

IMPLEMENTATION OF DSR ROUTING PROTOCOL ON SUNSPOT WIRELESS SENSORS

Jeegar V. Brahmakshatriya
Dept. of Computer Science
Rochester Institute of
Technology
jvb3350@cs.rit.edu

Mihir J. Daftari
Dept. of Computer Science
Rochester Institute of
Technology
mjd6427@rit.edu

Wisam F. Kadhim
Dept. of Computer Science
Rochester Institute of
Technology
wfk9421@cs.rit.edu

Abstract

An ad-hoc network is a collection of wireless mobile hosts coming together to form a complete network without the aid of any established infrastructure or centralized administration. Dynamic Source Routing protocol (DSR) is a simple and efficient routing protocol designed specifically for use in multi-hop wireless ad-hoc networks. In this paper we show DSR implementation on the Sun's SPOT sensors. We provide implementation details, minor modifications to RFC4728 message formats to make it work on the sensors and possible future optimizations of the protocol.

I. Introduction

Using wired hosts like phone and computers may not be always possible or preferable. Places like a disaster site or some remote place may not have infrastructure for wired network, or some people may just prefer to use wireless devices for convenience. Today we see a lot of wireless devices that are communicating with each other and also with traditional networks such as the Internet.

Routing in ad-hoc networks is difficult because of the changing dynamics of the network. The topology of the network may be changing constantly due to mobility of nodes. Wireless links may be created or destroyed as the nodes move. Also nodes used in ad-hoc networks are generally battery powered and have limited processing power. We would want to optimize their resource utilization. Ad-hoc networks may be deployed at unknown terrains which may have hostile environments or hazardous conditions which may cause frequent link failures [3].

Ad-hoc routing protocols are basically of two types: Pro-active Routing Protocols and Reactive Routing Protocols. In pro-active routing protocols, the nodes in the network maintain routing tables. They exchange these tables periodically to maintain the freshest routes available in the network. The nodes maintain routes to all other nodes regardless the need of

them. This method provides the node with a route immediately. The main disadvantages of such protocols are higher amount of data and network traffic for maintenance of routes and slow reaction on restructuring and failures. Some of the popular pro-active routing protocols are Destination-Sequenced Distance Vector Routing (DSDV) and Ad-hoc Wireless Distribution Service [2].

In reactive routing protocol, the nodes in the network do not maintain routes for all the nodes in the network. They maintain only the ones that are currently being used. When they need a route to a node to which they do not have a route to, they initiate a route discovery. Due to this method, the network traffic is reduced considerably. The main disadvantages of such protocols are high latency time in route finding and excessive flooding for route discovery can lead to network clogging. Some popular reactive routing protocols are Ad-hoc On-demand Distance Vector Routing Protocol (AODV) and Dynamic Source Routing (DSR) [2].

II. Sun SPOT Wireless Sensors

The Sun SPOT Devices are small, wireless, battery powered experimental platforms. They are programmed almost entirely in Java to allow regular programmers to create projects that used to require specialized embedded system development skills. The hardware platform includes a range of built-in sensors as well as the ability to easily interface to external devices [4].

The current configuration of the Sun SPOT platform, the eSPOT, has a main processor running the Java VM "Squawk" and which serves as an IEEE 802.15.4 wireless network node. The protocols are implemented on top of the MAC layer of the 802.15.4 implementation. Fig. 1 shows the network stack in the SPOT. It is designed to experiment with the wireless routing protocols and many other sensor applications. The top level of the stack implements protocols similar to TCP and UDP to provide reliable and datagram based applications. The layer underneath is the lowPan

layer which handles the packet fragmentation and reassembly and the layers underneath together implement the physical MAC layer. The lowPan layer is designed to talk any routing manager that implements the class IRoutingManager. An AODVManager is provided in the stack as the default routing manager [4]. In our project we created the DSRManager which replaced the AODVManager from the stack.

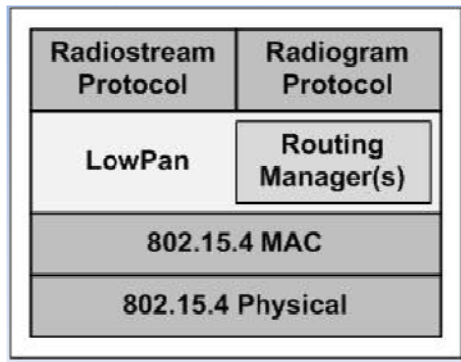


Figure 1: Radio stack in Sun SPOTs.

III. DSR Implementation

The implementation of DSR is based upon rfc4728 [1] with some minor modifications to make it work on the sun SPOT sensors. The modifications had to be made since, unlike other routing protocols, DSR adds its own headers to the packet sent across the network. But in sun SPOT sensors the lowPan layer overwrites some of the information in the IP header needed by DSR.

DSR operation can be divided into three main tasks: route discovery, in which DSR determines the route to the destination node in the network; route maintenance, in which DSR recognizes and fixes broken paths; and forwarding a data packet through the network. The header in each packet indicates the purpose of the packet.

Before we discuss the operation of DSR we will discuss the route cache. It is the central data structure for each node using DSR. It contains the node's knowledge of the current network topology. The DSR RFC suggests two different ways to implement the route cache. The first is called the "link cache" where in each node maintains one single unified graph data structure of this node's current view of the entire network topology. To search for a route the node has to use graph algorithms such as Dijkstra's shortest path algorithm. The other approach is the called the "path cache". Here node keeps track of the path to all destinations from itself [1]. This is easier to implement as it does not need any complex algorithms to calculate the path but the size of the cache can grow for a big network. The DSR specification also suggests that each node could have multiple routes to the destination [1]. We have implemented the path cache and for simplicity we have changed it so that each node just maintains a single route to the destination. The routing cache is implemented as a hash table which maps a destination

address to the path to that destination. Expiration times are used to ensure that the routes in the cache are not stale. The expiration times are updated each time a route is used. Fig.2 shows the data structure of our implementation for routing cache.

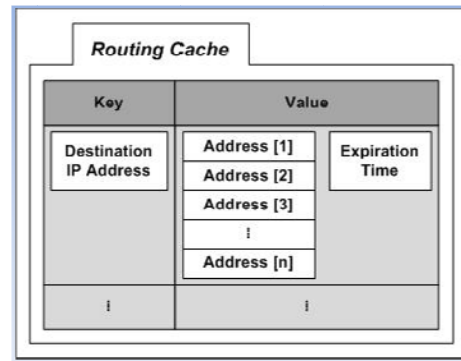


Figure 2: Routing cache structure.

III.a. Route Discovery:

Route discovery is used to find a route to another node in the network. Since DSR is an on-demand protocol it will initiate route discovery only when it needs to send a packet to the destination and it does not already have a route to that node. The host initiating the route discovery will broadcast a route request packet which will be received by all the nodes within the wireless transmission range of it. Fig.3 shows the RREQ message format.

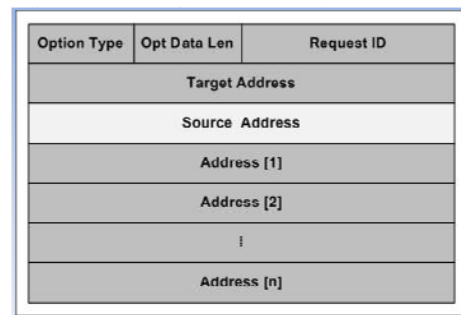
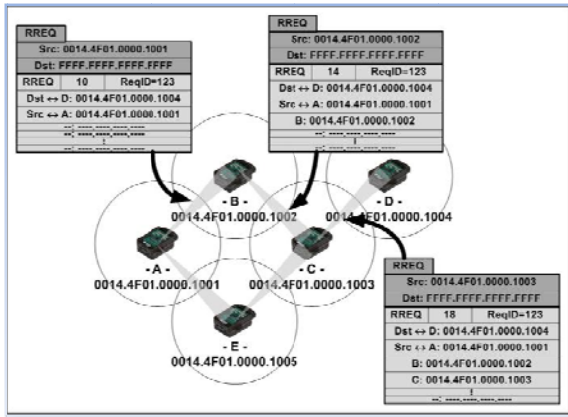


Figure 3: RREQ message format.

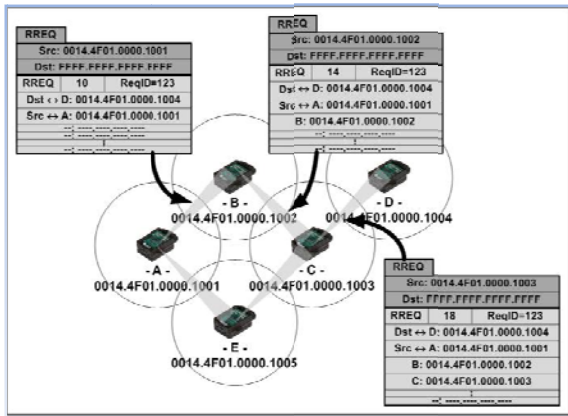
The source address in the IP header identifies the initiator of the request and the target address in the packet identifies the node for which the route is requested. In addition to the source and the destination IP addresses the packet also contains a route record which contain accumulated addresses of the sequence of hops taken by the route request packet as it is propagated through the ad-hoc network during this route discovery. Each node, which is not the target node, just appends its address at the end and then rebroadcasts the packet. The route request thus propagates through the ad-hoc network until it reaches the target host, which then replies to the initiator.

Notice that this process requires that the source address in the IP header always remains the address of

the originator of the route request packet. This was not possible in the sun SPOTs without making changes to the lowPan layer. To bypass this we added the source address as the part of the RREQ message. Fig.4 (a) shows the process of RREQ flooding, where node A wants to send a packet to node D, so it initiates the Route Request message. Node B and C append their addresses in the RREQ message and rebroadcast it. When D receives the message it will just follow the reverse path back to the source. Notice the highlighted header in RREQ message contains the addresses that are added by lowPan layer and the remaining packet is created by the DSR routing layer.



(a) RREQ flooding through the network.



(b) RREP back to source with route path.
Figure 4: Route discovery process.

This flooding required for route request can really slow down the network. So it is very important to reduce this as much as possible. The two main techniques used to do this are route replies by intermediate nodes and maintaining the request table to avoid re-broadcasting of previously seen route requests. When a node receives a route request it checks in its route cache to see if it has the path to the target address. If it does then it just replies to the originator and does not re-broadcasting the route request.

The second technique uses a data structure called Request Table. The request table keeps track of the

recently seen route requests (identified by the unique request ID in RREQ message) and the originator of the route request. If a node receives a route request which it has already seen then it will not propagate it further. Our implementation of the request table is again a simple hash table with the destination IP address as the key and the originator and the request ID as the value. This table can grow as the node sees more and more route request and so we use expiry time to clean up the table. Fig.5 shows the data structure for request table.

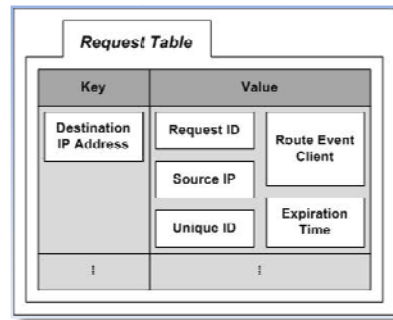


Figure 5: Request Table.

The flow chart shown in Fig.6 explains the operation that a node performs upon receiving a route request message.



Figure 6: Flow chart showing the logic for a node upon receiving a route request message.

Fig. 7 shows the Request Table in action. In the figure we can see how node C uses the Request Table to avoid re-broadcasting the same RREQ message twice. It receives the RREQ message for node D from node B at time t. The request ID for the message is 123. Node C enters this in its Request Table as it has not seen this message before. And so when it receives the RREQ message for node D again with request ID 123, it finds it in its Request Table and so does not re-broadcast it.

Fig. 8 shows the use of Routing Cache to reduce flooding. In the figure node E initiates a route request to node D. And so when C receives the RREQ message it checks its own route cache before forwarding the message. In this case node C already has learned the route to D and has it in its Routing Cache. So it replies to node E with the route and does not broadcast the RREQ any further.

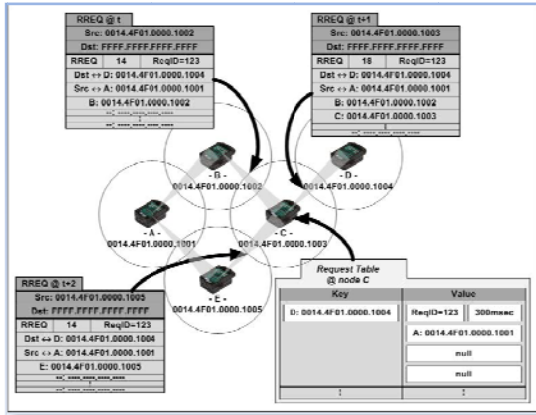


Figure 7: An example of route discovery with use of request table to reduce RREQ flooding.

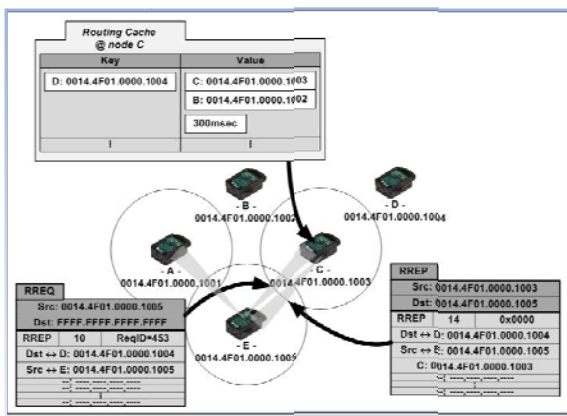


Figure 8: An example of route discovery with use of routing cache to reduce flooding.

Once the destination node receives route request it replies to the initiator with a route reply packet. The specification suggests that the destination node reply to the sender by using the route in its route cache, or by reversing the path found in the route request packet. We use the second approach where the destination node reverses the path since we have a device that has two way communications. Also the route replies are unicast messages. Fig.4 (b) shows the RREP path back to source.

Notice that the target address of the route reply is the address of the originator of the route request and the source address of the route reply is the address of the node replying to the route reply. Also all the route reply messages are unicast message to the next hop on the path to the target address. Here again the header of the packet was being overwritten by the lowPan and so we included the details in the packet itself. The part of the packet highlighted was modified.

III.b. Route Maintenance:

Every time a node sends a packet using a DSR source route, it is required to verify the reachability of the next hop on the route. To do this, it can add an acknowledgement option to the DSR header. The next-

hop node, on receiving the packet, will send back a DSR message containing an acknowledgement option. If the sender does not receive an acknowledgement in a certain timeout period, it retransmits the packet. The DSR specification mandates a data structure to hold the packets that are currently waiting for an acknowledgement [1].

Our implementation did not have to specify an acknowledgement request in the DSR header as acknowledgements are taken care of by the LowPan. LowPan on the receiver sends acknowledgements to the sender's lowPan. LowPan communicates this to DSR. If the DSR does not receive an acknowledgement from lowPan, it retransmits the message for maximum 3 times. If it does not receive an acknowledgement within those retransmits, it declares that link as broken and sends out a RERR to the previous hop. This RERR is propagated back to the originator. The originator removes all the entries from its route cache that uses that path so that it does not use that route again. The originator calls for another route discovery for the target. If the DSR receives an acknowledgement from lowPan before the maximum number for retries, it removes the message from the maintenance buffer.

III.c. Packet forwarding:

Forwarding a data packet in the network is really simple in DSR as the entire path is specified in the packet. The node just looks at the next hop and forwards the packet with acknowledgements taking place at every hop along the way. If the data packet comes from an application then the node uses its route cache and constructs a route message and encapsulates the data in the packet and forwards it. Fig.9 shows the logic used by node upon receiving data packet.

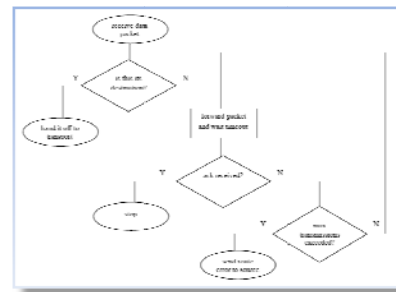


Figure 9: Flow chart showing the logic for a node upon receiving a data message.

IV. Future Enhancements

DSR can be designed to set up in promiscuous mode to hear about Route error messages. This will allow the node to learn about route failures from messages that are not destined for it. This way it will learn faster about route failures.

Traffic flow in the network can be reduced by piggybacking small data packets on route discovery packages. Traffic flow can also be reduced by specifying maximum number of hops over which a

route discovery packet can propagate. In the current scenario, if node A sends a route request to C, the entire network will be flooded with RREQ message where as that would not be need as C is just 1 hop away from it. B being on the other side forwards the message to others. This can be avoided using the Hop Count. When A initiates route discover, RREQ will have hop count as 1. If a route reply is not received in time out time, another RREQ is sent with a hop count of 2. This way the network will not be congested with RREQ packets.

V. Conclusions

Sun SPOTs are upcoming mobile ad-hoc devices with various applications. The learning curve was steep but we managed to implement Dynamic Source Routing protocol on Sun SPOTs. The challenging part was to adapt our DSR implementation to lowPan layer. LowPan is designed to work with Ad-hoc On-Demand Distance Vector Routing protocol (AODV). There were many places where we had to change the original specifications of DSR to implement it on the Sun SPOT devices. We made some changes in the message formats as mentioned earlier.

Compared to Pro-active routing protocols, there is negligible network traffic for maintaining routes as there is no periodic exchange between the nodes for maintaining routes. The most difficult part in DSR is route discovery. There is a fear of flooding the network with route request messages but that is taken care of by using unique request ID. After route discovery, routing is very simple. As the name suggests, the originator (source) does the routing. The originator tells intermediate nodes which route to take by putting the entire route in each message. The receiver pulls out the address and sees what the next hop should be. This makes routing easier for the nodes but the main drawback is that the packet size increases with the length of the route. This may cause high overheads. Using IPv6 with DSR is impractical because the address length is 128 bits.

References

- [1] D. Johnson, Y. Hu and D. Maltz, "The Dynamic Source Routing Protocol (DSR) for Mobile Ad-hoc Networks for IPv4", rfc4728, IETF, <http://www.ietf.org/rfc/rfc4728.txt>, 2007.
- [2] "List of Ad-hoc Routing Protocols", http://en.wikipedia.org/wiki/Ad_hoc_routing_protocol_list.
- [3] Stephen Mueller, Rose P. Tsang and Dipak Ghosal, "Multipath Routing in Mobile Ad-hoc Networks: Issues and Challenges", <http://www.cs.ucdavis.edu/~ghosal/Research/publications/stephen-lncs-multipath-survey-paper-2004.pdf>.
- [4] Sun Microsystems, "Sun Small Programmable Object Technology (Sun SPOT) Developer's Guide", V3 "Purple" release, <http://www.sunspotworld.com/docs/Purple/spot-developers-guide.pdf>, 2007.