Implementing AODV Ad Hoc Routing Protocol For IPv6

by

Meng Chunng Lee
B.Sc. E.E., University Of Manitoba

PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF ENGINEERING

In the School
of
Engineering Science

© Meng Chunng Lee 2003

SIMON FRASER UNIVERSITY

April 2003

All rights reserved. This work may not be
reproduced in whole or in part, by photocopy
or other means, without permission of the author.

APPROVAL

Name: **Lee, Meng Chunng**

Degree: Master of Engineering

Title of Project: Implementing AODV Ad Hoc Routing Protocol For IPv6

Examining Committee:

Chair: **Dr. Marek Syrzycki**
Professor

Dr. Steve Hardy
Senior Supervisor
Professor

Dr. Paul Ho
Supervisor
Professor

Dr. Tejinder Randhawa
Supervisor
NewMIC Scientist
New Media Innovation

Date Approved: April 11, 2003

PARTIAL COPYRIGHT LICENSE

I hereby grant to Simon Fraser University the right to lend my thesis, project or extended essay (the title of which is shown below) to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users. I further agree that permission for multiple copying of this work for scholarly purposes may be granted by me or the Dean of Graduate Studies. It is understood that copying or publication of this work for financial gain shall not be allowed without my written permission.

Title of Thesis/Project/Extended Essay

Implementing AODV Ad Hoc Routing Protocol For IPv6

Author:

(signature) [illegible]

(name) [illegible]

(date) [illegible]

ABSTRACT

Wireless networks have become increasingly popular in the past few years. It is anticipated that ad hoc networks will play an important role in the advancement of wireless networks. Among the various ad hoc routing protocols, the Ad-Hoc On-Demand Distance Vector (AODV) routing protocol is the most popular and common one. At the present time, IPv4 is the foundation of networking, as the most commonly used internet protocol standard. However, with the further growth of new wireless devices connecting to the Internet, IPv4 is proving inadequate to support this growth. The Internet Engineering Task Force (IETF) has developed IP Version 6 (IPv6) to enable a far larger number of systems to be deployed on the Internet. To ensure that the provisioning of wireless networks can keep pace with Internet growth, it is desirable to incorporate ad hoc routing protocols into IPv6. This would provide great advantages for both users and product developers.

The goal of this project is to implement an IP Version 6 (IPv6) AODV ad hoc routing protocol. We have chosen Uppsala University's AODV (AODV-UU) implementation as the baseline. In this project, we demonstrate the correctness of our IPv6 based AODV protocol by creating various network configurations in a test bed. More generally, this report provides valuable insight into the porting of routing protocols from IPv4 to IPv6.

DEDICATION

To my dad, mum and my girlfriend Bee Kim.

ACKNOWLEDGEMENTS

I know that this project would not have been possible except for several people. Firstly, I would like to thank Professor Steve Hardy for being my supervisor. Secondly, I would like to thank Dr. Tejinder S. Randhawa of NewMIC for providing valuable guidance throughout the project. I would also like to thank Antti Tuominen for answering my technical questions about AODV and IPv6 in a very responsive manner and also David Wilder for providing me with help on setting up the Linux Kernel Core Dump (LKCD) debugger. I would like to thank Warren Lee of NewMIC for assisting with setting up the laptop and helping with testing. Last but not least, I would like to thank the New Media Innovation Centre for providing resources for this project.

TABLE OF CONTENTS

Approval ii
Abstract iii
Dedication iv
Acknowledgements v
Table of Contents vi
List of Figures ix
List of Tables x
Chapter One Introduction 1
Chapter Two Introduction To Ad Hoc Networks 2
  2.1 Common Routing Protocol 2
  2.2 Ad Hoc Routing Protocol 2
    2.2.1 Reactive Routing Protocol 3
    2.2.2 Pro-Active Routing Protocol 3
Chapter Three Ad Hoc On-demand Distance Vector (AODV) Protocol 4
  3.1 AODV Operations 5
Chapter Four IP Version 6 Essentials 7
  4.1 Introduction To IPv6 7
  4.2 IPv6 Addresses 7
  4.3 IPv6 Address Format 7
    4.3.1 Leading Zeros Suppressed 8
    4.3.2 Zeros Field Collapsed 8
  4.4 IPv6 Address Types 8
  4.5 More About IPv6 8
Chapter Five IPv4 AODV Software Implementation 9
  5.1 Software Implementation 9
  5.2 Software Components In Details 10
    5.2.1 Core Program 10
    5.2.2 Kernel Module 13
    5.2.3 AODV Processing Module 14
    5.2.4 Packet Handler Module 15
    5.2.5 Route Table Module 15
    5.2.6 Miscellaneous Utilities Modules 15

5.2.7 IP Queue API Library 16
5.2.8 AODV Configuration Parameter 16
Chapter Six IPv6 AODV Implementation 17
6.1 AODV Control Messages For IPv4 And IPv6 17
6.1.1 Route Request Message (RREQ) 17
6.1.2 Route Reply Message (RREP) 18
6.1.3 Route Error Message (RERR) 19
6.1.4 Route Reply Acknowledgement (RREP-ACK) 20
6.2 New Implementation For IPv6 AODV 20
6.2.1 Local Host Initialization 20
6.2.2 Routing Table 21
6.2.3 Kernel Module 22
6.2.4 Packet Queue Library API 23
6.2.5 Socket Changes 23
6.2.6 ICMP Control Messages Handling 23
6.3 Porting Strategic 24
Chapter Seven Development Environment And Test-Bed Set Up 25
7.1 Mobile Node Set-up 25
7.2 Test Environment Set-up 25
7.3 Debugging Tools 26
Chapter Eight Test Results 27
8.1 Mobile Nodes Configuration 27
8.2 Test Scenario 1 27
8.3 Test Scenario 2 29
8.4 Test Scenario 3 33
8.5 Test Scenario 4 35
8.6 Test Scenario 5 37
Chapter Nine Conclusions 38
Appendices 39
A.1 Linux Kernel. 39
A.2 802.11b Wireless Device 40
A.3 Set up IPv6 Site Local Address 40
A.4 Running IPv6 AODV 41
A.5 Testing Utilities 41

A.6 Linux Kernel Core Dump (LKCD) Tools 41
A.7 IPv4 AODV Implementation 42
Reference List 43

LIST OF FIGURES

Figure 1 AODV Route Discovery Process 6
Figure 2 Main Loop Suede-Code 12
Figure 3 RREQ Messages for IPv4 18
Figure 4 RREQ messages for IPv6 18
Figure 5 RREP message for IPv4 18
Figure 6 RREP message for IPv6 19
Figure 7 RERR message for IPv4 19
Figure 8 RERR message for IPv6 20
Figure 9 Route Reply RREP-ACK for IPv4 and IPv6 20
Figure 10 Hash Function For IPv4 and IPv6 21
Figure 11 Test Scenario 1 Set Up 27
Figure 12 Results Before AODV Started 28
Figure 13 Results After AODV Started 29
Figure 14 Test Scenario 2 Set up 30
Figure 15 Mobil Nodes' Route Table For Test Scenario 2 31
Figure 16 Ping Results 32
Figure 17 Node A and Node C Route Table After Ping Stop 33
Figure 18 Traceroute6 Results From Node A To Node C 33
Figure 19 Test Results for Test Scenario 3 34
Figure 20 "traceroute6" Results 35
Figure 21 Test Results for Test Scenario 4 36
Figure 22 "traceroute6" Results 36
Figure 23 SSH Session From Node A To Node C 37

LIST OF TABLES

Table 1 Examples of Ad Hoc Routing Protocols 3
Table 2 Summary of IPv6 Address Type 8
Table 3 AODV-UU Software Components Summary 10
Table 4 Netfilter Hook Name for IPv4 and IPv6 22
Table 5 Socket Options Different For IPv4 And IPv6 23

CHAPTER ONE
INTRODUCTION

Wireless networks have become a favoured subject in academic research as well as in commercial product development. This is mainly because wireless mobile devices are rapidly gaining popularity due to their compact size and portability. In general, wireless networks can be classified into two categories: *Infrastructure Networks* and *Non-Infrastructure Mobile Networks.* Infrastructure Networks normally have a fixed wired gateway and mobile nodes can communicate with the network through a base station. An example application of this kind of network is a Wireless LAN. Non-Infrastructure Mobile Networks are also known as ad hoc networks. They do not use a fixed gateway for packet routing. Each mobile node acts as a router and maintains routes to other nodes in the network. This type of wireless network is very attractive for various purposes and applications such as convention meetings, electronic classrooms, search-and-rescue and related uses.

The most dramatic issue in the current IP world is the exponential growth of the number of Internet users along with growth in traffic. This implicitly indicates that the current IPv4 address space will be exhausted some time soon. Therefore, IPng or IPv6 has been introduced not only to solve this problem but also to enhance the network by including capabilities such as control of Quality-of-Services (QoS), auto configuration, security and other features.

This project is the product of the combination of these two hottest subjects in the networking world, namely ad hoc wireless networks and Ipv6. This combination is reflected in the title: *Implementing AODV Ad Hoc Routing Protocol for IPv6.*

CHAPTER TWO
INTRODUCTION TO AD HOC NETWORKS

An ad hoc wireless network is a self-maintaining network and all the mobile nodes are interconnected in an arbitrary manner. Hence, the routing in ad hoc networks differs from fixed line protocols in that optimum routing is not the most important requirement for ad hoc routing. Features like rapid route convergence and high reactivity are deemed more important.

2.1 Common Routing Protocol

There are two common routing algorithms in use on today's Internet: Link State and Distance-Vector algorithms. The main difference between these two algorithms is that routers using the Link State algorithm will flood all the nodes in the network with a small packet of information describing the state of its own links, whereas a router using the Distance-Vector algorithm sends its entire routing table to only its neighbors. The trade off between these two algorithms is that Link State algorithm converges faster and is less prone to routing loops while the Distance-Vector algorithm requires less memory and CPU power.

2.2 Ad Hoc Routing Protocol

Several routing protocols have been proposed for ad hoc mobile networks since the advent of DARPA packet routing protocols in the early 1970's. Ad hoc routing protocols can be classified into two categories: Re-Active Routing Protocols (also referred to as On-Demand Routing Protocols) and Pro-Active Routing Protocols (also refer as Table-Driven Routing Protocols).

2.2.1 Reactive Routing Protocol

Re-Active protocols create a route only when the source desires a packet to be routed. In other words, a node will initiate a route discovery process within the network when it requires a route to a destination. It will maintain the route until it is no longer required.

2.2.2 Pro-Active Routing Protocol

Pro-Active protocols will try to maintain up-to-date routing information for all nodes in the network. This is achieved through maintaining a set of routing tables. When there is a change in network topology, the node will propagate updated information throughout the network to maintain a consistent view.

The following table shows the existing ad hoc routing protocols and their category:

Re-Active	Pro-Active
Ad Hoc On-Demand Distance Vector (AODV)	Destination-Sequenced Distance-Vector (DSDV)
Dynamic Source Routing (DSR)	Optimized Link State Routing (OLSR)
Lightweight Mobile Routing (LMR)	Fisheye State Routing (FSR)
Temporally Ordered Routing Algorithm (TORA)	Topology Broadcast Based on Reverse-Path Forwarding (TBRPF)

Table 1 Examples of Ad Hoc Routing Protocols

A very well written paper called "A Review of Current Routing Protocols for Ad Hoc Mobile Wireless Networks" [Ref. 9], give a very good description of several routing schemes and also presents a comparison between the reactive and pro-active routing protocols, highlighting their features, differences and characteristics.

CHAPTER THREE
AD HOC ON-DEMAND DISTANCE VECTOR (AODV) PROTOCOL

Ad hoc On-Demand Distance Vector is one of the most popular ad hoc routing protocols. It is basically the combination of Dynamic Source Routing (DSR) and Destination-Sequenced Distance Vector (DSDV). The major strengths of AODV are:

- quick adaptation to dynamic link conditions,
- low CPU consumption and memory overhead,
- low network utilization,
- routing loop free by using destination sequence numbers.

Destination Sequenced Distance Vector (DSDV) protocol is based on the well-known Bellman-Ford routing algorithm but improved through the avoidance of routing loops in a mobile network of routers. Each node maintains a routing table which contains routes to all possible nodes. A sequence numbering system is used to allow a node to distinguish stale routes from new ones. Since routing table updates are sent periodically throughout the network, a considerable amount of control traffic is generated in the network.

AODV is based on a DSDV implementation but reduces the control traffic in the network by creating routes on an on-demand basis, instead of maintaining a complete list of routes. It is a pure on-demand route acquisition system. The specification of AODV is documented as an Internet-Draft submitted by the Mobile Ad Hoc Networking Working Group of the Internet Engineering Task Force (IETF) [Ref. 2]. The next section gives a brief description of how AODV works in general.

3.1 AODV Operations

There are 3 message types defined for AODV: Route Request (RREQ), Route Reply (RREP) and Route Error (RERR). These message types are communicated through a UDP connection and normal IP header processing is still required. When a source node wants to communicate with a destination node and it does not have a valid route, it will generate a RREQ message to its neighbours in control manner (Figure 1 a). Its neighbours will forward the request to their neighbours until the request reaches a node that has a route to the destination. Each node that forwards the route request creates a reverse route for itself back to the source node.

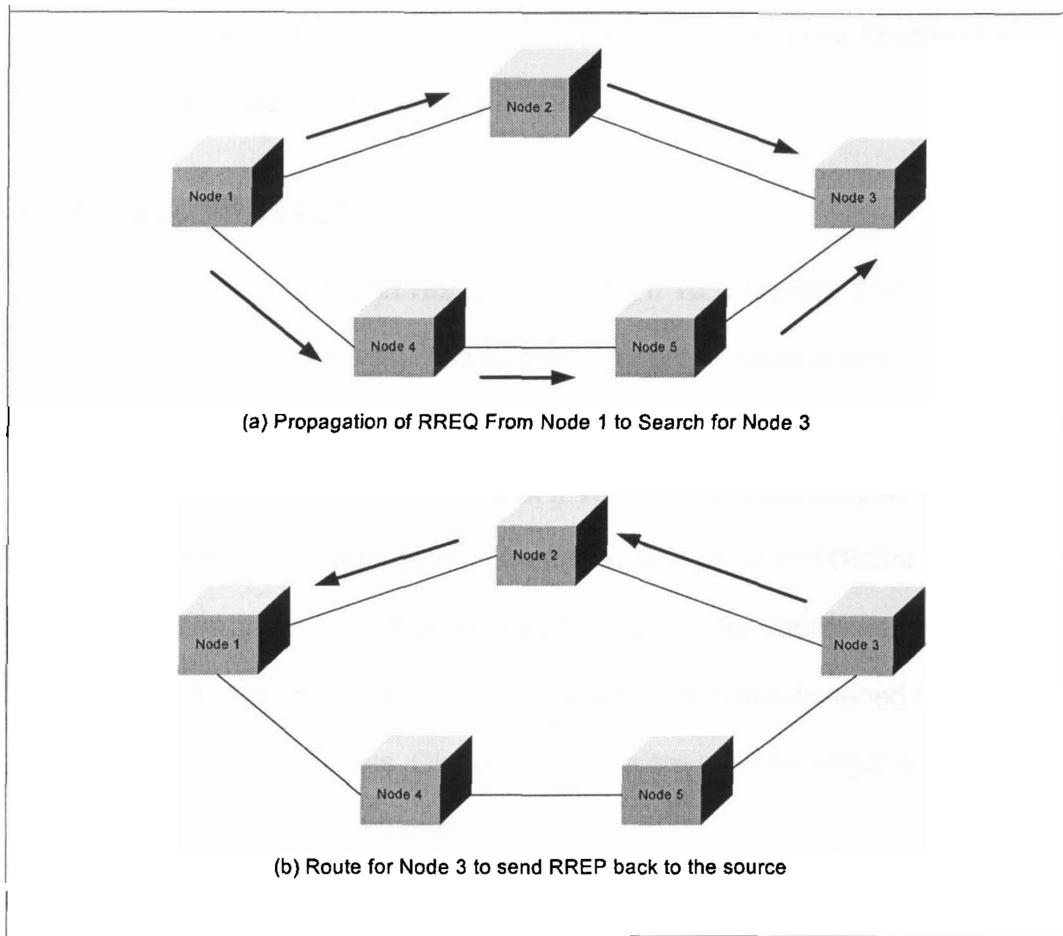
When the RREQ packet reaches a node with a route to the destination node, it generates a RREP (Figure 1 b) which contains the number of hops to reach the destination and the sequence number for the destination most recently seen by the node generating RREP. AODV uses destination sequence numbers to ensure that all routes are loop free and contain the most recent route information. Each node maintains its own sequence number and broadcast ID. The broadcast ID is incremented for every RREQ the node initiates.

Each of the nodes, that forward the reply back to the source, creates a forward route to the destination. The state created in each node is hop-by-hop along the path from the source to the destination. In other words, each node only remembers the next hop and not the whole path. If the source later receives a RREP that contains a greater or equal sequence number, but with a smaller hop count, it will update its route information for that destination and use the new route. A route will be deleted if it was inactive for a specified lifetime.

If a source moves, it will regenerate the route discovery protocol to find the route to the destination. If a node along the route moves, its upstream neighbours

notice the move and propagate a link failure notification message, a RREP with an infinite metric, to its upstream neighbours. The node which receives this message will also propagate the message to its upstream neighbours and continue the propagation until the source node is reached.

AODV periodically transmits "hello" messages (a RREQ with hop count equal to 1) to maintain a route to its neighbours. If a node misses three consecutive "hello" messages from a neighbour, it considers the link to the neighbour to be down.



(a) Propagation of RREQ From Node 1 to Search for Node 3

(b) Route for Node 3 to send RREP back to the source

Figure 1 AODV Route Discovery Process

CHAPTER FOUR
IP VERSION 6 ESSENTIALS

This chapter gives a basic introduction to IPv6 and provides a reference point for more IPv6 information.

4.1 Introduction To IPv6

The work for IP Version 6 (IPv6) standardization began in 1991 and the main part was completed in 1996. The specification of IPv6 is documented in RFC 2460. IPv6 is a successor to IPv4 with improvements to address space, security, Quality of Service (QoS), autoconfiguration and related areas.

4.2 IPv6 Addresses

One of the main objectives of introducing IPv6 to replace IPv4 is to supply a IP address space to last "forever". The address space has been increased from 32 to 128 bits. With 128 bits, there are approximately 10^{38} addresses. One of the major differences between IPv4 and IPv6 addresses is that IPv6 addresses are scoped. There are 3 different scopes for IPv6 addresses: Link Local, Site Local and Global address. Link Local addresses are designed to be used on each link for address autoconfiguration and for neighbour discovery functions. Site Local addresses are designed to replace IPv4 addresses for use in Intranets. Global addresses are used to uniquely identify a node that is connected to a IPv6 Internet.

4.3 IPv6 Address Format

IPv6 addresses are 128-bits long and are represented by eight fields of up to four hexadecimal digits. Each field is separated by a colon ":". For example: "FEC0:1234:0000:0000:0001:0002:0003:0004".

4.3.1 Leading Zeros Suppressed

The leading zeros of each field can be suppressed. The previous example can be written as "FEC0:1234:0:0:1:2:3:4".

4.3.2 Zeros Field Collapsed

If the IPv6 address has contiguous fields that contain only zeros, the field can be collapsed and simplified as "::". The previous example can be written as "FEC0:1234::1:2:3:4".

4.4 IPv6 Address Types

Since IPv6 introduced the concept of address scope, the address types can be identified from the leading bits of the IPv6 addresses. Table 2 gives a summary of IPv6 address types with examples.

Types	Prefix (Binary)	Prefix (Hex)	Examples
Link Local	1111 1110 10	FE80	FE80::260:1DFF:FEF0:F1AE
Site Local	1111 1110 11	FEC0	FEC0:1234::2222:1
Multicast	1111 1111	FF	FF02::1

Table 2 Summary of IPv6 Address Type

4.5 More About IPv6

Since fully explaining IPv6 is not within the scope of this report, readers can find more information about IPv6 from the following books: "Implementing IPv6: Supporting the Next Generation Internet Protocols" [Ref. 4] and "Big Book of IPv6 Addressing RFCs" [Ref. 5].

CHAPTER FIVE
IPV4 AODV SOFTWARE IMPLEMENTATION

The objective of this project is to implement IPv6 functionalities on an existing IPv4 AODV implementation. There are three recommended IPv4 AODV implementations: UCSB Implementation, NIST Implementation and Uppsala University Implementation. Links to this implementation can be found in the IPv4 AODV section in the Appendix A of this report.

This project is using Uppsala University AODV Implementation (AODV-UU) as the based to add in IPv6 functionalities. This chapter will describe the software architecture and implementation of AODV-UU.

5.1 Software Implementation

AODV-UU is implemented based on AODV IETF draft [Ref.. 2] version 10 and version 11. At the time of this project started, only version 10 is available, and correspond to AODV-UU Version 0.5. AODV-UU Version 0.6 is released early 2003 and is based on AODV IETF draft version 11. This project will based on AODV-UU Version 0.5 and will merge the AODV-UU Version 0.6 changes in the future. The pros of AODV-UU implementation is: it is very stable and very well tested. The cons is: there is no design document and not much comments in the code.

AODV-UU is designed to run on Linux's user space. The processing of AODV-UU is also relies on two kernel modules. AODV-UU required a Netfilter module "ip_queue.o" to run in kernel space to capture IP packets. Information on how to build this module can be found in the reference section of this report. AODV-UU implementation can be discussed in six different components: Core program, AODV messages processing modules, kernel modules, packet processing modules, routing

table modules and miscellaneous utility. The main functionalities of each component can be summarized in Table 3.

Component	Functionality
Core Program	program initialization, process user specified parameter when AODV start-up, attached call-back function and fall in a main loop to listen to the attached socket
Kernel Module	responsible for accepting the packet, or queue the packet to user space or forwarding the packet
AODV Processing Module	responsible for the generating AODV messages and processing the received AODV messages
Packet Handling Module	responsible for processing the packet that pass from the kernel module to the user space, it will decide whether to forward the packet based on the AODV routing table or queue the packet if no route can be found for the destination and generate a request for route discovery
Routing Table Module	responsible for maintaining a user space routing table as well as controlling the kernel routing table
Misc. Utilities	Provide useful function for general use, for example debug log, timer queue etc.

Table 3 AODV-UU Software Components Summary

Detail of each of these components will be explained in next section.

5.2 Software Components In Details

5.2.1 Core Program

The core program contains 1 file: “main.c”. This is the core of the AODV user space program. The main functionality of the core program are listed as follow:

- process user input parameter
- detect from the terminal and fork a new process (if applicable)
- initialize various components
- enter a main loop

This section will further discuss the critical components initialization including Local Host Initialization, Packet Input Initialization and AODV Socket Initialization. It will

also discuss the mechanism used for the AODV main loop with the "Call-Back" function implementation.

For host initialization, this is a critical area to be make for ADOV to support multiple interfaces. The main function of host initialization is to retrieve network interface information for AODV use, for example, interface address, netmask, interface index and so on. It will also load the "ip6_queue" and "kaodv" kernel modules (will be discuss in next section) and also turn on kernel IP forwarding option. Due to IPv6 introduce address scope concept, the host initialization code cannot be reuse directly from IPv4 to IPv6, therefore AODV for IPv6 will only support only one interface for current release.

Packet input initialization will create and initialize an IPQ handler is using IPQ standard API. This is actually opening a Netlink socket for the user space program to retrieve IP packets that captured by the kernel module. The initialization process will then call the attach_callback_func to attach the socket file descriptor and the associated processing function "packet_input()" defined in "packet_input.c".

AODV socket initialization will create a UDP socket and bind to port 654. It will add the host address to multicast/broadcast group for neighbor discovering purpose. It also set the IP_PKTINFO and IP_RECVTTL socket option so that when it receives an IP packet, it can retrieve the TTL (or HOPLIMIT for IPv6) information and the payload. The AODV message is actually store in the payload of the packet.

The "Call-Back Function" implementation for AODV-UU is using the "select() function" for I/O multiplexing. The "callbacks" data structure, "attach_callback_func" and the main loop suede-code are shown in Figure 2.

```
#define CALLBACK_FUNCS 4
static struct callback {
    int fd;
    callback_func_t func;
} callbacks[CALLBACK_FUNCS];

static int nr_callbacks = 0;

int attach_callback_func(int fd, callback_func_t func)
{
    if (nr_callbacks >= CALLBACK_FUNCS) {
      fprintf(stderr, "callback attach limit reached!!\n");
      exit(-1);
    }
    callbacks[nr_callbacks].fd = fd;
    callbacks[nr_callbacks].func = func;
    nr_callbacks++;
    return 0;
}

main()
{
    fd_set rfds, readers;
    int nfds = 0;

    /* Set sockets to watch... */
    FD_ZERO(&readers);
    for (i = 0; i < nr_callbacks; i++)
      {
      FD_SET(callbacks[i].fd, &readers);
      if (callbacks[i].fd >= nfds)
        nfds = callbacks[i].fd + 1;
      }

    while (1)
      {
      memcpy((char *) &rfds, (char *) &readers, sizeof(rfds));

      if ((n = select(nfds, &rfds, NULL, NULL, timeout)) < 0)
        continue;

      if (n > 0)
        {
          for (i = 0; i < nr_callbacks; i++)
            {
            if (FD_ISSET(callbacks[i].fd, &rfds))
              (*callbacks[i].func) (callbacks[i].fd);
            }
        }
      }
}
```

Figure 2 Main Loop Suede-Code

The following show the flow of the main loop:

- use FD_ZERO() to initialize the set
- use FD_SET() to add the socket to be monitor
- use "select function" to poll for processing
- the return value of select is the total count of the number of descriptors that are ready.
- use FD_ISSET to determine which socket is ready
- call the processing function accordingly.

5.2.2 Kernel Module

Besides the Netfilter kernel module "ip_queue.o", AODV also has its own kernel module. It is "kaodv.c". The main functionality of "kaodv.c" is to register the following netfilter hooks: NF_IP6_PRE_ROUTING, NF_IP6_LOCAL_OUT, NF_IP6_POST_ROUTING to nf_aodv_hook(). When Netfilter module receive an IP packet, nf_aodv_hook function will be called. It will check if this is AODV control messages. If the receive packet is AODV message, it will accept the packet and let go to AODV program to handle it. If it is not AODV message, it will check the hook value. If the hook value is NF_IP6_PRE_ROUTING or NF_IP6_LOCAL_OUT, it will put this in the queue and let the user program to pick this up. If the hook is NF_IP6_POST_ROUTING, it will reroute all packets before sending on interface. This will make sure queued packets are routed on a newly installed route (after a successful RREQ-cycle).

5.2.3 AODV Processing Module

AODV messages processing contains the following files: “seek_list.c” , “aodv_socket.c”, “aodv_rreq.c”, “aodv_rrep.c”, “aodv_rerr.c”, “aodv_hello.c”, “aodv_timeout.c”.

“seek_list.c” is used to keep a “precursor list” which contains the IP address for each of its neighbor that are likely to use it as a next hop towards each destination.

“aodv_socket.c” is responsible for sending and receiving AODV control messages and call the corresponding AODV messages handler to process the message.

“aodv_rreq.c” is responsible for the main functionality of AODV RREQ operation such as creating RREQ message, process received RREQ message, and RREQ route discovery.

“aodv_rrep.c” is responsible for the main functionality of AODV RREP operation such as creating RREP message, process received RREP message, create RREP-ACK as well as process received RREP-ACK.

“aodv_rerr.c” is responsible for the main functionality of AODV RERR operation such as creating RERR message and processing received RERR message.

“aodv_timeout.c” is responsible for the timeout of the following function: Route Delete, Route Discovery, Route Expire, Hello Timeout, RREQ Record, RREQ Blacklist, RREP-ACK and Wait On Reboot.

“aodv_hello.c” is responsible for sending Hello message and process received Hello message.

5.2.4 Packet Handler Module

There are 2 files associated with AODV packet handler: "packet_input.c" and "packet_queue.c".

"packet_input.c" is mainly responsible for processing IP packets that passed from the kernel space module. The socket communication is set up during packet handler initialization as described in the packet handler initialization part. It will decide whether the packet should be queue while waiting for route discovery process or drop the packet if no route can be found.

"packet_queue.c" is responsible for handling the queued packet.

5.2.5 Route Table Module

There are two files associated with AODV routing table handler: "k_route.c" and "routing_table.c". "routing_table.c" is serving the routing entry modification request that come from the AODV operation. It maintains a separate routing table in user space for AODV use, which is similar to the kernel routing table except it store some other information that specified to AODV like destination sequence number, last lifetime and so on. After updating the user space routing table, it will call the functions defined in "k_route.c" to add or delete route entry to or from the kernel routing table.

5.2.6 Miscellaneous Utilities Modules

The following files served as a utilities add on for AODV implementation: "timer_queue.c", "debug.c", "icmp6.c", "print_route6.c" and "ipv6_utils.c".

"timer_queue.c" provide general timer function the AODV operation such as sending hello message periodically and so on.

"debug.c" provide a logging mechanism for AODV to log debug message, general information and also the route table changes history.

"icmp6.c" provide a function that responsible for sending an ICMP message to tell the source that it request is not reachable.

"print_route6.c" is new for IPv6 for printing the kernel routing table. This is mainly use for debugging purpose. The implemenation is based on the net-tools route show implementation, which is inspecting the /proc/net/ipv6_route system file.

"ipv6_utils.c" provide a few IPv6 utilities for AODV such as getting host interface information, converting IPv6 address to string for display, and also a function to print IPv6 address.

5.2.7 IP Queue API Library

IP Queue Library contain one file: "libipq.c". This is standard IP Queue Library for user program to build with for the IP Queue standard interfaces.

5.2.8 AODV Configuration Parameter

AODV operation parameters are specified in "params.h" and served as the user configurable AODV file. It has the value of the parameters such as "Hello Interval", "Active Route Timeout", "Net Diameter" and so on. Since this is a header file, this parameter will be hard coded and build into the AODV object file. One way to improve this in the future is, make a separate configuration AODV configuration file and AODV program will read these parameters from configuration file. This will definitely save user time if he/she want to reconfigure the AODV operation parameters.

CHAPTER SIX
IPV6 AODV IMPLEMENTATION

This chapter will talk about the IPv6 implementation of AODV on top of AODV-UU. The IPv6 socket API changes and new features are documented in two IETF documents: RFC 2553 – Basic socket interface extension for IPv6 and RFC 2292 – Advanced sockets API for IPv6. There are also two good IPv6 porting guide provided by Sun Microsystems and HP. Links to these two tutorials can be found in the reference section. AODV specification for IPv6 is documented in "AODV for IPv6 Internet-Draft" [Ref. 3].

6.1 AODV Control Messages For IPv4 And IPv6

The operation of AODV for IPv6 are similar with AODV for IPv4. The main difference is the AODV control messages. AODV operations are quite straightforward and rely on 4 different control message types: Route Request (RREQ), Route Reply (RREP), Route Error (RERR) and Route Reply Acknowledge (RREP-ACK). The detail explanation of the AODV control messages can be found in the AODV IETF draft [Ref. 2]. This section will highlight the differences between IPv4 and IPv6.

6.1.1 Route Request Message (RREQ)

The main different of RREQ message for IPv4 and IPv6 is the type value. The type value for IPv4 is 1 and for IPv6 is 16. Besides that, instead of 32-bit source and destination, it replaced by 128-bit IPv6 address, and the fields order are rearranged for IPv6 to enable alignment along 128-bit boundaries.

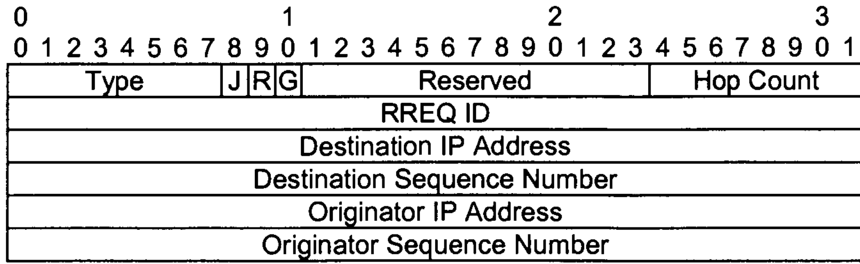


Figure 3 RREQ Messages for IPv4

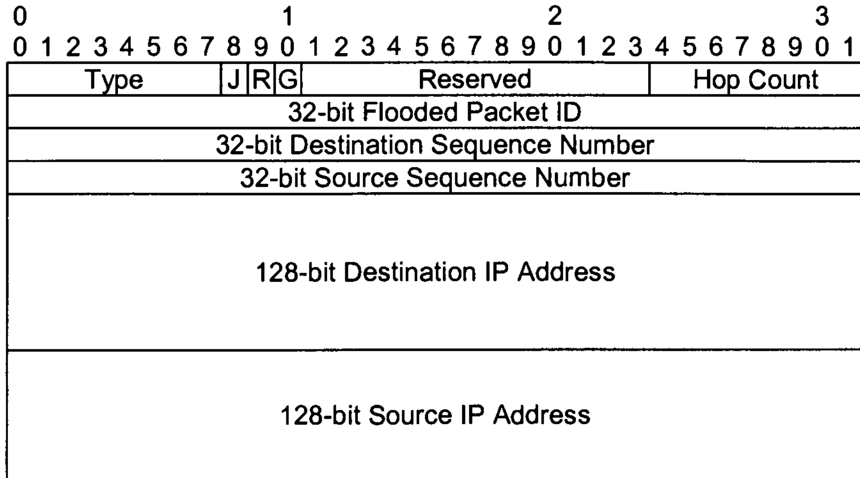


Figure 4 RREQ messages for IPv6

6.1.2 Route Reply Message (RREP)

The main different of RREP message for IPv4 and IPv6 is the type value. The value for IPv4 is 2 and for IPv6 is 17. There is 5 bits dedicated for prefix size in IPv4 and 8 bits for IPv6. Besides that, instead of 32-bit source and destination, it replaced by 128-bit IPv6 address, and the fields order are rearranged for IPv6 to enable alignment along 128-bit boundaries.

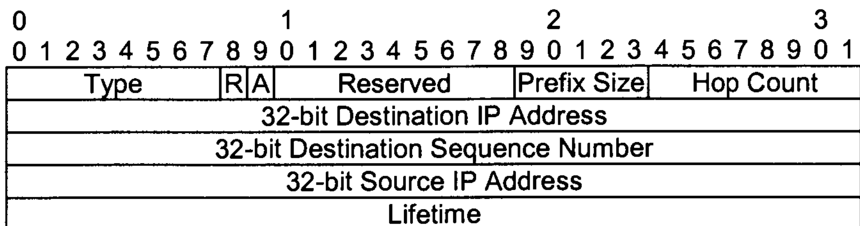


Figure 5 RREP message for IPv4

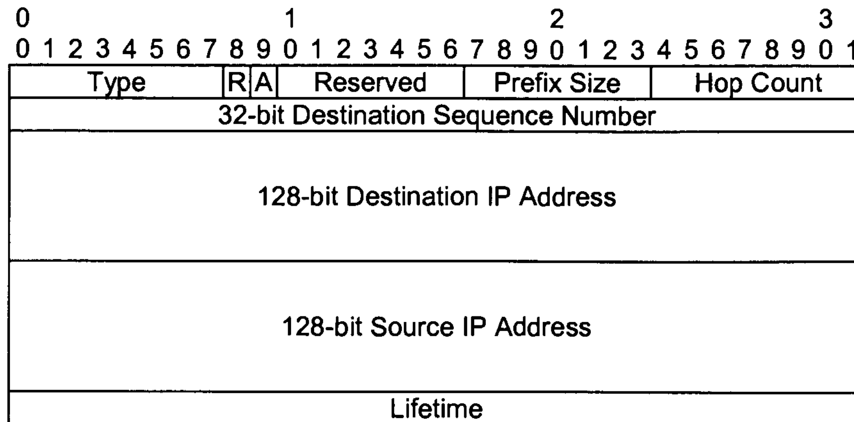


Figure 6 RREP message for IPv6

6.1.3 Route Error Message (RERR)

The main different of RERR message for IPv4 and IPv6 is the type value. The type value for IPv4 is 3 and for IPv6 is not yet determined (based on AODV for IPv6 IETF draft [Ref. 3]). For current implementation, value 18 is used and will be updated once the value is finalized. Besides that, instead of 32-bit source and destination, it replaced by 128-bit IPv6 address, and the fields order are rearranged for IPv6 to enable alignment along 128-bit boundaries.

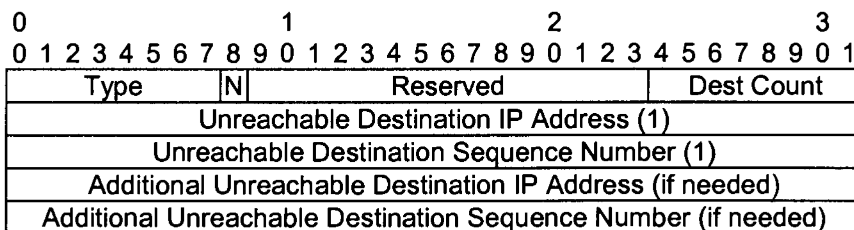


Figure 7 RERR message for IPv4

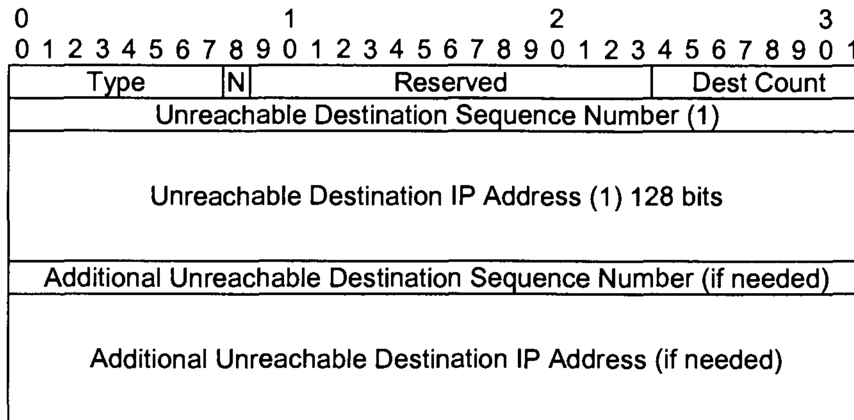


Figure 8 RERR message for IPv6

6.1.4 Route Reply Acknowledgement (RREP-ACK)

The main different of RREP message for IPv4 and IPv6 is the type value. The type value for IPv4 is 4 and for IPv6 is 19.

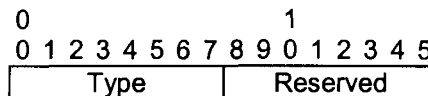


Figure 9 Route Reply RREP-ACK for IPv4 and IPv6

6.2 New Implementation For IPv6 AODV

6.2.1 Local Host Initialization

For IPv4, the interface information such as IP address, interface index, netmask & etc can be retrieve using ioctl() system call. Unfortunately, Linux's ioctl does not support the following actions: SIOCGIFCONF, SIOGIFADDR and SIOCGIFFLAGS. Using Netlink socket is one of the solution, and reading the network interface information from the /proc/net/if_inet6 system file is the easier and most efficient method.

Local host initialization also responsible to enable IP packet forwarding for the node. For IPv4, the IP packet forwarding option can be turn on by writing "1" to the following Linux system file: "/proc/sys/net/ipv4/ip_forward". For IPv6, this file is located at "/proc/sys/net/ipv6/conf/all/forwarding".

6.2.2 Routing Table

The Hashing function for routing table lookup should be different for IPv4 and IPv6. Since AODV-UU is using hash function for routing entry lookup for IPv4, a different hash function is needed for IPv6. I reuse an IPv6 hashing function that found in /usr/src/linux/include/net/addrconf.h which is one of the hashing algorithm for IPv6. The code is as shown as follow:

```
/* Original hash function for IPv4 AODV */
unsigned int hashing(u_int32_t * addr, hash_value * hash)
{
    *hash = *addr;
    return (*hash & RT_TABLEMASK);
}

/* hashing function from /usr/src/linux/include/net/addrconf.h */
static __inline__ u8 ipv6_addr_hash(struct in6_addr *addr)
{
    __u32 word;

    word = addr->s6_addr[2] ^ addr->s6_addr32[3];
    word  ^= (word>>16);
    word ^= (word >> 8);

    return ((word ^ (word >> 4)) & 0x0f);
}

/* New IPv6 hashing function for IPv6 AODV */
unsigned int hashing(struct in6_addr * addr, hash_value * hash)
{

    *hash = addr->s6_addr[2] ^ addr->s6_addr32[3];
    *hash ^= (*hash >> 16);
    *hash ^= (*hash >> 8);

    return(( (*hash) ^ (*hash >> 4) ) & RT_TABLEMASK);
}
```

Figure 10 Hash Function For IPv4 and IPv6

RT_TABLEMASK is defined as RT_TABLESIZE – 1. RT_TABLESIZE is currently defined as 64.

There is another major changes make for routing table module. k_del_rte is a internal function called by routing_table module to delete a kernel route. The original prototype of k_del_rte function is as follow:

int k_del_rte(u_int32_t dst, u_int32_t gw, u_int32_t nm);

Since k_del_rte() function will call ioctl to delete a kernel route, it always pass it 0 for interface index and route metric. This causing a problem in IPv6 in the following situation: When the kernel route table is actively being using, for example ping, then call ioctl to delete the route with metric 0, ioctl will return no error but the route does not get deleted. The metric that correspond to the route that store in the kernel route table need to be identified and store in the in6_rtmsg structure before calling ioctl. Since AODV does keep track of the metric for each route, it will pass in this parameter when calling k_del_rte() function. The new k_del_rte prototype is look like follow:

int k_del_rte(struct in6_addr dst, struct in6_addr gw, u_int16_t plen, unsigned int ifindex, unsigned int metric);

6.2.3 Kernel Module

kaodv6.c is functional similar to kaodv.c where kaodv6.c is for IPv6. Since this is a kernel loadable module, a new file and object is created for IPv6 so that kaodv.o and kaodv6.o can be loaded to the kernel at the same time. For kaodv6 module, the main difference is the netfilter hook names. The following are the names difference for IPv4 and IPv6:

IPv4	IPv6
NF_IP_PRE_ROUTING	NF_IP6_PRE_ROUTING
NF_IP_LOCAL_OUT	NF_IP6_LOCAL_OUT
NF_IP_POST_ROUTING	NF_IP6_POST_ROUTING

Table 4 Netfilter Hook Name for IPv4 and IPv6

6.2.4 Packet Queue Library API

libipq.c and libipq.h has the standard IP packet queue interfaces between the user and the kernel space. The original code is using Version 1.x for libipq.c and version 1.x For libipq.h. These two files do not support IPv6 features. The latest version of these two files can be downloaded form the Internet that has IPv6 support. Version 1.7 for libipq.c and version 1.6 for libipq.h are used for AODV for IPv6 implementation.

6.2.5 Socket Changes

AODV-UU is using sendto and recvfrom function for AODV messages communication. On the receiver on of the AODV node, it use recvmsg to retrieve the IP control messages such as TTL information and then using recvfrom to retrieve the IP payload, which is the AODV message. This does not work for IPv6 when the transmit side is sending the message using sendto, the IP control message cannot be retrieve from the receiver end. Therefore, for the IPv6 implementation of AODV, the AODV control message is send and receive using sendmsg and recvmsg.

The socket options for IPv4 and IPv6 are also different. The following table shown those are used in AODV socket:

IPv4		IPv6	
Level	Option	Level	Option
SOL_IP	IP_PKTINFO	IPPROTO_IPV6	IPV6_PKTINFO
	IP_RECVTTL		IPV6_HOPLIMIT

Table 5 Socket Options Different For IPv4 And IPv6

6.2.6 ICMP Control Messages Handling

In IPv6, there are 2 new ICMP message types: ND_NEIGHBOR_SOLICIT and ND_NEIGHBOR_ADVERT and this is basically replaced IPv4 ARP to resolve MAC addresses from IP address. The ND_NEIGHBOR_SOLICIT message will be sent when the host need to find a new destination that it does not have a record. This is part of the

neighbor discovery mechanism that build in the Linux kernel for IPv6. The host will multicast the message to the following address: 0xFF02::1:FFxx:xxxx, when 0xff02::1:ff/104 is the standard prefix and xx:xxxx will be replaced by the last 24 bits of the destination host address. For example, if a host try to ping the following address: 0xFFC0:1234::2222. The multicast address will become0xFF02::1:FF00:2222. When the host 0xFFC0:1234::2222 receive the ND_NEIGHBOR_SOLICIT, it will reply with ND_NEIGHBOR_ADVERT. This causing a problem with the AODV since it receive a packet and the destination is 0xFF02::1:FF00:2222, AODV first check this address is its own address and then check against its own routing table. Since this entry does not exist in its routing table and will not able to find a route to 0xFF02::1:FF00:2222, the ND_NEIGHBOR_SOLICIT message will never be processed. The solution to this problem is, packet_input() who handle in coming packet should let the ND_NEIGHBOR_SOLICIT and ND_NEIGHBOR_ADVERT message go through instead using AODV to process the packet.

6.3 Porting Strategic

Since AODV-UU is build using more than 20 files, including the header files, the approach that I used is, port one single file at a time. I first construct a dummy main file which will include the new header for the new file ported over. I also defined a dummy header file that defined the functions that needed for the new file but yet ported. Then I update the Makefile to compile the whole program with new ported file. I continue with the same process until all the files are ported over and remove the dummy header file and the dummy main file with the AODV main file.

CHAPTER SEVEN
DEVELOPMENT ENVIRONMENT AND TEST-BED SET UP

This chapter will talk about general setup for development and testing environment for this project. It also covers the debugging tools used in this project.

7.1 Mobile Node Set-up

The system requirement for a Mobile Node are stated as follow:

- Linux Kernel Version 2.4.18-3 (RedHat 7.3) and Up
- 802.11b Wireless Device running in Ad Hoc Mode
- Linux Kernel source must be installed in the proper place
- Must have Netfilter module: ip6_queue.o build as a kernel loadable module
- Must turn on IPv6 support for the Kernel
- Must configure a site local address to the interface
- Must be able to compile and run IPv6 AODV code
- Optional utilities for AODV testing: nc6 and mtvp

Refer to Appendix A for detail set up procedures for each of the components.

7.2 Test Environment Set-up

AODV does not restrict to run only with wireless interface. AODV is running on top of Network Layer, which mean it will also work with Ethernet interface. During the early development phase, a laptop and a desktop communicating with Ethernet interface is used running as infrastructure mode for IPv4. The final testing environment are using 3 laptops, which are running Linux Kernel Version 2.4.19 and 2.4.20, with 802.11b

wireless device that running in ad hoc mode. Each of the mobile nodes must set to have the same site local prefix. Net-tools utility is the major tools that use for testing, for instant, ping6, traceroute6 and route commands. Source node can use ping6 to figure out whether it can reach a destination that is not in its routing table. User can use traceroute6 to figure out the path of a packet travel. User can also use route to show the current IPv6 kernel route table. We also use nc6 (netcat6) to stream data through IPv6 connection. We set up a server node to send a MPEG file and run client on another node to stream the MPEG file to a MPEG player (Please refer to Appendix A for more information). We also try to run Secure Shell using IPv6 to remotely login to a host that is running AODV for IPv6.

7.3 Debugging Tools

There are two major debugging tools that were used for this project. One is the Ethereal to monitor packet that send or receive on a host. Ethereal version 0.9.97 does support for AODV for IPv4 and IPv6. Since AODV also has kernel module, and unfortunately this module does cause a kernel panic, a kernel debugging tool called Linux Kernel Core Dump (LKCD) is used to solve this problem (more information can be found in Appendix A).

CHAPTER EIGHT
TEST RESULTS

This chapter will talk about the test result that prove that the AODV for IPv6 is working as stated.

8.1 Mobile Nodes Configuration

Setup 3 laptops that meet the system requirements as stated in previous chapter. Configure each individual laptop to have an IPv6 site local address. The site local network prefix must be the same site local prefix. In our test environment, we are using 0xFEC0:1234::/64 as site local prefix and Node A, B and C are using local site address 0xFEC0:1234::1, 0xFEC0:1234::2 and 0xFEC0:1234::3 respectively.

8.2 Test Scenario 1

This test is to show the routing entries of each Mobile Node before and after IPv6 AODV start up where 3 Mobile Node can see each other (as shown in Figure 11).

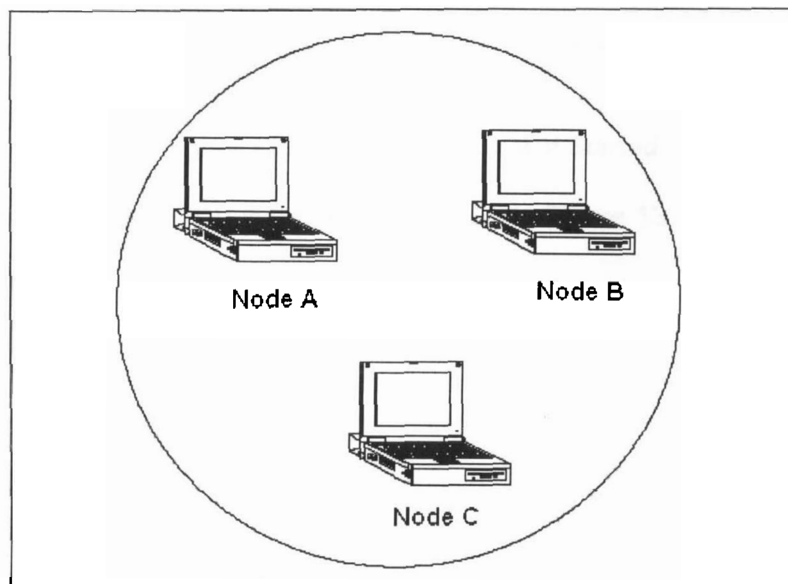


Figure 11 Test Scenario 1 Set Up

Use “route –A inet6” command to show the kernel route table of each mobile node. The results before AODV is started are shown as Figure 12.

Node A Route Table Before AODV Start

```
[root@localhost dev1]# route -A inet6
Kernel IPv6 routing table
Destination                     Next Hop          Flags Metric Ref    Use Iface
::1/128                         ::                U     0      0        0 lo
fe80::260:1dff:fef0:e56a/128    ::                U     0      0        0 lo
fe80::/10                       ::                UA    256    0        0 eth1
fec0:1234::1/128                ::                U     0      0        0 lo
fec0:1234::/64                  ::                UA    256    0        0 eth1
ff00::/8                        ::                UA    256    0        0 eth1
```

Node B Route Table Before AODV Start

```
[root@localhost root]# route -A inet6
Kernel IPv6 routing table
Destination                     Next Hop          Flags Metric Ref    Use Iface
::1/128                         ::                U     0      0        0 lo
fe80::202:a5ff:fe6f:293/128     ::                U     0      0        0 lo
fe80::/10                       ::                UA    256    0        0 eth1
fec0:1234::2/128                ::                U     0      0        0 lo
fec0:1234::/64                  ::                UA    256    0        0 eth1
ff00::/8                        ::                UA    256    0        0 eth1
```

Node C Route Table Before AODV Start

```
[root@localhost root]# route -A inet6
Kernel IPv6 routing table
Destination                     Next Hop          Flags Metric Ref    Use Iface
::1/128                         ::                U     0      0        0 lo
fe80::260:1dff:fef0:f1ae/128    ::                U     0      0        0 lo
fe80::/10                       ::                UA    256    0        0 eth1
fec0:1234::3/128                ::                U     0      0        0 lo
fec0:1234::/64                  ::                UA    256    0        0 eth1
ff00::/8                        ::                UA    256    0        0 eth1
```

Figure 12 Results Before AODV Started

The results after AODV is started are shown in Figure 13.

Node A Route Table After IPv6 AODV Started

```
[root@localhost root]# route -A inet6
Kernel IPv6 routing table
Destination                    Next Hop          Flags Metric Ref    Use Iface
::1/128                        ::                U     0      0        0 lo
fe80::260:1dff:fef0:e56a/128   ::                U     0      0        0 lo
fe80::/10                      ::                UA    256    0        0 eth1
fec0:1234::1/128               ::                U     0      0        0 lo
fec0:1234::2/128               ::                UH    2      0        0 eth1
fec0:1234::3/128               ::                UH    2      0        0 eth1
fec0:1234::/64                 ::                UA    256    0        0 eth1
ff05::11/128                   ff05::11          UAC   0      527      1 eth1
ff00::/8                       ::                UA    256    0        0 eth1
```

Node B Route Table After IPv6 AODV Started

```
[root@localhost root]# route -A inet6
Kernel IPv6 routing table
Destination                    Next Hop          Flags Metric Ref    Use Iface
::1/128                        ::                U     0      0        0 lo
fe80::202:a5ff:fe6f:293/128    ::                U     0      0        0 lo
fe80::/10                      ::                UA    256    0        0 eth1
fec0:1234::1/128               ::                UH    2      0        0 eth1
fec0:1234::2/128               ::                U     0      0        0 lo
fec0:1234::3/128               ::                UH    2      0        0 eth1
fec0:1234::/64                 ::                UA    256    0        0 eth1
ff05::11/128                   ff05::11          UAC   0      368      1 eth1
ff00::/8                       ::                UA    256    0        0 eth1
```

Node C Route Table After IPv6 AODV Started

```
[root@localhost root]# route -A inet6
Kernel IPv6 routing table
Destination                    Next Hop          Flags Metric Ref    Use Iface
::1/128                        ::                U     0      0        0 lo
fe80::260:1dff:fef0:f1ae/128   ::                U     0      0        0 lo
fe80::/10                      ::                UA    256    0        0 eth1
fec0:1234::1/128               ::                UH    2      0        0 eth1
fec0:1234::2/128               ::                UH    2      0        0 eth1
fec0:1234::3/128               ::                U     0      0        0 lo
fec0:1234::/64                 ::                UA    256    0        0 eth1
ff05::11/128                   ff05::11          UAC   0      24       0 eth1
ff00::/8                       ::                UA    256    0        0 eth1
```

Figure 13 Results After AODV Started

8.3 Test Scenario 2

Continue the set up from Test Scenario1, move the laptops such that Node A can only see Node B and Node B can see both Node A and C. Node C can only Node B as shown in Figure 14.

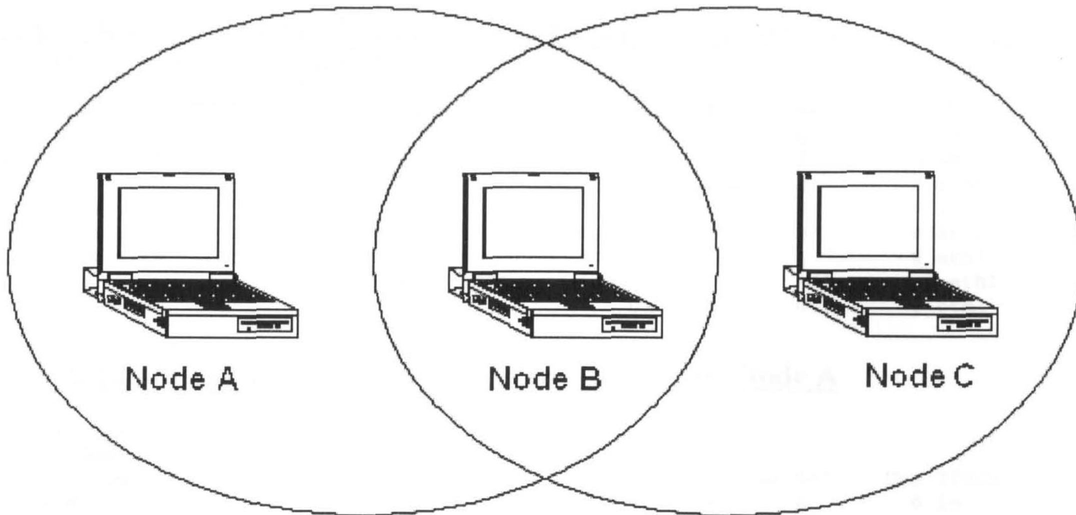


Figure 14 Test Scenario 2 Set up

Display the route table for each mobile node and the results are shown in Figure 15.

Use “ping6” command on Mobile Node A to ping Node C. The results of the ping and the route table for Node A and Node C is shown in Figure 16.

As the route table shown in Figure 16, Node A create a new route to Node C via Node B. Similarly, Node C also create a new route to Node A via Node B. The route table for Node A and Node C are shown in Figure 17 after ping is stop for a while.

As shown in Figure 17, the route to Node C in Node A is deleted. Similarly, the route to Node A in Node C also deleted. Use “traceroute6” command to trace the packet travel from Node A to Node C and the results is shown in Figure 18.

Node A Route Table After Node C Moved Away From Node A

```
[root@localhost root]# route -A inet6
Kernel Ipv6 routing table
Destination                      Next Hop          Flags Metric Ref    Use Iface
::1/128                          ::                U     0      0        0 lo
fe80::260:1dff:fef0:e56a/128     ::                U     0      0        0 lo
fe80::/10                        ::                UA    256    0        0 eth1
fec0:1234::1/128                 ::                U     0      0        0 lo
fec0:1234::2/128                 ::                UH    2      0        0 eth1
fec0:1234::/64                   ::                UA    256    0        0 eth1
ff05::11/128                     ff05::11          UAC   0      807      1 eth1
ff00::/8                         ::                UA    256    0        0 eth1
```

Node B Route Table After Node C Moved Away From Node A

```
[root@localhost root]# route -A inet6
Kernel Ipv6 routing table
Destination                      Next Hop          Flags Metric Ref    Use Iface
::1/128                          ::                U     0      0        0 lo
fe80::202:a5ff:fe6f:293/128      ::                U     0      0        0 lo
fe80::/10                        ::                UA    256    0        0 eth1
fec0:1234::1/128                 ::                UH    2      0        0 eth1
fec0:1234::2/128                 ::                U     0      0        0 lo
fec0:1234::3/128                 ::                UH    2      0        0 eth1
fec0:1234::/64                   ::                UA    256    0        0 eth1
ff05::11/128                     ff05::11          UAC   0      968      1 eth1
ff00::/8                         ::                UA    256    0        0 eth1
```

Node C Route Table After Node C Moved Away From Node A

```
[root@localhost root]# route -A inet6
Kernel Ipv6 routing table
Destination                      Next Hop          Flags Metric Ref    Use Iface
::1/128                          ::                U     0      0        0 lo
fe80::260:1dff:fef0:f1ae/128     ::                U     0      0        0 lo
fe80::/10                        ::                UA    256    0        0 eth1
fec0:1234::2/128                 ::                UH    2      0        0 eth1
fec0:1234::3/128                 ::                U     0      0        0 lo
fec0:1234::/64                   ::                UA    256    0        0 eth1
ff05::11/128                     ff05::11          UAC   0      675      1 eth1
ff00::/8                         ::                UA    256    0        0 eth1
```

Figure 15 Mobil Nodes' Route Table For Test Scenario 2

The Results Of Node A Ping Node C

```
[root@localhost root]# ping6 fec0:1234::3
PING fec0:1234::3(fec0:1234::3) from fec0:1234::1 : 56 data bytes
64 bytes from fec0:1234::3: icmp_seq=1 ttl=63 time=18.5 ms
64 bytes from fec0:1234::3: icmp_seq=2 ttl=63 time=1.29 ms
64 bytes from fec0:1234::3: icmp_seq=3 ttl=63 time=7.40 ms
64 bytes from fec0:1234::3: icmp_seq=4 ttl=63 time=1.19 ms
64 bytes from fec0:1234::3: icmp_seq=5 ttl=63 time=2.41 ms
64 bytes from fec0:1234::3: icmp_seq=6 ttl=63 time=6.00 ms
64 bytes from fec0:1234::3: icmp_seq=7 ttl=63 time=1.07 ms
64 bytes from fec0:1234::3: icmp_seq=8 ttl=63 time=1.07 ms
64 bytes from fec0:1234::3: icmp_seq=9 ttl=63 time=1.11 ms
64 bytes from fec0:1234::3: icmp_seq=10 ttl=63 time=11.5 ms
64 bytes from fec0:1234::3: icmp_seq=11 ttl=63 time=4.04 ms
64 bytes from fec0:1234::3: icmp_seq=12 ttl=63 time=1.27 ms
64 bytes from fec0:1234::3: icmp_seq=13 ttl=63 time=1.11 ms
64 bytes from fec0:1234::3: icmp_seq=14 ttl=63 time=1.15 ms
64 bytes from fec0:1234::3: icmp_seq=15 ttl=63 time=1.45 ms
64 bytes from fec0:1234::3: icmp_seq=16 ttl=63 time=10.1 ms
64 bytes from fec0:1234::3: icmp_seq=17 ttl=63 time=1.12 ms
64 bytes from fec0:1234::3: icmp_seq=18 ttl=63 time=1.12 ms
64 bytes from fec0:1234::3: icmp_seq=19 ttl=63 time=6.45 ms
64 bytes from fec0:1234::3: icmp_seq=20 ttl=63 time=5.86 ms

--- fec0:1234::3 ping statistics ---
20 packets transmitted, 20 received, 0% loss, time 19196ms
rtt min/avg/max/mdev = 1.070/4.270/18.528/4.581 ms
```

Node A Route Table When Ping In Progress

```
[root@localhost root]# route -A inet6
Kernel IPv6 routing table
Destination                      Next Hop          Flags Metric Ref    Use Iface
::1/128                          ::                U     0      0        0 lo
fe80::260:1dff:fef0:e56a/128     ::                U     0      0        0 lo
fe80::/10                        ::                UA    256    0        0 eth1
fec0:1234::1/128                 ::                U     0      34       0 lo
fec0:1234::2/128                 ::                UH    2      1        0 eth1
fec0:1234::3/128                 fec0:1234::2      UGH   3      17       1 eth1
fec0:1234::/64                   ::                UA    256    0        0 eth1
ff02::1:ff00:1/128               ff02::1:ff00:1    UAC   0      1        0 eth1
ff05::11/128                     ff05::11          UAC   0      959      1 eth1
ff00::/8                         ::                UA    256    0        0 eth1
```

Node C Route Table When Ping In Progress

```
[root@localhost root]# route -A inet6
Kernel IPv6 routing table
Destination                      Next Hop          Flags Metric Ref    Use Iface
::1/128                          ::                U     0      0        0 lo
fe80::260:1dff:fef0:f1ae/128     ::                U     0      2        0 lo
fe80::/10                        ::                UA    256    0        0 eth1
fec0:1234::1/128                 fec0:1234::2      UGH   3      24       1 eth1
fec0:1234::2/128                 ::                UH    2      2        0 eth1
fec0:1234::3/128                 ::                U     0      55       0 lo
fec0:1234::/64                   ::                UA    256    0        0 eth1
ff02::1:ff00:3/128               ff02::1:ff00:3    UAC   0      1        0 eth1
ff05::11/128                     ff05::11          UAC   0      926       1 eth1
ff00::/8                         ::                UA    256    0        0 eth1
```

Figure 16 Ping Results

Node A Route Table After Ping Stop For A While

```
[root@localhost root]# route -A inet6
Kernel IPv6 routing table
Destination                      Next Hop          Flags Metric Ref     Use Iface
::1/128                          ::                U     0      0         0 lo
fe80::260:1dff:fef0:e56a/128     ::                U     0      0         0 lo
fe80::/10                        ::                UA    256    0         0 eth1
fec0:1234::1/128                 ::                U     0      42        0 lo
fec0:1234::2/128                 ::                UH    2      1         0 eth1
fec0:1234::/64                   ::                UA    256    0         0 eth1
ff05::11/128                     ff05::11          UAC   0      974        1 eth1
ff00::/8                         ::                UA    256    0         0 eth1
```

Node C Route Table After Ping Stop For A While

```
[root@localhost root]# route -A inet6
Kernel IPv6 routing table
Destination                      Next Hop          Flags Metric Ref     Use Iface
::1/128                          ::                U     0      0         0 lo
fe80::260:1dff:fef0:f1ae/128     ::                U     0      2         0 lo
fe80::/10                        ::                UA    256    0         0 eth1
fec0:1234::2/128                 ::                UH    2      2         0 eth1
fec0:1234::3/128                 ::                U     0      120        0 lo
fec0:1234::/64                   ::                UA    256    0         0 eth1
ff05::11/128                     ff05::11          UAC   0      1016        1 eth1
ff00::/8                         ::                UA    256    0         0 eth1
```

Figure 17 Node A and Node C Route Table After Ping Stop

Traceroute6 Results From Node A To Node C

```
[root@localhost root]# traceroute6 fec0:1234::3
traceroute to fec0:1234::3 (fec0:1234::3) from fec0:1234::1, 30 hops max, 16 byte packets
 1  fec0:1234::2 (fec0:1234::2)  7.067 ms  1.287 ms  0.907 ms
 2  fec0:1234::3 (fec0:1234::3)  5.468 ms  1.145 ms  8.355 ms
```

Figure 18 Traceroute6 Results From Node A To Node C

8.4 Test Scenario 3

Continue with the Test Scenario 2 setup and have Node A to continue to ping Node C. Move Node A close to Node C. Ping continue to work and the time for ping will be shorter and the route table for Node A also updated with the new route entry where Node A can now see Node C directly. The results are shown in Figure 19.

Ping Results From Node A To Node C

```
[root@localhost root]# ping6 fec0:1234::3
PING fec0:1234::3(fec0:1234::3) from fec0:1234::1 : 56 data bytes
64 bytes from fec0:1234::3: icmp_seq=1 ttl=63 time=16.8 ms
64 bytes from fec0:1234::3: icmp_seq=2 ttl=63 time=1.34 ms
64 bytes from fec0:1234::3: icmp_seq=3 ttl=63 time=1.24 ms
64 bytes from fec0:1234::3: icmp_seq=4 ttl=63 time=1.20 ms
64 bytes from fec0:1234::3: icmp_seq=5 ttl=63 time=1.17 ms
.
.
.
```

Node A Route Table When Node A Pinging Node C

```
[root@localhost root]# route -A inet6
Kernel Ipv6 routing table
Destination                     Next Hop          Flags Metric Ref     Use Iface
::1/128                         ::                U     0      0         0 lo
fe80::260:1dff:fef0:e56a/128    ::                U     0      0         0 lo
fe80::/10                       ::                UA    256    0         0 eth1
fec0:1234::1/128                ::                U     0      207        0 lo
fec0:1234::2/128                ::                UH    2      3         0 eth1
fec0:1234::3/128                fec0:1234::2      UGH   3      5         1 eth1
fec0:1234::/64                  ::                UA    256    0         0 eth1
ff05::11/128                    ff05::11          UAC   0      1321        1 eth1
ff00::/8                        ::                UA    256    0         0 eth1
```

Ping Results From Node A To Node C When Node A Come Close To Node C

```
[root@localhost root]# ping6 fec0:1234::3
PING fec0:1234::3(fec0:1234::3) from fec0:1234::1 : 56 data bytes
64 bytes from fec0:1234::3: icmp_seq=1 ttl=63 time=16.8 ms
64 bytes from fec0:1234::3: icmp_seq=2 ttl=63 time=1.34 ms
64 bytes from fec0:1234::3: icmp_seq=3 ttl=63 time=1.24 ms
64 bytes from fec0:1234::3: icmp_seq=4 ttl=63 time=1.20 ms
64 bytes from fec0:1234::3: icmp_seq=5 ttl=63 time=1.17 ms
64 bytes from fec0:1234::3: icmp_seq=12 ttl=64 time=1001 ms   ← route change
64 bytes from fec0:1234::3: icmp_seq=13 ttl=64 time=9.56 ms
64 bytes from fec0:1234::3: icmp_seq=14 ttl=64 time=0.821 ms
64 bytes from fec0:1234::3: icmp_seq=15 ttl=64 time=0.780 ms
64 bytes from fec0:1234::3: icmp_seq=16 ttl=64 time=0.992 ms
64 bytes from fec0:1234::3: icmp_seq=17 ttl=64 time=0.774 ms
64 bytes from fec0:1234::3: icmp_seq=18 ttl=64 time=0.880 ms
64 bytes from fec0:1234::3: icmp_seq=19 ttl=64 time=1.01 ms
64 bytes from fec0:1234::3: icmp_seq=20 ttl=64 time=0.779 ms
64 bytes from fec0:1234::3: icmp_seq=21 ttl=64 time=0.822 ms
64 bytes from fec0:1234::3: icmp_seq=22 ttl=64 time=0.776 ms
64 bytes from fec0:1234::3: icmp_seq=24 ttl=64 time=0.778 ms
.
.
.
```

Node A Route Table When Node A Come Close To Node C

```
[root@localhost root]# route -A inet6
Kernel Ipv6 routing table
Destination                     Next Hop          Flags Metric Ref     Use Iface
::1/128                         ::                U     0      0         0 lo
fe80::260:1dff:fef0:e56a/128    ::                U     0      0         0 lo
fe80::/10                       ::                UA    256    0         0 eth1
fec0:1234::1/128                ::                U     0      255        0 lo
fec0:1234::2/128                ::                UH    2      3         0 eth1
fec0:1234::3/128                fec0:1234::3      UHC   0      3         1 eth1
fec0:1234::3/128                ::                UH    2      0         0 eth1
fec0:1234::/64                  ::                UA    256    0         0 eth1
ff05::11/128                    ff05::11          UAC   0      1356        1 eth1
ff00::/8                        ::                UA    256    0         0 eth1
```

Figure 19 Test Results for Test Scenario 3

The results shown in Figure 8.9 indicated that the route entry to Node C in Node A has changed from going though Node B to going to Node C directly instead. The ping result also show that the pinging time is shorter after the route changed. Use "traceroute6" to show the path for a packet travel after the route changed. The results is shown in Figure 20.

Traceroute6 Results From Node A To Node C

```
[root@localhost root]# traceroute6 fec0:1234::3
traceroute to fec0:1234::3 (fec0:1234::3) from fec0:1234::1, 30 hops max, 16 byte packets
 1   fec0:1234::3 (fec0:1234::3)  6.755 ms  1.279 ms  6.047 ms
```

Figure 20 "traceroute6" Results

The result shown in Figure 20 indicated that Node A is sending packet directly to Node C.

8.5 Test Scenario 4

Continue with Test Scenario 3 setup with ping is still running. Move Node A back to the location as in Test Scenario 2 where Node A cannot see Node C directly. Ping continues to work and the time for ping will be longer and the route table for Node A also updated with the new route entry where Node A can now see Node C via Node B. The results are shown in Figure 21.

The results shown that the ping time is now longer and the route to Node C in Node A is now go through Node B. Use "traceroute6" command to display the path of the packet travel from Node A to Node C and the results is shown in Figure 22.

The results shown in Figure 22 indicated that the packet from Node A is travelled to Node C via Node B.

Ping Results From Node A To Node C When Node A Move Away From Node C

```
[root@localhost root]# ping6 fec0:1234::3
PING fec0:1234::3(fec0:1234::3) from fec0:1234::1 : 56 data bytes
64 bytes from fec0:1234::3: icmp_seq=1 ttl=63 time=16.8 ms
64 bytes from fec0:1234::3: icmp_seq=2 ttl=63 time=1.34 ms
64 bytes from fec0:1234::3: icmp_seq=3 ttl=63 time=1.24 ms
64 bytes from fec0:1234::3: icmp_seq=4 ttl=63 time=1.20 ms
64 bytes from fec0:1234::3: icmp_seq=5 ttl=63 time=1.17 ms
64 bytes from fec0:1234::3: icmp_seq=12 ttl=64 time=1001 ms
64 bytes from fec0:1234::3: icmp_seq=13 ttl=64 time=9.56 ms
64 bytes from fec0:1234::3: icmp_seq=14 ttl=64 time=0.821 ms
64 bytes from fec0:1234::3: icmp_seq=15 ttl=64 time=0.780 ms
64 bytes from fec0:1234::3: icmp_seq=16 ttl=64 time=0.992 ms
64 bytes from fec0:1234::3: icmp_seq=17 ttl=64 time=0.774 ms
64 bytes from fec0:1234::3: icmp_seq=18 ttl=64 time=0.880 ms
64 bytes from fec0:1234::3: icmp_seq=19 ttl=64 time=1.01 ms
64 bytes from fec0:1234::3: icmp_seq=20 ttl=64 time=0.779 ms
64 bytes from fec0:1234::3: icmp_seq=21 ttl=64 time=0.822 ms
64 bytes from fec0:1234::3: icmp_seq=22 ttl=64 time=0.776 ms
64 bytes from fec0:1234::3: icmp_seq=24 ttl=64 time=0.778 ms
64 bytes from fec0:1234::3: icmp_seq=25 ttl=64 time=16.6 ms  <-- route change again
64 bytes from fec0:1234::3: icmp_seq=27 ttl=64 time=1.01 ms
64 bytes from fec0:1234::3: icmp_seq=28 ttl=63 time=14.8 ms
64 bytes from fec0:1234::3: icmp_seq=32 ttl=63 time=10.3 ms
64 bytes from fec0:1234::3: icmp_seq=33 ttl=63 time=1.48 ms
64 bytes from fec0:1234::3: icmp_seq=34 ttl=63 time=1.12 ms
64 bytes from fec0:1234::3: icmp_seq=35 ttl=63 time=1.12 ms
64 bytes from fec0:1234::3: icmp_seq=36 ttl=63 time=1.13 ms
64 bytes from fec0:1234::3: icmp_seq=37 ttl=63 time=1.12 ms
64 bytes from fec0:1234::3: icmp_seq=38 ttl=63 time=1.13 ms
64 bytes from fec0:1234::3: icmp_seq=39 ttl=63 time=2.81 ms

--- fec0:1234::3 ping statistics ---
39 packets transmitted, 28 received, 28% loss, time 38265ms
rtt min/avg/max/mdev = 0.774/39.062/1001.218/185.233 ms, pipe 2
```

Node A Route Table When Node A Move Away From Node C

```
[root@localhost root]# route -A inet6
Kernel IPv6 routing table
Destination                       Next Hop          Flags Metric Ref    Use Iface
::1/128                           ::                U     0      0        0 lo
fe80::260:1dff:fef0:e56a/128      ::                U     0      0        0 lo
fe80::/10                         ::                UA    256    0        0 eth1
fec0:1234::1/128                  ::                U     0      300       0 lo
fec0:1234::2/128                  ::                UH    2      4        0 eth1
fec0:1234::3/128                  fec0:1234::2      UGH   3      18       1 eth1
fec0:1234::/64                    ::                UA    256    0        0 eth1
ff05::11/128                      ff05::11          UAC   0      1389       1 eth1
ff00::/8                          ::                UA    256    0        0 eth1
```

Figure 21 Test Results for Test Scenario 4

Traceroute6 Results From Node A To Node C

```
[root@localhost root]# traceroute6 fec0:1234::3
traceroute to fec0:1234::3 (fec0:1234::3) from fec0:1234::1, 30 hops max, 16 byte packets
 1  fec0:1234::2 (fec0:1234::2)  13.957 ms  3.003 ms  2.423 ms
 2  fec0:1234::3 (fec0:1234::3)  13.582 ms  5.311 ms  0.774 ms
```

Figure 22 "traceroute6" Results

8.6 Test Scenario 5

Continue with Test Scenario 4, stop the ping command. Initiate Secure Shell login to Node C from A: "ssh –6 fec0:1234::3" and log in as root user. After login to Node C, display the route table of Node C: "route –A inet6" and display the interface configuration using "ifconfig". The results is shown in Figure 23.

Run Secure Shell On Node C From Node A

```
[root@localhost root]# ssh -6 fec0:1234::3
root@fec0:1234::3's password:
Last login: Thu Mar 20 13:50:18 2003 from fec0:1234::1
```

Dump Node C Route Table From The SSH Session

```
[root@localhost root]# route -A inet6
Kernel IPv6 routing table
Destination                    Next Hop          Flags Metric Ref    Use Iface
::1/128                        ::                U     0      0        0 lo
fe80::260:1dff:fef0:f1ae/128   ::                U     0      14       0 lo
fe80::/10                      ::                UA    256    0        0 eth1
fec0:1234::1/128               fec0:1234::2      UGH   3      5        1 eth1
fec0:1234::2/128               fec0:1234::2      UHC   0      2        0 eth1
fec0:1234::2/128               ::                UH    2      7        0 eth1
fec0:1234::3/128               ::                U     0      843       1 lo
fec0:1234::/64                 ::                UA    256    0        0 eth1
ff05::11/128                   ff05::11          UAC   0      2956       1 eth1
ff00::/8                       ::                UA    256    0        0 eth1
```

Display Node C Interface Information From The SSH Session

```
[root@localhost root]# ifconfig
eth1      Link encap:Ethernet  HWaddr 00:60:1D:F0:F1:AE
          inet6 addr: fec0:1234::3/64 Scope:Site
          inet6 addr: fe80::260:1dff:fef0:f1ae/10 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:3668 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3034 errors:16 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:385034 (376.0 Kb)  TX bytes:494529 (482.9 Kb)
          Interrupt:3 Base address:0x100

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:178 errors:0 dropped:0 overruns:0 frame:0
          TX packets:178 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:25418 (24.8 Kb)  TX bytes:25418 (24.8 Kb)
```

Figure 23 SSH Session From Node A To Node C

CHAPTER NINE
CONCLUSIONS

The goal of this project was to implement the first AODV routing protocol to support IPv6 functionalities. The study phase of this project was started in August 2002. No IPv6 AODV implementation was found at that time. The objective of this project was refined to implement IPv6 functionalities on an existing IPv4 AODV routing protocol. It was decided to implement IPv6 functionalities on Uppsala University's AODV (referred to as AODV-UU) implementation. The positive aspects of AODV-UU are that the protocol is stable and very well tested. The negative aspects are lack of design documentation and lack of comments in the code.

IPv6 functionalities have been successfully implemented on AODV-UU and completed with comprehensive testing. There are many opportunities for further work, such as providing multiple interface support, ns-2 support, performance analysis and so on.

The value of this project is not only to provide a functioning IPv6 AODV but also to provide a valuable guide for porting IPv4 protocols to support IPv6. This project also gives guidance in setting up a test bed for IPv6 AODV.

APPENDICES

A.1 Linux Kernel

IPv6 AODV is implemented to run on Linux Kernel Version 2.4.18-3 (Red Hat Linux 7.3) and up. IPv6 AODV has been tested running on Linux Kernel 2.4.18-3, 2.4.19 and 2.4.20. User can find out the kernel version using "uname" command. On the user prompt, type: *"uname –a"*. The system will return all the kernel version information. If the version is lower than 2.4.18-3 and would like to run IPv6 AODV, user can download the latest kernel source code from www.kernel.org. User can also find more information on how to compile and install the new kernel from the "Linux Kernel HOWTO Tutorial" (www.tldp.org/HOWTO/Kernel-HOWTO.html).

IPv6 AODV are required the Linux Kernel to have the IPv6 feature enable. User also need build the IPv6 Netfilter module as a kernel loadable module. In order for the user to build the kernel, use must have the kernel source installed in "/usr/src/linux-2.4.xx" directory. User can turn on these features using the following steps:

1. Go to the /usr/src/linux-2.4.xx directory
2. Type "make xconfig"
3. Click on the "Code maturity level options" menu and select "y" for development and/or incomplete code/driver.
4. Click on the "Networking Options" menu, select "y" for Network Packet Filtering, IPv6 protocol and click on "IPv6: Netfilter Configuration". Select "m" for all the options.
5. Follow the Linux Kernel Howto tutorial to build the kernel

After the kernel is rebuild and boot successfully, user need to add the following line: "NETWORKING_IPV6=yes" to the following file: "/etc/sysconfig/network". User can restart the network setup without rebooting the system using "/etc/init.d/network restart" command. User can verified that the Netfilter modules are build successfully using the following command "modprobe ip6_queue". If no error messages pop out, that means the ip6_queue module is build successfully.

A.2 802.11b Wireless Device

AODV is not restricted to run only on wireless network but it is intend for ad hoc wireless purpose. User can use 802.11b wireless device to form an ad hoc wireless network. Since 802.11b is the most popular and affordable technology at this point of time, IPv6 AODV are tested using Lucent 802.11b PCMCIA wireless card. Since this is an ad hoc network, user does not need an access point for the setup. User can find more information on how to install and set up a wireless device driver on Linux from the "Wireless Howto" tutorial (www.tldp.org/HOWTO/Wireless-HOWTO.html). The only thing that user need to be aware is, all the wireless devices must be running in ad hoc mode.

A.3 Set up IPv6 Site Local Address

Once the IPv6 kernel option is turn on for the Linux kernel, user can assigned an Site Local IPv6 address to an interface using ifconfig command. For example, "ifconfig eth1 add fec0:1234::1/64". This mean that the site local prefix address is fec0:1234 and the prefix length is 64. The site local host can be identified by fec0:1234::1 IPv6 address. User can refer to "IPv6 Howto" tutorial (www.bieringer.de/linux/IPv6/) to learn more about IPv6 configuration.

A.4 Running IPv6 AODV

User can download the IPv6 AODV code from www.newmic.com/research/aodv.html. Once the user untar the AODV tar file to a specified directory, user can build the code by typing "make clean install" in the specified AODV directory. "aodvd6" will be build and install to the /bin directory. To run IPv6 AODV, type "aodvd6 –i eth1" where "eth1" is the interface that AODV will be used for communication. Before aodvd6 can be run, the specified interface must have been assigned an site local address. User can find out more options by typing "aodvd6 –h" to print out all the available options on the screen.

A.5 Testing Utilities

This project are also using some other IPv6 utilities for testing other than the Linux net tool utilities. For example, netcat6, a utility to reads and writes data across network connections and it is fully support IPv6. Netcat6 can be download from www.deepspace6.net/projects/netcat6.html web site. This project is also using a Linux MPEG player call "MTV" together with netcat6 for testing. MTV can be downloaded from www.mpegtv.com. User can play MPEG file thorough the network using these two tools. For example, Node A want to play a MPEG movie from Node B, both of these nodes must have netcat6 install and Node A must have MTV player install. Type the following command on Node B "nc6 –l –p 5000 < ./test.mpg". On Node A, type "nc6 <Node B IP Address> <Port #> | mtvp –". The test.mpg will be played on Node A.

A.6 Linux Kernel Core Dump (LKCD) Tools

The Linux Kernel Core Dump project has created a set of utilities and kernel patches for debugging Linux kernel crash. It is designed to provide kernel developers as well as system administrators a reliable method of detecting, saving and examining

system crashes. Developers can use these tools to analyse the crash dump. LKCD utilities, including well written tutorials, can be download from http://lkcd.sourceforge.net web site.

A.7 IPv4 AODV Implementation

The are couples of IPv4 AODV implementation that can be downloaded for free from the Internet. The following implementations are recommended:

UCSB Implementation: moment.cs.ucsb.edu/AODV/aodv.html

NIST Implementation: w3.antd.nist.gov/wctg/aodv_kernel

Uppsala University Implementation: www.docs.uu.se/~henrikl/aodv

REFERENCE LIST

[1] C.K. Toh, "Ad Hoc Mobile Wireless Networks: Protocols and Systems", Prentice Hall PTR, 2001.

[2] Charles E. Perkins, Elizabeth M. Royer & Samir R. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing – Internet-Draft", January 2002, Work-in-progress.

[3] Charles E. Perkins, Elizabeth M. Royer & Samir R. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing For IP Version 6 – Internet-Draft", November 2001, Work-in-progress.

[4] Mark A. Miller, "Implementing IPv6, Second Edition: Supporting the Next Generation Internet Protocols", M&T Books, 2000.

[5] Peter H. Salus, "Big Book Of IPv6 Addressing RFCs", Morgan Kaufmann, 2000.

[6] Online, "Porting Networking Applications to the IPv6 APIs, Solaris™ Version 8", Sun Microsystems, October 1999.

Can be downloaded from: wwws.sun.com/software/solaris/ipv6/porting_guide_ipv6.pdf

[7] Online, "HP-UX 11i IPv6 Porting Guide", Hewlett-Packard, February 2001.

Can be downloaded from: www.docs.hp.com/hpux/onlinedocs/netcom/ipv6portingguide.pdf

[8] S. Corson, J. Macker, "Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations – RFC 2501", IETF, January 1999.

[9] Elizabeth M. Royer, C.K. Toh, "A Review of Current Routing Protocols for Ad Hoc Mobile Wireless Networks", IEEE Personal Communications, April 1999.