# Design Strategies for AODV Implementation in Linux

Ms. Prinima Gupta
MCA dept., Manav Rachna College of Engineering,
Sector-43, Faridabad. INDIA
prinima_mail@rediffmail.com

Dr. R. K Tuteja
MCA dept., N.C. Institute of Computer Sciences, Israna,
Panipat. INDIA
rk_tuteja2006@yahoo.co.in

***Abstract*—In a Mobile Ad hoc Network (MANET), mobile nodes constructing a network, nodes may join and leave at any time, and the topology changes dynamically. Routing in a MANET is challenging because of the dynamic topology and the lack of an existing fixed infrastructure. In this paper, we explore the difficulties encountered in implementing MANET routing protocols in real operating systems, and study the common requirements imposed by MANET routing on the underlying operating system services. Also, it explains implementation techniques of the AODV protocol to determine the needed events, such as: Snooping, Kernel Modification, and Netfilter. In addition, this paper presents a discussion of the advantages as well as disadvantages of each implementation of this architecture in Linux.**

***Keywords- Ad-hoc Networking, AODV, MANET.***

## I. INTRODUCTION

AODV is an on demand algorithm, meaning that it builds routes between nodes only as desired by source nodes. It maintains these routes as long as they are needed by the sources. Hence, it is considered as a reactive routing protocol. The Ad-hoc On Demand Distance Vector Routing (AODV) protocol is an algorithm used for the implementation of such networks. The connection between nodes is established for the duration of one session, so no need to have a base station in order to establish such a connection between nodes. Nodes discover other target nodes that are out of range by broadcasting the network with Rout Requests (RREQ) that are forwarded by each node. If the destination node get the RREQ, then it sends back Route Reply (RREP) to the source node. After the route has been discovered between source node and destination node, then it's the time to start sending data thru that route.

There are a limited number of such implementations, mostly for the Linux operating system. This paper is an exploration and comparison of several AODV implementations including: National Institute of Science and Technology (NIST), the University of California, Santa Barbara (UCSB), Uppsala University (UU) in Sweden and the University of Illinois, Urbana-Champaign (UIUC). The earliest implementation of AODV is the Mad-hoc implementation.

In this paper we first give an overview of the challenges implementers face when implementing an on-demand ad hoc routing protocol. These challenges emerge, as the on-demand routing model does not easily fit into the standard operating system routing and packet forwarding model. We describe the problems with the routing model of current operating systems and identify the necessary extra events that must be recognized to ensure correct behaviour of on-demand ad hoc routing protocols. Then we describe and discuss the different design strategies that have previously been deployed in implementations of on-demand ad hoc routing protocols, focusing on AODV implementations in Linux. The intention is to give an overview of the developed solutions and point out best practices and experiences learnt.

## II. BACKGROUND

First, we describe the AODV routing protocol and its basic operation. Then, we describe the available implementations of AODV for the Linux platform. It describes their available functionality and their design philosophy.

### *A. AODV Protocol Overview*

The AODV routing protocol is a reactive routing protocol; therefore, routes are determined only when needed. Hello messages may be used to detect and monitor links to neighbors. If Hello messages are used, each active node periodically broadcasts a Hello message that all its neighbors receive. Because nodes periodically send Hello messages, if a node fails to receive several Hello messages from a neighbor, a link break is detected. When a source has data to transmit to an unknown destination, it broadcasts a Route Request (RREQ) for that destination. When a RREQ is received by an intermediate node, a route to the source is created. If the receiving node has not received the RREQ before, is not the destination and does not have a current route to the destination, it rebroadcasts the RREQ. If the receiving node is the destination or has a current route to the destination, it generates a Route Reply (RREP). The RREP is unicast in a hop-by-hop fashion to the source. As the RREP propagates, each intermediate node creates a route to the destination. When the source receives the RREP, it records the route to the destination and begins sending data. If multiple RREPs are received by the source, the route with the shortest hop count is chosen.

As data flows from the source to the destination, each node along the route updates the timers associated with the routes to the source and destination, maintaining the routes in the routing table. If a route is not used for some period of time, a node

cannot be sure whether the route is still valid; consequently, the node removes the route from its routing table. If data is flowing and a link break is detected, a Route Error (RERR) message is sent to the source of the data in a hop-by-hop fashion. As the RERR propagates towards the source, each intermediate node invalidates routes to any unreachable destinations. When the source of the data receives the RERR, it invalidates the route and reinitiates route discovery, if necessary.

### B. AODV Implementations

Each implementation was developed and designed independently, but they all perform the same operations. The field studies using the AODV routing protocol have thus far been limited to devices running the Linux operating system, as the current implementations of AODV have all been developed for that platform. Thus real-world testing of the protocol has been limited to homogenous network environments. There are two types of different implementations, user space daemons and kernel modules.

*1) Mad-Hoc Implementation:* was the first available implementation of AODV by Fredrik Lilieblad, Oskar Mattsson, Petra Nylund, Dan Ouchterlony, and Anders Roxenhag running on a Linux 2.2 kernel but does not supports multicast. It uses the method of snooping ARP and data packets, using the libpcap Linux packet capturing facility. It is a user space only solution. It does not comply with an up-to-date version of the AODV specification, and is no longer supported. As such, it does not interoperate properly with the later implementations, and is not recommended for use.

*2) NIST Implementation:* was the only kernel implementation done by the NIST, Department of Commerce's Technology Administration U.S., and Wireless Communications Technologies Group running on a Linux with a 2.4 kernel. It is very fast and efficient reaching the best performance of all implementations. The NIST Implementation of AODV is currently at version 2.1 at time of writing. The latest version has support for multicast AODV, as well as multi-hop Internet gatewaying. The protocol is implemented completely as a Linux kernel module. If uses Netfilter from the 2.4 kernel to capture packets going in and out of the node instead of using the libcap library. It uses a Proc file to update the user about current routes and statistics for that node.

*3) Uppsala University Implementation*: The University of Uppsala also published a user space daemon implementation called AODV-UU which runs on Linux with a 2.4 kernel. Multicast support is available via a patch implemented by a group of researchers from the University of Maryland. The protocol is implemented as a user space daemon, and two loadable Linux kernel modules (kaodv and ip_queue_aodv). It uses the Netfilter library to intercept incoming and outgoing packets, but this is performed in user space.

*4) University California, Santa Barbara Implementation:* was the newest daemon published on 2nd of April 2002 by the University of California, Santa Barbara, running on a Linux with a 2.4 kernel. Similar to the UU implementation, the UCSB version is implemented as a user space daemon. It similarly uses the Netfilter library for intercepting incoming and outgoing packets from the chosen interface. In fact, the implementation uses directly the UU packet input user space packet queuing module and the kaodv/packet_queue_aodv kernel modules. As such, it suffers from exactly the same problems as the UU implementation in that all packets must pass the boundary between kernel and user space twice.

*5) University of Illinois, Urbana-Champaign Implementation:* The UIUC implementation, is based on their ad-hoc support library (ASL), which is a Linux specific library designed to provide all the services required by ad-hoc routing protocols. As such, the UIUC AODV implementation is a user space daemon compiled against the ASL library. The implementation has not been interoperability tested against the other protocol implementations. Much of the complexity of the user space ad-hoc routing module has been removed to the ASL library, a very desirable feature, as this should allow other, different, ad-hoc routing protocols to be developed using the same library.

## III. Identifying the Challenges of On-Demand AD-HOC Protocols

When we are discussing the challenges faced when implementing an on-demand ad hoc routing protocol, it is of relevance to recap the routing architecture of current operating systems. In particular, how the functionality is divided and why implementing on-demand protocols is a challenge compared to implementing traditional routing protocols or proactive ad hoc routing protocols.

The routing functionality in modern operating systems is typically divided in two parts:

### A. Packet forwarding function

Consists of the routing function within the kernel, located within the IP layer of the TCP/IP stack, in which packets are directed to the appropriate outgoing network interfaces, or local applications, according to the entries in the kernel routing table. When the IP-layer receives a packet, it inspects a table called the forwarding table. Based on the IP destination address the packet is either directed to a local application listening on the specified port number, dropped, or sent out to the corresponding next-hop neighbor on the specified network interface according to the destination IP address of the packet.

### B. Packet routing function

It typically consists of a user-level program responsible for populating the kernel routing table. The definition of an optimal route is dependent on the routing algorithm; the number of hops to the destination is usually the chosen metric. The program performing the routing is typically implemented in user space as a program running in the background (the routing daemon).

On Linux the route selection process is carried out the following way: when selecting a route for a packet, the kernel

first searches the routing cache for an entry matching the destination IP address of the packet, and if found it forwards the packet to the next hop specified in the routing cache entry. Entries are deleted which have not been used for some time. If no entry for the destination is found in the routing cache, the kernel makes a look-up for the destination in the kernel routing (FIB) table using longest prefix matching. If an entry is found in the table, a new entry for the destination is created and inserted into the routing cache, i.e., the kernel routing table is used to populate the routing cache.

In on-demand routing protocols not all routes are known in advance, they must be discovered as they are needed. In such cases a mechanism is required to notify the on-demand routing protocol that a route discovery cycle must take place for the destination, and any packets already being sent to the destination must be queued while the route discovery cycle completes. For the AODV routing daemon to function, it must be determined when to trigger AODV protocol events. Since the IP stack was designed for static networks where link disconnections are infrequent and packet losses are unreported, most of these triggers are not readily available. Therefore, these events must be extrapolated and communicated to the routing daemon via other means. The events that must be determined are:

*1) When to initiate a RREQ:* Route Requests are needed when the IP layer receives a packet to be transmitted to an unknown destination, i.e., a destination with no matching entry in the route table. The problem of the current network stack architecture is that we only know we need a route after the packet has already crossed the boundary between user space and kernel space.

*2) When and how to buffer packets waiting for a route discovery cycle (or for some other reason) to complete:* When an application attempts to send a packet to a destination for which the routing table has not a valid route, the IP layer should buffer the packet for a period of time while a route discovery cycle takes place. If a route is found, the packets should be reinserted into the IP layer and sent to the destination. If a route is not found, the packets should be discarded and the application program should be notified.

*3) When to update the lifetime of an active route:* On-demand routing protocols typically cache a route that has been discovered for a period of time before deleting it if it is inactive. The IP layer therefore must have the capability to notify the routing protocol when an on-demand route has been used, so that the routing protocol can update its timers for the route.

*4) When to generate a route error message if a valid route does not exist for the next-hop IP address of a received packet:* If a data packet is received from another host and there is no valid route to the destination, the node must send a RERR so that the previous hops and the source stop transmitting data packets along this invalid route. Under normal operation of the IP layer on receiving a packet is to send a destination host unreachable ICMP message to the source of the transmission, and silently drop the packet. Instead, the IP layer must give notification to the AODV routing protocol such that it knows it should send a route error message to the original source or the packet.

*5) When to generate a RERR during daemon restart:* When a node reboots, the AODV specification requires that it sends Route Error messages to any nodes attempting to communicate with it up until the end of DELETE_PERIOD seconds. This behavior is required in order to ensure no routing loops occur.

These notification and capabilities are not explicitly present in the protocol stacks of modern operating system. The existing implementations have taken a number of different approaches to solving this problem. The next section describes a number of possible approaches that implementers have taken in Linux.

## IV. Design Stratergies of LINUX

Special emphasis is given on implementation techniques for Linux. We highlight the advantages and disadvantages of each technique and discuss how the implementation techniques and the techniques available for programmers have evolved. The alternatives described in this section are:

- *Kernel Modifications:* Modify the source code of an operating system kernel to produce new API for implementation.
- *Snooping:* Use packet capturing facilities.
- *Netfilter:* Use the packet filter and packet mangling architecture.

The Linux netfilter framework can also partly be attributed to the fact that Linux has become the most popular platform for on-demand ad hoc routing protocol implementation development of the thirteen listed implementations on the AODV web page, eight are for Linux only.

### A. Kernel Modification

Another possibility to determine the AODV events is to modify the networking code of an operating system kernel. Royer and Perkins modified the Linux kernel to support their implementation of the AODV protocol. Such an API would require mechanisms as they would require protocols to register interest with the kernel in the relevant routing events (such as the requirement for a new Route Request). The kernel would then inform the ad-hoc routing protocol when a route Request is required; to initiate route discovery, code is added in the kernel at the point where route lookup failures occur. The kernel source was modified so that a route look-up failure would result in a notification to a user space daemon that was a part of the implementation, it would provide a mechanism to buffer packets for which a route request is being performed and to later reinject them; it would maintain timers associated with the route, etc. Figure 1 shows the architecture of the AODV daemon and the required support logic.

Advantages

- The events are explicitly determined and there are no wasted overhead.
- By modifying the kernel data structures and support code directly, there is no overhead of additional protocol accounting, compared to a user space implementation or even a Linux kernel module.

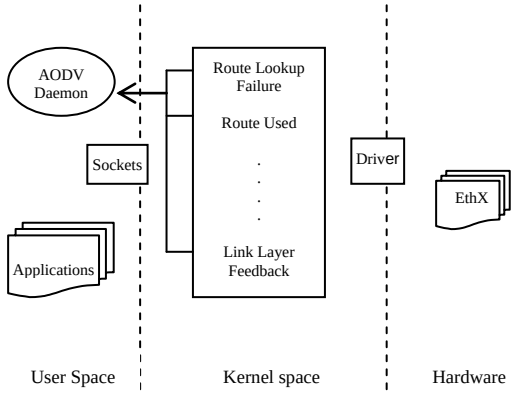


Figure 1. Kernel Modification Architecture

Disadvantages

- *Difficult Installation procedure*: Installation of the necessary kernel modifications requires a complete kernel recompilation.
- *Less Portable:* A drawback of this approach is that it will require major changes to the operating system kernels, and will not be very portable for existing operating system kernels without requiring users to install a new kernel.
- *Difficult to maintain:* Patches (modifications) might only apply cleanly against a certain version of the Linux kernel. There could even be problems with kernels with the same version number as distributions apply their own set of patches to the Linux kernel source.

The first release of the AODV-UCSB implementation used the kernel modifications approach. Desilva and Das also made an implementation of AODV by modifying the Linux kernel and the in-kernel ARP implementation.

### *B. Snooping*

Using code built into the kernel of most operating systems, a user space program can capture all incoming and outgoing packets on a network interface. The process of capturing packets is also known as sniffing or snooping. The code to perform snooping is built into the kernel and is available to user-space programs by using the Packet Capture Library (libpcap). Each packet that is transmitted is passed to the routing daemon using libpcap. When the daemon sees that a packet was transmitted along an active route, the lifetime for that route is updated so that it does not expire, since it is in use. In a similar manner, all the other AODV events may be determined by monitoring the incoming and outgoing packets. By snooping the Address Resolution Protocol (ARP) packets and data packets, AODV can be implemented without any kernel modifications. As such, the routing protocol can be implemented easily in either kernel space or user space. The routing protocol can determine when a route discovery cycle is needed by snooping ARP request packets, as an ARP request is sent to resolve the hardware address for an unknown IP address (if there is an appropriate subnet route entry set up for the correct interface).

The routing protocol can observe incoming and outgoing data packets, and as such can determine when a route is being used, or when a packet is received for which we have no routing information.

Advantage

- *Simple installing and execution:* It does not require any code to run in the kernel-space.

Disadvantages

- *Overhead:* An ARP packet is generated when a node does not know the MAC address of the next hop. Using this inference, if an ARP request packet is seen for an unknown destination and it is originated by the local host, and then a route discovery needs to be initiated. Since route discovery is initiated by outgoing ARP packets, these outgoing packets are unnecessary overhead, and they waste bandwidth.
- *Dependence on ARP:* If the routing table and ARP cache become out of sync, it is possible that the routing protocol may not function properly. For example, if the ARP cache contains an entry for a particular unknown destination, then an ARP packet will not be generated for this destination even though the destination is not known by the routing daemon. Consequently, route discovery will not be initiated. For proper operation the routing protocol must monitor and control the ARP cache in addition to the IP routing table, because disagreement between the two can cause the routing protocol to function incorrectly.

### *C. Netfilter*

Netfilter is a packet filtering framework implemented as a set of hooks at well defined places in the Linux TCP/IP networking stack. Netfilter redirects packet flow through user defined code, which can examine, drop, discard, modify or queue the packets for the user-space daemon. Netfilter is similar to the snooping method; however, it does not have the disadvantage of unnecessary overhead or dependence on ARP.

It consists of a number of hooks in the IP layers that are well-defined points in a packet's traversal of the protocol stack. The IPV4 stack has five hooks. The two hooks NF_IP_LOCAL_OUT and NF_IP_LOCAL_IN are for packets incoming to and outgoing from local processes on the current host. Here a routing decision on what to do with the packet is made. If it is an incoming packet, it may be sent to the NF_IP_FORWARD hook before forwarding, sent up to the

NF_IP_LOCAL_IN hook for delivery to a local process, or dropped. If it is an outgoing packet, it is dropped or sent on the NF_IP_POST_ROUTING hook before being released to the appropriate network interface driver for transmission across the network. Incoming packets also traverse the NF_IP_PRE_ROUTING hook as they enter the IP layer, before being subjected to kernel routing. Thus, packets going from and to other hosts can be captured at these two hooks. Routing decisions are made for packets arriving at the network interface of the host after traversing NF_IP_PRE_ROUTING, to see if they are bound for this host or destined to be forwarded. Routing decisions are made for packets sent by local processes after traversing NF_IP_POST_ROUTING.

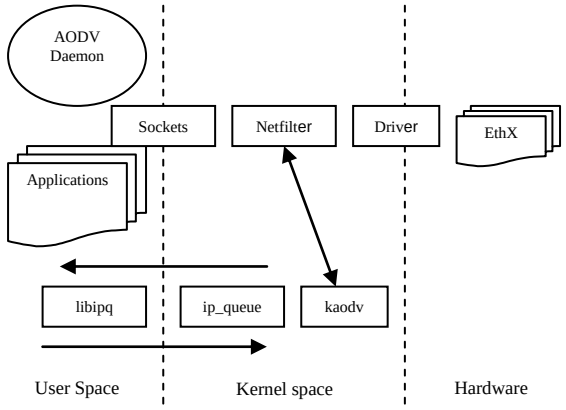


Figure 2. Netfilter Architecture

These functions can show the direction packets travel through the network stack as they enter from a local process or a network interface.

- NF_ACCEPT: allow the packet to pass to the next registered hook
- NF_DROP: have the choice to either discard the packet
- NF_QUEUE: request that these packets be queued for later reinsertion into the IP layer otherwise this is equal to NF_DROP
- NF_STOLEN: grab the packet. We may reinsert the packet at a later point in time
- NF_REPEAT: call this hook again

These hooks are used by the kaodv kernel module. The kernel module driver, ip_queue module is used to queue these packets for the user-space daemon. There the AODV daemon uses a user space library libipq to make control decisions about each packet. Finally packets that are queued (by returning NF_QUEUE) are buffered by the ip_queue driver, typically (though not necessarily) for user space, see figure 2. These packets are handled asynchronously and thus they can be returned to the IP layer at any later time, or discarded.

The Netfilter architecture can be used for firewall filtering (the Linux ip_tables tool uses Netfilter in version 1.4 of the kernel), all kinds of Network Address Translation (NAT) services, or for other advanced packet processing requirements.

Advantages

- *Highly portable*
- *Easy to install*

Disadvantage

- *Requires a kernel module*: A kernel module is more portable than a kernel modification because it depends only on the Netfilter interface. This interface does not change from one kernel version to the next.

## V. Conclusion

AODV is currently one of the most popular ad-hoc routing protocols. These indicate that AODV performs very well both during high mobility and high network traffic load, making it one of the most interesting candidates among today's ad-hoc routing protocols. Implementing a routing protocol is very important to validate its design. Coming up with a clean implementation not only helps better understanding of the protocol nuances, but also allows extensions to explore the protocol design space. In this paper we analysed design possibilities for AODV implementations. We then examined the advantages and disadvantages of three strategies for determining this information. This analysis supported our decision to use small kernel modules with a user-space daemon. We hope that the information in this paper aids researchers in understanding the trade-offs in ad hoc routing protocol implementation development. Further, the description of the design structure and performance of each implementation can assist users in deciding which implementation best fits their needs.

## References

[1] Luke Klein-Berndt, NIST Kernel AODV homepage, http://w3.antd.nist.gov/wctg/aodv_kernel/. September 2003.

[2] Mad-hoc AODV homepage. http://mad-ho.flyinglinux.net/. September 2003.

[3] Luke Klein-Berndt, "Kernel AODV". National Institute of Standards and Technology. http://w3.antd.nist.gov/wctg/aodvkernel. 30 Oct 2008.

[4] Ian Chakeres, UCSB AODV homepage. http://moment.cs.ucsb.edu/AODV/aodv. html, September 2003.

[5] Erik Nordström, UU AODV homepage. http://user.it.uu.se/~henrikl/aodv/, September 2003.

[6] Binita Gupta, UIUC AODV homepage. Including ASL library. http://sourceforge.net/projects/aslib/, September 2003.

[7] E. M. Belding-Royer, "Report on the AODV Interop," University of California Santa Barbara, Tech. Rep. 2002-18, June 2003.

[8] E. Borgia, "Experimental evaluation of ad hoc routing protocols," in Proc. of IEEE PerCom 2005 Workshops, Kauai Island, Hawaii, March, 8–12 2005.

[9] "Running AODV-UU in the Network Simulator NS-2.". https://prj.tzi.org/repos/ dmn/aodv-uu-dtn/trunk/ README.ns. 2 Nov 2008

[10] Erik.Nordstr¨om., AODV-UU. ttp://core.it.uu.se/core/index.php/AODV-UU. Last accessed December 2006.

[11] Ian D. Chakeres and Elizabeth M. Belding-Royer. "AODV routing protocol implementation design". In ICDCSW '04: Proceedings of the 24th International Conference on Distributed Computing Systems Workshops - W7: EC (ICDCSW'04), pages 698–703, Washington, DC, USA, 2004. IEEE.

[12] Douglas E. Comer., "Internetworking with TCP/IP: Principles, Protocols, and Architecture". Prentice-Hall, Inc., Upper Saddle River, NJ, USA, fourth edition, 2000.

[13] Saman Desilva and Samir R. Das., Experimental evaluation of a wireless ad hoc network. In Proceedings of the 9th Int. Conf. on Computer Communications and Networks (IC3N), pages 528–534, Las Vegas, NV, USA, October 2000.

[14] Nova Engineering, "NovaRoam," http://www.novaroam.com/.

[15] C. E. Perkins and E. M. Royer, "The Ad hoc On-Demand Distance Vector Protocol," in *Ad hoc Networking*, C. E. Perkins, Ed. Addison-Wesley, 2000, pp. 173–219.

[16] C. E. Perkins, E. M. Belding-Royer, and S. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing," *RFC 3561*, July 2003.

[17] E.M. Royer & C.E Perkins, "An Implementation Study of the AODV Routing Protocol", Proceedings of the IEEE Wireless Communications and Networking Conference, Chicago, IL, September 2000.

[18] Netfilter homepage. http://www.netfilter.org/. September 2003.

[19] J. Kadlecsik, H. Welte, J. Morris, M. Boucher, and R. Russell, "The netfilter/iptables Project," http://www.netfilter.org/.

[20] IEEE Computer Society, "IEEE 802.11 Standard, IEEE Standard For Information Technology," 1999.

[21] J. Tourrilhes, "Wireless Tools for Linux,". http://www.hpl.hp.com/personal/Jean Tourrilhes/Linux/Tools.html.

[22] H. Lundgren, D. Lundberg, J. Nielsen, E. Nordstrm, and C. F. Tschudin, "A Large-scale Testbed for Reproducible Ad hoc Protocol Evaluations," in IEEE Wireless Communications and Networking Conference 2002 (WCNC), March 2002.

[23] V. Kawadia, Y. Zhang, and B. Gupta, "System Services for Implementing Ad-Hoc Routing: Architecture, Implementation and Experiences," in Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys), San Francisco, CA, June 2003, pp. 99–112.

[24] G. Bianchi, "Performance Analysis of the IEEE 802.11 Distributed Coordination Function," IEEE Journal Selected Areas in Communications, vol. 18, March 2000.

AUTHORS PROFILE

**Prinima Gupta** is presently working as Asst. Professor in MCA Department, Manav Rachna College of Engineering, Faridabad. She has completed Master of Computer Application from Kurukshetra University, Kurukshetra, M.Phil from Vinayaka Mission University, Tamil Nadu and presently doing PhD in Computer Science. She has 5+ years of teaching experience. She published 03 papers in National conferences. Her area of specialization includes Computer Networks and Computer Architecture

**Prof. (Dr.) R. K Tuteja** is presently working as Director (Academics) in NCICS, Israna, Panipat. He has 45 years of teaching experience. He was successfully guided 30 PhD 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.