

Experimental Performance Evaluation of AODV Implementations in Static Environments

Koojana Kuladinithi, Asanga Udugama, Nikolaus A. Fikouras, Carmelita Görg
ComNets, Universität Bremen, Otto-Hahn-Allee NW1, 28359 Bremen, Germany
{koo|adu|niko|cg}@comnets.uni-bremen.de

ABSTRACT

The Ad hoc On-Demand Distance Vector (AODV) routing protocol is designed for use in mobile ad-hoc networks. As of the writing of this article, there exist several implementations of the AODV protocol for a range of operating systems (e.g., Unix/Linux, Windows), different devices (e.g. Notebook computers, PDAs) each developed using different techniques and programming languages. This paper describes the architecture of 2 AODV implementations and experimentally compares them with respect to their effect on transport layer protocol (UDP and TCP) performance. The investigated scenario involves a stationary AODV test-bed that consists of 6 nodes connected using WLAN 802.11b. The considered AODV implementations are the JAdhoc, a Java based platform independent implementation and the AODV-UU, a Linux based user space implementation developed using C that utilizes kernel space functionalities. The performance of both implementations is compared against that of a manually configured test-bed.

1. INTRODUCTION

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 [1, 2]. 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. As of the writing of this document AODV has been published as an experimental RFC [3].

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 [4, 5] also in comparison to the performance of other ad hoc routing protocols [6].

There exist currently several AODV implementations [7] for different operating systems. These implementations comply with a varying degree to the protocol description defined in [3]. 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.

This study gives an insight in implementation aspects of two popular implementations compliant to RFC 3561 [3], namely of JAdhoc [8] an inter platform AODV protocol handler in java and of AODV-UU a Linux protocol handler written in C, working both on user space and kernel space [9].

The investigated scenario involves the evaluation of TCP and UDP performance between AODV nodes in a static ad-hoc environment. This scenario resembles the conditions encountered in conference rooms, class rooms, hotel rooms or home networks.

The performance of UDP was evaluated with respect to parameters such as packet delivery fraction, offered load, out of order sequence packets and jitter. The performance of TCP was evaluated with respect to throughput together with TCP parameters of Retransmission Time Out (RTO) and congestion window size (cwnd) [10].

The rest of this study is organized as follows: First, AODV operations are briefly discussed. Implementation considerations of an AODV protocol handler are explained in the next section for both JAdhoc and AODV-UU implementations. The fourth section details experimental set up of the test-bed and all test scenarios. The fifth section shows the obtained results, together with the analysis. The final section is a concluding summary.

2. AODV OPERATIONS

Ad hoc network routing protocols are all about discovering, establishing, recovering and maintaining routing paths. To that end AODV uses 4 types of control messages. They are the Route Request (RREQ), Route Reply (RREP), Route Acknowledgment (RREP-ACK) and Route Error (RERR) messages. Sequence numbers in AODV play a key role in ensuring loop freedom. Every node maintains a monotonically increasing sequence number for itself. It also maintains the highest known sequence number for each destination in the routing table, called destination sequence number which is included with RREQ, RREP and RERR messages [1, 3].

When an originator needs a route to a destination, it initiates the route discovery process. Route discovery involves a network wide flood of RREQ targeting the destination and awaiting for a RREP. An intermediate node receiving the RREQ, is required to first set up a reverse path to the originator designating the previous hop of the RREQ as the gateway to it. If the intermediate node has a valid route to the destination and the 'D' (*Destination Only*) bit is not set in the RREQ, the intermediate node generates a RREP to the originator and a gratuitous RREP to the destination; otherwise the RREQ is re-broadcasted. Duplicate copies of RREQs (identified by RREQ ID maintained for the originator) received at any node are discarded. When the destination receives a RREQ, it is required to respond with a RREP. The RREP is routed back to the originator via the reverse path. As the RREP propagates towards the originator, a forward path to the destination is established. Optionally, the originator may issue a RREP-ACK to the destination upon receiving the RREP in order to make certain of the reliability of the bi-directional path. Each node maintaining active routes is required to periodically multicast HELLO message, i.e. a special RREP with the Time To Live (TTL) of IP header field set to one. The connectivity to immediate neighbors can be determined via link layer information or by listening to HELLO messages. When a node detects link failure, it is required to issue a route error (RERR) message back to all active nodes that are dependent on the failed link (route). These nodes are found via a separately maintained precursor lists to all originators that are affected by the failed link. When the originator receives the RERR, it can start a route discovery process, again. Unused routes in the routing table are expired using a timer-based technique.

3. IMPLEMENTATION CONSIDERATIONS

AODV is a protocol that operates at the network layer of the OSI protocol stack. An AODV protocol handler can either operate as a user space application or a kernel space application or using a combination. Irrespective of what type of application, the programming language and the operating system environment should have the following capabilities for the AODV protocol handler to operate.

Unicast and broadcast communications: The AODV controlling messages are UDP and should be destined to port 654 of the recipient. The RREQ is a broadcast message that needs to be heard by all the devices in the vicinity. RREP and RREP-ACK are unicast messages directed to a known device. The RERR, the last of AODV messages can be unicast or broadcast depending on the requirement.

Table 1 Methods of implementation of JAdhoc & AODV-UU

	<i>JAdhoc</i> <i>Developed in Java Language for Unix and Windows platforms and uses Jpcap to capture packets.</i>	<i>AODV-UU</i> <i>Developed in C for Linux platform. Run as a user space application using kernel space functionality</i>
<i>Unicast and broadcast communications</i>	<i>Using standard Java class library that provides a set of classes which can be used to send and receive UDP packets to a unicast, multicast or broadcast IP address.</i>	<i>Uses the Berkeley Socket Interface provide as part of the standard Linux development environment.</i>
<i>Intercepting IP packets that require a route (to commence the Route Discovery process)</i>	<i>Unix Environment: Set the loop-back network device as the networking device of the default route. This is done at the beginning as part of JAdhoc's initialization. Once this is done, JAdhoc continues to monitor the loop-back device using the packet capture class library, for packets that reach this interface.</i> <i>Windows Environment: JAdhoc first sets a dummy MAC address (00-00-00-00-00-00) to the IP address specified in the default route at initialization. Once the dummy address is set, it continues to monitor the network interface that the default route takes.</i>	<i>Uses the Netfilter framework to install hooks to intercept packets. In particular, it uses the NF_IP_LOCAL_OUT hook that is implemented through a kernel module, to handle packets This functionality is obtained using the libipq module.</i>
<i>Information of IP packets that utilize the existing routes</i>	<i>Continually monitors the given network interface using the packet capture class library.</i>	<i>Uses the NF_IP_LOCAL_OUT & NF_IP_PRE_ROUTIN hooks of the Netfilter framework. Install a kernel module code that services these hooks.</i>
<i>Manipulation of the Routing Table</i>	<i>Using System Commands. It provides a general purpose interface to be implemented for each of the operating environments that it runs on. As an example, JAdhoc uses a class called OSperationsWindowsIPv4 to handle all operating environment specific commands for Windows IPv4 environment</i>	<i>Uses IOCTL (Input-Output Control) commands to manipulate the routing table.</i>
<i>Support for a Timer Mechanism</i>	<i>Threading mechanism of Java language to implement the myriad of timers required by the AODV protocol,</i>	<i>Uses the pthread library for multi-threading and thread synchronization.</i>
<i>Packet Queuing (Buffering)</i>	<i>Linked List implementation of the Java class library to buffer packets that are being generated by the operating system</i>	<i>Uses a user space based buffe, but which is accessible to the kernel based Netfilter framework.</i>
<i>Link break detection</i>	<i>Using Hello message</i>	<i>Using Hello message</i>

Intercepting IP packets that require a route: The protocol handler needs to know when an IP packet needs a route. It needs to know the destination IP address of the packet to initiate a Route Discovery and to hold this packet to be sent out once the route is made.

Information of IP packets that utilize the existing routes: The protocol handler requires listening to IP traffic that uses the current routes. This information is required to maintain or expire routes.

Manipulation of the Routing Table: The AODV protocol requires that the routing table of the local device be set according to the routing requirements as requested by the AODV protocol handler, based on the routes made or discarded.

Support for a Timer Mechanism: The AODV protocol requires several timers to follow the status of different processes. Some of these timers perform different tasks after they reach a certain time and expire themselves. Others continue to be active as long as the protocol handler is active. For e.g., the route discovery process requires maintaining a timer for issuing RREQs in order to determine when a destination is not reachable. Similarly, the periodic transmission of HELLO messages requires the existence of a timer.

Packet Queuing (Buffering): The route discovery process is initiated and when a transport layer session issues datagrams addressed to a destination for which the protocol handler maintains no known routing path. For the duration of this process such data can not be transmitted and hence have to be buffered. The system facility to support this functionality is critical to the performance of a protocol handler.

Link break detection: The AODV describes two alternatives for the discovery of link failures with adjacent AODV nodes. The first deploys periodic HELLO messages while the second relies on link-layer information.

Table 1 shows that how above mentioned requirements has been implemented in JAdhoc and AODV-UU. More details of design architecture and techniques used can be found at [11] [12].

4. EXPERIMENTAL SET UP

In order to do the experimental performance analysis of UDP and TCP in a static AODV network, the experimental test-bed illustrated in fig. 1 was constructed. It consists of 6 notebooks with following hardware and software configurations.

Hardware: All computers are Sony Vaio notebooks which have Mobile AMD Duron processors running at 1.1 GHz and 256 MB of RAM. For wireless connectivity, Cisco Aironet IEEE 802.11b wireless cards were used. The wireless cards were set in ad-hoc mode on channel 1. The RTS/CTS setting was set to 1 byte. No WEP encryption was utilized.

Software: All notebooks have been installed with the Linux Mandrake 8.2 distributions and the Linux Kernel 2.4.19. The WLAN interface of each notebook has been allocated a different IP address from a different sub network as shown in the fig. 1. In order to perform the experimental evaluation while maintaining the notebooks in close physical distance to each other the MACKill software [13] was installed on each notebook. MACKill performs MAC level filtering. The Iperf traffic generator software [14] was utilized for generation of UDP and TCP flows. Web100 [15] was used to measure a selected set of TCP parameters.

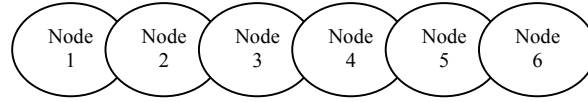


Figure 1 AODV test-bed set up

Test Scenarios: Both JAdhoc version 0.11 and AODV-UU version 0.7.1 were installed for AODV ad hoc routing support. AODV protocol options were configured as indicated in RFC 3561 [3] except for the HELLO Loss option which was set to three (reasons for this is explained in section 5.3). The evaluation was based on the performance of UDP and TCP flows which are generated by the iperf traffic generator, varying the generation rates. MACKill was used to prevent the hearing of IP packets that get generated from nodes other than the immediate neighbors (refer fig.1). Test-bed is configured to work in three test modes.

- **Static Mode:** Routing environment of all nodes are set manually as if they are in an AODV network,
- **UU Mode:** Each node sets routes on demand using AODV-UU protocol handler,
- **JAdhoc Mode:** Each node sets routes on demand using JAdhoc protocol handler.

UDP test: Various data rates of UDP streams were evaluated. For the test-bed in static mode no packet loss was identified even for the maximum evaluated data transmission rate of 3.6Mbps. Given that in IEEE802.11 ad-hoc mode the same frequency channel is used by all six ad-hoc nodes as shown in fig. 1, the maximum end-to-end throughput is given by 3.6 Mbps divided by 6 (512000 bps), when using all six nodes simultaneously. Therefore, UDP data rates of 512000 bps was chosen to measure the key attributes of a UDP communication of packet delivery ratio, actual load, jitter and analysis of out of sequence packets. Following measurements were taken for both UU and JAdhoc, when varying the number of nodes from two to six.

Packet delivery ratio: The ratio given by the number of UDP packets originating from the source over the number of packets received by the destination. This describes the loss rate that will be witnessed by the transport protocols; in turn this affects the maximum throughput that the network can support.

Offered load vs actual load: Actual load is measured as total volume of data received during the period of UDP communication (i.e. 60 sec).

Jitter: The jitter of a packet stream is defined as the mean deviation of the difference in packet spacing at the receiver compared to the sender, for a pair of packets. If S_i is the time packet i was sent from the sender, and R_i is the time it was received by the receiver, the jitter sample J_i is given by:

$$J_i = \frac{|(R_{i+1} - R_i) - (S_{i+1} - S_i)|}{|(R_{i+1} - S_{i+1}) - (R_i - S_i)|} \quad \text{----- (A)}$$

and the average jitter is the average value over n packets. The jitter of a UDP stream is a particularly important quantity for time sensitive data such as real-time audio and video, since a large jitter can have a profound effect on the perceived quality.

Out of sequence packets: Out of sequence packets refer to the unordered packet receipt by the receiver of a communication session. Certain applications are sensitive to packets that are received out of order.

TCP test: At first, TCP throughput was measured by varying the end-to-end number of hops between the sender and receiver from one (with two nodes) to five (with six nodes) in Static mode. It was found that TCP resorted to a considerable amount of retransmissions which was reflected on the session's throughput. This effect was made more profound as the number of hops increased. This was accredited to the inability of the air link to sustain an increasing number of nodes in the same frequency channel. To prove this case the experiment with two hops (three nodes) was repeated while the middle node was equipped with two IEEE802.11b wireless interfaces, each configured to an orthogonal frequency channel. It was observed that the TCP throughput achieved performance levels similar to those witnessed during the single hop (with two nodes) scenario.

In a multi-hop ad-hoc environment in an 802.11 network, TCP throughput can be affected by the exposed node problem [16] [17].

For the performance comparison of the three evaluated modes a number of TCP session parameters were taken into consideration, namely the session throughput, the retransmission timeout duration and the Congestion Window size which are explained below.

Throughput: The throughput of TCP is defined as the ratio of total number of data bytes sent by the source to the time taken for the transfer.

RTO: TCP uses a retransmission timer to ensure data delivery in the absence of any feedback from the remote data receiver. The duration of this timer is referred to as RTO (Retransmission Time Out, RFC 1122).

Congestion Window (CWND): The amount of data that may be unacknowledged, on flight in the network at any given time.

5. EXPERIMENTAL RESULTS

This section provides the results of the performance analysis done on the basis of the scenarios and the tests described in the previous section. First section analyses the UDP measurements and the second analyses the TCP measurements. All scripts (C programs and awk scripts) that were used to analyze the data are made available in [18].

5.1 UDP Results – Analysis

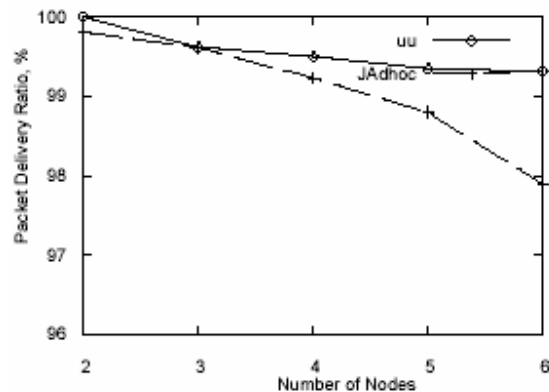


Figure 2 Packet Delivery Ratio

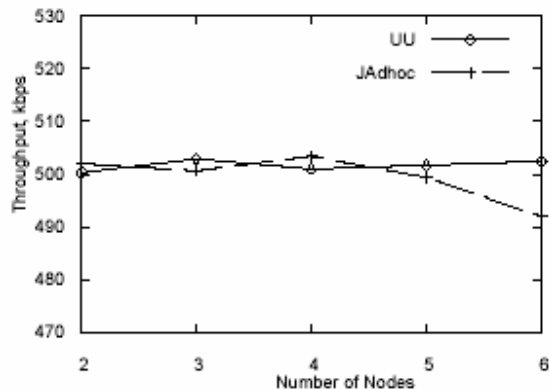


Figure 3 UDP Actual Load for given load of 512000bps

UDP results of Packet Delivery Ratio, UDP actual load for given load of 512000 bps, out of order sequence packets and jitter variations are shown both for AODV-UU (referred to henceforth as UU) and JAdhoc. UDP streams of 500Kbps were used to collect UDP measurements varying the hop count between the sender and the receiver from one to four. Each test was run for duration of 60 seconds and evaluation data were collected at the receiving node. Packet Delivery Ratio of both UU and JAdhoc protocol handlers was found to be between 98% and 100% (fig 2). Compared with the static mode environment, both UU and JAdhoc demonstrated increased packet loss relative to the increase of the hop count. It was observed that most of the packet losses occurred at the beginning of the UDP transmission, subsequently to the route discovery process, during the release of the buffered UDP datagrams. This effect is not repeated again during the course of the experiment but would be encountered should the route discovery process be initiated again. Due to the buffer issue (release of buffered packets at the beginning), both JAdhoc and UU demonstrate a higher throughput (compared to the offered load, see fig 4) as buffered packets are released (during first 600ms). Compared to UU, JAdhoc suffers out of order packets (fig. 5). This was identified as being due to the buffering mechanism used in JAdhoc (refer Table 1). JAdhoc uses a user-space buffering technique that initially introduces a delay in the transmitting early packets. Once the buffered packets are released then these are transmitted in parallel with newer packets causing a severe de-sequencing of received packets. Fig.6 and Table 2 shows the jitter variation in all environments as computed according to the equation A.

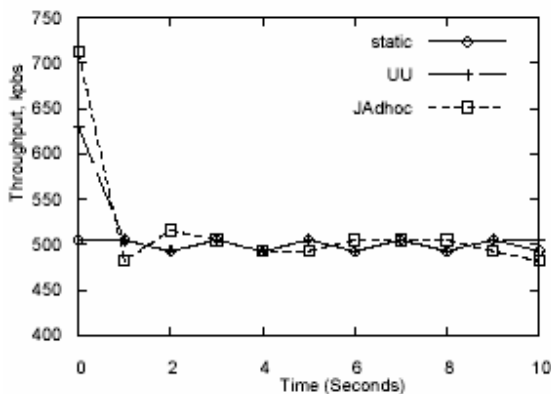


Figure 4 UDP Throughput Variations (4 hops)

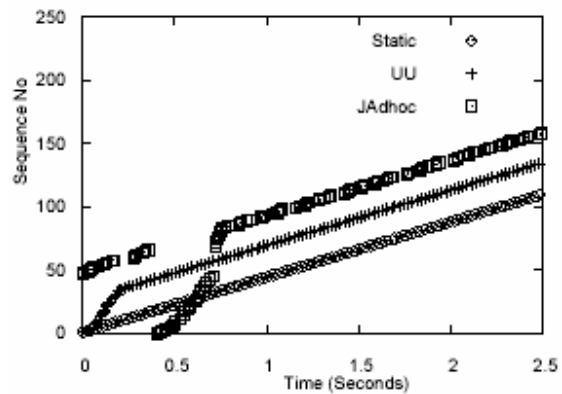


Figure 5 UDP Packet Trace (4 hops)

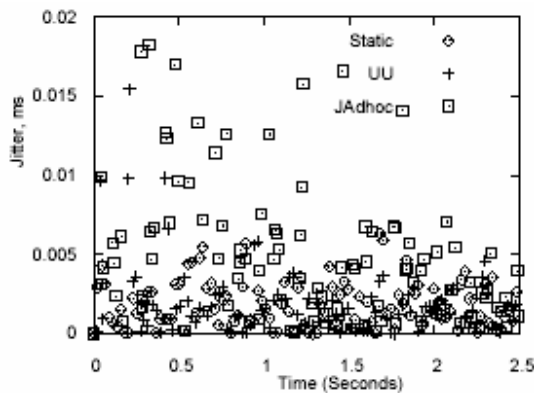


Figure 6 Jitter Variations (4 hops)

Table 2 Average Jitter measured during 60 seconds

	<i>Average Jitter</i>
<i>Static</i>	<i>1.57 ms</i>
<i>UU</i>	<i>1.85 ms</i>
<i>JAdhoc</i>	<i>5.85 ms</i>

5.2 TCP-Results - Analysis

TCP throughput variation with the increase of hop counts is shown for all modes of static, UU and JAdhoc. Each measurement was taken for the duration of 60 seconds. The performance of TCP parameters RTO, cwnd and DupAcks (using packet trace) [19] was shown for an experimental test scenario with three hops between the sender and the receiver. Though the tests were done for all hop possibilities (up to 5 hops), a representative diagram of the three hop test is shown.

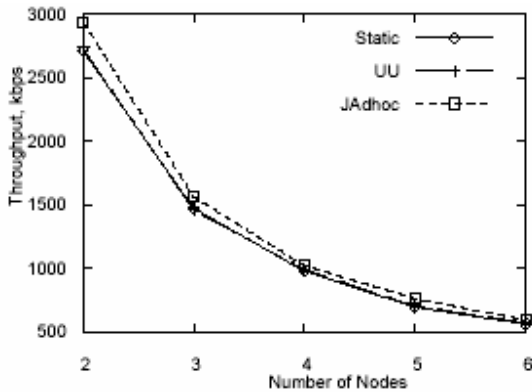


Figure 7 TCP Throughput variations with the increase of nodes

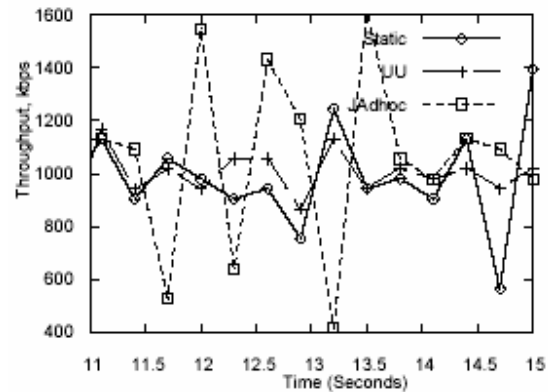


Figure 8 TCP throughput variations between 4 nodes

In contrast to UDP results shown in the previous section, TCP throughput shows that JAdhoc achieves higher throughput (Fig. 7, Fig. 8) values. In order to determine the reason behind this effect the number of DupAcks that are transmitted in each evaluated configuration was investigated. The comparison of fig. 9-11, leads to the conclusion that a significantly smaller number of DupAcks was generated when using JAdhoc. DupAcks are known to affect the performance of cwnd [10] which explains the higher TCP throughput values of the JAdhoc test. This conclusion holds also for other scenarios with higher hop counts.

The reasons for having larger number of DupAcks in static environment was found by analyzing the RTO variations during the TCP transmission for all test cases. It was found that the static network configuration demonstrated the shortest RTO compared to both UU and JAdhoc (fig. 14-15). Smaller RTO values leads to more packets getting generated by the sender. This is bound to put the link-layer to strain and lead to higher packet loss rates. In turn, this will cause the transmission of more DupAcks which will affect the TCP

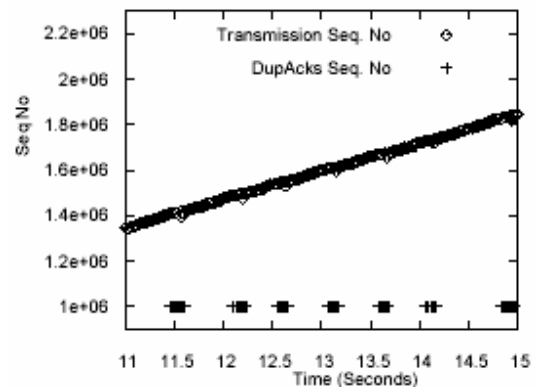


Figure 9 TCP packet trace, with DupAcks (Static)

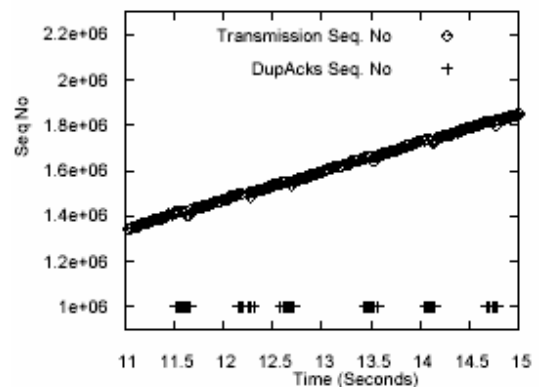


Figure 10 TCP packet trace, with DupAcks (UU)

cwnd and hence degrade the communication's throughput. In all environments, packets can be lost at the link layer due to the use of the same frequency channel and the "Exposed Node" problem [16] that is common in multi-hop ad hoc networks when using 802.11 technologies.

Additionally, it was noticed that both JAdhoc and UU take more CPU time when running, than in a static ad-hoc environment. This was found to affect the response time of the sender and receiver devices resulting in higher RTO values for JAdhoc and UU compared to the Static environment.

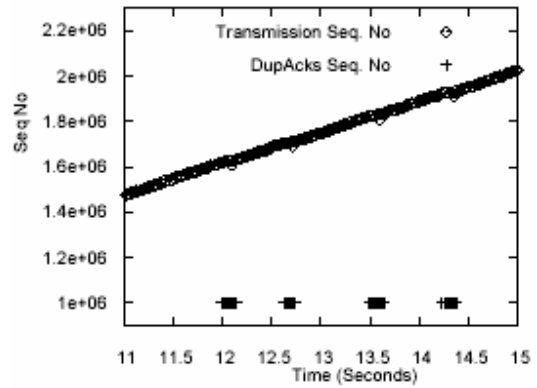


Figure 11 TCP packet trace, with DupAcks (JAdhoc)

5.3 Additional Observations

The throughput of TCP communications can be affected by altering certain AODV parameters. More specifically, it was found that by changing the 'HELLOLoss' value from three to two, the TCP throughput was reduced for both JAdhoc and UU experiments with higher data rates (e.g. 2.4 Mbps).

Both UU and JAdhoc detect link breaks based on HELLO messages. When detecting link breaks using HELLO messages, each node waits to receive at least one HELLO message for the duration of "HelloInterval*HelloLoss". And also, each AODV node should process each packet (both data & control messages) to update the route lifetime. When processing more data packets, the node takes a long time to process the HELLO message and if it is not within the pre-defined interval (i.e. HelloInterval*HelloLoss) when processing, the node invalidates the active route. Therefore, lower the detection duration, the higher the probability of route breaks, under the transmission of higher data rates. Detecting link breaks using link layer information avoids this situation. In a static AODV environment, it is always better to use a higher value for HelloLoss, when transmitting higher data rates.

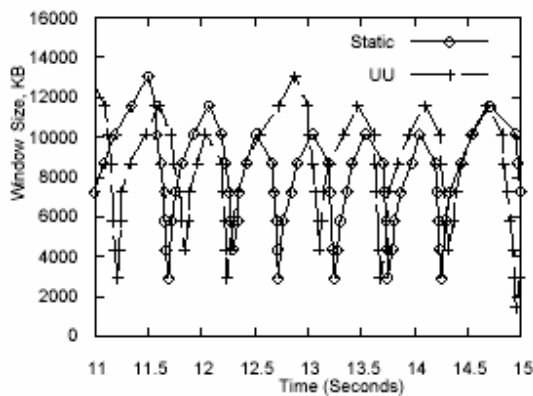


Figure 12 Variation of cwnd (Static & UU)

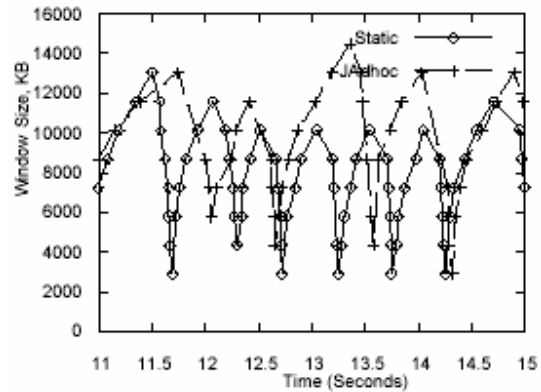


Figure 13 Variation of cwnd for (Static & JAdhoc)

When increasing the hop count between the sender and receiver up to five, it can be seen that route discovery initiations occur more than once, even when the TTL of Route Requests are set to 10. In AODV, lifetime of the reverse route is computed according to the equation (B).

$$\text{MinimalLifetime} = (\text{current time} + 2 * \text{NET_TRAVERSAL_TIME} - 2 * \text{HopCount} * \text{NODE_TRAVERSAL_TIME}) \text{ ----(B)}$$

When applying parameters that were used at the third node (HopCount=3, NODE_TRAVERSAL_TIME=10, Net Diameter =10), it sets the life time of originating route as 140ms. At the time of receiving RREP, this node has already invalidated the originating route. This is due to the more time taken to process RREP by JAdhoc, being a user space implementation and running on top of the JVM. Since UU has direct access to kernel level functionality, this problem is non-present. The equation B considers only the propagation delay. It might not be harmful to AODV operations to use MinimalLifetime as default value of one sec or to add the processing delays to the existing equation. This will avoid unnecessary route discoveries at the beginning.

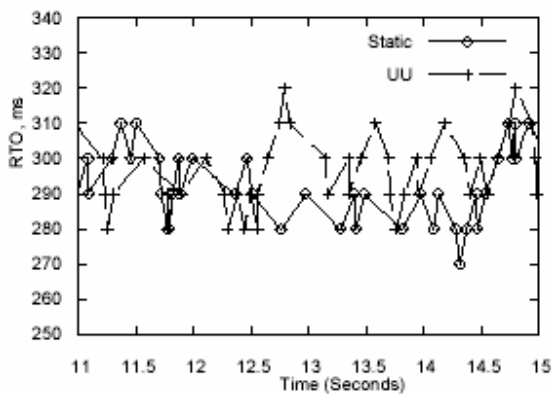


Figure 14 RTO Variations (Static & UU)

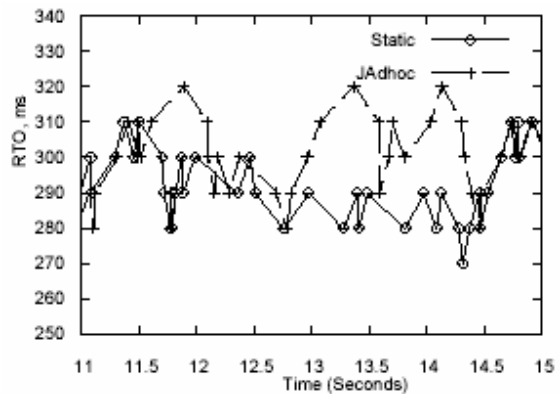


Figure 15 RTO Variations (Static & JAdhoc)

CONCLUSION

The area of ad hoc networking has been receiving increasing attention among researchers in recent years. This has been accompanied by an increasing availability of implementations for different operating systems. However, it has been largely unknown how such implementations perform against each other and how different design decisions affect the performance of transport protocols.

This paper makes contributions in two areas. First, it details the implementation architecture of two popular AODV implementations. Namely an AODV implementation of the Upsala University and a Java based implementation from the University of Bremen. Secondly, it provides some experimental results taken from comparing these two implementations together with a static (AODV-less) environment. These experimental results were based on the analysis of UDP and TCP communications. Further research issues include performing the same test in an environment, considering the movements and also determining the effect of values of HelloLoss and minimal lifetime that are discussed in section 5.3

ACKNOWLEDGEMENT

Implementation of UoB-JAdhoc has been supported by the European Commission within the IST project NOMAD (IST-2001-33292). The authors would like to acknowledge the contributions of Henrik Lundgren from Uppsala University, Sweden.

REFERENCES

1. Perkins, C.E., *Ad Hoc Networking*. 2000, NJ: Addison-Wesely.
2. 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.
3. 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.
4. 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.
5. Ramarathinam, V. and M.A. Labrador, *Performance Analysis of TCP over Static Ad Hoc Wireless Networks*. in *In Proceedings of the ISCA 15th International Conference on Parallel and Distributed Computing Systems (PDCS)*. 2002.
6. 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.
7. Implementations, P.A., available at <http://moment.cs.ucsb.edu/AODV/aodv.html#Implementations>. 2002.
8. UoB-JAdhoc, *AODV implementation in Java, developed by ComNets, University of Bremen, available at www.aodv.org*. 2003.
9. AODV-UU, *AODV implementation on Linux, by University of Uppsala, available at <http://user.it.uu.se/~henrikl/aodv>*. 2002.
10. Wright, G.R. and W.R. Stevens, *TCP/IP Illustrated*. Vol. 2. 1995: Addison-Wesley.
11. Kuladinithi, K. and A. Udugama, *JAdhoc System Design Manual*. 2003, ComNets, University of Bremen, available at http://www.aodv.org/index.php?name=EZCMS&menu=4&page_id=2.
12. Wiberg, B., *Porting AODV-UU Implementation to ns-2 and Enabling Trace-based Simulation, Chapter 4*. 2002, Uppsala University, Sweden, available at <http://bwiberg.dyndns.org/kurser/exjobb/exjobb/exjobb.pdf>.
13. Lundgren, H., *MacKill - Tool to filter packets at Link Layer, available at <http://user.it.uu.se/~henrikl/aodv/>*.
14. Tirumala, A., et al., *Iperf: The TCP/UDP Bandwidth Measurement Tool, available at <http://dast.nlanr.net/Projects/Iperf>*.
15. *Web100, available at www.web100.org*.
16. Uskela, S., *Link Technology Aspects in Multihop Ad Hoc Networks*. 2002, Networking Laboratory, Helsinki University of Technology.
17. Xu, S. and T. Saadawi, *Does the IEEE 802.11 MAC Protocol Work Well in Multihop Wireless Ad Hoc Networks*, in *IEEE Communications Magazine*. 2001.
18. ComNets, U.o.B., *Scripts for TCP & UDP Results Analysis, available at <http://www.aodv.org/modules.php?op=modload&name=UpDownload&file=index&req=viewdownload&cid=2>*. 2004.
19. Xylomenos, G., et al., *TCP Performance Issues over Wireless Links*. IEEE, Communications Magazine, 2001. **39**.