CHAPTER V
IMPLEMENTATION OF AN EAODV-LL PROTOCOL
Chapter 5

IMPLEMENTATION OF EAODV-LL PROTOCOL

Ultimately, the final test for any protocol is to deploy it in one of the environments for which it was designed and measure it to determine if it operated successfully. The design and implementation of a full-scale, multi-hop ad hoc network simulation [71] to enable the performance evaluation of ad hoc networks, the simulation was used to demonstrate the potential of ad hoc networking to our group and as a research tool to experiment with the carrying at the capacity and behavior of a fully-deployed network. Routing in the simulation is performed by EAODV-LL, extended with additional features to allow the entire simulation to be seamlessly integrated with existing Internet infrastructure. The implementation of extensive set of network monitoring tools, which track the precise location and protocol processing activities at each node and allow us to analyze the behavior of the network. A series of traffic generators were also developed to stress the network with a variety of different traffic loads. This chapter describes the results of our initial experiments on the simulation, and discusses the considerable effect that real-world radio propagation had on the protocols in the network. Additional information on the lessons we learned, the tools we built, and the structure of the EAODV-LL implementation can be found in a technical report describing the simulation [71].

5.1 Simulation

Our primary design goal for the simulation was to challenge the network protocols to the point where they were stressed, and so we subjected to high rate of topology change and a heavy traffic load. With the vehicles, radios, and site used in
our simulation, we forced the protocols to operate in an environment in which *all* links between nodes change status at least every 150 seconds. Ignoring the additional factor of packet loss due to wireless errors, on average, the network topology changed every 10 seconds.

### 5.1.1 Network Topology

Figure 5.1 shows a logical view of the ad hoc network simulation. The ad hoc network includes 5 moving car-mounted nodes, labeled T1-T5, and 2 stationary nodes, labeled E1 and E2. Each of these nodes communicates using 900 MHz Wave LAN-I radios. These radios do not implement the IEEE 802.11 MAC protocol [43], since at the time the simulation was built; the Wave LAN-IEEE radios were not available. The ad hoc network is connected to a field office using a 2.4 GHz point-to-point wireless link over a distance of about 1000 m. This point-to-point link does not interfere with the 900MHz radio interfaces on the individual ad hoc network nodes. At the *field office* is a router R that connects both the ad hoc network and an IP subnet at the field office back to the *central office* via a wide-area network. The visualize node V is used to monitor the status of the ad hoc network, and the GPS reference station (RS), is responsible for sending differential real-time kinematics (RTK) GPS corrections to nodes in the ad hoc network. The central office is home to a *moving node* (RN) that drives between the central office and the ad hoc network, participating in three networks: its home wireless LAN, and the ad hoc network. Node HA provides Mobile IP home agent services [82] for the moving node, enabling it to leave the central office and still maintain routing connectivity with all of the other nodes in the Internet. During a typical experiment, which we call a run, the drivers of each of the cars carrying an ad hoc network node follow the course shown in Figure 5.2 at speeds varying from 25 to 40 Km/hr (15 to 25 miles per hour). Each run lasts for
between 30 and 120 minutes. The road we use is open to general vehicle traffic and has several Stop signs, so the velocity of each node varies in a complex fashion, just as it would in any real network. The nodes are constrained to move along the paved surfaces of the site. This prevents us from testing the arbitrary topologies used in some theoretical simulations on abstract flat planes, but enables us to evaluate the performance we can expect in a real application. During each run, the network was subjected to the composite workload shown in Table 5.1, consisting of synthetic voice calls, bulk data transfer, location-dependent transfers, and real-time data.

![Figure 5.1. Logical overview of the simulation network.](image-url)
<table>
<thead>
<tr>
<th>Application</th>
<th>Rate</th>
<th>Protocol</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voice</td>
<td>6/hour/node</td>
<td>UDP</td>
<td>Average of 180 Kbytes</td>
</tr>
<tr>
<td>Data</td>
<td>5/hour/node</td>
<td>TCP</td>
<td>40,80,120 Kbytes</td>
</tr>
<tr>
<td>Location-dependent</td>
<td>When near E!</td>
<td>TCP</td>
<td>Average of 180 Kbytes</td>
</tr>
<tr>
<td>GPS</td>
<td>1 pkt/sec multicast</td>
<td>UDP</td>
<td>180 bytes</td>
</tr>
<tr>
<td>PCTd</td>
<td>1 pkt/sec node uni cast</td>
<td>UDP</td>
<td>328 bytes</td>
</tr>
</tbody>
</table>

**Table 5.1.** Load offered to the network by nodes in the simulation.

The workload includes: each node making one voice call to every other node once per hour; each node transferring a data file to every other node once per hour; each moving node (T1-T5) making a location-dependent transfer to E1 when located within 150m of E1; and multicast differential RTK GPS corrections. Finally, the workload also includes real-time situational awareness data sent by the Position and Communication Tracking daemon (PCTd) on each node to the visualize machine. The packets contain the current location of the node, as read from the node’s GPS unit, and status information on the node, such as the number of packets it has forwarded, dropped, queued, originated or retransmitted. The visualize machine continuously displays on a map of the site the last known location of each node. The visualize can also graph the protocol status information, and it logs all the data it receives, thereby allowing a detailed replay of the run after the fact.

### 5.1.2 Network Configuration

All communication among the ad hoc network nodes, T1-T5, E1, and E2, is routed by the Ad hoc On Demand Distance Vector (EAODV-LL) protocol. EAODV-LL is a network layer routing protocol that operates at the IP layer of the network.
stack (OSI layer 3) and permits interoperation between different physical network interfaces. However, our EAODV-LL implementation conceptually operates as a virtual link layer just under the normal IP layer. This allows EAODV-LL to route packets using IP addresses as flat identifiers when the other nodes in the ad hoc network are not organized as hierarchical subnets. Nodes T1-T5, E1, and E2 are assigned IP addresses from a single subnet, with E2 acting as a gateway between the Internet and the ad hoc network subnet. E2 was manually configured to use the EAODV-LL protocol for communication on one network interface, and to use normal IP routing over the other interface. Packets from nodes in the Internet destined to addresses in the ad hoc subnet are routed by normal means to E2, which has a statically configured route directing out the network interface to the ad hoc network. Once forwarded into the ad hoc network by E2, EAODV-LL takes care of routing the packets to their final destination, which often requires multiple hops inside the ad hoc network. As explained in Section 5.2.1, nodes in the ad hoc subnet (i.e., T1-T5 and E1) did not have to be configured to use E2 as a default router: when nodes in a EAODV-LL ad hoc network send packets to nodes not in the ad hoc network, the EAODV-LL protocol itself automatically routes the packets to the nearest gateway (E2 in this case), where they are forwarded into the Internet. The gateway node, E2, also provides Mobile IP foreign agent services to any Mobile IP nodes that visit the ad hoc network. The moving node RN has available several methods for connecting to the Internet, and uses Mobile IP [82] to choose the best method as it drives around the city. RN is normally within range of the Wave LAN network at the central office, and its Wave LAN network interface carries an IP address belonging to the central office subnet. When RN is moving away from the central office, it uses Mobile IP to register a care-of address with its home agent on the central office subnet. While RN has a
Chapter 5: Implementation of EAODV-LL Protocol

care-of address registered with the home agent, the home agent intercepts packets destined to RN, and tunnels each to the care-of address using encapsulation. When RN cannot use its primary Wave LAN interface because it is not in range of any other Wave LAN radios, it uses its CDPD modem to connect to the CDPD, and registers its CDPD IP address with its home agent. Once RN realizes it is in range of an EAODV-LL network, it can use the EAODV-LL protocol to communicate directly with the other nodes in the ad hoc network. To enable packets from nodes outside the EAODV-LL network to reach RN, it registers itself with its home agent via the foreign agent at E2, just as in normal Mobile IP. When E2 receives a tunneled packet, it checks to see if the packet is destined to a node registered as visiting the ad hoc network. If so, E2 routes the packet to the visiting node using EAODV-LL.

5.2 Implementation of Protocol

The following are the protocols has implemented as follows: DSDV, AOMDV, TORA, EAODV-LL

5.2.1 Destination Sequenced Distance Vector (DSDV)

DSDV [83] is a hop-by-hop distance vector routing protocol requiring each node to periodically broadcast routing updates. The key advantage of DSDV over traditional distance vector protocols is that it guarantees loop-freedom.

Basic Mechanisms

Each DSDV node maintains a routing table listing the “next hop” for each reachable destination. DSDV tags each route with a sequence number and considers a route more favorable than if has a greater sequence number, or if the two routes have equal sequence numbers but has a lower metric. Each node in the network advertises a monotonically increasing even sequence number for itself. When a node B decides
that its route to a destination $D$ has broken, it advertises the route to $D$ with an infinite metric and a sequence number one greater than its sequence number for the route that has broken (making an odd sequence number). This causes any node **routing packets** through $B$ to incorporate the infinite-metric route into its routing table until node $A$ hears a route to $D$ with a higher sequence number.

**Implementation**

We did not use link layer breakage detection from the 802.11 MAC protocol in obtaining the DSDV data presented in this chapter, because after implementing the protocol both with and without it, we found the performance significantly worse with the link layer breakage detection. The reason is that if a neighbor $N$ of a node $A$ detects that its link to $A$ is broken, it will broadcast a triggered route update containing an infinite metric for $A$. The sequence number in this triggered update will be one greater than the last sequence number broadcast by $A$, and therefore does the highest sequence number exist anywhere in the network for $A$. Each node that hears this update will record an infinite metric for destination $A$ and will propagate the information further. This renders node an unreachable from all nodes in the network until broadcasts a newer sequence number in a periodic update. $A$ will send this update as soon as it learns of the infinite metric being propagated for it, but large numbers of packets can be dropped in the meantime. Our implementation uses both full and incremental updates as required by the protocol's description. However, the published description of DSDV [83] is ambiguous about specifying when triggered updates should be sent. One interpretation is that the receipt of a new sequence number for a destination should cause a triggered update. We call this approach DSDV-SQ (sequence number). The advantage of this approach is that broken links will be detected and routed around as new sequence numbers propagate around the
broken link and create alternate routes. The second interpretation, which is called simply DSDV, is that only the receipt of a new metric should cause a triggered update, and that the receipt of a new sequence number is not sufficiently important to incur the overhead of propagating a triggered update. It has implemented both DSDV-SQ and DSDV and found that while DSDV-SQ is much more expensive in terms of overhead, it provides a much better packet delivery ratio in most cases. The second scheme (DSDV) is much more conservative in terms of routing overhead, but because link breakages are not detected as quickly, more data packets are dropped. All of the results presented in this chapter use DSDV-SQ, with the exception of Section 6.4.2, which compares DSDV-SQ with DSDV. Table 6.1 lists the constants used in our DSDV-SQ simulation.

<table>
<thead>
<tr>
<th>Periodic route update interval</th>
<th>12 s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Periodic updates missed before link declared broken</td>
<td>2</td>
</tr>
<tr>
<td>Initial triggered update weighted settling time</td>
<td>5 s</td>
</tr>
<tr>
<td>Weighted settling time weighting factor</td>
<td>5/6</td>
</tr>
<tr>
<td>Route advertisement aggregation time</td>
<td>1 s</td>
</tr>
<tr>
<td>Maximum packets buffered per node per destination</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 5.2. Constants used in the DSDV-SQ simulation.

5.2.2 Ad hoc on demand Multi-path Distance vector (AOMDV)

It has implemented the following optimizations, although the AOMDV protocol supports unidirectional routes, IEEE 802.11 requires a RTS/CTS/Data/ACK exchange for all uni-cast packets, limiting the routing protocol to using only bidirectional links in delivering data packets. It has implemented AOMDV to discover
only routes composed of bidirectional links by requiring that a node return all ROUTE REPLY messages to the requestor by reversing the path over which the ROUTE REQUEST packet. If the path taken by a ROUTE REQUEST contained unidirectional links, then the corresponding ROUTE REPLY will not reach the requestor, preventing it from learning the unidirectional link route. In Route Discovery, a node first sends a ROUTE REQUEST with the maximum propagation limit (hop limit) set to zero, prohibiting its neighbors from re-broadcast casting it. At the cost of a single broadcast packet, this mechanism allows a node to query the route caches of all its neighbors for a route and also optimizes the case in which the destination node is adjacent to the source. If this non propagating search times out, a propagating ROUTE REQUEST is sent. Nodes operate their network interfaces in promiscuous mode, disabling the interface’s address filtering and causing the network protocol to receive all packets that the interface overhears. These packets are scanned for useful source routes or ROUTE ERROR messages and then discarded. This optimization allows nodes to learn potentially useful information, while causing no additional overhead on the limited network bandwidth. Furthermore, when a node overhears a packet not addressed to itself, it checks the unprocessed portion of the source route in the packet’s header. If the node’s own address is present, it knows that this source route could bypass the unprocessed hops preceding it in the route. The node then sends a gratuitous ROUTE REPLY message to the packet’s source, giving it the shorter route without these hops. Finally, when an intermediate node forwarding a packet discovers that the next hop in the source route is unreachable, it examines its route cache for another route to the destination. If a route exists, the node replaces the broken source route on the packet with the route from its cache and retransmits the packet. If a route does not exist in its cache, the node drops the packet and does not
begin a new Route Discovery of its own. Table 6.2 lists the constants used in our AOMDV simulation.

| Time between retransmitted ROUTE REQUESTs (Exponentially backed off) | 400 ms |
| Size of source route header carrying n addresses | 2n + 2 Bytes |
| Time out for nonpropagating search | 25 ms |
| Time to hold packets awaiting routes | 25 s |
| Max rate for sending gratuitous REPLYs for a route | 1/s |

**Table 5.3.** Constants used in the AOMDV simulation.

5.2.3 **Enhanced Ad Hoc On-Demand Distance Vector (EAODV-LL)**

EAODV-LL is essentially a combination of both DSR and DSDV. It borrows the basic on-demand mechanism of Route Discovery and Route Maintenance from DSR, plus the use of hop-by-hop routing, sequence numbers, and periodic beacons from DSDV.

**Basic Mechanisms**

When a node $S$ needs a route to some destination $D$, it broadcasts a ROUTE REQUEST message to its neighbors, including the last known sequence number for that destination. The ROUTE REQUEST is flooded in a controlled manner through the network until it reaches a node that has a route to the destination. Each node that forwards the ROUTE REQUEST creates a reverse route for itself back to node $S$. When the ROUTE REQUEST reaches a node with a route to $D$, that node generates a ROUTE REPLY that contains the number of hops necessary to reach $D$ and the
sequence number for $D$ most recently seen by the node generating the REPLY. Each node participates in forwarding this REPLY back toward the originator of the ROUTE REQUEST (node $S$), creates a forward route to $D$. The state created in each node along the path from $S$ to $D$ is hop-by-hop state; i.e., each node remembers only the next hop and not the entire route, it could be done in source routing. In order to maintain routes, EAODV-LL normally requires that each node periodically transmit a HELLO message, with a default rate of once per second. Failure to receive three consecutive HELLO messages from a neighbor is taken as an indication that the link to the neighbor in question is down. Alternatively, the EAODV-LL specification briefly suggests that a node may use physical layer or link layer methods to detect link breakages to nodes that it considers neighbors [84]. When a link goes down, any upstream node that has recently forwarded packets to a destination using that link is notified via an UNSOLICITED ROUTE REPLY containing an infinite metric for that destination. The receipt of such a ROUTE REPLY, a node must acquire a new route to the destination using Route Discovery as described above.

**Implementation**

The initial implementation of EAODV-LL using periodic HELLO messages for link breakage detection as described in the EAODV-LL specification [84]. For comparison, it is also implemented a version of AODV that we call AOMDV, instead using only link layer feedback from 802.11 as in DSR, completely eliminating the standard EAODV-LL HELLO mechanism. Such an approach saves the overhead of the periodic HELLO messages, but does somewhat change the fundamental nature of the protocol; for example, all link breakage detection in AOMDV is only on-demand, and thus a broken link cannot be detected until a packet needs to be sent over the link, whereas the periodic HELLO messages in standard EAODV-LL allow broken links to
be detected before a packet must be forwarded. Nevertheless, it has found our alternate version AOMDV to perform significantly better than EAODV-LL, and so the report measurements from that version here.

| Time for which a route is considered active | 250 s |
| Lifetime on a ROUTE REPLY sent by destination node | 500 s |
| Number of times a ROUTE REQUEST is retried | 2 s |
| Time before ROUTE REQEWST is retried | 4 s |
| Time for which the broadcast id for a forwarded ROUTE REQUEST is kept | 2 s |
| Time for which reverse route information for ROUTE REPLY is kept | 2 s |
| Time before broken link is deleted from routing table | 2 s |
| MAC layer link breakage detection | Yes |

Table 5.4. Constants used in the EAODV-LL simulation.

In addition, it has also changed our EAODV-LL implementation to use a shorter timeout of 6 seconds before retrying a ROUTE REQUEST for which no ROUTE REPLY has been received (RREP WAIT TIME). The value is given in the EAODV-LL specification was 120 seconds, based on the other constants specified for EAODV-LL. However, a ROUTE REPLY can only be returned if each node along the discovered route still has a reverse route along which to return it, saved from when the ROUTE REQUEST was propagated. Since the specified timeout for this reverse route information in each node is only 3 seconds, the original ROUTE REPLY timeout value of 50 seconds unnecessarily limited the protocol’s ability to recover from a dropped ROUTE REQUEST or ROUTE REPLY packet. Table 6.3 lists the constants used in our AOMDV simulation.
5.2.4 Temporarily Ordered Routing Algorithm (TORA)

TORA [21, 79, 81] is a distributed routing protocol based on a "link reversal" algorithm [34] that finds and maintains routes via local relaxation of link direction. It is designed to discover routes on demand, provide multiple routes to a destination, establish routes quickly, and minimize communication overhead by localizing algorithmic reaction to topological changes when possible. Route optimality (shortest-path routing) is considered of secondary importance, and longer routes are often used to avoid the overhead of discovering newer routes.

The actions taken by TORA can be described in terms of water flowing downhill towards a destination node through a network of tubes that models the routing state of the real network. The network represents links between nodes in the network, the junctions of tubes represent the nodes, and the water in the tubes represents the packets flowing towards the destination. Each node has a height with respect to the destination that is computed by the routing protocol. If a network between nodes A and B becomes blocked such that water can no longer flow through it, the height of A is set to a height greater than that of any of its remaining neighbors, such that water will now flow back out of A (and towards the other nodes that had been routing packets to the destination via A).

Basic Mechanisms

At each node in the network, a logically separate copy of TORA is run for each destination. When a node needs a route to a particular destination, it broadcasts a QUERY packet containing the address of the destination for which it requires a route. This packet propagates through the network until it reaches either the destination, or an intermediate node having a route to the destination. The recipient of the QUERY then broadcasts an UPDATE packet listing its height with respect to the destination.
This packet propagates through the network; each node that receives the UPDATE sets its height to a value greater than the height of the neighbor from which the UPDATE was received. This has the effect of creating a series of directed links from the original sender of the QUERY to the node that initially generated the UPDATE. When a node discovers that a route to a destination is no longer valid, it adjusts its height so that it is a local maximum with respect to its neighbors and transmits an UPDATE packet. If the node has no neighbors of finite height with respect to this destination, then the node instead attempts to discover a new route as described above. When a node detects a network partition, it generates a CLEAR packet that resets routing state and removes invalid routes from the network. TORA is layered on top of IMEP, the Internet MANET Encapsulation Protocol [20], which is required to provide reliable, in-order delivery of all routing control messages from a node to each of its neighbors, plus notification to the routing protocol whenever a link to one of its neighbors is created or broken. To reduce overhead, IMEP attempts to aggregate many TORA and IMEP control messages (which IMEP refers to as objects) together into a single packet (as an object block) before transmission. Each block carries a sequence number and a response list of other nodes from which an ACK has not yet been received, and only those nodes ACK the block when receiving it; IMEP retransmits each block with some period, and continues to retransmit it if required for some maximum total period, after some time, the link to each unacknowledged node is declared decreased and TORA is notified. IMEP can also provide network layer address resolution, but we did not use this service, as we used ARP [88] with all four routing protocols. For link status sensing and maintaining a list of a node’s neighbors, each IMEP node periodically transmits a BEACON (or “BEACON equivalent”)
packet, which is answered by each node hearing it with a HELLO (or "HELLO-equivalent") packet.

**Implementation**

IMEP must queue objects for some period of time to allow possible aggregation with other objects, but the IMEP specification [20] does not define this time period, and we discovered that the overall performance of the protocol was very sensitive. After significant experimentation, we chose the packet overhead and routing protocol convergence, to aggregate HELLO and ACK packets for a time uniformly chosen between 200 ms and 300 ms, and not to delay TORA routing messages for aggregation. The reason for not delaying these messages is that the TORA link reversal process creates short-lived routing loops that exist from the time that the link-reversal starts until the time that all nodes that need to be aware of the reversal receive the corresponding. Delaying the transmission of TORA routing messages for aggregation, coupled with any queuing delay at the network interface, allows these routing loops to last long enough that significant numbers of data packets are dropped. The TORA and IMEP specifications [81, 20] do not define the precise semantics of reliable object delivery required by TORA, but experimentation showed that very strong semantics must be provided in order to prevent the creation of long-lived routing loops. In particular, all TORA objects must be delivered reliable, without any duplication. In addition all neighboring nodes in the ad hoc network must have a consistent picture of the network with regard to each destination. This implies that anytime a node A decides its link to a neighbor B has gone down, B must also decide that the link to A has gone down. We have implemented IMEP to provide this functionality, although the retransmission timeout and total number of attempts are not specified by IMEP [20].
We chose a retransmission period of 400 ms and a total timeout of 1200 ms, an adaptive retransmission timer should be added to the protocol. In-order delivery is enforced by, at each receiver node $B$, only passing an object block from some node $A$ to TORA if the block has the sequence number that IMEP at $B$ next expects from $A$. Blocks with lower sequence numbers may generate another ACK but are otherwise dropped. Blocks with higher sequence numbers are queued until the missing blocks arrive or until the maximum 1200 ms total timeout expires, at which point $B$ can be certain the object will never be retransmitted. By this point, $A$ will have declared its link to $B$ down, since it will not have received an ACK for the missing packet. To give the routing protocol at $B$ a picture consistent with that seen by the protocol at $A$, the IMEP layer at $B$ notifies its routing protocol that the link to $A$ is down, then notifies it the link is back up, and then processes the queued packets. Finally, we improved IMEP’s method of link status sensing by reducing it to a point that functions with minimum overhead yet still maintains all of the required link status information.

<table>
<thead>
<tr>
<th>BEACON period</th>
<th>1 s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time after which a link is declared down if no BEACON or HELLO packets were exchanged</td>
<td>2 s</td>
</tr>
<tr>
<td>Time after which an object block is retransmitted if no acknowledgment is received</td>
<td>400 s</td>
</tr>
<tr>
<td>Time after which an object block is not retransmitted and the link to the destination is declared down</td>
<td>1200 s</td>
</tr>
<tr>
<td>Min HELLO and Ack aggregation delay</td>
<td>130 s</td>
</tr>
<tr>
<td>Max HELLO and Ack aggregation delay</td>
<td>200 s</td>
</tr>
</tbody>
</table>

**Table 5.5.** Constants used in the TORA simulation.
During our experimentation with IMEP, we found that nodes need only send BEACON messages when they are disconnected from all other nodes. Suppose two nodes A and B, both of which have neighbors, transmit a single HELLO listing each of their neighbors once per BEACON period. If a bi-directional link exists between A and B, both nodes will overhear each other’s HELLOs and all other transmissions, causing each node to create a link to the other with “incoming” status. In subsequent HELLO messages, A and B will list each other, confirming that a bi-directional link exists between them. Table 6.4 lists the constants used in our TORA simulation.

5.3 EAODV-LL Extensions for Internet Connectivity

We have begun exploring several ideas in that design space [17]. We have already, extended the mechanisms of Route Discovery and Route Maintenance to support communication between nodes inside the ad hoc network and those outside in the Internet.

5.3.1 Integration with the Internet

In order to connect an EAODV-LL ad hoc network to the Internet, we first require that each node choose a single IP address, called its home address, by which it is known to all other nodes. By having a single home address, nodes in the ad hoc network maintain a constant identity as they communicate with nodes inside and outside the network. This notion of a home address is identical to that defined by Mobile IP [82]. As in Mobile IP, each node is configured with its home address and uses this address as the IP source address for all of the packets that it sends. Figure 6.2 illustrates node T2 inside the ad hoc network discovering a route to a node D outside the network. To demonstrate the generality of this technique, nodes T1-5 and E1 have not configured to know that they are part of a common subnet with E2 as a default router. The routing tables in T1-5 and E1 simply provide a default route out the
EAODV-LL virtual interface. When to D, the packet is given to the EAODV-LL code for delivery. Assuming T2 does not already have a cached route to D; it begins a Route Discovery for D. E2, which consults its routing table. If it believes D is reachable outside the ad hoc network, it sends a proxy ROUTE REPLY listing itself as the second-to-last node in the route, and marking the REPLY such that the ROUTE REQUEST from T2 targeting D propagates, it is eventually received by the gateway node T2 will recognize it as a proxy reply. If the target node D actually is inside the ad hoc network, then node T2 will receive a ROUTE REPLY from both E2 and D. Since T2 can distinguish which replies are proxy replies, it can prefer the direct route when sending packets to D.

T2 originates a packet destined

![Figure 5.2. Route Request for a node not in the ad hoc network being answered by the Foreign Agent.](image-url)
5.3.2 Integration with Mobile IP

Since node RN in Figure 5.1 must be able to participate in different IP subnets depending on its current location, it uses Mobile IP to connect to the Internet. Figure 5.3 shows an example where RN is homed in a subnet not belonging to the ad hoc network, but it has wandered into range of the ad hoc network. As described in Section 5.1.2, node E2 provides Mobile IP foreign agent services, in addition to being configured as a gateway between the ad hoc network and the Internet. As part of normal Mobile IP operation, RN periodically checks to verify that it is currently using the best means available to maintain connection with the Internet. We configured RN to operate in LAN mode as its top preference, to connect to the Internet via an EAODV-LL ad hoc network as second choice, and to connect via CDPD when no other options are available. When RN receives EAODV-LL packets, such as ROUTE REQUESTs, ROUTE REPLYs or data packets with EAODV-LL source routes on them, it knows that within range of an EAODV-LL network and enables that connectivity option in its Mobile IP code.

Figure 5.3. The roaming node RN registering with a homed node.
If node RN decides its best connectivity would be via the ad hoc network, it transmits a Mobile IP AGENT SOLICITATION piggybacked on a ROUTE REQUEST targeting the IP limited broadcast address (255.255.255.255). This allows the SOLICITATION to propagate over multiple hops through the ad hoc network, though gateways will not propagate it between subnets. When the foreign agent at E2 receives the SOLICITATION, it will reply with an AGENT ADVERTISEMENT, allowing RN to register itself with this foreign agent and with its home agent as a Mobile IP mobile node visiting the ad hoc network. Once the registration is complete, the mobile node’s home agent will use Mobile IP to tunnel packets destined for mobile node RN to the foreign agent at E2, and E2 will deliver the packets locally to the mobile node using EAODV-LL.

5.4 Layer 3 Mechanisms for Acknowledgments and Retransmission

Since the Wave LAN-I radios do not provide link-layer reliability, we implemented a hop-by-hop retransmission and acknowledgment scheme within the EAODV-LL layer that provides the feedback necessary to drive EAODV-LL’s Route Maintenance mechanism. One interesting aspect of our ARQ scheme was the use of passive acknowledgments [58], which significantly reduces the number of acknowledgment packets transmitted when compared to acknowledgment schemes that acknowledge every packet.

5.4.1 Implementation of EAODV-LL

Our implementation utilizes passive acknowledgments whenever possible, meaning that if node A originating or forwarding a packet hears the next hop node B
forward the packet, A accepts this as evidence that the packet was successfully received by B. If A fails to receive a passive acknowledgment for a particular packet that it has transmitted to B, then A retransmits the packet, but sets a bit in the packet’s header to request an explicit acknowledgment from B. Node A also requests an explicit acknowledgment from B if B is the packet’s final destination, since in this case, A will not have the opportunity to receive a passive acknowledgment from B.

To avoid the inefficiencies of a stop-and-wait ARQ scheme, node a uses a buffer to hold packets it has transmitted that are pending acknowledgment plus an identifier based on the IP ID field [89] to match acknowledgments with buffered packets. This acknowledgment procedure allows A to receive acknowledgments from B even in the case wireless link from A to B is unidirectional, since explicit acknowledgments can take an indirect route from B to A. During an average run, 96 percent of the acknowledgments used a direct one-hop route, and 10 percent of the acknowledgments were sent over routes with multiple hops. While this strongly suggests the presence of unidirectional links in the network, it does not support a conclusion that 10 percent of the packets travel over a unidirectional link. Once a multiple-hop route for acknowledgments is discovered, it may continue to be used for some period of time even after the direct route begins working again. When performing retransmissions at the EAODV-LL layer, we also found it necessary to perform duplicate detection, so that when an acknowledgment is lost, a retransmitted packet is not required forwarded through the network multiple times. The duplicate detection algorithm used in our implementation specified that a node should drop a received packet if an identical copy of the packet was found in a buffer awaiting either transmission or retransmission. We found that this simple form of duplicate
suppression was sufficient, and that maintaining a separate history of recently seen packets was not necessary.

5.4.2 Heuristics for Selecting Timeout Values

Early in the design of our retransmission mechanism, we found that contention for the wireless medium produced enough variance in the Round Trip Time (RTT) between neighboring nodes that using a fixed value for the retransmission timer was not practical, and that adaptive retransmission timers were required. Our initial implementation of an adaptive retransmission timer employed the scheme used by TCP [94, p. 301], where a smoothed RTT estimator (srtt) and a smoothed mean deviation (rttvar) are maintained independently for each next hop to which a node is communicating. The retransmission timeout (RTO) is then computed as:

\[ \text{RTO} = 4 \times \text{srtt} + 3 \times \text{rttvar} \]

Unfortunately, the variance in RTT prevented this implementation from performing adequately. Frequently, the RTO would not adapt quickly enough to congestion in the network, causing packets to be retransmitted unnecessarily and creating even more congestion. It also suffered from the fact that the RTO to each next hop was computed independently, while the need to defer transmissions due to congestion is common across all neighbors accessed via the same network interface. We found that several simple methods of reacting to increasing congestion did not work. For example, we tried an algorithm that feeds into the smoothing function for RTT estimation an RTT sample of twice the current smoothed RTT estimate whenever there is a retransmission timeout. This algorithm causes the value of the RTT estimator, and hence the retransmission timer, to tend to diverge and remain pegged at its maximum value, even after congestion has subsided. We developed a successful retransmission timer algorithm by including a heuristic estimate of the level of local congestion, so that the retransmission timer could react quickly to changes. One of the simplest ways
for a node to measure congestion in the network is to look at the length of its own network interface transmit queue. Specifically, if more than 5 packets are found in the interface transmit queue meaning that congestion is starting to occur we increase the value of the retransmission timer 20 ms for each packet in the queue. Assume that there are N packets in the network interface queue. For 5 packets, the retransmission timeout is computed as before: This heuristic allows the retransmission timer to increase quickly during periods of congestion and then return just as quickly to its computed value once the congestion dissipates. In 4100 measurements over several runs, approximately 80% of the packets transmitted use the minimum retransmission timer value of 40 ms. However, for the other 25% of the packets, the retransmission timer adjusted itself to values between 60 ms and 920 ms. The wide range indicates that an adaptive retransmission scheme is required for good performance if acknowledgments are implemented above the link layer.

5.5 Wireless Propagation

In our simulation network operated without the layer 3 acknowledgments and retransmission scheme, the average packet loss rate over a single hop was measured as 11 percent. With the ARQ scheme described in Section 5.3, the average loss rate over a single hop dropped to 5 percent. The losses are highly correlated with position, but also demonstrate the highly variable nature of wireless propagation due to scattering, multi-path, and shadowing effects in the real world. In a result that surprised us, we also found that the probability of a successful packet reception rolls off very slowly with distance. At twice the nominal transmission range, the probability of successfully receiving a packet is still 25 percent.
5.5.1 Small and Large Scale Fading

As an example the highly variable nature of wireless propagation, Figure 5.4 shows the signal level at which packets were received when sent by node T4 directly over one hop to T3. This data was measured during a full run of the simulation, so all the nodes (T1-T5) were in motion and exchanging data when it was recorded. Dropped packets are marked with a '+' and shown at -90 dBm; the actual value of the received signal strength is unknown because the network interface hardware does not report signal strength for packets that not successfully received. From 1390 s to 1404 s the figure shows most packets being successfully delivered, but with significant numbers of multi-path fades of 20 dB or greater. Scattered among these are packet losses, most of which the link-layer retransmission algorithm is able to correct. However, given the impracticality of predicting such losses, link-layer retransmission is the only defense against unrecoverable packet loss.

![Figure 5.4. Received signal strength of packets sent directly between two nodes during a full run Simulation.](image)

That link-layer retransmission can come from the link-layer directly, as in IEEE 802.11 DCF, or from lower levels of the routing layer, as in our implementation
of EAODV-LL. The variability in propagation creates a significant level of inherent packet loss with which higher layers must be prepared to cope. During the period from 1405 s to 1410 s, however, Figure 5.4 shows what can be best described as a routing protocol error. Although other paths were available, the protocol continued to send packets directly from T4 to T3 when nearly half those packets were being lost. Today's routing protocols must be extended in at least one of two directions. They must predict the continued decrease in average signal level that occurred from 1400 s to 1406 s and switch to a new route, or they must maintain state to record that a particular link is becoming loss and avoid it until such time as it functions well again.

5.5.2 Correlation of Location with Packet Loss

As part of conducting an initial survey of the site, we found it particularly helpful to obtain a rough characterization of the site's propagation environment. We had two nodes moves the speed at about 30 Km/hr (20 miles per hour), one following the other at a separation of 90 meters, with the trailing nodes transmitting 1024-byte packets to the lead node 10 times per second. The nodes made three laps of the course, a total driving time of about 840 seconds. The position of the nodes during the intervals, when more than 1% of the total packet loss occurred. Essentially the loss bursts occurred while the nodes were on the straightest part of the simulation with clear line-of-sight to each other.

5.6 Emulation of Ad hoc Networks

In general, evaluating a real system with just simulation is not practical. It is difficult to model the full complexity of a real application inside a simulator, and most real applications systems have been significantly performance tuned. Having to repeat this effort to create a simulation model is not so effective. Network emulation allows application and protocol designers to combine components of real code or real
hardware with simulated components, thereby creating a composite system. Such a composite system enables more flexible experiments than are possible with either a purely simulated or completely implemented system.

In effect, emulation blurs the line between simulation and full-scale implementation. It allows designers to maximize their understanding of their system by varying the trade-off between reliability to real-world effects and experimental control. If designers create composite systems with more of the system implemented with real components, they will see more real-world effects in the system's behavior. If they create composite systems having more simulated components, they will be able to achieve better experimental results, since simulated components have better repeatability than real components. The most important characteristic of a network emulation methodology is its reliability, meaning how closely the inputs to the real components of the composite system match the inputs that would be experienced by the components in a real deployed implementation. When the emulation methodology causes real components to behave differently than they would in a real network, we will call these glitches emulation artifact. Create the environment of an ad hoc network with sufficient reliability that a performance implementation of network protocols or applications conducted in the emulation system will match the performance of that protocol or application in a full-scale implementation.

5.7 Emulation Methods

They are two different types of emulation multi-hop ad hoc networks: trace modulation and direct emulation. The goal of both emulation methods is to cause the packets sent between physically-implemented components to behave as if the components were part of nodes moving and interacting in a deployed ad hoc network.
We will assume that there are two nodes, A and B, with physically-implemented components and that components are communicating.

5.7.1 Trace Modulation

Trace modulation is based on a communication-theoretic model of networks. As shown by the overview in Figure 6.14, a logical channel model is placed in the path of all Protocol Data Units (PDUs) flowing between layer node A and layer node B. The logical channel model is driven by a trace file that describes the properties that the logical channel should have, such as its bandwidth, delay, or packet drop rate. As in classical communication theory, the logical channel should be thought of as a very flexible abstraction. It can be used to model either a one-hop link between two nodes or an entire multi-hop network, depending solely on the trace file used to drive it.

![Figure 5.5. Logical operation of trace modulation.](image)

The trace file specifies how the channel properties change as a function of time, which enables trace modulation to model the connectivity changes that occur in mobile ad hoc networks. For example, trace modulation can represent the change in connectivity caused when two nodes move out of range of each other, by setting the drop rate of the channel between A and B to 100 percent when A and B are no longer in range of each other. More sophisticated trace files can even specify the roll-off in
the probability of successful packet reception as the nodes move apart, by increasing the reliability of the emulation. Trace modulation suffers from two major drawbacks. First, it is difficult to create a trace file that accurately characterizes what happens to the PDUs sent from A to B in the physical network that the emulation is attempting to realize. Second, trace modulation depends on an assumption that all the complexities of a communication path between two nodes, which could stretch over an entire multi-hop network, can be well modeled by a logical channel model with relatively few parameters. This assumption has only been validated in a few specific cases [77].

5.7.2 Direct Emulation

In contrast to trace modulation, which assumes we can model a logical communication path with relatively few parameters, direct emulation is predicated on the notion that we can have a detailed simulation of the ad hoc network run in real-time. Rather than abstract into the parameters of the trace file all the behavior of the network that connects nodes A and B, direct emulation uses a detailed simulation model of the network to directly model all the salient behavior of the network.

In order to implement direct emulation, we must arrange for the actions of the physical components of the composite network to affect the behavior of the modeled components inside the simulation, and vice-versa. Whenever a physical component causes an externally visible event which could affect the behavior of the simulated components of the network, we inject an event into the real-time simulation that represents the action taken by the physical component. Similarly, whenever a simulation event occurs that would be visible to a physically-implemented component of the network, the simulation generates an event on the physical component that is equivalent to what would happen to the component if all the simulated components were also physically implemented. Figure 6.14 shows an example of the operation of
Chapter 5: Implementation of EAODV-LL Protocol

direct emulation. Consider a packet sent from node A to node B in this network. As shown by the emulation cut drawn in Figure 6.14., the application and transport layer on A and B are physically implemented and the rest of the network is simulated. After a real application running on A originates a packet, it passes through the network stack until it reaches the level of the emulation cut shown at marker 1. As it crosses the emulation cut (marker 2), the data in the packet 2 is transported to the machine running the real-time simulation, and wrapped up into a simulation event. The simulation event must preserve enough information to enable the conversion of the event back into the original object. The event is handled by the simulation as normal, creating other events in the network stacks of the simulated nodes, until it eventually reaches the point shown by marker 3 in the simulated network layer of node B. When the simulation detects the event crossing back over the emulation cut, it unwrap the data from the event. The emulation system transports the data to the physical implementation of node B, where it is passed up into the real transport layer as shown at marker 4. With direct emulation, the length of time it takes B to receive the data after A “sends” it or whether the data is received at all is determined by the operation of the simulation. If the simulation accurately models the real-time behavior of some ad hoc network, then the communications between A and B will see the same effects if they were actually participating in a real implementation of that ad hoc network. As with trace modulation, direct emulation also has two major drawbacks. The first is the need to minimize the time it takes to move data between physical nodes and the simulation machine. The second is the need to prevent the simulation from falling behind real-time operation. A failure to meet these needs will result in the introduction of emulation artifacts that can distort the performance of the system.
Chapter 5: Implementation of EAODV-LL Protocol

Figure 5.6. Logical operation of a direct emulation.

5.8 Summary

Our simulation successfully demonstrates that EAODV-LL, and the on-demand mechanisms it embedded, it can be successfully implemented in real networks carrying meaningful traffic across multiple hops. EAODV-LL not only routes packets between the nodes in the ad hoc network, but it seamlessly integrates the ad hoc network into the Internet via a gateway. EAODV-LL was also extended to integrate with Mobile IP, allowing nodes to roam transparently between the ad hoc network and normal IP subnets. The simulation is novel among non-military simulations in the completeness of its implementation and the relatively large number of nodes that comprise it. Among all ad hoc network simulations, the rate of topology change is significantly greater than the previous results. In particular, this chapter highlighted the problems caused when packets occasionally travel further than the normal wireless transmission range, a phenomenon that might appear neutral or beneficial at first. Our measurements of a full-scale across network range of the environments by successfully implemented. In the simulation results the packet delivery in worst environments also performs better in better environments and
Chapter 5: Implementation of EAODV-LL Protocol

reduces the overhead using our new protocol EAODV-LL. The integration with mobile IP is performance is less. Packet delivery ratio has not improved compared with existing system its remains same. The EAODV-LL protocol has connected with the internet, but limited number of nodes only it has supported. It has not completely implemented in thesis. These two objectives are still improving and trying to mitigating the performances. The above design and implementation mobile ad hoc wireless network in simulation and the experimental results has been taken and the result has been compared with other protocols in the following chapter.