AN IMPLEMENTATION POSSIBILITIES FOR AODV ROUTING PROTOCOL IN REAL WORLD

Nitiket N Mhala[1] and N K Choudhari [2]

[1]Associate Professor, Department of Electronics Engg., BDCOE, Sevagram,India
nitiket_m@rediffmail.com
[2]Principal, Bhagwati Chadurvedi COE, Nagpur,India
drnitinchoudhari@gmail.com

ABSTRACT

Most of the adhoc routing protocol research work has been done using simulation only because of the difficulty of creating real implementation. In simulation the developer controls the whole system, which is in effect only a single component. An Implementation, on the other hand, needs to interoperate with a large complex system and the system components. In this paper we focus on working implementation of AODV routing protocol by means of certain design possibilities and possible opportunities for obtaining needed AODV events. We discuss the socket based mechanism particularly when AODV routing daemon communicates changes to the IP route table. The paper suggests the need of implementation of Generic Netlink Family.

KEYWORDS

AODV, netlink, routing daemon, Linux kernel, ***snooping,*** *netfilter*

1. INTRODUCTION

Mobile wireless devices are rapidly gaining popularity due to recent improvements in the portability and power of these products. There is a growing need for communication protocols which allow users of these devices to communicate over wireless links. To allow such on-the-fly formation of networks, the Ad hoc On-Demand Distance Vector (AODV) routing protocol has been developed [1], [2], [3]. AODV has been designed for use in ad hoc mobile networks. It allows users to find and maintain routes to other users in the network whenever such routes are needed. Testing mobile wireless protocols in a real-world environment presents numerous difficulties. These difficulties include creating repeatable scenarios with tens, hundreds, or even thousands of mobile nodes. Creating multiple scenarios with only small variances is also quite challenging. Because of these difficulties, simulations of AODV have been created to test the protocol in a variety of repeatable scenarios [1] [3]. However, while simulating a protocol can aid in the basic design and testing of the protocol, certain assumptions and simplifications can be made in a simulation that are not valid in a real-world scenario. Hence, it is important to implement the protocol, once the simulation is complete.

Creating a working implementation of an ad hoc routing protocol is non-trivial and more difficult than developing a simulation. In simulation, the developer controls the whole system, which is in effect only a single component. An implementation, on the other hand, needs to interoperate with a large, complex system. Some components of this system are the operating system, sockets, and network interfaces. Additional implementation problems surface because current operating systems are not built to support ad hoc routing protocols. A number of required events are unsupported; support for these events must be added. Because these events encompass many system components, the components and their interactions must also be explored. For these rea-

sons it takes significantly more effort to create an ad hoc routing protocol implementation than a simulation.

2. RELATED WORK

Conventional IP based routing protocols are not appropriate for ad hoc mobile networks because of the temporary nature of the network links and additional constraints on mobile nodes i.e. limited bandwidth and power [13, 14].Routing protocols for such environments must be able to keep up with the high degree of node mobility that often changes the network topology drastically and unpredictably. The mobile ad hoc networking (MANET) working group has been formed within the IETF to develop a routing framework for IP based protocols in mobile ad hoc networks. AODV (Ad hoc On-demand Distance Vector routing) is one such protocol, which is widely established. The AODV has been published as an experimental RFC [15].AODV is a widely researched protocol among the research community. Most of the research effort has focused on simulations aimed at determining the performance of AODV [16,17] also in comparison to the performance of other ad hoc routing protocols [18].There exist currently several AODV implementations [19] for different operating systems. These implementations comply with a varying degree to the protocol description defined in [15]. Even though all are considered protocol compliant, different design decisions (e.g., kernel level implementations perform efficiently, compared to user level implementations) can give certain protocol handlers an advantage over others. Recently there have been many AODV routing protocol implementations, including Mad-hoc [20],AODV-UCSB [21], AODV-UU [22], Kernel-AODV [23] and AODV-UIUC [24]. Each implementation was developed and designed independently; but, they all perform the same operations and many interoperate.

2. 1Brief AODV Protocol Overview

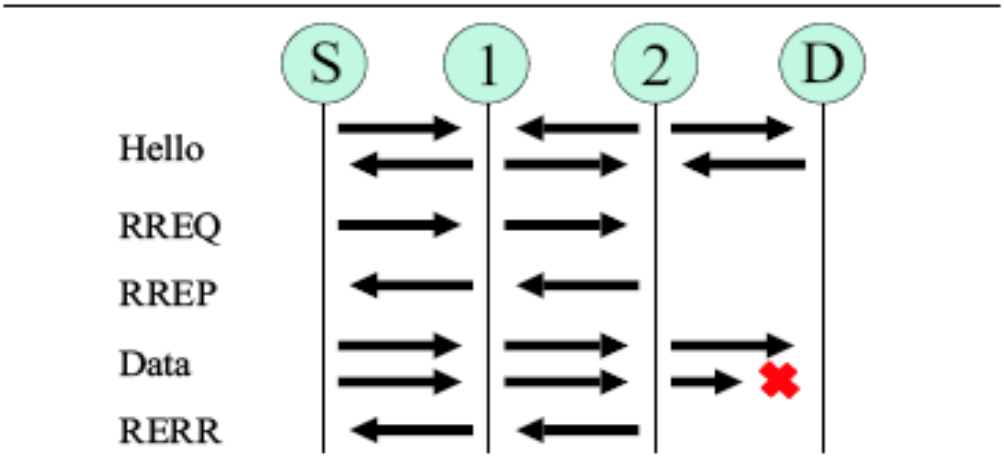

Figure 1. AODV Protocol Messaging.

The AODV routing protocol [11][12] is a reactive routing protocol; therefore, routes are determined only when needed. Figure 1 shows the message exchanges of the AODV protocol. 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. At each intermediate node, when a RREQ is received a route to the source is created. If the receiving node has not received this 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 can begin 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) 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.

2.2 Logical Structure of Implementation

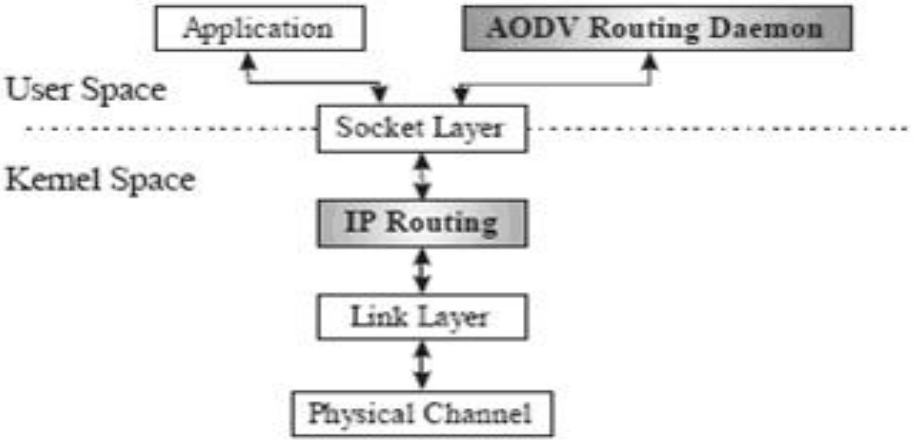

Figure 2. Implementation Structure

While implementation in real world, certain changes are necessary both protocol and kernel in order to allow to operate AODV correctly. The figure highlights where the modifications can occur. We can choose to implement AODV in Linux kernel because of inherent mobility and open loop characteristics of Linux.Similarly; the alternative to implementing a routing protocol in user space is to incorporate the existing protocol into the existing kernel as in, [4]. Protocol modifications suggest most basic changes made to AODV in route Reply and Route Table. When a node receives a route request, it replies if it either is the destination or it has a current route to the destination. In the simulation, RREPs were originally unicast from responding node to the source. As the RREP was propagated, intermediate nodes update their routing tables to include the routes to the destination. In implementation however this does not work, because if RREP is unicast from responding node to the source, the intermediate nodes use IP forwarding and do not process the packet. Hence the protocol needed to be changed so that RREPs are un-

icast on a hop-by-hop basis. Additionally, a source IP address field was added to the RREP so that ultimate destination of the RREP would be retained. .Similarly, AODV routing daemon communicates changes to the IP routing table through the use of netlink socket. Whenever AODV has route addition, modification or deletion, it transmits a message to IP through this socket and route is updated accordingly.

2.3 Implementation Strategy

In order to function for AODV routing daemon, it is essential to determine when to trigger AODV protocol events. The events must be extrapolated and communicated to the routing daemon via other means. The events that must be determined are

When to initiate a route request:
This is indicated by a locally generated packet that needs to be sent to a destination for which a valid route is not known.

When and how to buffer packets during route discovery:
During route discovery packets destined for the unknown destination should be queued. If a route is found the packets are be sent.

When to update the lifetime of an active route:
This is indicated by a packet being received from, sent to or forwarded to a known destination.

When to generate a RERR if a valid route does not exist:
If a data packet is received from another host and there is no known route to the destination, the node must send a RERR so that the previous hops and the source halt transmitting data packets along this invalid route.

When to generate a RERR during daemon restart:
After the AODV routing protocol restarts, it must send a RERR message to other nodes attempting to use it as a router. This behavior is required in order to ensure no routing loops occur.

3. IMPLEMENTATION DESIGN POSSIBILITIES

3.1 Possible opportunities for obtaining the said events include

- Snooping
- Netfilter
- Kernel Modification

3.1.1 Snooping

In order to determine the needed events is to promiscuously snoop all incoming and outgoing packets.[5] The code to perform snooping is built into the kernel and is available to user space programs. For e.g. An ARP packet is generated when a node does not know the MAC layer address of the next hop. Using this interface, if an ARP request packet is seen for an unknown destination and it is originated by the local host, then a route discovery needs to be initiated.Similarly, all other AODV events may be determined by monitoring incoming and outgoing packets. The most important advantage of this solution is it does not require any code to run in the kernel space. Hence it allows for simple installation and execution .But two disadvantages are overhead and dependence over ARP.

3.1.2 Netfilter

Netfilter [6] is a set of hooks at a various points inside the Linux protocol stack. Netfilter redirects packet flow through user defined code, which can examine, drop, discard, modify or queue the packets for user space daemon. Using Netfilter is similar to snooping method however it does not have the disadvantage of unnecessary overhead or dependence on ARP. This solution has the strength such as there is no unnecessary communication; it is highly portable, it is easy to install and user space daemon can determine all the required events. On the other hand, the disadvantage of this solution is that it requires a kernel module. However kernel module is easier than kernel modifications. A kernel module is more portable than kernel modifications because it depends only on the Netfilter interface. This interface does not change from one kernel version to next.

3.1.3 Kernel Modification

In order to determine the AODV events is to modify the kernel. Code can be placed in the kernel to communicate the events to an AODV user-space daemon. For example, to initiate route discovery, code is added in the kernel at the point where route lookup failures occur. Given this code in the kernel, if a route lookup failure happens, then a method is called in the user-space daemon.

Figure shows the architecture of the AODV daemon and the required support logic. The advantages of this solution are that the events are explicitly determined and there are no wasted overhead. The main disadvantages of this solution are user installation and portability. Installation of the necessary kernel modifications requires a complete kernel recompilation. This is a difficult procedure for many users. Also, kernel patches are often not portable between one kernel version and the next. Finally, understanding the Linux kernel [7] and network protocol stack requires examining a significant amount of uncommented, complex code.

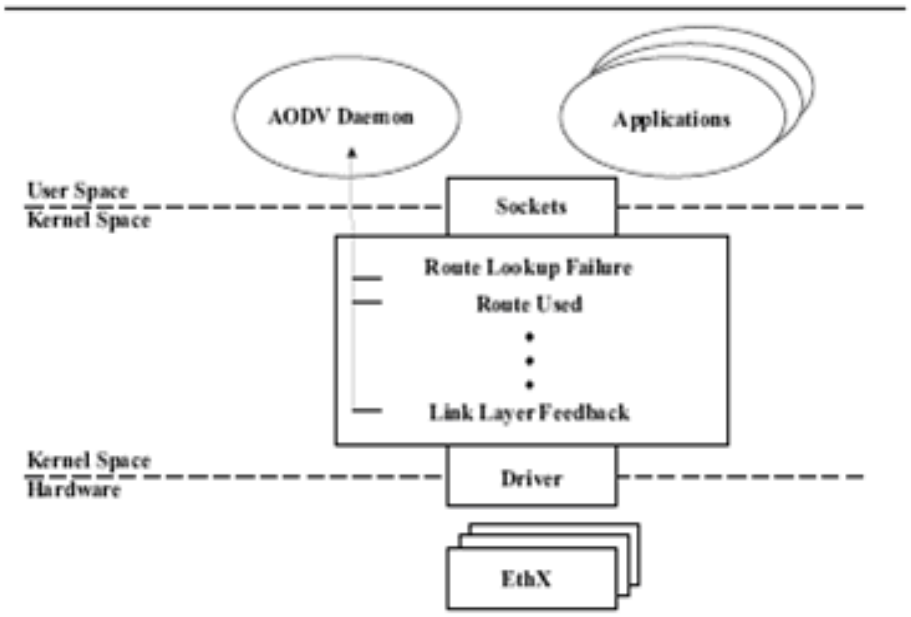

Figure 3. Kernal Modification Architecture

3.2 Basic Considerations in kernel modifications to meet out the required aodv Events

3.2.1 IP routing

When a packet arrives at a node's IP-Layer from the application layer, Ip checks whether it has a route to the destination by consulting its routing table. If it has either a route or a default router, it forwards the packet. If neither of these exists, IP informs the application that a route does not exist, and the session is aborted. In adhoc routing, default routes typically does not exist, except possibly for specific connections to an infrastructure. Often, due to node mobility and specifically with on-demand protocols, a valid route is not known for a given destination. Instead of notifying the application, IP must be changed to notify the routing daemon that route needs to be found for the destination.

3.2.2 Route Tables

AODV maintains its own route table of destination for which it has a route. Each route table entry has associated with it a lifetime field. When an entry's lifetime expires, that entry is invalidated. Each time a route to a destination is used, the life time associated with that route is updated so that the route table entry is not prematurely deleted. Because IP is responsible for forwarding data packets, however, AODV does not know when route entries are used. Hence it cannot accurately use this feature within the routing daemon. To enable this functionality of AODV to be maintained , the kernel can be modified so that IP maintains a structure parallel to the IP route table in which it stores a Last use field.

4. SOCKET BASED MECHANISM

The socket based mechanisms allow the applications to listen on a socket, and the kernel can send those messages at any time. This leads to a communication mechanism in which user space and kernel space are equal partners.

4.1 Netlink Sockets

Netlink is a special IPC used for transferring information between kernel and user space processes, and provides a full-duplex communication link between the Linux kernel and user space. It makes use of the standard socket APIs for user-space processes, and a special kernel API for kernel modules. Netlink sockets use the address family AF_NETLINK, as compared to AF_INET used by a TCP/IP socket.

Why Netlink sockets?

It is simple to interact with the standard Linux kernel as only a constant has to be added to the Linux kernel source code. There is no risk to pollute the kernel or to drive it in instability, since the socket can immediately be used.

- Netlink sockets are asynchronous as they provide queues, meaning they do not disturb kernel scheduling.
- Netlink sockets provide the possibility of multicast.
- Netlink sockets provide a truly bidirectional communication channel: A message transfer can be initiated by either the kernel or the user space application.
- They have less overhead (header and processing) compared to standard UDP sockets.

4.2 Role of Socket mechanism in AODV Implementation

AODV can be implemented as a routing daemon in use space. The daemon communicates with the Linux kernel through the use of sockets. At initialization, AODV opens a UDP socket to the kernel. This socket is used for both the transmission and reception of AODV control messages. AODV has been issued port number 654 and hence binds to this port when opening the socket.

The AODV routing daemon communicates changes to the IP route table through the use of net-link socket.[8] Whenever AODV has a route addition, modification or deletion, it transmits a message to IP through this socket and the route is updated accordingly .In order to prevent the premature deletion of routes in the kernel routing table, AODV's route table maintenance is altered to include a periodic refresh of the kernel route table entries .This may be accomplish through the use of periodic timer. When the timer expires, AODV sends a message to IP on the netlink socket telling it to update, or refresh, the route.

4.3 Identified drawbacks

- Each entity using netlink sockets has to define its own protocol type (family) in the kernel header file include/linux/netlink.h, necessitating a kernel re-compilation before it can be used.
- The maximum number of netlink families is fixed to 32. If everyone registers its own protocol this number will be exhausted.

4.4 Implementation of Generic Netlink Family

In order to eliminate the above two drawbacks ,We suggest to implement "Generic Netlink Family" It acts as a Netlink multiplexer, in a sense that different applications may use the generic netlink address family. Generic Netlink communications are essentially a series of different communication channels which are multiplexed on a single Netlink family. Communication channels are uniquely identified by channel numbers which are dynamically allocated by the Generic Netlink controller. Kernel or user space [9] users which provide services, establish new communication channels by registering their services with the Generic Netlink controller. Users of the service, then query the controller to see if the service exists and to determine the correct channel number. Each generic netlink family can provide different "attributes" and "commands". Each command has its own callback function in the kernel module and may receive messages with different attributes. Both commands and attributes, are "addressed" by an identifier

4.5 User Space Sending Phase

1. Create a socket
2. Connect to the NETLINK_GENERIC socket family.
3. Resolve the ID for the particular generic netlink family we want to talk with.
4. Create the generic netlink message header. This specifies which callback function of your kernel module gets executed.
5. Put the data into the message. The second argument is used by the kernel module to distinguish which attribute was sent.
6. Send the message to the kernel

4.6 Receiving Phase

1. Add a callback function to the socket. This callback function gets executed when the socket receives a message. In the callback function the message needs to be decoded.
2. Wait until a message is received.

4.7 Recommendations

The Generic Netlink mechanism is a very flexible communications mechanism and as a result there are many different ways it can be used. The following recommendations are based on conventions within the Linux kernel and should be followed whenever possible. While not all existing kernel code follows the recommendations outlined here all new code should consider these recommendations [10] as requirements.

4.7.1 Attributes and Message Payloads

The Netlink attribute mechanism has been carefully designed to allow for future message expansion while preserving backward compatibility. There are also additional benefits to using Netlink attributes which include developer familiarity and basic input checking

4.7.2 Operation Granularity

While it may be tempting to register a single operation for a Generic Netlink family and multiplex multiple sub-commands on the single operation this is strongly discouraged for security reasons. Combining multiple behaviors into one operation makes it difficult to restrict the operations using the existing Linux kernel security mechanisms

4.7.3 Acknowledgment and Error Reporting

It is often necessary for Generic Netlink services to return an ACK or error code to the client.

5. CONCLUSION

AODV is a widely researched protocol among the research community. Most of the research effort has focused on simulations aimed at determining the performance of AODV. However, while simulating a protocol can aid in the basic design and testing of the protocol, certain assumptions and simplifications can be made in a simulation that are not valid in a real-world scenario. Currently several AODV implementations exist for different operating systems. In real sense, creating a working implementation of an ad hoc routing protocol is non-trivial and more difficult than developing a simulation. In view of this, our paper efforts implementation possibilities for AODV routing protocol in a real world. Here, we first identified the unsupported events needed for AODV to perform routing. We then examined the advantages and disadvantages of three strategies for determining this information. Among the three implementation strategies, we emphasizes on kernel modifications strategy. This paper strongly interpreted on the basic considerations should be carefully considered for kernel modifications to meet out the required AODV events. Therefore, we briefly introduced Socket Based Mechanism and explain its major role in AODV implementation which is most important when AODV routing daemon communicates changes to the IP route table through the use of netlink socket. In conclusion for future kernel modification, we suggest the implementation of Generic Netlink Family.

ACKNOWLEDGEMENTS

The authors would like to thank everyone, including the anonymous reviewers.

REFERENCES

[1] C.E Perkins and E.M. Royer. Ad-hoc On-Demand Distance Vector Routing.Proceeding of the

2nd IEEE Workshop on mobile Computing systems and Applications,pages 90-100.New Orleans,L.A.February 1999.

[2] C.E Perkins and E.M. Royer and S.R. Das.Ad-hoc On Demand Distance Vector (AODV) routing.IETF Internet Draft,draft-ietf-manet-aodv-05.txt,March 2000.

[3] E.M. royer and C.E. Perkins Multicast Operation of Adhoc distance vector Routing Protocol. Proceedings of the 5th ACM/IEEE International Conference on Mobile Computing and Net working,pages 207-218 ,Seattle,WA,august,1999.

[4] D.A. Maltz ,J.Broach and D.B.Jonson Experiences Designing and Building, a Multihop Wire less Adhoc Test Bed Technical Report CMU, CMU school of computer science

[5] F. Lilieblad, O. Mattsson, P. Nylund, D. Ouchterlony, and A. Roxenhag. Mad-hoc AODV Implementation and Documentation. http://mad-hoc.flyinglinux.net.

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

[7] http://www.kernel.org

[8] www.linuxjournal.com/article/

[9] http://people.ee.ethz.ch/~arkeller/linux/kernel_user_space_howto.html

[10] http://lwn.net/Articles/

[11] C.E.Perkins,E.M. Belding-Royer and S.Das. Adhoc On Demand Distance vector (AODV) Routing.RFC3561,July 2003.

[12] C.E.Perkins & E.M. Royer,The Adhoc On Demand Distance Vector Protocol In C.E Perkins,editor,Adhoc networking pages 173-219.Addision-Westly,2000.

[13] Perkins, C.E., *Ad Hoc Networking*. 2000, NJ: Addison-Wesely.

[14] Royer, E.M. and C.-K. Toh, *A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks.*IEEE Personal Communications Magazine, 1999: p. 46-55.

[15] Perkins, C., B.-R. E., and D. S., *Ad hoc On-demand Distance Vector routing*. 2003, Request For Comments (Proposed Standard) 3561, Internet Engineering Task Force.

[16] Lee, S.J., E.M. Royer, and C.E. Perkins, *Scalability Study of the Ad Hoc on-Demand Distance Vector Routing.* ACM/Wiley International Journal of Network Management, 2003: p. 97-114.

[17] Ramarathinam, V. and M.A. Labrador. *Performance Analysis of TCP over Static Ad Hoc Wire Less Networks*. in *In Proceedings of the ISCA 15th International Conference on Parallel and Distributed Computing Systems (PDCS)*. 2002.

[18] Samir R. Das, et al., *Performance Comparison of Two On-demand routing Protocols for Ad hoc Networks.*IEEE Personal Communications Magazine special issue on Ad hoc Networking, 2001: p. 16-28.

[19] Implementations, P.A., *available at http://moment.cs.ucsb.edu/AODV/aodv.html #Implementa tions*. 2002.

[20] F. Lilieblad, O. Mattsson, P. Nylund, D. Ouchterlony, and A. Roxenhag. Mad-hoc AODV Imple mentation and Documentation. http://mad-hoc._yinglinux.net.

[21] I. D. Chakeres. AODV-UCSB Implementation from University of California Santa Barbara. http://moment.cs.ucsb.edu/ AODV/aodv.html

[22] H. Lundgren, D. Lundberg, J. Nielsen, E. Nordstr¨om, and C. F. Tschudin. A Large-scale Testbed for Reproducible Ad hoc Protocol Evaluations. In *IEEEWireless Communications*

and Networking Conference 2002 (WCNC), March 2002.

[23] L. Klein-Berndt. Kernel AODV from National Institute of Standards and Technology (NIST). http://w3.antd.nist.gov/wctg/aodv kernel/.

[24] V. Kawadia, Y. Zhang, and B. Gupta. System Services for Implementing Ad-Hoc Routing: Archi tecture, Implementation and Experiences. In *Proceedings of the* 1st *International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 99.112, San Francisco, CA, June 2003.

Authors

Mr. Nitiket N. Mhala is PhD student and also working as Associate Professor & Head in the Department of Electronic Engineering, Sevagram, India. He received his ME Degree from RM Institute of Research and Technology, Badnera, Amravati University and BE Degree from Govt. College of Engineering, Amravati, Amravati University. He published a Book Entitled PC Architecture and Maintenance. He published research papers at National and International level. He is a member of Institute of Electronics and Telecommunication Engineer (IETE). His area of interest spans Data communication, Computer network and Wireless Ad hoc networks.

Dr. N. K. Choudhari is a Professor and completed his Ph.D degree in Electronics Engineering from J.M.I., New Delhi and received his M.Tech in Electronics Engineering from Visveswaraya regional Engineering College, Nagpur. He received his BE in Power Electronics from B.D.C.O.E., Sevagram. Presently he is Principal at Smt.Bhagwati Chaturvedi COE, Nagpur, India. He is guiding few research scholars for persuing Ph.D degree in RTM Nagpur University, Nagpur, India. He has worked as members of different advisory committees and was a member of Board of Studies Electronics Engg. of RTM Nagpur University, Nagpur, India.