Exploration and Comparison of Several AODV Implementations: A Survey

Abdallah Rasheed & Khader Mohammad
Division of Computing Studies, Arizona State University
aarashid@asu.edu, hajkhader@gmail.com

ABSTRACT

An ad-hoc network consists of mobile nodes constructing a network that the topology changes dynamically. 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. This paper is an exploration and comparison of several AODV implementations including: Mad-hoc, AODV-UCSB, AODV-UU, Kernel-AODV and AODV-UIUC. It gives details on how AODV works. In addition, this paper presents a discussion of the advantages as well as disadvantages of each implementation. Also, it explains implementation techniques of the AODV protocol to determine the needed events, such as: Snooping, Kernel Modification, and Netfilter. AODV-UU has been used as a routing agent for mobile nodes in the Network Simulator NS-2. Because of AODV-UU is publicly recommended implementation of AODV and more portable than the other implementations, I choose to install and configure AODV-UU implementation and run it in Network Simulator NS-2.

***Keywords*:** Ad-hoc Networking, AODV, AODV-UU, NS-2,Unicasting, Multicasting.

I. INTRODUCTION

The Ad hoc On-Demand Distance Vector (AODV) routing protocol is designed for mobile ad hoc networks (MANETs) and other wireless ad-hoc networks with large numbers of mobile nodes. The protocol's algorithm creates routes between nodes only when the routes are requested by the source nodes, giving the network the flexibility to allow nodes to enter and leave the network as will. Routes remain active only as long as data packets are traveling along the paths from the source to the destination. If the source stops sending packets, the path will time out and close. AODV was developed at the Nokia Research Center of University of California, Santa Barbara and University of Cincinnati by C. Perkins and S. Das [1].

AODV supports both Unicast and Multicast routing. Basically, the difference between Unicast and Multicast is that Unicast is a Communication that takes place over a network between a single sender and a single receiver. In contrast, Multicast is referred to the process of sending a message to a select group. For example, sending an e-mail message to a mailing list and videoconferencing are regarded as multicasting.

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. In contrast, the most common routing protocols of the Internet are proactive, meaning they find routing paths independently of the usage of the paths. Additionally, AODV forms trees which connect multicast group members. The trees are composed of the group members and the nodes needed to

connect the members. AODV uses sequence numbers to ensure the freshness of routes.

II. How AODV works

AODV utilizes routing tables to store routing information; one routing table for uncast routes as well as one for multicast routes. These tables hold information like: destination address, next-hop address, hop count, destination sequence number, and life time.

AODV discovers routes as needed and when it is necessary, which means no need to maintain routes from every node to all other nodes. And routes should be maintained as long as it's necessary. AODV nodes have four types of messages to communicate between each other:

- Route Request (RREQ)
- Route Reply (RREP).
- Route Error (RERR)
- HELLO messages.

RREQ and RREP messages are used for route discovery, whereas RERR and HELLO messages are used for route maintenance. The following flow chart illustrates the basic AODV protocol:

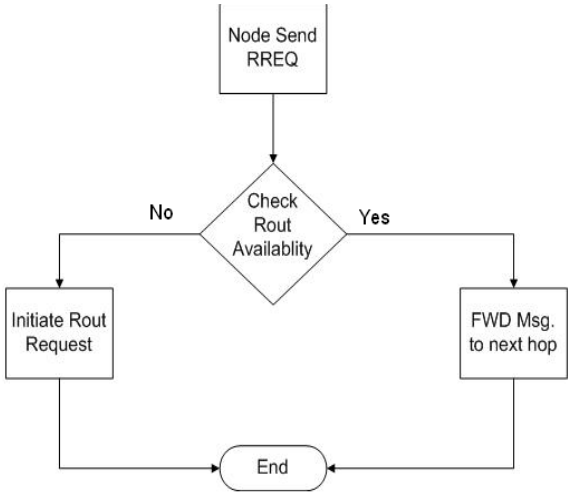


Figure 1: Basic AODV protocol.

When a source node (S) wishes to send a packet to a destination node (D), first it checks its routing table to determine if that node has a route to the destination node. If it has, then the packet is forwarded to the next hop node. If it doesn't have a route, then it initiates the AODV- Route Discovery steps, which is explained in more details in the following:

- AODV builds routes using a route request/reply query. For example, when a source node (S) wants a route to a destination node (D) and it does not already have a route, it broadcasts a route request packet across the network, which is called (RREQ).

- The packet sent from S node to D node contains: the source node's IP address, destination node's IP address , source node's sequence number, destination node's sequence number, and broadcast ID number. Each time a source node uses RREQ, broadcast ID number will be incremented.

- Depending on the information carried by the packet sent from S node to D, nodes receiving this packet update their information for the source node and set up backwards pointers to the source node in the routing tables. RREQ also contains the most recent sequence number for the destination of which the source node is aware. A node receiving the RREQ may send RREP if and only if it is either the destination or if it has a route to the destination with corresponding sequence number greater than or equal to that contained in the RREQ. If so, it unicasts a RREP back to the source. Otherwise, it rebroadcasts the RREQ. Nodes keep track of the RREQ's source IP address and broadcast ID. If they receive a RREQ which they have already processed, they discard the RREQ and do not forward it.

- Then RREP propagates back to the source, hence nodes set up forward pointers to the destination. Once the source node receives the RREP, it may begin to forward data packets to the destination. If the source later receives a RREP that has a greater sequence number or contains the same sequence

number with a smaller hop count, it updates its routing information in the routing table for that destination and begin using the better route.

- The route will continue to be maintained as long as the route remains active, and the route is considered to be active as long as there is packet traffic periodically between the source and the destination node. If the source node stops sending data packets, then the links will time out and it will be deleted from the intermediate node routing tables. If a link break occurs while the route is active, the node upstream of the break propagates a route error (RERR) message to the source node to inform it of the now unreachable destination. After receiving the RERR, if the source node still desires the route, it can reinitiate route discovery.

- To summarize the previous steps see Figure 2 that shows the basic functioning of AODV.

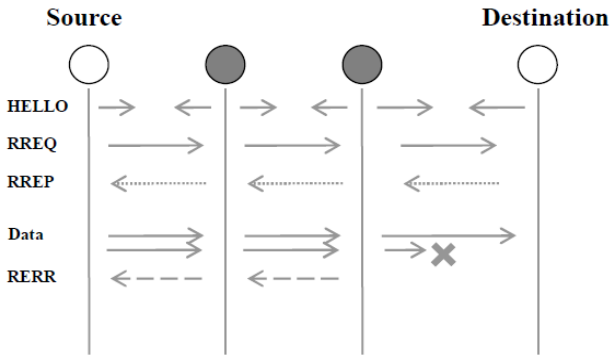


Figure 2: Basic AODV Functioning [5]

Multicast Route discovery:

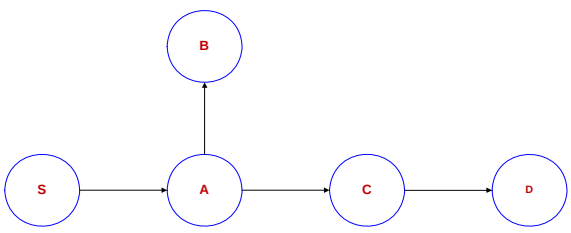
- Multicast routes discovery are pretty much the same as the unicast. If a node (S) wishes to join a multicast group, it broadcasts a RREQ with the destination IP address of the multicast group and with the flag (join) set to indicate that it would like to join the group.

- Then any node receiving this RREQ that is a member of the multicast tree that has a fresh enough sequence number for the multicast group may send a RREP. As the RREPs propagate back to the source, the nodes forwarding the message set up pointers in their multicast route tables. As the source node receives the RREPs, it keeps track of the route with the freshest sequence number, as well as the smallest hop count to the next multicast group member.

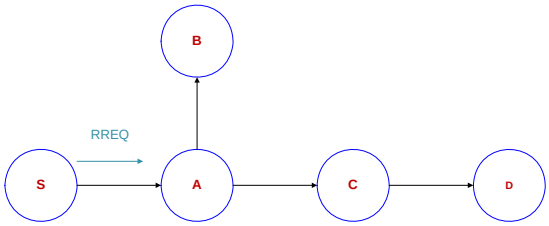
- After a specified discovery period, the source node will unicast a Multicast Activation (MACT) message to its selected next hop. This message serves the purpose of activating the route. A node that does not receive this message that had set up a multicast route pointer will timeout and delete the pointer. If the node receiving the MACT was not already a part of the multicast tree, it will also have been keeping track of the best route from the RREPs it received. Hence it must also unicast a MACT to its next hop, and so on until a node that was previously a member of the multicast tree is reached.

The following example explains AODV Route Discovery Algorithm:

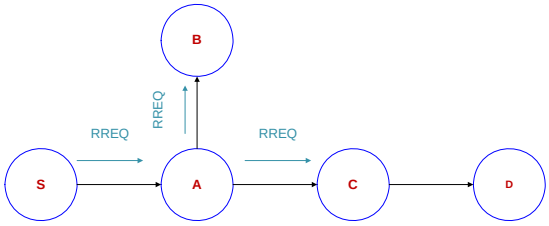
1. Node S needs a route to D
2. Creates a Route Request (RREQ)
   - Enters D's IP addr, seq#, S's IP addr, seq#, hopcount=0



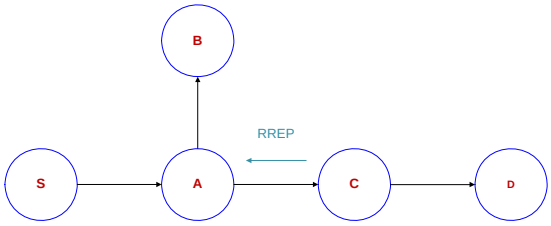
3. Node S broadcasts RREQ to neighbors



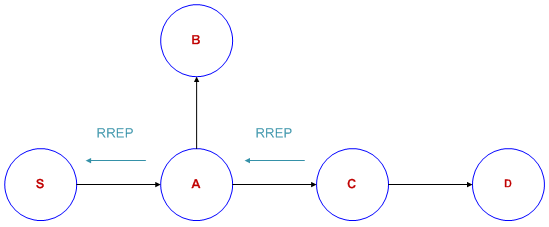
4. Node A receives RREQ
   - Makes a reverse route entry for S dest=S, nexthop=S, hopcount=1
   - It has no routes to D, so it rebroadcasts RREQ



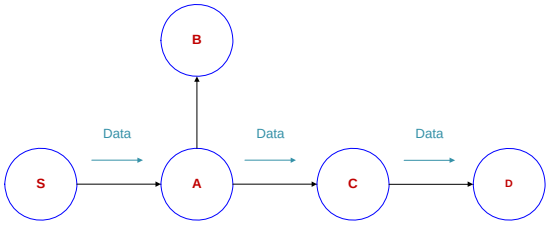
5. Node C receives RREQ
   - Makes a reverse route entry for S dest=S, nexthop=A, hopcount=2
   - It has a route to D, and the seq# for route to D is >= D's seq# in RREQ
   - C creates a Route Reply (RREP), enters D's IP addr, seq#, S's IP addr, hopcount to D=1
   - Unicasts RREP to A



6. Node A receives RREP
   - Makes a forward route entry to D dest=D, nexthop=C, hopcount=2
   - Unicasts RREP to S



7. Node S receives RREP
   - Makes a forward route entry to D dest=D, nexthop =A, hopcount = 3
   - Sends data packet on route to D



III. Finding and obtaining various AODV implementations

There are various AODV routing protocol implementations, such as Mad-hoc, AODV-UCSB, AODV-UU, Kernel-AODV, and AODV-UIUC [4]. All of the implementations use HELLO messages to determine local connectivity and detect link breaks. In addition, all implementations (except Mad-hoc) support the expanding ring search and local repair optimizations.

Despite the fact that all of the AODV implementations share the same operations, each AODV implementation was developed independently. Hence, there are some differences between these implementations, which will be explained in detail in the following paragraphs:

Mad-hoc was the first AODV implementation, and it was written in Java. Mad-hoc is a metropolitan mobile ad hoc network simulator. It allows the simulation of large networks and

features realistic models for node mobility and propagation of radio waves. Mad-hoc focus on the highest network layers. It was designed to provide a wireless network simulator that is flexible enough so as it can be used in a variety of contexts. Following are some example that Mad-hoc has been successfully applied to perform:

- Multi-objective optimization of broadcast protocols.
- IEEE802.11b network emulation.
- Studies on mobile agents.
- Generation of dynamic graphs.
- Co-evolutionary optimization of topology control protocols for hybrid ad hoc networks.

The Mad-hoc implementation resides completely in user-space and uses the snooping strategy to determine AODV events. Mad-hoc has bugs that cause it to fail to perform properly, and it doesn't have queuing of data packets while route discovery is in process [6].

The first release of **AODV-UCSB** (University of California, Santa-Barbara) used the kernel modification strategy. Then after the Netfilter implementation was done, AODV-UCSB was updated to use Netfilter.

AODV-UU has the same design as AODV-UCSB. The main protocol logic resides in a user-space daemon, it uses kernel modules to utilize the Netfilter. In addition, AODV-UU (Uppsala University) includes Internet gatewaying support as well as multiple interface support. AODV-UU has been simulated in real world environment using NS2 simulator[4].
The throughput achieved by AODV-UU is higher than that of AODV-UCSB, and that latency associated with AODV-UCSB is greater than AODV-UU. Thus AODV-UU seems to perform better than AODV-UCSB [2].

The **AODV-UIUC** implementation is similar to AODV-UCSB and AODV-UU except it explicitly separates the routing and forwarding functions. Routing protocol logic takes place in the user-space daemon, while packet forwarding is handled in the kernel. This is efficient because forwarded packets are handled immediately and fewer packets traverse the kernel to user-space boundary.

Kernel AODV is a loadable kernel module for Linux. It implements AODV routing between computers equipped with WLAN devices. No need for user-space daemon because it uses Netfilter, and it places the routing protocol logic inside the kernel module. At the same time no need to traverse packets from kernel to user-space, which improves the performance of the implantation. Kernel AODV supports Internet gatewaying, multiple interfaces and a basic multicast protocol [3].

IV. AODV PROTOCOL ARCHITECTURES

This section explains the ways of designing the AODV protocol, which are listed in details bellow:

- **Snooping:**

  To determine the needed events all incoming and outgoing packets should be snooped. Snooping can be performed by the code which built into the kernel and is available to user-space programs. The most important advantage of this solution is it does not require any code to run in the kernel-space. Hence, this solution is simple in terms of installation and execution.

  The disadvantages of this solution are overhead and dependence on ARP. For example, in order to determine whether there is a need for route discovery or not, ARP request should be initiated. Since route discovery is initiated by outgoing ARP packets, these outgoing packets are overhead, and they waste bandwidth [4].

- **Kernel modification:**

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

- **Netfilter:**

Linux netfilter is the powerful packet filtering framework inside the Linux kernel. It consists of three parts:

- Hooks: which are points in a packet's traversal of that protocol stack where netfilter gets called.
- The ability for parts of the kernel to register for listening to these hooks and then process packets which traverse the hook. After processing, they tell the kernel what to do with the Packet: Discard it, accept it, forget about it or queue it for user-space.
- A queue where packets can be collected for sending to user-space.

Netfilter solution has many advantages: there is no unnecessary communication, it is highly portable, it is easy to install and the user-space daemon can determine all the required events.

One disadvantage of netfilter is that it requires a kernel module. However, a kernel module is easier to install than a kernel modification. Since only the kernel module must be recompiled, there is no need to recompile the complete kernel. Also, the kernel module can be loaded or unloaded at any time [4].

V. ADVANTAGES AND DISADVANTAGES OF EACH IMPLEMENTATION

There are a lot of advantages and disadvantages of AODV protocols. One advantage of AODV is that it doesn't create extra traffic for communication among existing links, that's because routes are established on demand and destination sequence numbers are used to find the latest route to the destination. Also connection setup delay is less. In addition, distance vector routing is simple, and it doesn't require much memory or calculation. However AODV requires more time to establish a connection, and the initial communication to establish a route is heavier than some other approaches. In contrast of DSR, the source node and the intermediate nodes of AODV store the next-hop information corresponding to each flow for data packet transmission.

AODV is simple and may be more efficient than other protocols when rate of information transmission is low enough that the overhead of explicit route discovery and maintenance happened by other protocols is relatively higher. Potentially higher reliability of data delivery and that's because packets may be delivered to the destination on multiple paths.

The main difference between AODV and other on-demand routing protocols is that it uses a destination sequence number, which keeps an up to date path to the destination node, so a node updates its path information if the destination sequence number of the current packet received is greater than the last one stored at the node.

The disadvantage of AODV protocol is that intermediate nodes can lead to inconsistent routes if the source sequence number isn't updated for a while, hence they don't have the latest destination sequence number. In addition, multiple RREP packets in response to a single RREQ packet can lead to heavy control overhead. Another disadvantage of AODV is that the periodic beaconing leads to unnecessary bandwidth consumption. Also very high overhead since data packets may be delivered to too many nodes who do not need to receive them.

Kernel AODV has few advantages: it operates faster than user space implementation because it does not require any mechanism for transferring packets from kernel to user space and vice-versa.

Some of the disadvantages of the Kernel AODV are less portable and difficult to maintain, and reduces the protocol functionality and weakens memory management. Also kernel AODV has some bugs like memory leaks, mishandling malloc () function, assertion failures, routing loops [3].

The work in [5] is a comparison of AODV-UCSB vs. AODV-UU based on the throughput and latency. As seen from that paper the throughput achieved by AODV-UU is higher than that of AODVUCSB, so the latency associated with AODV-UCSB is greater than AODV-UU. Thus AODV-UU seems to perform better than AODV-UCSB. Also, both AODV-UU and AODV-UCSB

respond equally to link breakage and promptly find another path to their destination.

VI. INSTALLATION AND CONFIGURATION OF AODV-UU

AODV-UU can be used as a routing agent for mobile nodes in the Network Simulator NS-2 instead of AODV implementation that comes by default with NS-2. AODV-UU was chosen over the other AODV implementations because it is one of the most publicly recommended implementations of AODV i.e. (kernel AODV, AODV-UU and AODV-UCSB). Moreover, AODV is more portable than Kernel AODV and easy to maintain. Also, Kernel AODV has some bugs such as: memory leaks, routing loops, and mishandling of malloc function.

This section explains the installation of the AODV-UU protocol as well as some of the installation difficulties and bugs in the AODV-UU routing protocol. So the main goal of this section is to install/ configure AODV-UU implementation and run it in Network Simulator NS-2.

A. Installing NS-2

To use AODV-UU in NS-2, you need a Linux system (or other UNIX system which NS-2 will run on) and the 2.33 version of NS-2. For windows users, you might use Cygwin, which is a Linux like environment for windows.

Download NS-2 Network Simulator from this homepage:http://www.isi.edu/nsnam/ns/nsbuild.html#allinone, and the latest version so far is NS-2.33, and the good thing that the patch for dynamic libraries was integrated the official ns-allinone-2.33 release, so if you use that version you don't need to apply the patch again, and it should be automatically built when you run the install script. It's recommended to download the "ns-allinone" package since it contains all necessary software components, hence no need to build the software individually.

The Following steps explains ns-allinone installation in home directory for Linux system:

- Unpack the ns-allinone archive using "tar" command:
  $ cd ~
  $ tar zxvf ns-allinone-2.33.tar.gz
- Then run the installation script supplied with NS-2:
  $ cd ~/ns-allinone-2.33
  $ ./install
- Now try to run the simulator using this command:
  $ cd ~/ns-allinone-2.33/bin
  $ ./ns
- If the installation was successful, NS-2 should display
  a prompt sign "$ % "

B. Installing AODV-UU

- Download AODV-UU version 0.9.5 from the sourceforge homepage:
  http://sourceforge.net/
- Unpack it in the "NS-2.33" directory using "tar" command:
  $ cd ~/ns-allinone-2.33/NS-2.33
  $ tar zxvf aodv-uu-0.9.5.tar.gz
- Create a symbolic link named "AODV-UU" that points to the "aodv-uu-0.9.5" directory, and this step is necessary for NS-2 to be able to find the AODV-UU files.
  $ ln -s ./aodv-uu-0.9.5 ./AODV-UU
- This step is necessary if an old version of ns is used i.e. less than allinone-2.33.
  $ patch -p1 < AODV-UU/NS-2.31-aodv-uu-0.8.patch
- If you are installing AODV-UU into an existing, already compiled NS-2 installation, ns-2 needs to be recompiled before the changes take effect:
  $ cd ~/ns-allinone-2.33/NS-2.33
  $ ./configure
  $ ./make distclean
  $ ./configure
  $ ./make

C. How to use AODV-UU in NS-2 Simulator?

AODV-UU is easy to use and is straightforward. However, mobile nodes should be configured to use "AODVUU" protocol as their routing agent (instead of "DSDV" or "AODV" routing agent) in the OTcl simulation scenario scripts [7]. This can be done with the node-config command, before creating any mobile nodes:

```
$ns_ node-config -adhocRouting AODVUU \
                 -llType LL \
                 -macType Mac/802_11 \
                 -ifqType Queue/DropTail/PriQueue \
                 -ifqLen 50 \
                 -antType Antenna/OmniAntenna \
                 -propType
                -Propagation/TwoRayGround \
                -phyType Phy/WirelessPhy \
                -topoInstance $topo \
                -channel Channel/WirelessChannel \
```

The remaining node configuration can be used for Real-Time simulation, so they should be OFF since no need to use them in NS-2 simulator.

```
                 -agentTrace OFF \
                 -routerTrace OFF \
                 -macTrace OFF \
                 -movementTrace OFF\
```

D. AODV-UU Configuration

For each mobile node in the simulation scenario script, AODV-UU routing agent instance is configured individually by setting the values of its configuration variables from OTcl. The available configuration variables are explained in [7], and should be set before the simulation begins, which are located before the "$ns_ run" line at the end of OTcl script. The default values can be found in the "NS-2.33/tcl/lib/ns-default.tcl" file:

*debug_*	*Print log messages on standard output (stdout)*
*expanding_ring_search_*	*Expanding ring search for RREQs*
*hello_jittering_*	*Jittering of HELLO messages*
*llfeedback_*	*Use link layer feedback instead of HELLOs*
*local_repair_*	*Use local repair*
*log_to_file_*	*Write log messages to logfile*
*optimized_hellos_*	*Only use HELLOs when there are active routes*
*ratelimit_*	*Use rate limiting for RREQs and RERRs*
*receive_n_hellos_*	*Receive N HELLOs before treating as neighbor.(Should be set to at least 2 if you use it)*
*rreq_gratuitous_*	*Force gratuitous flag on all RREQs*
*rt_log_interval_*	*Periodically log routing table to routing table logfile, value in interval in msecs (0 = off).*
*unidir_hack_*	*Detect and avoid unidirectional links*
*wait_on_reboot_*	*Wait 15 sec. on reboot delay*
*internet_gw_mode_*	*Run this node as a gateway*

To configure the AODV-UU routing agent, the values of AODV-UU routing agent instance variables should be changed. The instance variables are described below:

- *debug_* determines whether the events of the “general” logfile should be printed to the console (stdout).
- *hello_jittering_* determines whether jittering of HELLO messages should be used.
- *rreq_gratuitous_* determines whether the gratuitous RREP flag should be set in RREQs. This flag is described in section 3.6.3.
- *receive_n_hellos_* requires that a certain number of HELLO messages should be received from a node before it is treated as a neighbor. If used, it should be set to a value greater than or equal to 2.
- *expanding_ring_search_* determines whether expanding ring search should be used for RREQs.

- *wait_on_reboot_* determines whether a 15-second wait-on-reboot delay should be used on startup.
- *rt_log_interval_* determines the interval between loggings of the internal routing table of the AODV-UU.
- *unidir_hack_* determines whether uni-directional links
  should be detected and avoided.

Some bugs in AODV-UU were existed and corrected during initial testing of the AODV-UU routing agent. One of the most serious bugs was a log buffer overflow in the debug module, which causes the simulator to crash with a segmentation fault when a large simulation was conducted and routing table logging had been enabled. This bug was solved by flushing the routing table log after each line. Simulations also revealed a bug in the processing of AODV messages, where the reception of a message could cause buffered packets to be scheduled for transmission even though an active route to the destination did not exist. Finally, errors related to RERR processing were found and corrected by the author of AODV-UU.

VII. 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.

Several independent AODV implementations exist, such as AODV-UU, Kernel-AODV, AODV-UCSB, AODV-UIUC, and Mad-hoc AODV. This paper discussed advantages and disadvantages of each implementation. Finally, the AODV-UU installation/configuration on NS-2 has been explained in details.

Acknowledgmen

The author would like to thank Dr. Bruce Millard, Professor of practice, Division of Computing studies, Arizona State University at the Polytechnic Campus, for his advising during this project, and appreciate his effort for supplying references to complete this study. Also, the author thanks Dr. Harry Koehnemann and Dr. Alan Skousen the committee members for their feedback and help with this project. I would like to thank all of my family members specially my parents for their support and encouragement.

References

[1] C. Perkins, E. Belding-Royer, S. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing", Internet experimental RFC 3561, July 2003.

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

[3] V. Kawadia, Y. Zhang and B. Gupta. "System Services for Implementing Ad-hoc Routing Protocols", International Conference on Parallel Processing Workshops, April 2002.

[4] I. D. Chakeres and E. M. Belding-Royer. "AODV Routing Protocol Implementation Design". In Proc. of WWAN, March 2004

[5] Evaluating MANET in real world environment." Clevelan State University. 30 Oct 2008. http://www.csuohio.edu/engineering/ece/research/theses/2004/kalepu.pdf .

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

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