Implementation study of AODV for Microsoft Windows CE platform

Ms. Prinima Gupta[1], Dr. R. K Tuteja[2]

**[1]MCA, Manav Rachna College of Engineering,
Faridabad, India-121001**

**[2]MCA, N.C. Institute of Computer Sciences,
Israna, Panipat, India-121001**

Abstract

The Ad hoc On-Demand Distance Vector (AODV) routing protocol is designed for use in mobile ad-hoc networks. There are a number of implementations of the Ad-hoc routing protocols available for the Linux platform, but not for any other platform. Windows CE .NET, the successor to Windows CE 3.0, combines an advanced real-time embedded operating system with the most powerful tools for rapidly creating the next generation of smart, connected, and small-footprint devices. This paper presents a design, implementation, and evaluation of AODV protocol for the Windows CE .NET operating system and describes the salient characteristics of the Windows CE platform for those unfamiliar with the system. It also focuses on the application level implementation of the algorithm and provides the framework for integrating the protocol in Windows CE .NET in form of an NDIS intermediate driver.

Keywords: *AODV, Windows CE .NET, Emulator, NDIS Intermediate Driver.*

1. Introduction

The field of ad-hoc networks is an area of much active research at the moment. The AODV [1], routing protocol is an on-demand, or reactive protocol for finding routes, that is, a route is established only when it is required by a source node for transmitting data packets. It has been shown to have promising characteristics, including performance figures, in simulation studies compared with other proposed ad-hoc routing protocols. This dissertation presents the design, implementation, and evaluation of the AODV [5] routing protocol for the Windows CE platform.

The real-world testing of the ad hoc routing protocol has been limited to the Linux Platform. The protocol has not been accessible to non-technical users of mobile devices, as the majority of such users are not familiar with the Linux operating system. Users will be able to install our version on their Windows CE mobile devices, giving them the ability to connect to any network running AODV. They may, for example, wish to communicate with other users during a meeting where no pre-existing infrastructure is in place.

In an on-demand ad-hoc network, these two processes are closely linked, as the routing protocol must be able to handle situations where packets are to be forwarded to a previously unknown destination by initiating a route discovery cycle. The network protocol stacks of modern operating systems have not been designed to deal with this situation; they do not provide adequate system services for the implementation of ad-hoc routing protocols. This greatly complicates the implementation of on-demand ad-hoc routing protocols, and has slowed their development. The implementation strategy of existing ad-hoc protocols in Linux is examined earlier. Most such protocols rely on the packet filtering and mangling architecture called Netfilter to handle packets for an ad-hoc routing protocol. Unfortunately the Windows protocol stack has no direct counterpart to the Netfilter framework.

Windows CE is a newer generation of operating system from Microsoft. The Windows CE operating system is a 32-bit, multitasking, multithreaded operating system that has a scalable, open architecture design, providing support for a variety of devices [3]. Some new features are introduced in Windows CE comparing to windows desktop operating system, such as limitation of memory, unicode problem, componentization, supporting a range of embedded, mobile or multimedia product lines [2], etc. Unlike desktop operating system has mass storage device, Windows CE programs almost run on devices that never have disks. Standard communications support is built into Windows CE, enabling access to the Internet to send and receive e-mail or browse the World Wide Web. Windows CE 5.0 is billed as a low-cost, compact, fast-to-market, real-time operating system available for supporting four different CPU families, spanning the ARM, MIPS, SuperH (SH), and x86 architectures.

2. Window CE Overview

Microsoft Windows CE .NET is designed to meet the needs of a broad range of intelligent hardware devices, from enterprise tools such as industrial controllers, communications hubs, and point-of-sale terminals to

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.

consumer products such as cameras, Internet appliances, interactive televisions and mobile-computing devices [3]. The networking protocol stack in Windows CE operating system is a "subset" of windows desktop networking protocol stack. Windows CE .NET offers the application developer the ease of scripting languages, along with the versatile environment of the Microsoft Win32 application programming interface (API).

Windows CE is adapted for a specific hardware platform by creating a thin layer of code that resides between the kernel and the hardware platform. This layer is known as the OEM Adaptation Layer (OAL). The operating system has been designed using a component based structure such that Original Equipment Manufacturers (OEMs) can choose only the operating system features that they require for their specific hardware platform. The OAL isolates device-specific hardware features from the kernel. The Windows CE kernel, in turn, contains processor-specific code to handle processor core functions. The OAL is specific for a particular CPU and hardware platform. In only choosing the components they are interested in, OEMs can keep the footprint of their devices small. For example, the Pocket PC operating system is based on the building blocks from the previous version of Windows CE, version 3. It is not however the same as the Windows CE based operating system on all other Windows CE 3 based devices.

The Windows CE TCP/IP consists of *core protocol elements*, *services*, and the *interfaces* between them

- The Network Device Interface Specification (NDIS) is a public interface, documented on the Microsoft Developer Network (MSDN), which governs the communication between interface device drivers controlling hardware adapters, and the upper-level protocols, the most common being TCP/IP.
- The Transport Driver Interface (TDI), in the Microsoft® Windows® CE operating system architecture, is an interface that serves as an adaptation layer to Winsock-based user APIs. It isolates the highly asynchronous callback-based architecture of the stack presenting a Windows Sockets Specification 1.1 interface.
- Winsock 2.0 is a networking API that provides access to multiple transport protocols, including support for creating applications that support multiple socket types.
- The Winsock DLLs communicate with the TCP/IP stack through the TDI interface. Winsock is the Microsoft Windows implementation of the Berkeley Sockets interface, with some Windows specific extensions. The Winsock interface is part of the win32 API, and is most commonly used by applications to send TCP/IP traffic to other hosts.

Windows CE removes the barrier between kernel space and user space for device drivers. As such all the networking device drivers in the Windows CE architecture effectively run in protected user mode. Hence, such drivers can link with the Winsock DLLs, and do not need to use the TDI interface directly as in Windows XP.

2.1 Modular Structure

We can build a Microsoft Windows CE .NET based platform using a number of discrete modules. This minimizes the memory needed by the platform. By selecting only those modules that your platform requires, you can minimize the amount of memory that your device requires. A module contains a collection of related application programming interface (API) functions. Some modules are composed of components. Each component can contain a collection of API functions.

2.2 Features

The Microsoft Platform Builder version 4.2 Catalog consists of a list of board support packages (BSPs), drivers, configurations for core operating systems (OSs), and Platform Manager Transports. The items in the Catalog represent the technologies you can select when designing your Microsoft Windows CE .NET based platform [2] [3] [4]. These technologies are organized and displayed as features. Features are specific implementations of the technologies available in the Windows CE .NET 4.2 OS.

2.3 Operating Systems Features

The core operating system (OS) services provide a common foundation for all Microsoft Windows CE .NET OSs. The services enable low-level tasks such as process, thread, and memory management, and provide some file system functionality. The Windows CE OS offers a rich set of features. Component services, networking capabilities, multimedia support, and many other capabilities are contained within individual OS features.

The kernel, which is represented by the Nk.exe module, is the core of the Microsoft Windows CE operating system (OS). The kernel provides the base OS functionality for any Windows CE based device. This functionality includes process, thread, and memory management. We can use the kernel process and thread functions to create, terminate, and synchronize processes and threads and to schedule and suspend a thread. Processes, which represent single instances of running applications, enable users to work on more than one application at a time. Threads enable an application to perform more than one task at a time. Thread priority levels, priority inversion handling, interrupt support, and timing and scheduling are all included in the Windows

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.

CE kernel architecture. Together, they provide real-time application capability for time-critical systems.

3. User level Implementation

The user level implementation is developed for testing the algorithm on actual mobile devices on actual operating system and gives us a fair idea of the use and performance of the algorithm when it will be implemented at the kernel level. It provides us an opportunity where we can perform simulations of the algorithm on a test bed set up. Such simulations provide more realistic results than the ns simulations. This implementation shows how we would integrate our protocol with Windows CE, so that our protocol can run directly above the data link layer and applications can be built over our protocol.

3.1 Data Structures

The data structures made by us to implement the AODV algorithm will cover the entities that are exchanged during the protocol and that are used by the nodes for management purposes. The structures used by us can be basically classified into the following groups:

- Lists of nodes. A linked list is maintained for nodes directly accessible over the wireless interface. Each neighbour_list struct entry contains the neighbour's IP address, hardware address, the interface through which it can be contacted, and the route table entry for this neighbour (i.e. within one hop). When entries in this list are timed out, this may initiate the sending of a Route Error message.
- Packets. There are different types of packets that can be transferred during a successful operation of the algorithm. The various packet types are hello, route request, route reply, route error, and control packet. All these packets have different properties and hence need to be represented uniquely.
- Events. The control Packets are placed in the different event_queue list which contains events such as EVENT_RREQ, EVENT_RREP, EVENT_RERR and EVENT_CLEANUP.

3.2 Timers

Nearly every entity attached with a protocol for ad hoc networks has an expiration time. This is due to the mobility experienced by such networks. There are a number of operations within an AODV implementation that require specific timing. For example, HELLO messages are sent at a periodic interval, Route Requests are rebroadcast after a certain interval, etc. The timed events are sorted in increasing time of occurrence, such that the timed item to occur soonest is always at the front. The following timers possible:

- EVENT_HELLO Timer: This timer is used to trigger hello events at each interface. When a HELLO message is sent, another EVENT_HELLO message event is placed in the timer queue, and set to occur in HELLO_INTERVAL seconds. Its value serves as basic units for other timers, thus, the expiration values of the other timers are in a way multiples of the expiration value here.
- EVENT_RREQ Timer: This timer is used after a Route Request has been sent, but no Route Reply has been received in a certain time.
- EVENT_NEIGHBOUR Timer: This timer is used when, a result of the receipt of a HELLO message, a new neighbour is added to the neighbour list queue, or the lifetime of an existing one is updated, then this entry must be set to expire after a certain timeout.
- EVENT_CLEANUP Timer: This timer is used such that cleanups of the routing table and flood_id_queue occurs at periodic intervals.

3.3 Simulations

We will test the user level application developed by us by using the facility of Emulator. With the Emulator, you can design and build a Windows CE based platform and test it using software that mimics hardware rather than testing the platform on hardware. In the absence of PDAs, we can run the application on normal PCs of our LAN by running the emulators on them, see Figure 1. The emulator can run our application in the same manner a normal PC will do, only with slight degradation in performance. On an average an emulator gives 80% of performance of the normal PC on which it runs. Now we will refer to the emulators running on various PCs as nodes. So after hard coding the neighbors in the application for every node, we can run the software and test the performance of the protocol.

Fig-1 Emulator for Window CE

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.

4. Design Approaches

Two approaches were considered. The main approach used is the intermediate driver approach. This approach was chosen as it is the only approach that does not require large changed to the TCP/IP protocol driver, and as such it is the only form of such a driver that would be easy to install and distribute. In addition, extensive kernel modifications were performed to alter the filter-hook mechanism to provide asynchronous packet filtering facilities directly in the IP layer.

4.1 Implementing AODV as NDIS Intermediate Driver

In the Windows networking architecture, the Network Driver Interface Specification (NDIS) facilitates communication between the operating system, upper level protocol drivers (such as TCP/IP), and network drivers (that control the hardware network interface cards). The NDIS interface is located between an upper-level protocol driver on the top of the communications architecture, the intermediate and miniport drivers in the middle of the communications architecture, and the hardware network adaptors at the bottom. Thus an NDIS protocol driver like TCP/IP calls functions in the intermediate or miniport drivers, fully abstracted through NDIS, and vice versa.

The most important driver among all is the intermediate driver. NDIS intermediate drivers include a protocol driver interface at their lower edge and a miniport driver interface at their upper edge. The protocol interface of intermediate driver allows it to load above a driver with a miniport driver interface. The miniport interface of intermediate driver allows it to load below a driver with a protocol lower edge interface. For the upper layer protocol drivers, NDIS intermediate driver works like a miniport driver, but for the lower layer miniport drivers NDIS intermediate driver works like a protocol driver. Figure 2 shows relationship among NDIS intermediate driver protocol driver and miniport driver.

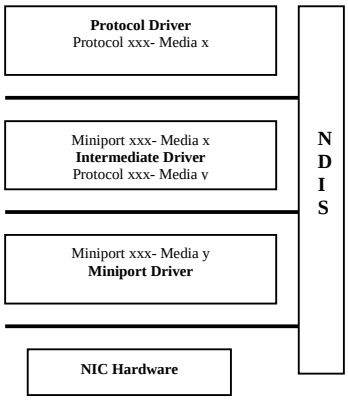

Fig-2 NDIS Intermediate Driver [3]

Before data packets flow to the under layer networks or upper layer applications, the mangling operation can be performed inside the intermediate driver. NDIS intermediate drivers can be used to filter packets and perform data mangling operations on them. For example, it can be used to encrypt or decrypt packets.

The advantages of this approach is that it is easy to install, it would be easy to port to other Windows versions which use NDIS, including Windows XP, and it would be possible to get such a driver signed by Microsoft under the Windows Hardware Quality Labs (WHQL) scheme. The disadvantages include that such an implementation could be seen as being too low in the networking stack: as a filter mechanism between the networking layer and data link layer of the OSI model. Packets for an unknown route will have to be rerouted by the AODV intermediate driver after they are 'coaxed' out of the IP layer. Also, because the routing protocol is tightly coupled with the data link layer in this implementation, it is not independent of the specific transmission mechanism being used (e.g. 802.3, 802.11, HiperLAN, etc).

4.2 Modifying the Filter Hook Mechanism

The filter-hook driver mechanism comes close to meeting the requirements for an on-demand ad-hoc routing protocol. The main drawback of the filter –hook driver is that it cannot be used to deal with packets asynchronously: it must either accept the packet for transmission immediately, or discard the packet. Thus packets cannot be buffered while a route discovery cycle takes place. It is possible with significant effort to modify the filter-hook mechanism, or more likely to introduce a new similar mechanism, such that packets can be removed and later re-injected into the IP layer, to provide functionality similar to that of Netfilter in the Linux operating system.

By this mechanism an implementation of AODV would consist of a separate driver that communicates with the TCP/IP driver using I/O Control Codes (IOCTLs). The IP layer would export IOCTLs for registering a call-back function to be called with a packet as a parameter as the packet traverses the relevant hooks. The AODV driver on initialization would use the exported IOCTL to register its call-back function. The function would return a value indicating the packet should immediately continue its traversal of the network stack, should be immediately discarded, or should be removed from its traversal to be reinjected at a later stage. The IP layer will also export a mechanism for the attached filter driver to reinject packets processed asynchronously.

The advantages of such an approach are that porting effort for future ad-hoc protocols between Linux and Windows would be greatly reduced, and the new hooking mechanism would be very suitable for meeting the requirements of ad-hoc routing protocols. Such a

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.

mechanism would also be useful to many other applications that require asynchronous packet filtering and mangling facilities, similar to those which use Linux Netfilter. The disadvantage of this approach include that the code for the TCP/IP driver in Windows is proprietary Microsoft code, and cannot be viewed or modified without special license. Distributing such a mechanism would not be possible unless the mechanism is adopted by Microsoft for future versions of their operating systems. Installing such a mechanism on existing operating systems would not be straight forward.

4.3 Module Description

The AODV code is written in C. As such it consists of a number of modules, whose functionality is described here.

aodvp.h
This is an include file containing some important macros and type definitions. A number of important structs are defined in here, including the AODV control message types (RERR, RREP, RREQ), and various linked list structures for the AODV route table, precursor entries, the timer queue, event queue, etc.

aodvp_driver {.c, .h}
This driver has the dual purpose of initialising the NDIS intermediate driver (or filter-hook driver), and the AODV structures. It contains the DriverEntry function which is the first entry point called in an intermediate driver. Its purpose is to register the intermediate driver with NDIS. It also initialises the AODV structures, and starts the event_queue thread. This module also contains the clean-up function which is called when the driver is unloaded by NDIS.

aodvp_thread {.c, .h}
As control packets are received in the intermediate driver (or filter driver) on an interrupt, they are placed as an entry in the event_queue structure. To prevent doing a lot of processing on interrupts, the packets are processed by a separate thread which is created and managed in this module. The thread sleeps until a new control packet arrives. The control packet is placed in the event_queue list, and the aodv thread is woken. The types of events to be processed are:
EVENT_RREQ: occurs when a Route Request message is received on one of the node's interfaces.
EVENT_RREP: occurs when a Route Reply message is received on one of the node's interfaces. Since HELLO messages are Route Reply messages with a hop count of zero, HELLO messages are also processed with this event.
EVENT_RREP_ACK: a node can request by setting a flag in its Route Reply message that it should receive an explicit acknowledgement in the form of a Route Reply Acknowledgement message. The acknowledgement is handled with this event.
EVENT_RERR: occurs when a Route Error message is received on one of the node's interfaces.
EVENT_CLEANUP: occurs periodically, and is used to clean up inactive routes in the route table, and the flood_id_queue.

event_queue {.h, .c}
The event_queue module maintains a linked list of event_queue_entry structs. The event queue is a First-In First-Out (FIFO) structure. The module contains functions to initialise the queue, insert entries, remove the next entry, and cleanup the queue. Event queue entries consist of AODV control packets and cleanup events, and are used such that the bulk of the AODV routing protocol processing occurs in the AODV thread, and not in an interrupt thread.

miniport {.c, .h}
This module is relevant to the NDIS intermediate driver implementation, and not the modified filter-hook driver. It contains the NDIS miniport interface that is exported at the upper edge of the NDIS intermediate driver. As such, packets being sent from the TCP/IP protocol driver arrive in this module. From here they are passed to the packet_out module for AODV processing, before being passed down to the underlying miniport (or another intermediate) driver.

neighbour_list {.c, .h}
This module maintains a linked list of nodes directly accessible over the wireless interface from this one (i.e. within one hop). The module contains functions for managing this list. When entries in this list are timed out, this may initiate the sending of a Route Error message.

packet_in {.c, .h}
When a packet is received, either through the intermediate driver's lower-edge protocol interface, or the modified filter-hook driver's incoming hook, it is sent to a function in this module for processing. Only AODV packets (those UDP packets destined for the AODV port) are examined. Firstly the format of the packet is checked to see that it is a properly formatted AODV packet. Next, if the packet is a unicast packet (such as a Route Reply message) destined for another node to which we no longer have a route table entry, a Route Error message is sent. Next, the lifetime of the route from the source is updated, and the packet placed in the event queue for processing.

packet_out {.c, .h}
Packets received through the intermediate driver's upper edge miniport interface, or the modified filter-hook driver's outgoing packet hook, are filtered in a function in this module. Unicast packets for which we have a route, or broadcast packets, are passed through without modification. Unicast packets for which we have no route, and hence for which a route discovery cycle is required, are passed to the packet_queue module for

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.

buffering, and a Route Request is initiated. The packets will later be reinjected (or dropped) when the route discovery cycle succeeds (or fails). The lifetime for valid routes is updated when unicast packets are sent on this route.

protocol {.c, .h}
Similar to the miniport module, this is specific to the NDIS intermediate driver implementation. Packets arriving at the lower edge of the intermediate driver (from the miniport driver, or a lower layered intermediate driver) from other hosts are filtered through this module. They are first passed to the packet_in module for filtering, and then released up to the overlying protocol driver for processing.

route_table {.c, .h}
This module maintains all the needed routing information for contacting other nodes. It provides functions for managing the route_table_entry structures, including the creating and deleting of entries, and deleting of invalid entries. It provides functionality for adding and deleting precursor entries to and from a route table entry. Finally, it uses the Windows IP Helper API to manage the kernel routing table, to add and delete appropriate entries.

rrep {.c, .h}
This module provides the functions necessary for correct handling of route reply messages, including receiving HELLO messages. It is passed packets from the AODV thread, and appropriate action is taken.

rreq {.c, .h}
This module provides the functionality for handling Route Requests. Received Route Request packets are passed into it from the AODV thread, and they are processed as required here. This module also exposes the function required for generating and sending a Route Request for a particular destination.

timer_queue {.c, .h}
There are a number of operations within an AODV implementation that require specific timing. For example, HELLO messages are sent at a periodic interval, Route Requests are rebroadcasted after a certain interval, etc. This module maintains a queue of timed events, sorted in increasing time of occurrence, such that the timed item to occur soonest is always at the front of the list. A separate thread runs to perform the timed operations. It sleeps until the time that the next timer_queue_entry is due (maintained as an absolute time in milliseconds according to the system clock). It then wakes up, performs any due timer items, and sleeps until the time the next new item is due. The possible timed items are EVENT_RREQ, EVENT_HELLO, EVENT_CLEANUP, and EVENT_NEIGHBOUR.

5. AODV Implementation Evaluation

Here we will discuss the testing of implementation of AODV at the user level. We tested the algorithm on emulators provided by Platform Builder. There were two levels of testing.

1.) This tests two directly connected nodes having emulators, exchanging hello messages with each other. Two computers running Platform Builder provided with Windows CE .NET 4.2 is used for this test. The Platform Builder is used to build our AODV application. It is then used to generate a platform, which is a specific implementation of Windows CE.NET based on an "Enterprise Web Pad" and using "Emulator: X86 BSP (Broad Support Package)".

We downloaded the OS image into the emulator running in "Virtual Switch - Fixed IP" mode. It is imperative that the two PCs running the Emulator have installed Microsoft Loopback Adapter, without which the emulators fail to run in Vitual Switch Fixed IP mode. When the emulators came up, we set their IP addresses & the subnet masks corresponding to the LAN and run the AODV application from the Platform Builder by selecting it from the drop down menu of the "Run Programs" tab in the Platform Builder. Each node periodically broadcasts HELLO messages. The connected node receives the HELLO messages, and installs a route to the other node and ensures the reply is received.

2.) In this test, we run the application simultaneously on three nodes having emulators configured in the topology 1-2-3 where node 1 is the source and 3 is the destination. Node 1 pings the last node and started sending a Route Request through the middle node. Node 2 receives the RREQ, and sends a Route Reply. Node 1 then installs the route and pings are correctly received. The actions are verified for above nodes.

6. Conclusions

The contribution of this work has been to produce a real-world implementation of AODV for Windows CE, suitable for running on mobile and embedded devices, such as palm-tops and PDAs. The paper showed that our implementation can run at the user level in Windows CE .NET. With this paper we provide a platform on which future performance studies of AODV will be performed.

The NDIS intermediate driver approach is easy to install and distribute. It does not require any changes to proprietary code. However there are drawbacks to this approach. One of the main drawbacks is that independence of the underlying data link layer is lost. When a packet for which a route discovery cycle takes place reaches the intermediate driver, it has already gone

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.

through kernel routing, and ARP. As such, when the route discovery cycle completes it is necessary to insert the hardware destination address on the Ethernet frame. This effectively limits this implementation to interoperating with Ethernet (802.3) and wireless Ethernet (802.11), and other data-link layers directly supported. If the routing protocol was to be used on any other data-link layer, then support would have to be explicitly added for this. This is not the case for an implementation within the IP layer, such as the packet filter-hook mechanism. In addition, there is a minor overhead associated with rerouting packets which are buffered during a route discovery cycle in the intermediate driver (involving correcting the hardware destination address). The filter-hook mechanism is well situated in the protocol stack for processing packets before kernel routing takes place. Its main drawback is that it cannot buffer packets while a route discovery cycle takes place.

Acknowledgment

I express my sincere gratitude and acknowledgement towards Prof. (Dr.) R. K Tuteja, Director (Academics), who guided me. It was his constant support and inspiration without which my efforts would not have taken this shape. I sincerely thank him for this, and seek his support for all my future endeavors.

References

[1] C.Perkins, E. Belding-Royer, S. Das, " Ad hoc On-Demand Distance Vector(AODV) Routing", http://www.ietf.org/rfc/rfc3561.txt, IETF, July 2003.
[2] D. Boling, "Programming Microsoft Windows CE .NET," Third Edition, Microsoft Press, 2003.
[3] MSDN homepage, http://msdn.microsoft.com, Internet, Sept.2007.
[4] A. Wigley, S. Wheelwright, R. Burbidge, "Microsoft .Net Compact Framework," Microsoft Press, 2003.
[5] David West. "An Implementation and Evaluation of the Ad-Hoc On-Demand Distance Vector Routing Protocol for Windows CE", Trinity College Dublin, 2003.
[6] UU AODV homepage. Erik Nordström. http://user.it.uu.se/~henrikl/aodv/. September 2003.
[7] NIST Kernel AODV homepage. Luke Klein-Berndt. http://w3.antd.nist.gov/wctg/aodv_kernel/. September 2003.
[8] Report on the AODV Interop. Elizabeth M. Belding-Royer. UCSB Tech Report 2002-18, June 2002.
[9] Windows Network Data and Packet Filtering. http://www.ndis.com/papers/winpktfilter.htm. September 2003.
[10] Microsoft Developer Network Documentation for Windows® CE .Net.
[11] V. Kawadia, Y. Zhang and B. Gupta, "System Services for Implementing Ad-hoc Routing Protocols," In Proceedings of International Conference on Parallel Processing Workshops, 2002.
[12] NDIS Intermediate Driver Samples For Windows NT, Windows 2000 and Higher. http://www.pcausa.com/pcasim/Default.htm.
[13] Writing an NDIS Intermediate Driver. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/network/hh/network/301int_3mg7.asp.
[14] NDIS Intermediate (IM) Driver Frequently Asked Questions. http://www.pcausa.com/resources/ndisimfaq.htm.

Ms. Prinima Gupta is pursuing PhD (computer science). She received her MPhil (computer science) degree in 2009. She holds an MCA from Kurukshetra University, Kurukshetra. She is currently working as Lecturer in MCA Department with Manav Rachna College of Engineering, Faridabad. She has 7 years of teaching experience. She published 03 papers in National conferences and 01 paper in International Journal. Her area of specialization includes Computer Networks and Computer Architecture.

Prof. (Dr.) R. K Tuteja is PhD from Kurukshetra University, Kurukshetra in 1969. He holds an MA (Mathematics) from KUK. He is currently working as Director (Academics) in NCICS, Israna, Panipat. He has 47 years of teaching experience. He was successfully guided 30 Ph.D research students and 17 students for M. Phil. Degree. He has published 126 Research papers in National/International Journals. He has worked as Head of Statistics/ Mathematics/ Computers Science & Application Department at M. D. University Rohtak.

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.