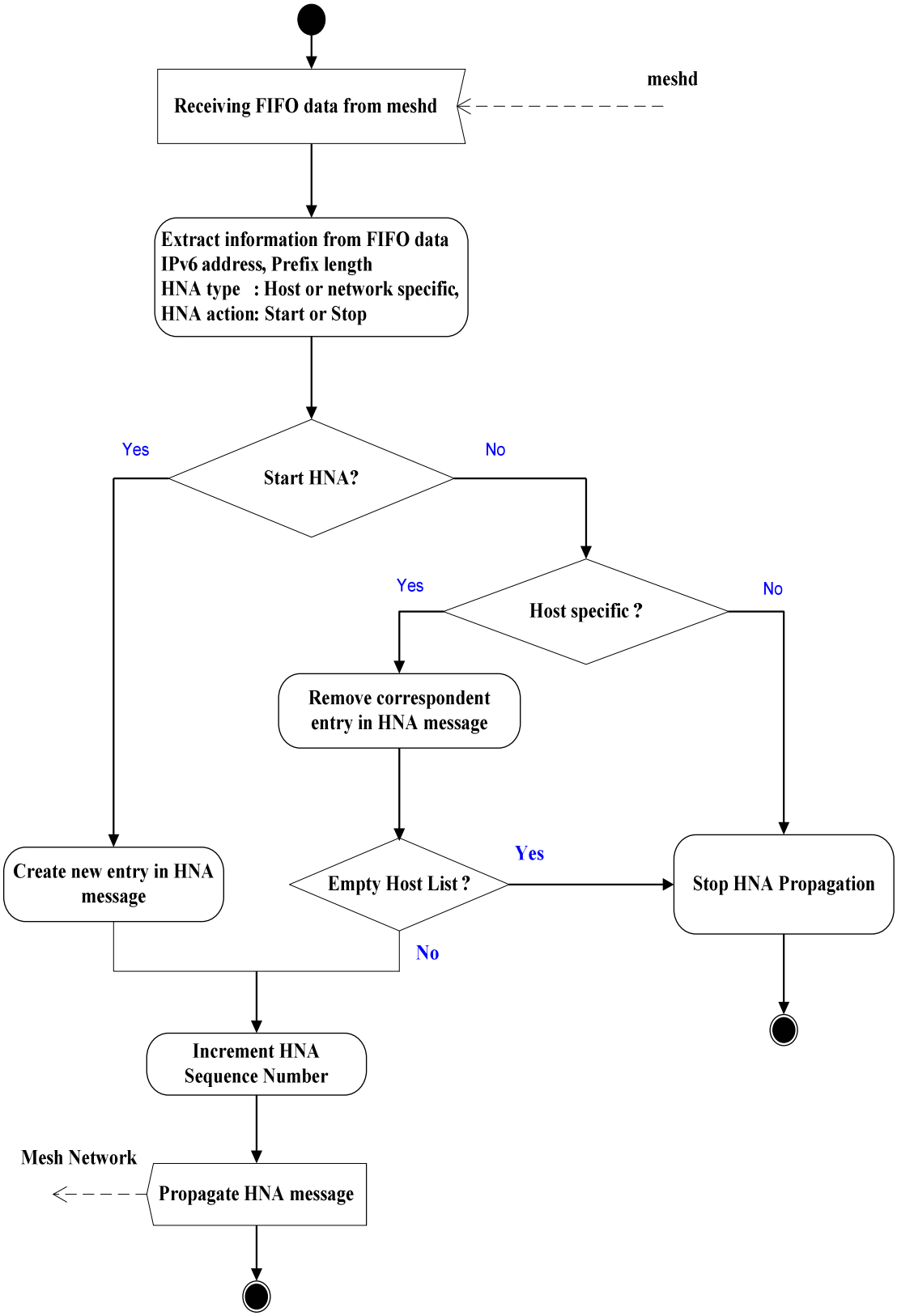

Figure 3.17 Control Flow of Modified CRC *olsrd* HNA Processing

3.5 Summary

The software structure of a mesh node is illustrated in the following diagram:

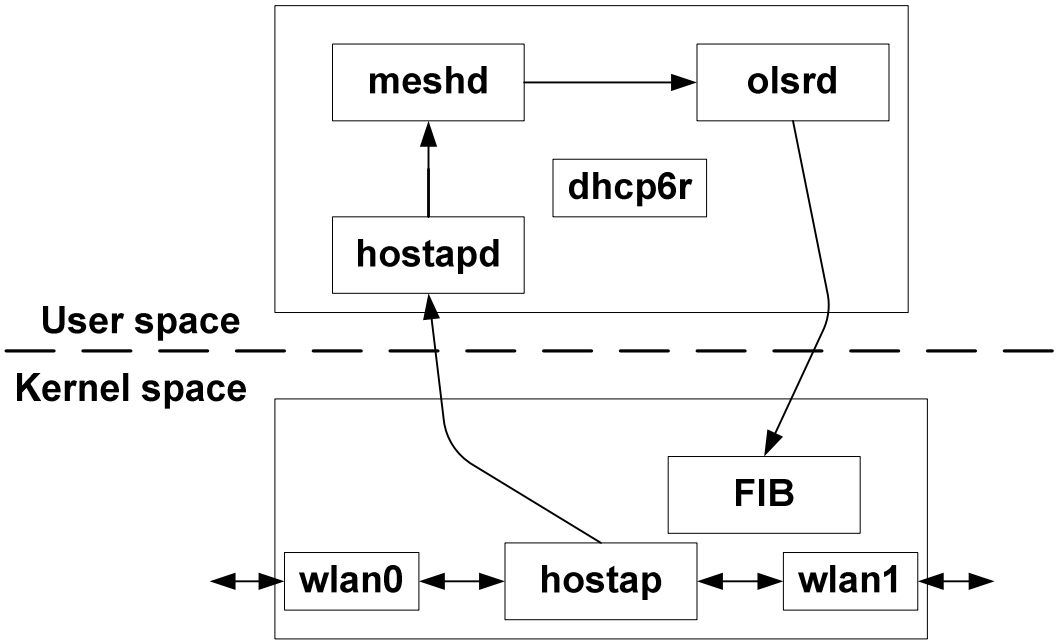

Figure 3.18 Software Structure of a Mesh Node

In the user space, there are four processes, *hostapd*, *dhcp6r*, *olsrd*, and *meshd*. In the kernel space, the HostAP driver is responsible for packet reception and transmission on both of the wireless interfaces. The driver also sends all the IEEE 802.11 management frames to *hostapd*. *Olsrd* manages routing by manipulating the kernel Forwarding Information Base (FIB).

Chapter 4

Experiments and Evaluation

In Chapter 3, we described the design and implementation of the proposed solution to integrate the local footprint and the mesh backhaul. In this chapter, we design and build a testbed with our implementation. Different test scenarios are designed to demonstrate the functionality of our implementation and evaluate the performance of our approach. Section 4.1 briefly describes the hardware of our testbed. In Section 4.2, we conduct a live test to verify the basic functionality. Section 4.3 discusses the result of an emulated test involving four mesh nodes and a wireless client. In Section 4.4, we use IPv6 enabled FTP software to test the TCP throughput in an emulated 6-hop network.

4.1 Testbed

All the nodes in the testbed are Linux laptops or desktops. Each consists of two IEEE 802.11 wireless interfaces and an Ethernet interface. Most of the wireless interfaces are Linksys WPC11 PC cards, which are based on Intersil's Prism 3 chipset. A few are Netgear Dual Band WAG511 PC cards or WAG311 PCI cards. These Netgear cards support IEEE 802.11b/a/g. To be compatible with the Linksys cards, we set them to work in IEEE 802.11b mode. Since Netgear cards are based on the Atheros chipset, the

MadWiFi driver is configured. Devices with Netgear cards are used only as clients, not as mesh nodes. Some of the machines use Linux kernel 2.4.26, while others use kernel 2.6.6 based on the Fedora core 2 distribution.

4.2 Test Scenario One: Basic Functionality Test

In this test scenario, we deploy two mesh nodes. On each node, we start the following processes: *olsrd*, *dhcp6r*, *hostap*d, and *meshd*. *Meshd* is configured to run in host-specific HNA mode. Then we start a client to associate with one mesh node. After it associated, we run the DHCPv6 client dhcp6c on the client, and it can successfully acquire an IPv6 address, and the HNA message is initiated.

By adding two more clients to the same local footprint, we find that the IPv6 addresses of the two clients are appended to the host list in the HNA messages. Next, we disassociate the three clients one by one. Since we set the IN_ACTIVITY parameter to one minute, gradually, all the three clients are disassociated by HostAP. Each time a client is disassociated, we notice that the corresponding IPv6 address is removed from the host list in the HNA message. And when the last client is disassociated, no further HNA message is generated.

Compared with the original CRC *olsrd* [5], our implementation reduces routing overhead. The CRC *olsrd* generates an HNA message every HNA_INTERVAL time, even if all the clients have left the local footprint. Let us look at an N-node mesh network with a string topology as shown in Figure 4.1.

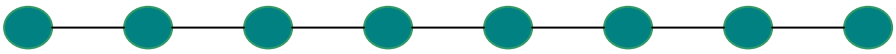

Figure 4.1 A network with a string topology

Except for the first and the last node, all the other N-2 nodes are MPRs. When one of the two end nodes generates an HNA message, all the N-2 MPRs will forward this HNA message. So there are totally N-1 HNA messages flooded into the whole network. If the HNA message is originated from nodes other than the two end nodes, the total number of HNA messages flooded into the network is N-2. With N being a large number, we can assume each node will flood approximately N-2 HNA messages into the network by ignoring the special case with the two end nodes. In CRC *olsrd*, all the nodes must advertise HNA messages. So the network experiences N*(N-2) HNA messages every HNA_INTERVAL time. In our implementation, the number of HNA messages generated depends on the number of active local footprints. If all the local footprints are empty, our new *olsrd* will not propagate any HNA message at all. With K local footprints that are not empty, the new *olsrd* will inject K*(N-2) HNA messages into the network. The following diagram illustrates the difference between the two implementations.

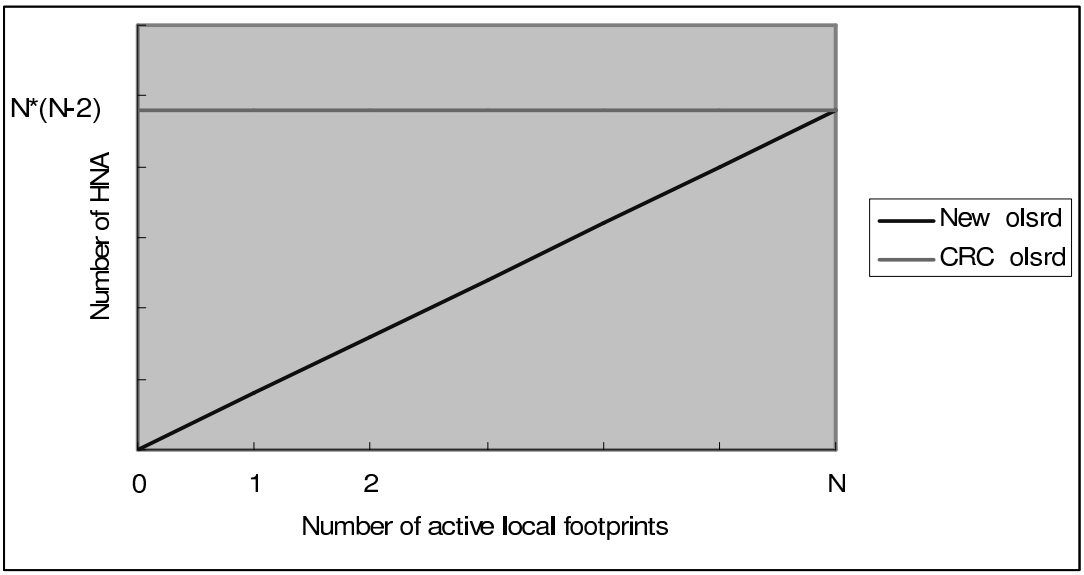

Figure 4.2 Comparison of HNA messages, String Topology

The network with a string topology does not benefit much from the MPR mechanism. The MPR mechanism performs best in a star network as shown in the following diagram. There is only one MPR node, which is the center node.

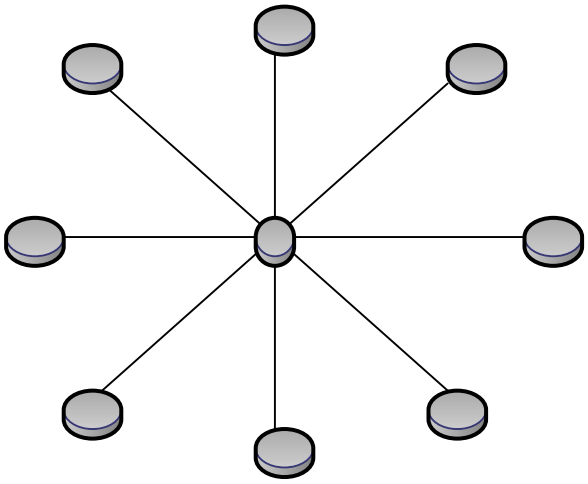

Figure 4.3 A network with a Star Topology

If the center node generates an HNA message, other nodes will not relay this HNA message. If the HNA message is originated from nodes other than the center node, the center node will relay the HNA message. So in this case, the network experiences totally 2*N-1 HNA messages every HNA_INTERVAL time. With N being a large number, approximately 2*N HNA messages will flood the network every HNA_INTERVAL. Similar to the analysis of the above string topology, CRC *olsrd* always injects 2*N HNA messages into a star network, while our implementation generates 2*K HNA messages, where K is the number of active local footprints.

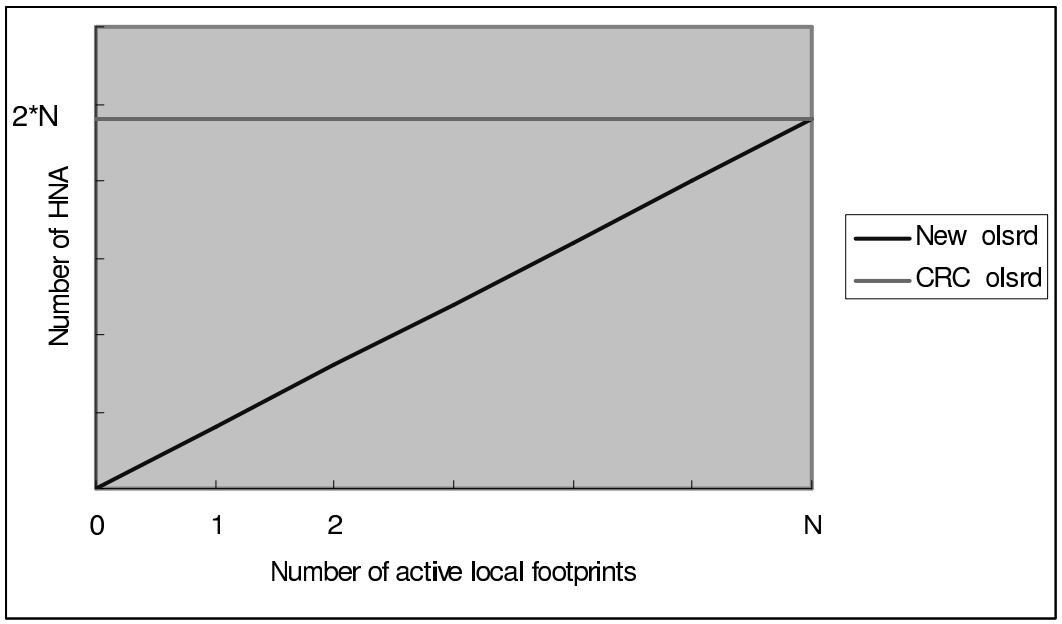

Figure 4.4 Comparison of HNA Messages, Star Topology

Since each HNA message is 20 bytes plus 20 bytes per pair of network address and network prefix, for a network-specific HNA message, it will emit a data frame of 146 bytes into the mesh network every HNA_INTERVAL time. No matter whether the local

footprint is empty or not, each mesh node emits HNA messages and all the MPR nodes retransmit them in CRC *olsrd* implementation. In a large mesh network, these HNA messages not only consume quite a few bandwidth, but also bring more latency to the network due to their contentions to access the media. Our implementation avoids unnecessary HNA message propagation. Therefore, it helps to improve the network performance.

In order to test the possibility of deploying such a mesh network in the real world, we conduct a live experiment to measure TCP throughput. A FTP client downloads a big data file (about 100M bytes) from its associated mesh node directly. We conduct ten runs of FTP sessions, and the throughput we measured is shown in Figure 4.5.

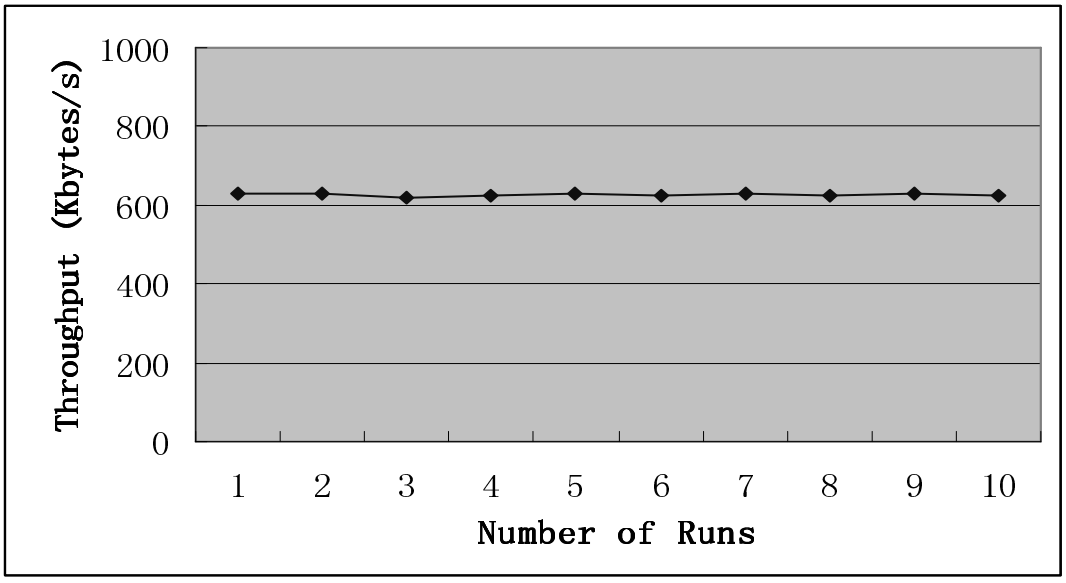

Figure 4.5 TCP Throughputs in Kbytes/s

Both the client and server are configured with IEEE 802.11b cards. In theory, the throughput is 11 Mbps. In our live test, the average throughput we can achieve is about 628 Kilobytes per second, which is about 5 Mbps. Why can we only make use of about half the capacity?

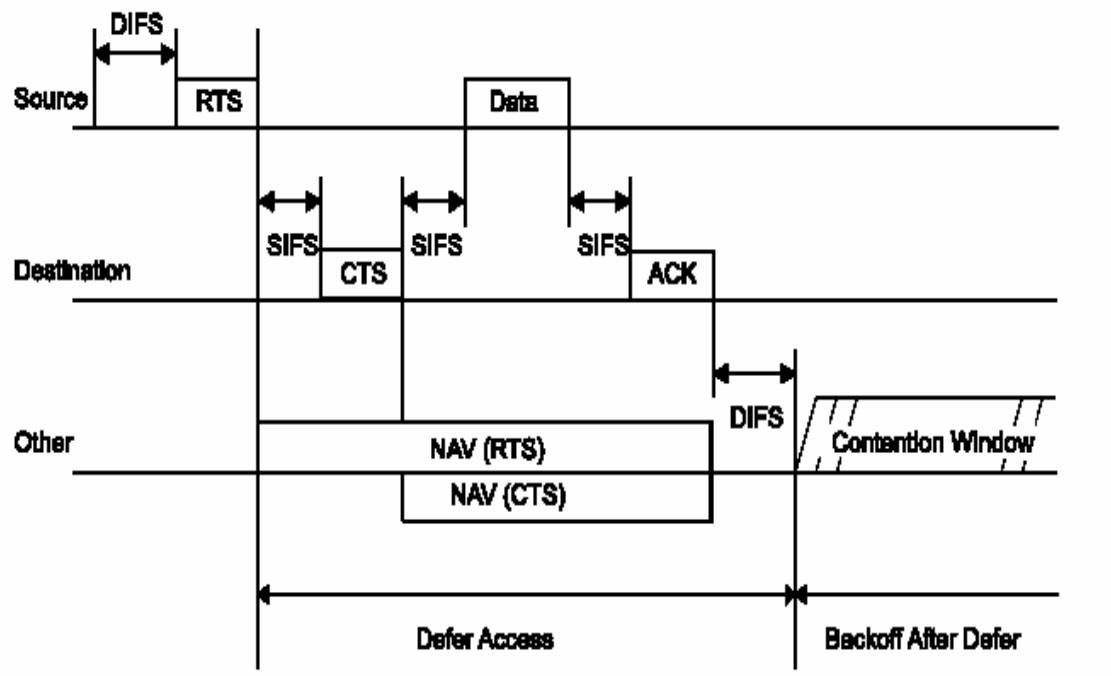

Figure 4.6 MAC Layer Delay (Source: IEEE 802.11 Spec. [14])

As the above figure shows, IEEE 802.11 stations have to contend for the channel. To simplify the analysis, we ignore the RTS/CTS mechanism. First, all stations wait for a DIFS (DCF inter frame space) time, which is 50μs for 802.11b. Then there is an average random back-off time. An ideal scenario would be that the sender and the receiver take turns in transmitting TCP data or TCP ACK segments, which causes an average back-off

delay of 80μs. With more nodes contending for the medium, the average back-off time will be longer than our two-node scenario. Next is the transmission of physical layer preamble and header, which is 192 bits in total. Since they are transmitted at 1 Mbit/s, it will take 192μs. The actual data frame transmission follows. Suppose we want to send a 1500-byte IP packet, the data frame is 1534 bytes with the MAC header. The transmission time is 1116μs.

When the destination station receives the frame, it will first wait for a SIFS (short inter frame space) time, which is 10μs. Then it sends a 14-byte ACK frame, which takes 10.2μs plus the 192μs needed for the physical preamble and header. The destination station also needs to sends a TCP ACK packet. A similar delay happens again.

Roughly speaking, a 1500-byte IP packet takes 2.181 ms (or 2181μs) to reach the destination and receive a TCP ACK. Therefore every second we can transmit 459 TCP segments. Taking into account the 40-byte IPv6 header and 20-byte TCP header, each TCP segment can carry 1440 bytes payload. The TCP throughput is approximately 5.29 Mbps for a wireless IPv6 network. Our test results are close to the above analysis.

4.3 Test Scenario Two: Connectivity Test

4.3.1 Mobile Network Emulator (MNE)

Test case one only involves two mesh nodes. To expand the network and create multi-hop paths, we have to place the laptops far apart. For example, we put one laptop at the Minto Center, and its 5-hop neighbor has to stay at the Mackenzie Building. It is even harder to repeat a test. So instead we use an emulator, MNE. It is developed by the Naval Research Laboratory (NRL). With MNE the movement of laptops is emulated. Each node moves

virtually to a location specified in an NS-2 scenario file. To avoid interference, a supplemental Ethernet interface is used to transport control information. Figure 4.7 illustrates the network topology of test case two.

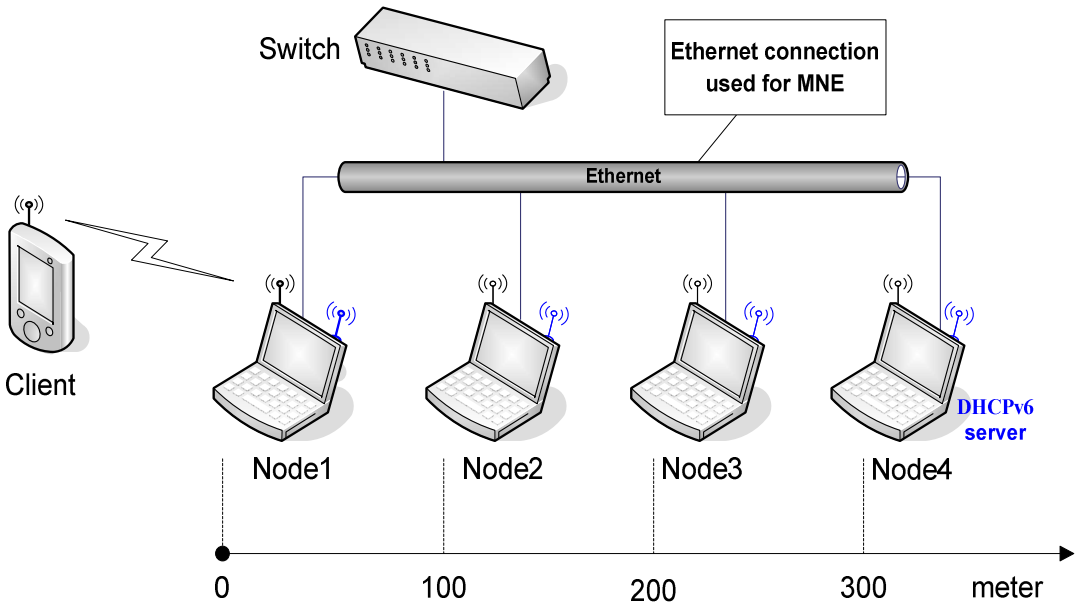

Figure 4.7 MNE Emulated Node Location

The four emulated mesh nodes are each 100 meters away in a line, as shown in Figure 4.7. IPv4 interface eth0 is the backchannel interface, and all the Ethernet interfaces are connected to a 16-port SMC switch. MNE is started with the link range set to 150 meters on all the nodes. That means node1 can only receive packets from node2, and node4 can only communicate to node3. Interfaces on the mesh nodes are assigned IP addresses shown in Table 1. All the prefix lengths are 64 bits.

NIC Function	NIC Name	Node1	Node2	Node3	Node4
AP	wlan0	fec1:100::1	fec1:200::1	fec1:300::1	fec1:400::1
OLSR	wlan1	fec0:1::1	fec0:2::1	fec0:3::1	fec0:4::1
MNE	eth0	192.168.0.1	192.168.0.2	192.168.0.3	192.168.0.4

Table 4.1 Table Mesh Nodes IP Address Configuration

In the local footprint of Node1, a Client associates with Node1 through its AP radio wlan0 (fec0:1::1). DHCPv6 server is running on Node2. When Node1 receives a DHCPv6 Solicit or other messages on wlan0 from a Client, it relays the DHCPv6 message to Node2 through unicast on wlan1. The reason of using unicast is due to the limitation of the current DHCPv6 implementation. With only one OLSR radio per mesh node, DHCPv6 messages that are relayed by a relay agent can not forward to another relay agent. However, even with this constraint, the approach is still reasonable. Since all the mesh nodes are routers, we need to configure them manually, and we also know the IP address of the DHCPv6 server.

The purpose of this test scenario is to test the functionality of OLSR, especially the HNA message propagation in a multi-hop environment. After running *crcolsrd*, we check the routing table in each node. All these diagrams of routing tables are screen shots acquired with GIMP.

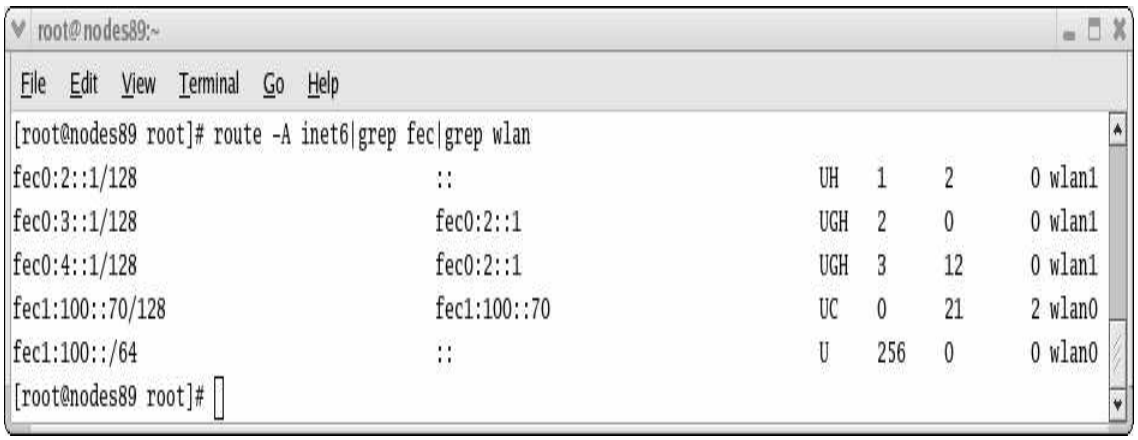

Figure 4.8 Routing Table on Node1

Node1's routing table tells us that it has an 1-hop neighbor, Node2 (fec0:2::1), an 2-hop neighbor, Node3 (fec0:3::1), and one 3-hop neighbor, Node4 (fec0:4::1). This exactly reflects the topology of the emulated network. It also shows that for Node1 to reach its 2-hop neighbor, and 3-hop neighbor, it needs to go through Node2 (fec0:2::1) as the next hop.

The network prefix, fec1:100::/64, is the subnet of the local footprint of Node1. And fec1:100::70/128 is the address of the associated client in Node1's local footprint.

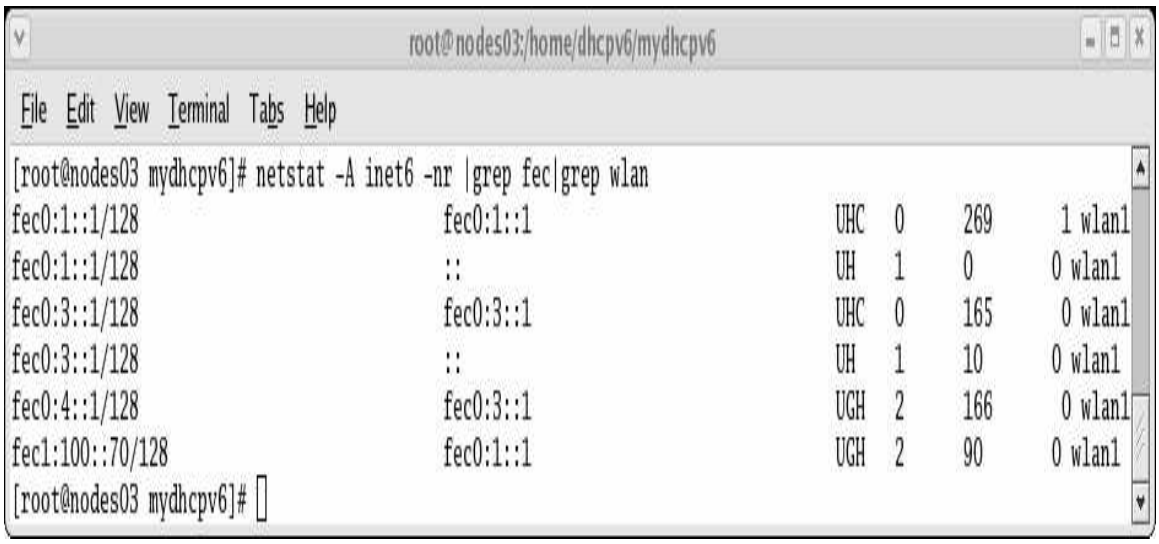

Figure 4.9 Routing Table on Node2

Figure 4.9 shows that Node2 has two 1-hop neighbors, Node1 (fec0:1::1) and Node3 (fec0:3::1). Node4 (fec0:4::1) is a 2-hop neighbor. To reach the client fec1:100::70 in Node1's local footprint, Node2 needs to go through Node1's wlan1 first. That is why it is two hops away.

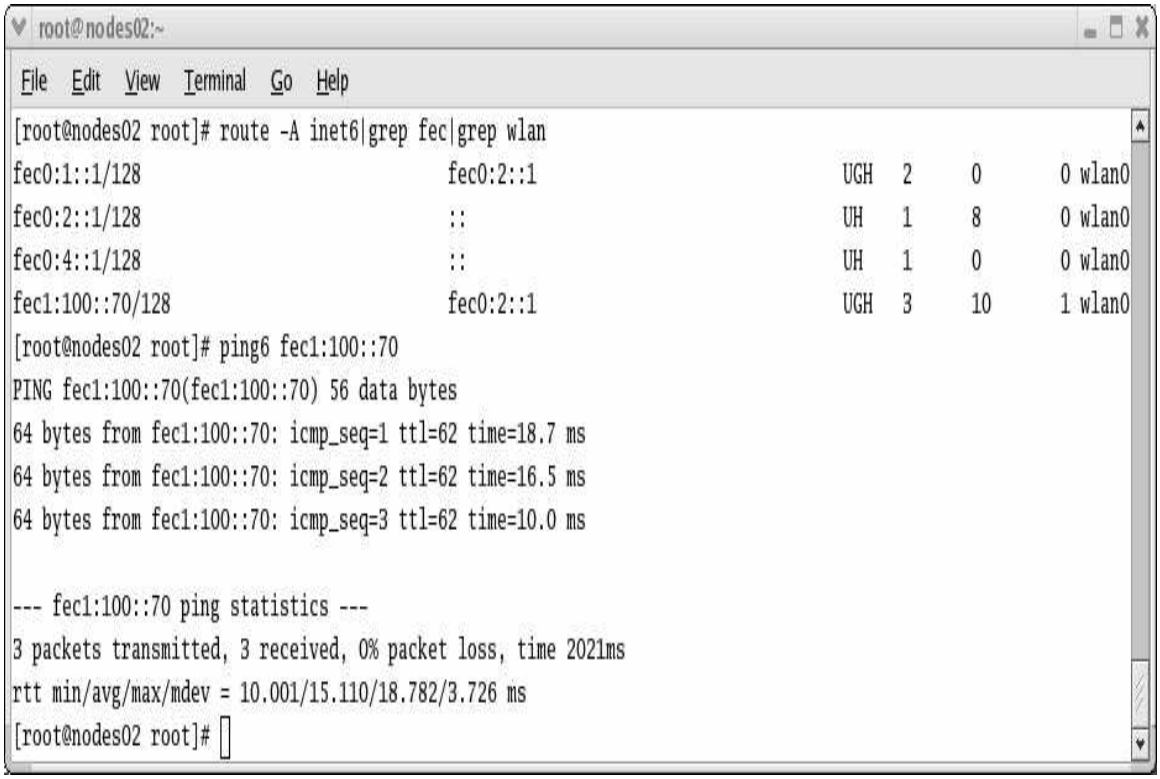

Figure 4.10 Routing Table on Node3

Node3's routing table also has an entry for Node1's client, fec1:100::70. The client is 3 hops away from Node3, because the client's AP, Node1 is 2 hops away. The output of ping6 demonstrates that the connectivity between Node3 and the client is fine.

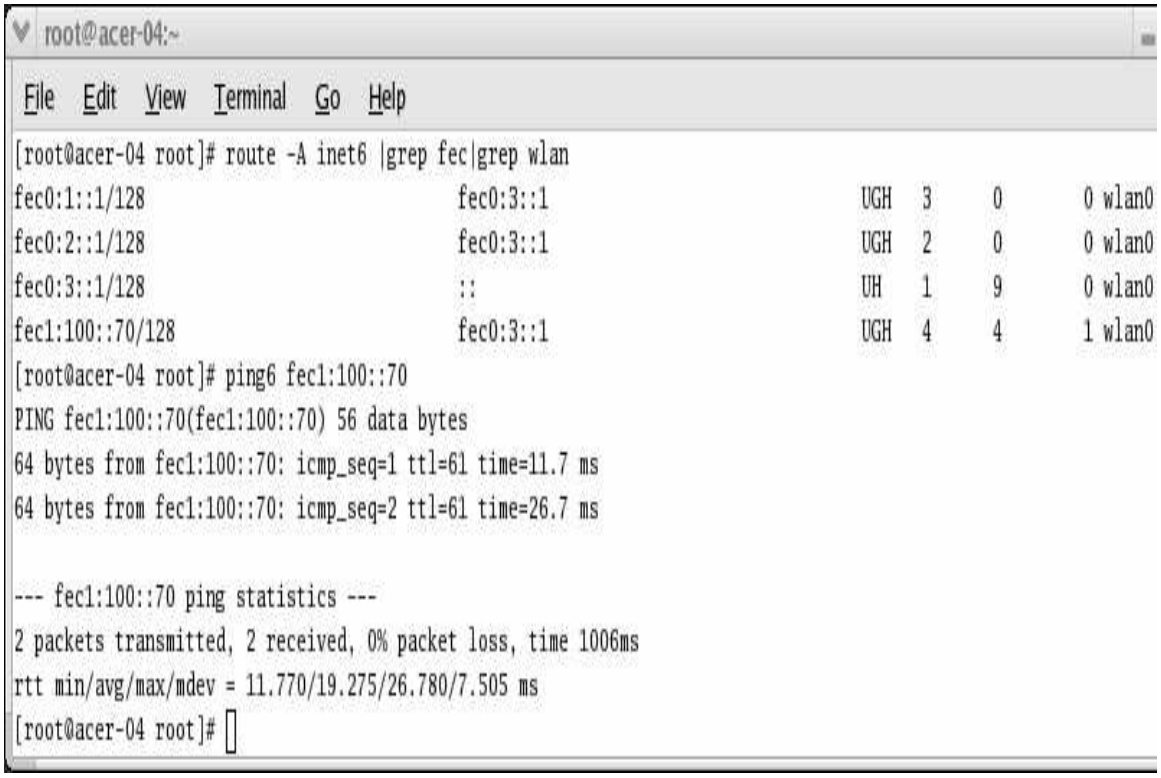

Figure 4.11 Routing Table on Node4

As shown in Figure 4.7, Node4 is the right most node in the emulated network. Again we use ping6 to test the reachability of the client, fec1:100::70, and it works well. This demonstrates that the local footprint and the mesh backhaul are integrated.

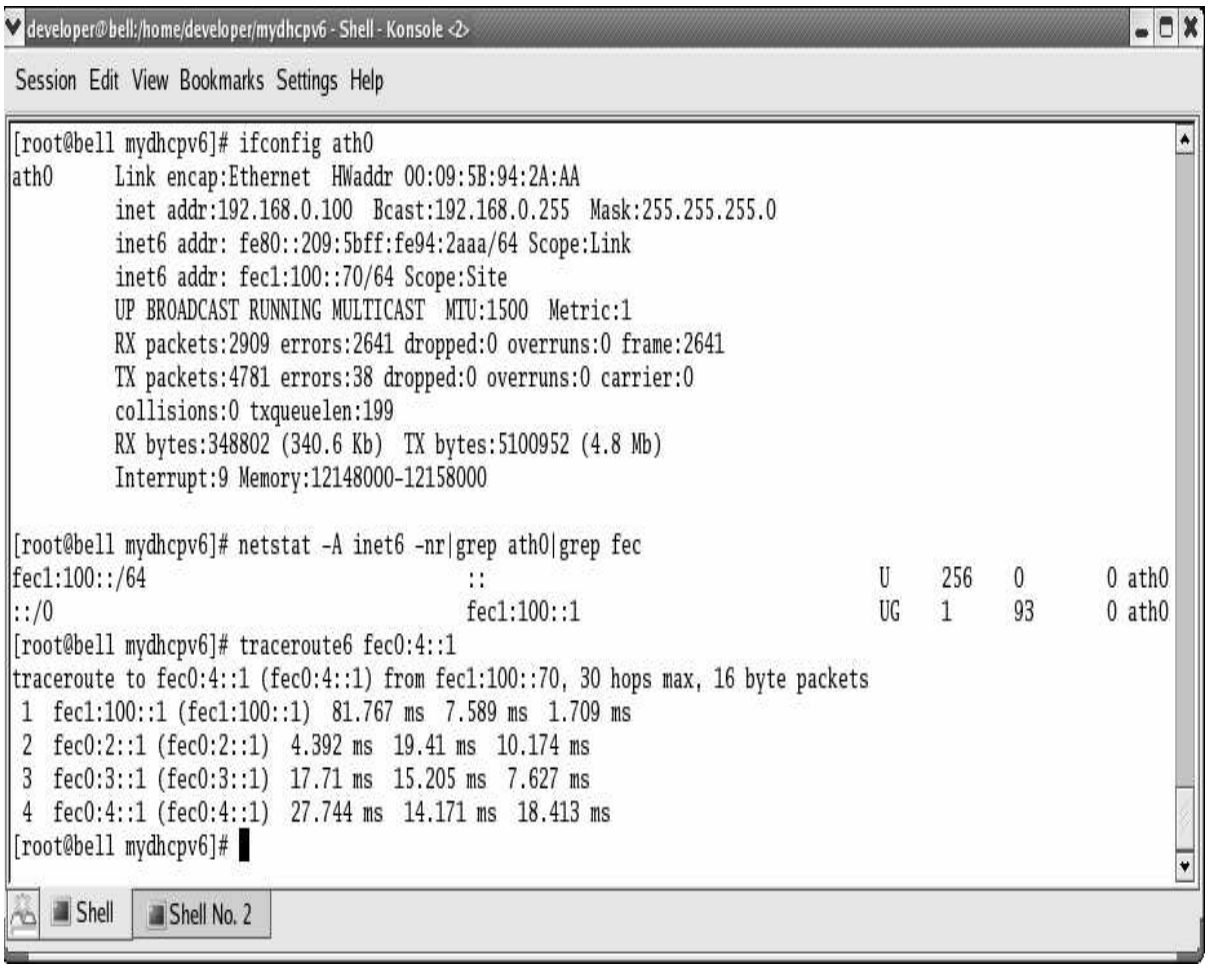

Figure 4.12 Routing Table on the Client

Figure 4.12 consists of three parts. The first is the output of command *ifconfig*, which displays the addresses the client has configured. The second displays the routing table, which tells us that its default gateway is fec1:100::1, the AP interface of Node1. The third part is the output of traceroute6. To send packets to Node4, fec0:4::1, the client follows a path as shown in Figure 4.13, which again exactly corresponds to the emulated network.

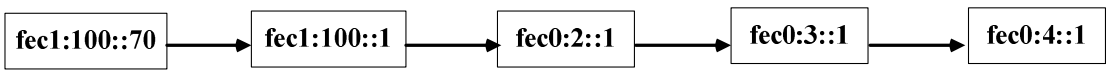

Figure 4.13 Multi-Hop Forwarding Path from the Client

4.4 Test Scenario Three: Multi-Hop TCP Throughput Test

In test case two we emulate a six-node mesh network that is a string topology. Similar to the second test case, the distance between two neighbor nodes is 100 meters. The goal of this scenario is to test the TCP throughput in a multi-hop mesh network. The emulated network is shown in Figure 4.14.

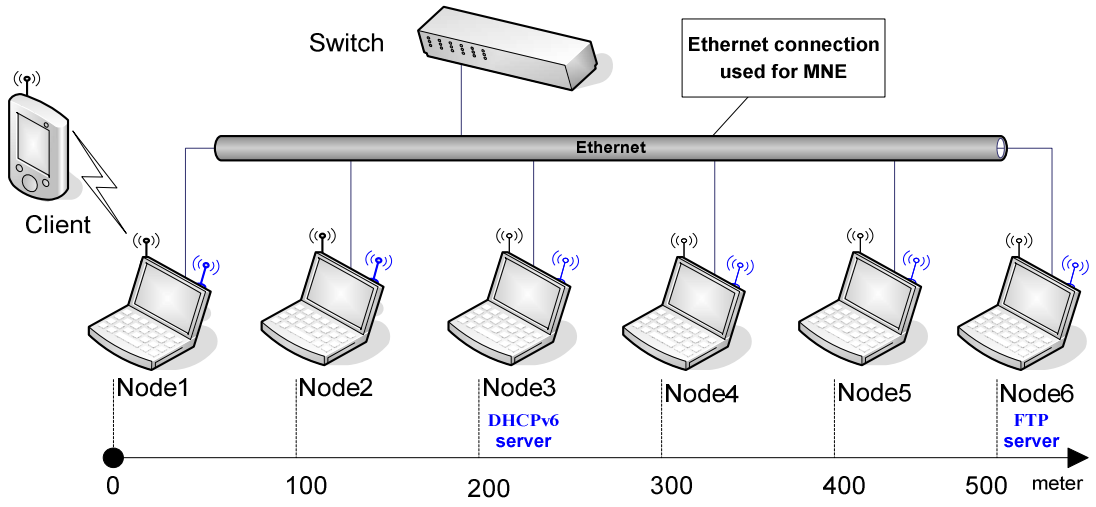

Figure 4.14 MNE Emulated 6-node Mesh Network Topology

Since our testbed is an IPv6 network, we need an IPv6 enabled application to generate the live traffic. Here we chose *vsftpd* 1.2.1 as the FTP server, and *tnftp* (formerly known as lukemftp) version 20030825 as the FTP client.

Through DHCPv6, the client acquires the IP address fec0:1::70. And a routing entry is generated automatically by the kernel for the subnet fec0:1, which is the local footprint of

Node1, the associated AP of the client. But it still can not have access to the FTP server, which is running in Node6 (fec0:3::1). The reason is that it lacks a default gateway to other networks. Unlike DHCPv4, DHCPv6 does not have a default router option. Unfortunately, a DHCPv6 client can not configure its default gateway automatically. Until recently, people in the DHCPv6 mailing list are still discussing the need to add such a default router option. The DHCPv6 working group believes that Router Advertisement in the stateless autoconfiguration should meet the need. But in a large network it is not desirable to have radvd daemon running on all the routers just to broadcast the default router information. Instead, having one or a few DHCPv6 servers to provide the information is a better approach.

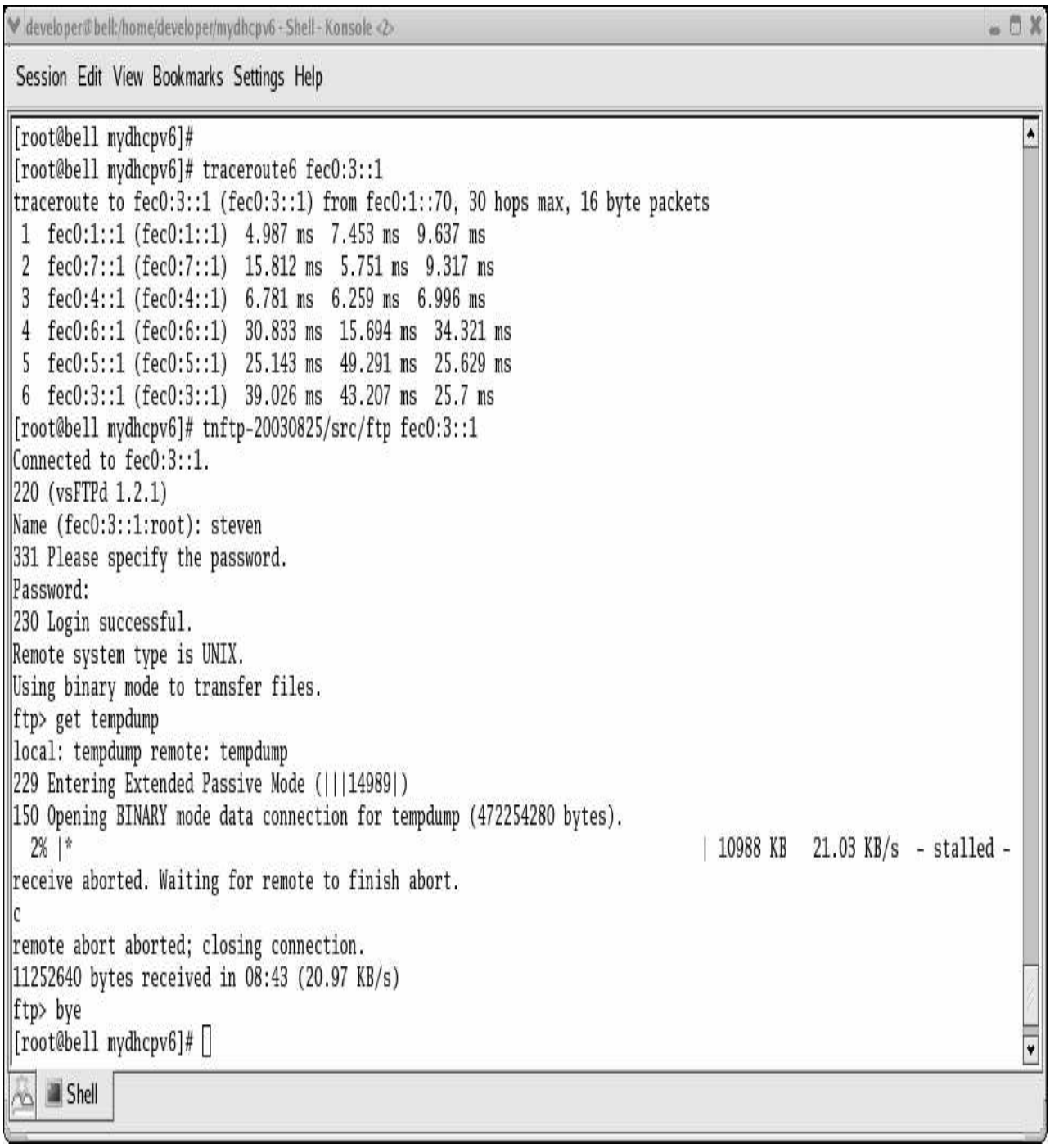

Figure 4.15 Output of Traceroute6 and FTP Connection on the client

From Figure 4.14 we can see that the FTP server is 6 hops away, and the throughput is 21.03 kilo bytes per seconds. But after about eight minutes, the FTP connection is broken.

Similarly, FTP connections from 1 hop to 6 hops were tested, and the results are shown in the following figure.

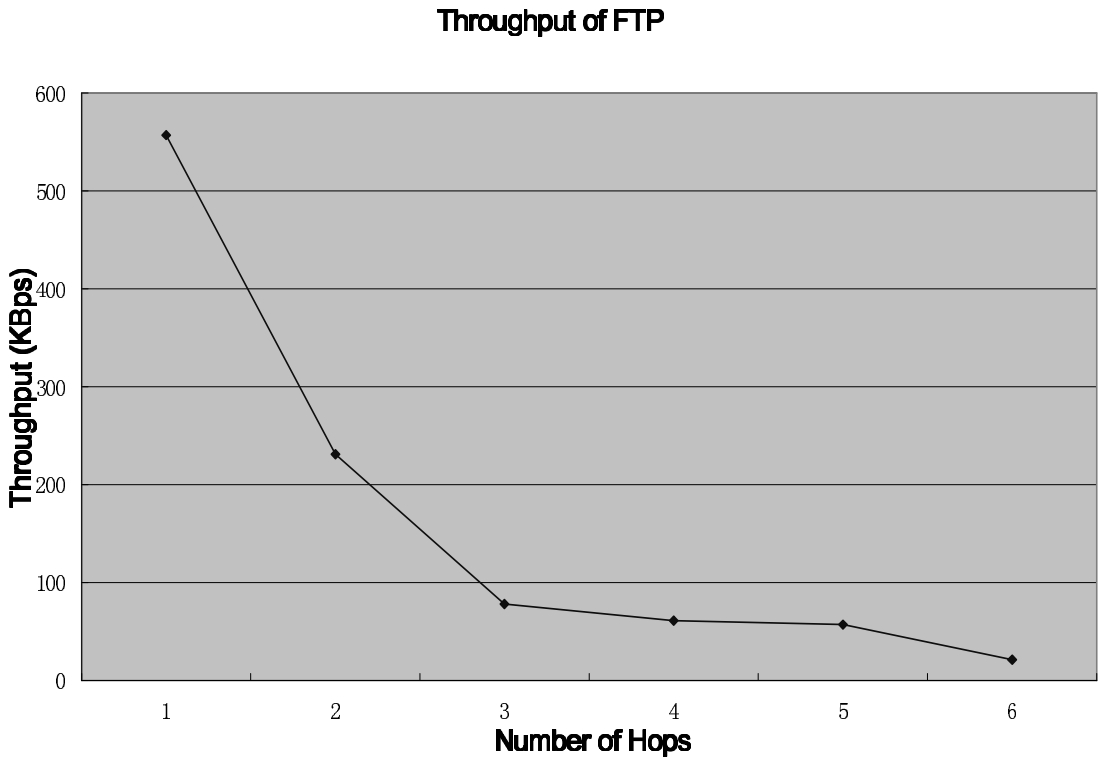

Figure 4.16 FTP Throughput vs Number of Hops.

Number of Hops	FTP Throughput
1	557
2	231
3	78
4	61
5	57
6	21

Table 4.2 FTP Throughput KBytes/s

With the increasing number of hops, the FTP throughput decreased dramatically. It can not even maintain a connection at 6-hop. The main reason is due to the IEEE 802.11 MAC contention. In our case, when Node 2 forwards packets to Node 3, Node1 can not continue transmitting packets. Because all the nodes in this test are within the transmission range, the RTS from one node will silence all the other nodes. Even though MNE blocks IP packets, it does not block IEEE 802.11 MAC layer frames. Therefore, in this emulation, a virtually 6-hop neighbor which is 500 meters away actually still contends with the first node for the access to the medium.

4.5 Summary

In both the live and emulated experiments, the modified OLSR implementation can create correct routing table entries, which ensures the mesh backbone function well. With the new HNA mechanism, local footprints also seamlessly integrate into the mesh backbone. In short, we have successfully built a fully functioned wireless mesh network. During the course of testing, we also learn some valuable lessons. As shown in the third test scenario, we notice that multi-hop wireless connections have potential performance problems. In order to build a scalable wireless mesh network, traditional (i.e. shortest path first) routing protocols need to be improved. Link qualities should be taken into account when selecting a forwarding path. Better transport layer protocols that adapted to the wireless environment may also boost the performance. Most importantly, a better MAC protocol, such as IEEE 802.16 [13], may more effectively solve the problem.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

This thesis presented the design and implementation of a wireless mesh network with mobility support. Because our motivation is to deploy such a mesh network in the real world, one of the key design decisions we made is to provide support to regular wireless clients without modifying any software or hardware. To support IP address autoconfiguration, DHCPv6 is employed. OLSR is chosen as the routing protocol due to its good performance in large and dense wireless networks.

Our implementation involves both network layer and link layer. It includes processing of the IEEE 802.11 management frame, DHCPv6 message, ICMPv6 Neighbor Discovery message, and OLSR HNA message.

To verify the functionality and evaluate the performance of our implementation, a testbed is constructed. Both live tests and emulations with MNE are conducted. In order to understand the practical issues of deployment of such a wireless network, we use real traffic instead of traffic generators even in emulated tests.

These tests indicated that our implementation achieved the expected design goals. Visiting clients can automatically acquire an IPv6 address, and corresponding routing entries are created on other mesh nodes. Since HNA messages are generated and propagated dynamically, control message overhead is reduced.

5.2 Future Work

Our implementation also has some limitations that could not be overcome due to the time limitation. If mesh nodes are configured to propagate host-specific HNA messages, the number of wireless clients that can be supported in the mesh network is limited. Another limit of the current implementation is that we only provide support to nomadic users. It would be very nice if users can roam seamlessly from one local footprint to another without the need to initiate the connection again.

5.2.1 IPv4 Client Support

Even though IPv6 is gaining more momentum recently, IPv4 will exist for several years before it is totally replaced by IPv6. So IPv4 support needs to be added. One solution is to use the Dual Stack Transition Method (DSTM) mechanism [2]. DSTM is a transition proposal that uses IPv4 over IPv6 dynamic tunnels, and allocates temporary IPv4 addresses to dual stack hosts. In our case, Gateway mesh nodes can be configured as DSTM border routers, and non-gateway mesh nodes as IPv4/IPv6 dual stack hosts. When an IPv4-only client transmits an IPv4 packet, the mesh node in that local footprint forwards the packet through the Dynamic Tunneling Interface (DTI) mechanism to the DSTM border routers. The DSTM border router is also called Tunnel End Point, which encapsulates the IPv4 packet in an IPv6 packet, and vice versa.

With the DSTM solution, only the gateway mesh nodes need permanent IPv4 addresses, and have connectivity to IPv4 networks. The mesh backbone is still an IPv6 only network. The existing addressing scheme does not need to be changed.

5.2.2 ETX Metric

Expected Transmission count (ETX) is proposed by MIT [6]. It is designed as a new metric that helps to find high-throughput paths on multi-hop wireless networks. Most of the wireless routing protocols, such as OLSR, DSR, or AODV use the minimum hop-count as the metric. But since IEEE 802.11 is a lossy medium by nature, the shortest path may not necessarily be the path that has the highest throughput.

The ETX metric takes into account more factors, such as link loss ratios, the asymmetry of the loss ratios in the two directions, and the reduction of throughput due to interference among intermediate nodes. Experiments show that routes selected based on ETX have significantly higher throughputs than the traditional minimum hop-count metric, particularly for long paths with more than two hops. To improve the performance of our mesh network, the ETX link measurement algorithm should be implemented in OLSR.

5.2.3 Reduce Routing Table Size

Host-specific HNA message is very flexible, but it also has one drawback. Mesh nodes have to create one entry for each wireless client. The result is a large routing table. One solution to this problem is to extend the DHCPv6 implementation. With the modified DHCP server, it will not randomly allocate an IP address from its address pool. Instead, it will assign addresses based on the first relay agent that has relayed the Solicit or Request message. In our case, the first relay agent of a DHCPv6 Solicit or Request message is the

mesh node that works as an AP in the local footprint. With the help of the configuration file on the DHCP server, we can ensure wireless clients in one local footprint obtain addresses in a specific range. For example, if the network prefix of a local footprint is 2001:3ffe:5005:ffff::/64, we can reserve the scope 2001:3ffe:5005:ffff:1234::/80 to the above local footprint. Then the HNA message needs not to be generated for each client. One HNA message that advertises the 2001:3ffe:5005:ffff:1234::/80 range for the whole local footprint will be enough. This is similar to assigning network prefixes to mesh nodes, but allows for more efficient allocation of address space.

5.2.4 Security

At present, we rely on the Wired Equivalent Privacy (WEP) mechanism to protect our mesh network. WEP is the encryption standard implemented in the IEEE 802.11's MAC Layer. The payload of each IEEE 802.11 frame is encrypted before transmission using the RC4 stream cipher provided by RSA Security [26]. The encryption key is the shared secret key created by the user plus a randomly generated 24-bit initialization vector (IV). Since the IV is only 24 bits, eventually the same IV will be reused. By collecting enough frames, a hacker can crack the key. To enhance the security, a new standard IEEE 802.11i was approved in June, 2004. IEEE 802.11i utilizes IEEE 802.1x for authentication and key management, and Advanced Encryption Standard (AES) as the improved encryption algorithm.

When there is an implementation of the IEEE 802.11i specification available, we will add the stronger security mechanism to our mesh network.

5.2.5 Roaming Support

Mobile IP (RFC 2002) [24] is a good solution to support wireless clients migrating between local footprints without losing connectivity or previously established sessions. However, Mobile IP has to be installed both on the mobile client side as well as on the mesh nodes. This is a contradiction to our zero-conf policy. We need to think of a better approach to implement roaming support on the server side (mesh nodes) without sacrificing the zero-conf feature.

References

[1] P. Bahl, “Opportunities and Challenges of Community Mesh Networking”, http://www.research.microsoft.com/users/bahl/Present/Bahl_keynote04.ppt/, Keynote @ MICS Workshop ETH Zurich. July, 2004.

[2] J. Bound,” Dual Stack Transition Mechanism”, IETF draft <draft-bound-dstm-exp-01.txt>, April, 2004.

[3] B. Chambers, “The Grid Roofnet: a Rooftop Ad Hoc Wireless Network”, MIT Master's Thesis, June 2002.

[4] T. Clausen, P. Jacque, “Optimized Link State Routing Protocol (OLSR)”, IETF RFC 3626, October 2003.

[5] CRColsrd, Communication Research Center(CRC), http://pf.itd.nrl.navy.mil/projects.php?name=olsr, April, 2003.

[6] D. De Couto, D. Aguayo, J. Bicket, R. Morris, “A High-Throughput Path Metric for Multi-Hop Wireless Routing”, Proceedings of the 9th ACM International Conference on Mobile Computing and Networking (MobiCom '03), San Diego, California, Pages:134-146, September 2003.

[7] R. Droms, J. Bound, B. Volz, T. Lemon, C. Perkins, M. Carney, ”Dynamic Host Configuration Protocol for IPv6 (DHCPv6)”, IETF RFC 3315,July 2003.

[8] Ethereal, http://www.ethereal.com.

[9] R. Hinden, S. Deering, “Internet Protocol, Version 6 (IPv6) specification”, RFC 2460, December 1998.

[10] R. Hinden, S. Deering, “Internet Protocol Version 6 (IPv6) Addressing Architecture”, IETF RFC 3513, April 2003.

[11] R. Hinden, M. O'Dell, S. Deering, “An IPv6 Aggregatable Global Unicast Address Format”, IETF RFC 2374, July 1998.

[12] Hostap device driver, http://hostap.epitest.fi/.

[13] IEEE 802.16, http://www.wimaxforum.org/home.

[14] IEEE Computer Society LAN MAN Standards Committee, “Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications”, ANSI/IEEE Std. 802.11, 1999 Edition.

[15] In-Stat/MDR report, “Joe Schmo Has Wi-Fi: The Wireless Home Becomes a Reality”, Report Number: IN030819RC, Publication Date: December 2003.

[16] D. Johnson, D. Maltz, Y. Hu, “The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR)”, draft-ietf-manet-dsr-10.txt, July 2004.

[17] MADWiFi device driver, http://sourceforge.net/projects/madwifi.

[18] D. Maltz, J. Broch, D. Johnson, “Experiences Designing and Building a Multi-Hop Wireless Ad Hoc Network Testbed”. CMU School of Computer Science Technical Report CMU-CS-99-116. March 1999.

[19] MeshNetworks, http://www.meshnetworks.com/.

[20] T. Narten, E. Nordmark, W. Simpson, ”Neighbor Discovery for IP Version 6 (IPv6)”, IETF RFC 2461, December 1998.

[21] R. Ogier, F. Templin, M. Lewis,” Topology Dissemination Based on Reverse-Path Forwarding (TBRPF)”, IETF RFC 3684, February 2004.

[22] C. Perkins, E. Belding-Royer, and S. Das, “Ad hoc On-Demand Distance Vector (AODV) Routing”, RFC 3561, July 2003.

[23] C. Perkins, P. Bhagwat, “Highly Dynamic Destination-Sequenced

Distance-Vector Routing (DSDV) for Mobile Computers", in Comp. Communication Rev., October 1994.

[24] C. Perkins, "IP Mobility Support", RFC 2002, October 1996.

[25] Raniwala, K. Gopalan, T. Chiueh, "Centralized Algorithms for Multi-channel Wireless Mesh Networks". ACM Mobile Computing and Communications Review (MC2R) Vol 8, No 2, April 2004.

[26] RSA Security Inc., http://www.rsasecurity.com/.

[27] SOWN (the Southampton Open Wireless Network), http://www.sown.org.uk/, Southampton, UK.

[28] W. Stevens, M. Thomas,"Advanced Sockets API for IPv6", IETF RFC 2292, February 1998.

[29] Tcpdump and libpcap, http://www.tcpdump.org.

[30] S. Thomson, T. Narten, " IPv6 Stateless Address Autoconfiguration", IETF RFC 2462, December 1998.

[31] US Department of Defense, "Internet Protocol version 6 (IPv6) Interim Transition Guidance", http://ipv6.disa.mil/docs/stenbit-ipv6-guidance-20030929.pdf, September, 2003.