Implementation of Dyamic Source Routing using 802.15.4 on XBee Series 1 Modules

Tanmay U. Sane, Samuel L.Shue, James M. Conard
Department of Electrical and Computer Engineering,
University of North Carolina Charlotte
Charlotte, NC, USA
slshue@uncc.edu, jmconrad@uncc.edu

***Abstract*— An implementation of Dynamic Source Routing on 802.15.4 using XBee Series 1 modules is presented. This implementation demonstrates the use of Dynamic Source Routing to determine the route from initiator (source) node to target (destination) node and used it to deliver message packets within an intra-network of wireless motes. The wireless motes compromise of Atmega 328P based microcontroller board (Red Board) interfaced with XBee Series 1. The algorithm itself searches for the desired route based on first come first serve basis and uses it to forward the message packet to the target node. Due to the dynamic nature of the protocol, the network has self-healing ability. The software library developed in the course of this implementation provides the user an interface to implement customized multi-hopping on XBee Series1 due to absence of any underlying operating system.**

Keywords*— *XBee Series 1 (XBee S1); Dynamic Source Routing (DSR); SubMinature version A (SMA)

I. INTRODUCTION

The past decade has seen a considerable rise in the creation of wireless nodes and their deployment in wireless sensor networks in a variety of applications and fields [9]. Wireless development platforms such as the MICAz, TelosB, Imote etc. have been fairly popular. Most of these platforms are supported by an underlying operating system or proprietary software. Proprietary software complicates research by adding unknown processes to an implemented idea, making performance evaluation more difficult [6]. Moreover, the existence of underlying operating system avoids user customization of wireless module firmware. Also a wireless node has limited range of transmission and relies on multi-hop communication to achieve end to end message delivery. Although, multi hopping is beneficial towards reliable message delivery, duplication of messages can cause increased network traffic which affects the network adversely. Lastly, wireless sensor nodes/modules are cost sensitive and require high initial infrastructure. This paper aims to address these issues in the light of point-to-point and point-to-multipoint communications.

In order to resolve driver and OS related issues an open source WSN development platform (Motesquito [6]) Motesquito is under development. The XBee Interface layer library and hardware abstraction layer library developed for this board form the basic backbone of this implementation [8]. The paper is based on developing a networking interface to the existing interface and abstraction layers, as illustrated in Fig 1. This networking layer makes function calls to the layers below it and reduces user interaction related to XBee and hardware layer initializations, transmission and reception of XBee packets. The networking layer is responsible for route searching, message delivery and message acknowledgment reception and for route maintenance. Thus, it provides routing functionality with the wireless modules discussed further. A suitable packet structure was created as per the specifications of the selected routing algorithm [2]. The Physical Layer compromises of the microcontrollers resources.

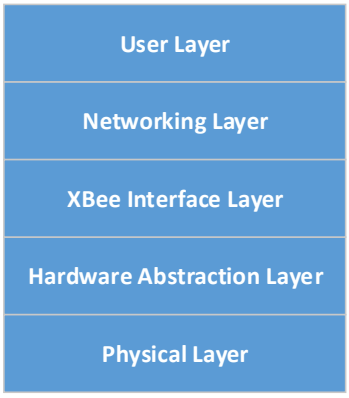

Fig. 1. *Software Layer Organization*

II. BACKGROUND

Wireless ad hoc networks use various protocols in order to achieve routing. In order to design a network layer implementation protocols such as the ADOV (Ad hoc On-demand Distance Vector) Routing and (DSR) Dynamic Source Routing were considered. The usage of periodic proactive messages to track neighboring nodes and higher latency involved to recover from broken routes [1] in case of AODV were found to be inadequate for the implementation. In contrast to this, DSR is a source-routing scheme, where the entire end-to-end route is included in the header of the message being sent [1]. The key advantage to source routing is that intermediate nodes do not need to maintain up-to-date routing information in order to route the packets they forward, since the packets themselves already contain all the routing decisions [3]. The on-demand nature of the protocol eliminates the need for the periodic route advertisement and neighbor detection packets present in other protocols [3]. This scheme consists of two

978-1-4673-7300-5/15/$31.00 ©2015 IEEE

methods, namely Route Request and Route Maintenance. Route Request is used to determine the route to the target node whereas Route Maintenance is used to maintain the validity of the routes.

In order to avoid excessive flooding in the network it is essential to limit the number of hops taken by a broadcasted Route Request packet. This requirement is easily met by the 802.15.4 firmware for point-to-point or star topology on Series 1 [4]. This feature isn't available on XBee Series 2 (ZigBee) due to existing AODV (DigiMesh) implementation [5]. Moreover, the XBee Interface and Hardware Abstraction layer libraries are portable for both XBee Series 1 as well as Series 2[6]. Due to this reason, XBee Series 1 was selected for this implementation.

The paper is further dived into various sections. Section III deals with the challenges faced and the design decisions made to resolve them. Section IV provides a detailed description of the results obtained during the various protocol steps. Section V concludes the paper with the inferences derived and sheds light onto the areas that need further research.

III. DESIGN

The design section deals with the Hardware and Software design aspects of the implementation.

A. Hardware Design:

This sections deals with interfacing the indicator LED, XBee radio to the microcontroller and the test bed creation. The ATmega328P based RedBoard was used for the implementation.

a) XBee Interfacing: Each wireless node or moduleis formed by interfacing a XBee Series 1 module to an ATmega 328P based microcontroller board. Herein the Rx pin of the microcontroller is connected to the DOUT pin of the XBee and Tx pin of the microcontroller is connected to the DIN Pin of the XBee. The XBee module is powered using the on board 3.3 V supply of the microcontroller board.

b) LED Indicator: An LED indicator was interfaced with the PORTD pin7 to indicate reception of messages over the USART.

B. Software Design

The DSR protocol can be divided into two phases. The first phase is the Route discovery phase followed by the Messaging and Route Maintenance phase. The design challenges included creation of the DSR packets, classification of packets, packet handling across layers (XBee Interface and DSR Layer) and status reporting. This section discusses in detail the DSR packet, its types and their function, the components of the route cache, the recent request list and the processing and handling of the various received message types.

a) Software Organization:

The XBee Transreceiver transmits and receives XBee command and data packets. Every time a DSR packet is transmitted it is encapsulated in the data section of the XBee packet and then transmitted through the radio (Fig.3). Similarly, whenever an XBee packet is received, the DSR acket section of its data is extracted by the networking layer (DSR) layer from the XBee data section for further processing. All the received or transmitted DSR packets are classified depending upon their identifier field value. An USART interrupt service routine (ISR) is invoked every time data is received. This ISR further invokes XBee Interface Layer functions which extract the XBee packet from the received data. A callback function at the XBee Interface Layer passes control packet to the new packet reception function in the networking layer. In the new packet reception function, the DSR packet is extracted from the XBee packet. And each packet's DSR identifier value is passed as an input to a switch case. Based on the identifier value, the switch case processes each packet differently. After completion of processing at this layer the control is passed onto the User Layer via another callback function existing in the networking layer. The User Layer thus provides the user with the necessary scope of developing application level interfaces. A similar passage of controls in reverse order is exercised when a user level interface intends to transmit a packet across the network. In this case, the control is passed from the User Layer to the Networking Layer and then to the XBee Interface Layer, which passes the packet to the Physical Layer (microcontroller resources, USART) which actually transmits the intended packet. The XBee Interface layer and Hardware Abstraction Layer function in close association with one another. The packet handling is illustrated in Fig.4.

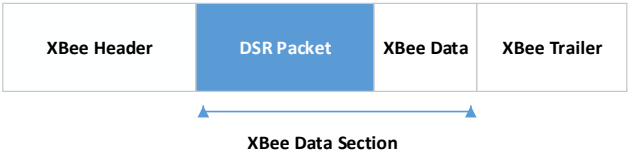

Fig. 2. XBee Packet Structure

b) DSR Packet Structure: The DSR packet structure consists of the Identifier, Initiator (source node),Target(destination node),Unique Id, Route pointer, Route length , Data length and Route Record , Data .

- *Identifier*: It consists of the packet type and is used to distinguish between various packets such as the Route discovery, Route Reply, Message Packet and Acknowledgement Packet. Each of these packet types is associated with distinct identifier.
- *Initiator*: This section specifies the transmitting node (source node) 16 bit network ID.
- *Target*: This section specifies the destination node (desired node) Network ID.
- *Unique ID*: Each Route discovery packet has a unique ID associated with it.
- *Expiration ID / Route Pointer*: This is a multifunctional section. During Route discovery it stores the hop count i.e. the number of hops traversed during the process of Route discovery. If the number of hops accumulated

exceeds the maximum hop count, then further route discovery is culminated and the route discovery packet is ignored. Thus the route discovery is considered to have expired. During the stage of Message *f*orwarding and Acknowledgement forwarding, this stage is used to point towards the next node/ *hop* position in the route record.

- *Route length and Data length:* These sections indicate the length of route and data section respectively.
- *Route Record*: The Route Record section is an array which holds either the route to the target node during messaging/ forwarding phase or the accumulated routes during route discovery phase.
- *Data*: This section stores the 64 bit address of the initiator node during the route discovery and route reply stage and has the data during the Message forwarding stage.

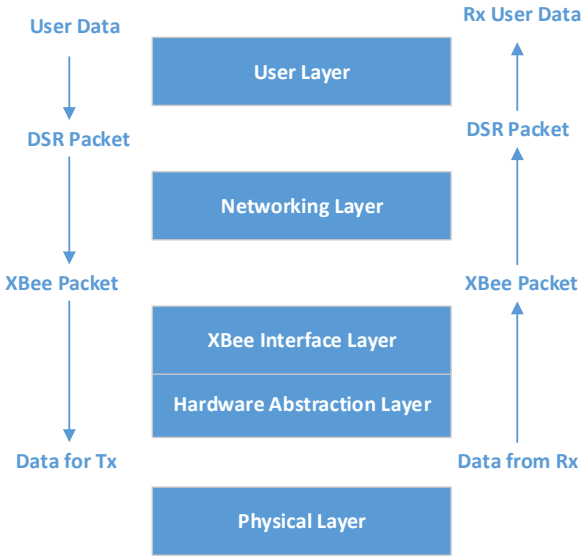

Fig. 3. Packet handling across layers

The DSRPacket structure is the most crucial part of this implementation. All routing decisions are based on the different values of the packet parameters. The DSRpacket is declared in software as follows,

```
struct{
        identifier;
        initiator;
        target;
        unique_id;
        expiration_period;
        route_record_length;
        dataLen;
        route_record[10];
        data[20];
       } DSRpacket;
```

The `createDSRpkt()` function is used to create a DSR packet and all the DSR packet elements are passed to it as arguments.

```
DSRpacket createDSRpkt
(Identifier, Initiator, Target, Unique_id,
Expiration_period, Route_record_length, DataLen,
* Route_record,* Data);
```

The `updated_TX_DSR_Packet()` is a generic function used to transmit various DSRpacket types to a intended destination node `netid_dest_node.`

```
 updated_TX_DSR_Packet
(DSRpacket r_packet1, netid_dest_node,
indentifier);
```

c) Route Cache: Each path node (module) maintains a route entry associated with the destination node. Each cached route entry consists of the target node, unique id, expiration time (time of caching), route length and route to the target node. This structure is illustrated as follows,

```
struct {

       dest_node[MAX_C_LEN];
       r_len[MAX_C_LEN];
       cached_route[MAX_C_HOP][ MAX_C_LEN];
       expir_id[MAX_C_LEN];
       uni_id[MAX_C_LEN];
} route_cache;
route_cache rc_ptr;
```

The maximum number of cached entries is user configurable. The variable `MAX_C_LEN` indicates the maximum number of route entries that can be cached and `MAX_C_HOP` indicates the maximum number of hops associated with a route entry. While caching routes, on board Timer0 assigns a current value to `expir_id`. Cached entries are searched for using the function

```
search_route_cache(desired_node);
```

This function returns the index of the cached route or reports its absence of route to the `desired_node`. The route cache is updated at a given location using the following function which uses a received `DSRpacket rec_pkt` and `location`as input

```
update_route_cache (rec_pkt,location);
```

A new entry is appended to the cache using the following function to which a `DSRpacket` is passed as an argument.

```
 app_seen_list_at_curr(rec_pkt);
```

d) Recent Request List (Seen List): The recent request list consists of the list of recently heard route request packets. This list consists of initiator and unique ID associated with the heard route request packet. This is illustrated by the structure

```
struct {
   initi_add[10];
```

```
reqID[10];
} recent_seen_reqs;
```

Whenever a new route request packet is heard, it is added to the recent request list and the packet is processed. If a duplicate request packet is heard, then it's discarded to curtail network traffic. It has a variable int `seen_list_len` associated with length of it .This variable is incremented every time a new (broadcast) request packet is heard. And the recent request is appended to the list by using the function.

```
app_seen_list_at_curr(rec_pkt);
```

The seen list is searched for a listed entry using the function

```
search_seen_list
      (initiator_node,uni_reqt_id);
```

This function is designed to return an integer value indicating either the presence or absence of the cached entry.

*e) Software Processes:*This section deals with the software processing and control flow of the various packet types. With the XBee S1 being interfaced through the UART all the packet types are received as serial interrupts. Depending upon the type of DSR packet received, a switch case based processing choice is made for a given packet.

- Route Discovery:

  The process of route discovery is the fundamental mechanism of DSR. Route discovery takes place on three levels. The first level is illustrated by Fig.5 Firstly; the initiator node initiates the process of Route discovery by appending the packets unique id and initiator to its seen list and broadcasting the message packet. Each discovery request generated had its own unique identification number. Upon broadcasting, the initiator initializes a timeout counter in order to wait for any possible replies. If a reply is received before the time out then the discovered route is extracted from the reply packet else route discovery function indicates failure of route discovery.

  The second phase of route discovery takes place at a non-target intermediate node as shown in Fig.6. Such a receiver node checks whether a request packet was heard previously. A request is heard if its entry is found in the seen list. A heard request is discarded and duplicate route formation is avoided. A packet is also discarded if the maximum numbers of hops have been reached. If request packet is not discarded it's appended to the seen list and any further re-transmissions of the request through this node are avoided and the receiver node's network id is appended to the received request's route record using the following function which returns a `DSRpacket` with updated route record and route record length.

  ```
  append_route_r
  (DSRpacket r_packet, net_id_update)
  ```

  The third phase of route discovery takes place at the target node as shown by Fig.6. A target node saves the traversed route to the data section and transmits (unicasts) a reply packet to its previous hop (node). The route length is decreased by one as the packet is sent to its previous hop.

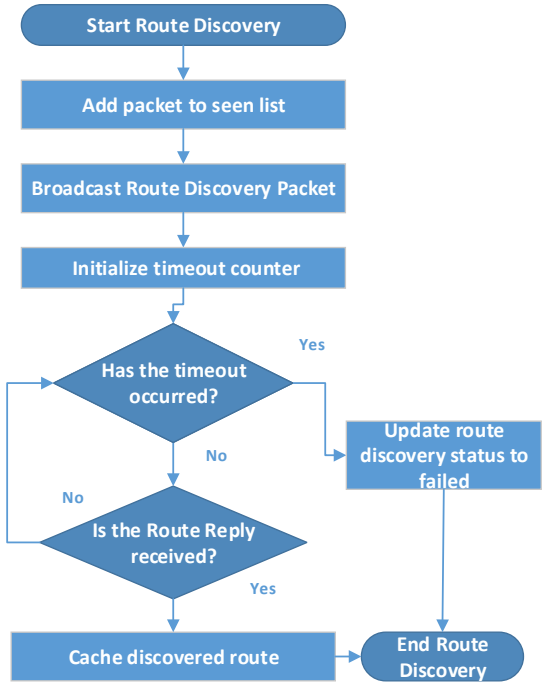

Fig. 4. Route Discovery at Initiator

- Route Reply:

  A reply packet is handled in two ways depending upon its reception at either the path node or initiator node. The process of handling the route reply is illustrated by Fig.7. A reply packet is unicasted from existing route record node (current path node) to the previous node (next node) listed on the route record till the route record length is zero and the next hop node is the initiator itself. The route record entries of the route reply are in the reverse order to that of the discovered route.

  If the receiver node is the initiator and if the time out hasn't occurred yet then the newly found route to the destination is either appended to the route cache or a previously expired route to the destination is updated. This step marks the completion of the Route Discovery phase.

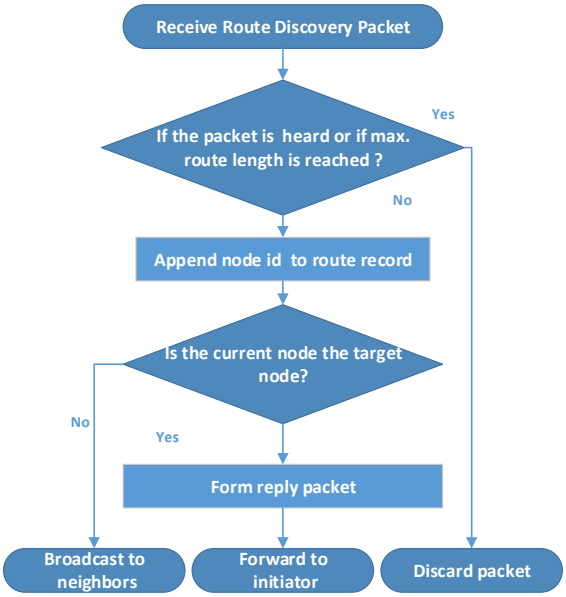

Fig. 5. Route Discovery at intermediate/ target node

- Message Forwarding:

In message forwarding the whole data section is used to store the message data as compared to the route reply packet where in it was used to store traversed nodes and the actual route record section was used for forwarding. Also for this packet type, the expiration id section of the DSRpackect acts as the route pointer and point to the next hop node on the route record. The process of message forwarding is illustrated in Fig.8.The discovered route is used for message forwarding..

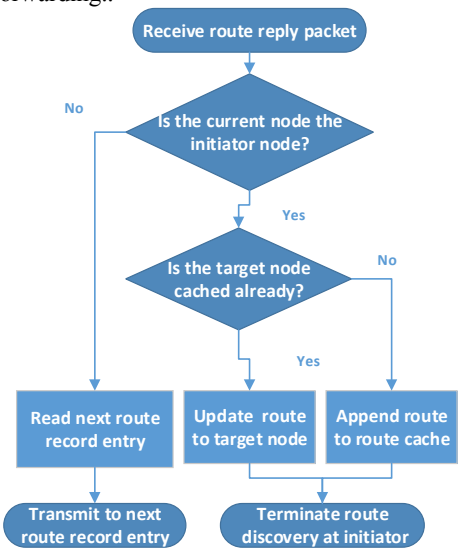

Fig. 6. Route Reply at intermediate/ initiator node

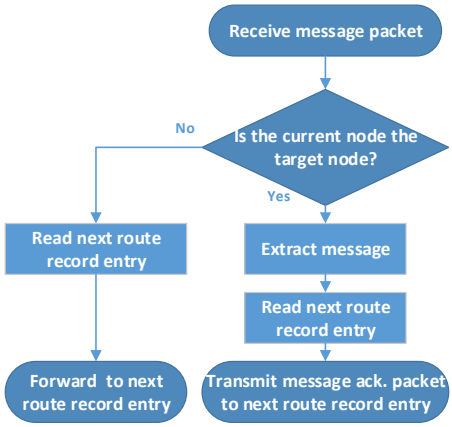

Fig. 7. Message packet process steps at intermediate/ target node

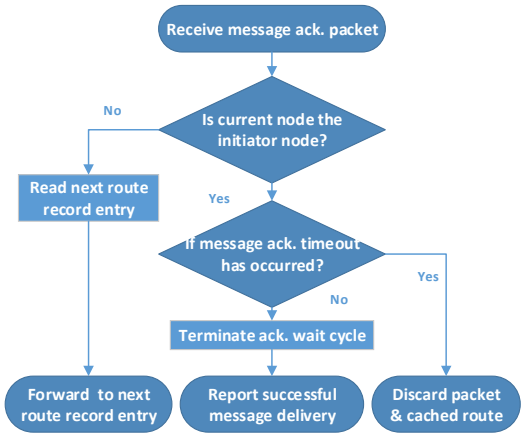

Fig. 8. Ack. message packet process steps at intermediate/ initiator node

- Acknowledgement Forwarding:

This packet type is responsible for route maintenance. The steps associated with forwarding the ack. packet are illustrated by Fig.9. The acknowledgement is handled in two ways. The path nodes continue decrementing the route pointer and forwarding the packet till it reaches the penultimate node with zero route pointer value and the next hop is the initiator itself. If the ack. packet reaches the initiator before the timeout then the wait cycle is halted and successful reception is reported by updating msg_status to value 1. If the message timeout counter times out before the reception of an ack. packet or if an ack. isn't received at all then a failure of reception / failure of existing route is reported by setting msg_status to value 3.

- Transmit DSR:

The process of transmitting a message to the destination node using DSR is managed by the Transmit_DSR function

```
Tranmsit_DSR
(dest_n, * msg, msg_siz );
```

In this function, the destination node, message/ data and message size are passed as arguments to the function. The

steps taken by Transmit DSR are illustrated in Fig.10. This interface relies on both the route discovery as well as route maintenance aspects of DSR.

Herein, the route cache is checked for any cached route to the destination. If a destination node is found then it's checked to verify whether the packet is stale or valid. If a cached route is found valid then it's used to for routing of the message packet. If a cached route is not found or if it's found to be stale then, route discovery is initiated and timeout counter is initialized. If route discovery is successful then the `discovered_route` is used for message routing. Once a message packet is transmitted towards the initiator, the function

`status_ack();`

is used to initialize a timeout counter and check the validity of the received acknowledgment. This function returns a msg_`status` depending upon the status of transmission. This status value is further returned as the return value for the `Tranmsit_DSR` function. If an acknowledgment is received, before the time out then successful transmission is notified to the user level using msg_`status` and if ack. is not received or received after the time out, a corresponding ack. failure is reported to the user. In case of route discovery failure, the failure status is reported to the user and the function is terminated.

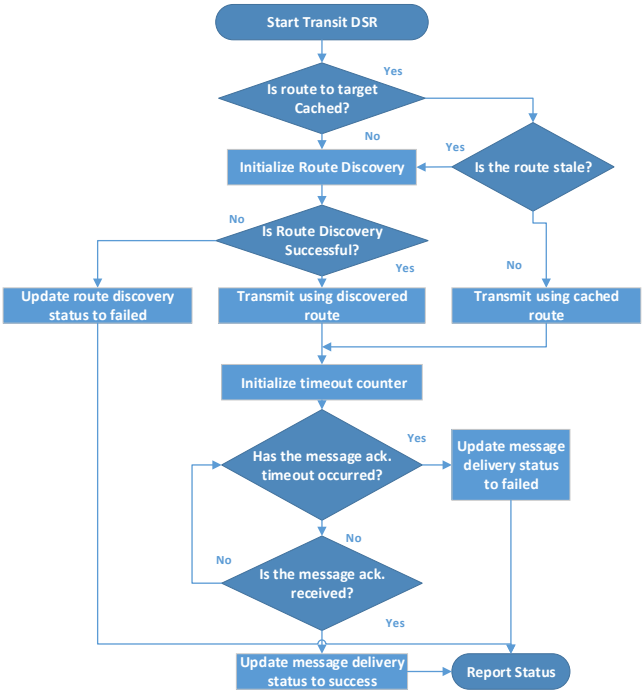

Fig. 9. Transmit_DSR steps

IV. IMPLEMENTATAION

a) Testbed creation: The XBee Series 1 modules The XBee Series 1 modules demonstrated receptions over 50 feet range in-spite of low power (Power Level 0) configuration. Due to this it was difficult to achieve isolation between modules that were two hops away. Thus, it was difficult to simulate a test environment using wireless transmissions and receptions. Hence there was a need for an attenuation based test-bed. Accordingly various tests were carried out to achieve isolation between nodes that were two hops using the variable 8495B /70 dB Attenuator. An attenuation value of 128 dB was found to provide the required isolation between nodes that were two hops away. This attenuation value was split equally between successive hops i.e. each hop was assigned with 64 dB of attenuation. Each XBee module was connected with a female SMA connector and then connected to a SMA male co-axial cable which was further connected to the attenuator. This test-bed block diagram is illustrated by Fig.11. The functioning of the each of the test-bed modules was monitored and tested using the CoolTerm serial termial application.The test-bed implementation is shown in Fig.12.

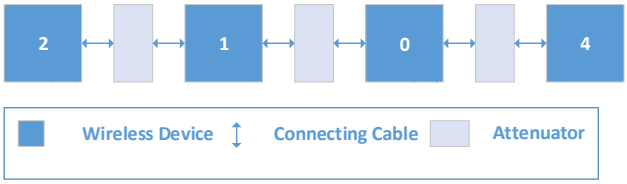

Fig. 10. *Test bed block diagram*

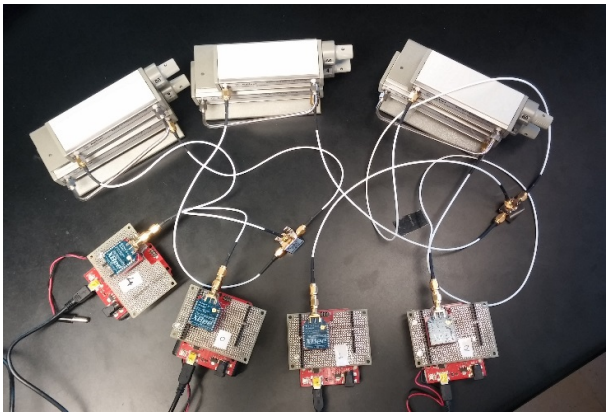

Fig. 11. *Test bed implementation*

b) Route Discovery Implementation and Testing:

After assembling a test-bed the whole software stack (Fig. 1) was deployed on 4 modules. These modules were preconfigured with the network ID's as per Fig. 12. For the implementation a test message 'MOTESQUITO' was passed as an argument to `Tranmsit_DSR()` function for target node 4 from source node 2. As the nodes were initialized, no cached routes were available hence a route discovery process (Fig.13 (a)) was initiated by node 2. Here it was ensured that none of the nodes had heard the request before the test case. The module with network ID 2 added the route request packet's initiator and unique ID to its heard list avoid further request loops. This discarding of duplicate packets is illustrated by Fig.13 (b) and (c). The route discovery continues and the route

record is appended with the path node's network ID during each successive hop, till the target node is found (Fig.13 (d)).

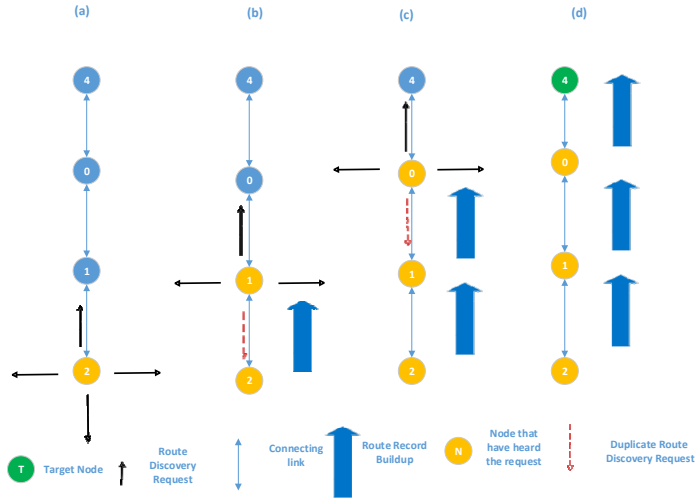

Fig. 12. *Route Discovery Implementation*

A reply packet was assembled at the target node and was forwarded through each of the path nodes on the accumulated route record till the initiator was reached. This process is illustrated in Fig.14 (a) through (d). In Fig. 13 and Fig.14 implementations it is assumed that Route Discovery and Route Reply mechanisms are completed before timeout occurs at the initiator node. The discovered route here is via 1, 0, and 4.

If a node connecting link was disconnected from the test-bed i.e. a node was isolated (Fig.15 (a) and (b)) or if an intermediate node had already heard a request packet (Fig.15 (c) and (d)), the route discovery function was halted and no reply packet was sent to the initiator. This resulted in the timeout of the route request and a route discovery failure was reported to the user by status of the LED indicator. Route Discovery failure also occurs and is reported, if the Route Reply isn't received before the timeout or if an intermediate link was lost, as illustrated by Fig.16 (a) and (c).

Also, if a path node is not functional or has moved out of range during the forwarding of the Route Reply packet, the reply packet is never received at the initiator and the timeout occurs. This leads to failure of route discovery.

c) Message and Acknowledgement Forwarding:

The message is forwarded along the discovered route 1, 0 and 4 and the target node responds with the acknowledgment along route 0, 1 and 2.This scenario is illustrated in Fig.16 and Fig.17. The timeout counter is initialized as the message is transmitted and halted either by reception of message acknowledgement or by timeout itself. If a message delivery acknowledgment is received before time out then the user is notified about the success of transmission by the return value of `Tranmsit_DSR()`. Based on this value, the indicator can be activated for a user defined period to report the status (Fig 17). This is the only way the transmission of message to the target node would be successful.

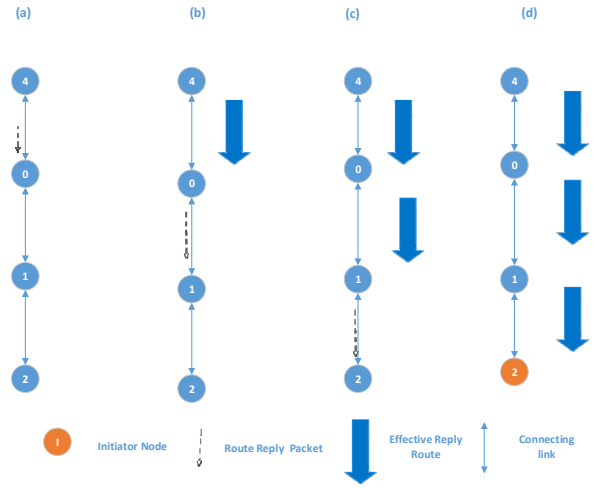

Fig. 13. *Route Reply Impementation*

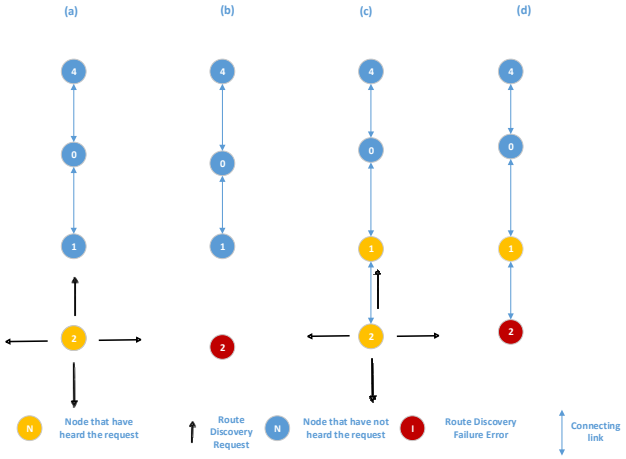

Fig. 14. *Route Discovery faliure during Route Request*

If an intermediate path node had moved out of range (connecting link missing) Fig. 18 (a), transmission failure is reported. If an acknowledgement arrives after the time out (Fig.18 (b)) or if a path node has moved out of range during the return of the acknowledgment packet (Fig.18 (c)), the ack. timeout counter times out and transmission failure is reported. The user is notified of this failure by the return value of `Tranmsit_DSR().` Based on this value the indicator status was made to report failure (Fig.18 (d)). The cases described in this section demonstrate the probable real-time scenarios that would occur. The possible event of a successful transmission can occur in only one way but numerous ways leading to possible failures were studied.

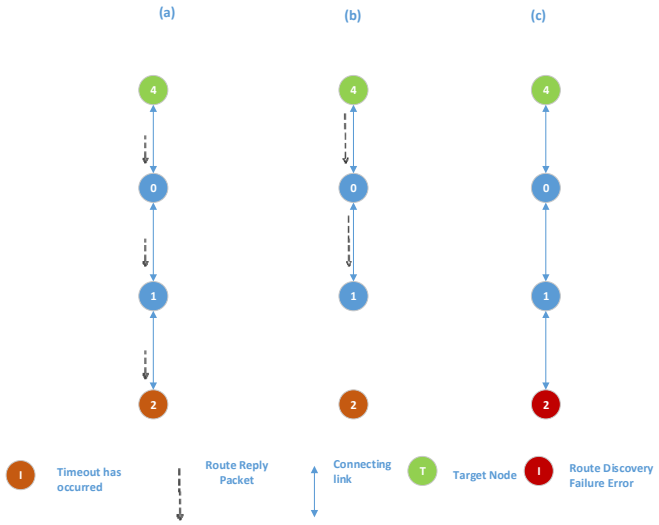

Fig. 15. *Route Discovery faliure during Route Reply*

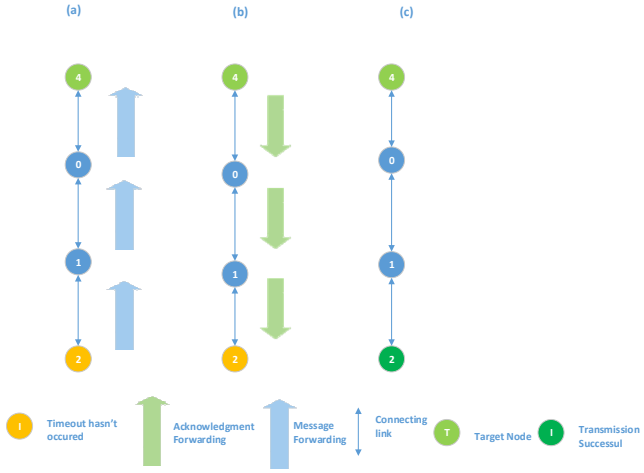

Fig. 16. Transmission Successful

*d) Self Healing:*The network layer does provide the network with self healing functionality. The test-bed under consideration consists of 4 modules but an additional node with network ID 3 was added as peer node of node 1. In the event of non availability path using node 1, the dynamic nature of the protocol select a route through node 3 instead of node 1, to reach target node 4.

V. Conclusion

The networking library developed during the course of this implementation does provide the user an interface to forward messages from one node to another using DSR. The functionality of this library is verified through physical deployment. The software stack as mentioned in Fig.1 does provide a huge scope for developing customized interfaces for the wireless modules used for this implementation. The XBee S1 (802.15.4) does prove to be an efficient choice for this implementation due its single hop functionality. The selection of ATmega 328P based RedBoard proved to be good choice for this implementation. However, a microcontroller with more memory would allow for larger route caches and more hops. The library is designed with a scope for scalability due to which more modules can easily be incorporated .Additional research and development is required to use the software stack on deployable wireless modules. Self-organizing feature is under current research.

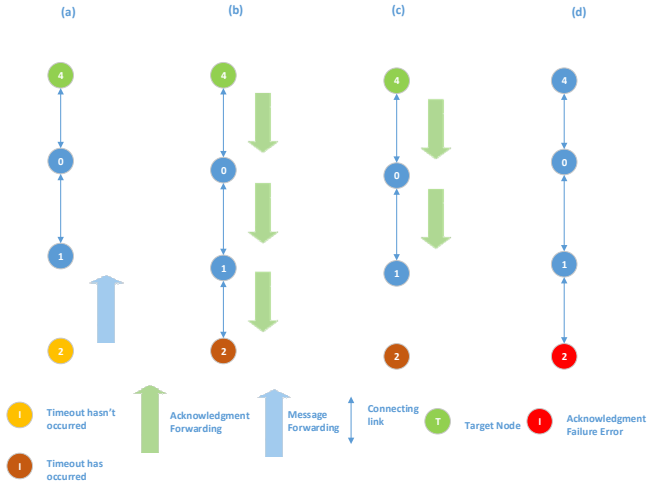

Fig. 17. Transmission Faliure

VI. References

[1] Poor, R. and Auburn, Charlotte B. and Bowman, C. (2003), Self-healing networks, QUEUE, May 2003, 52-59

[2] D. Johnson and D. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks", in Mobile Computing, edited by Tomasz Imielinski and Hank Korth, Chapter 5, pp. 153-181, Kluwer Academic Publishers, 1996.

[3] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, andJ. Jetcheva. A Performance Comparison of Multi-Hop Wireless Ad-Hoc Network Routing Protocols. In In Proceedings of the Fourth Annual ACM/IEEE Inter-national Conference on Mobile Computing and Net-working, Oct. 1998.

[4] XBee®/XBee-PRO® RF Modules, Product Manual v1.xEx - 802.15.4 Protocol.

[5] XBee™ ZNet 2.5/XBee-PRO™ ZNet 2.5 OEM RF Modules, Product Manual v1.x.4x - ZigBee Protocol

[6] Sultana Alimi, Samuel Shue and James M. Conrad,Design and Implementation of an Open-SourceWireless Sensor Network Development Platform, SOUTHEASTCON 2014, IEEE

[7] ATMEL, AVR microcontroller ATmega 328P datasheet, Oct. 2009 .

[8] Samuel L. Shue and James M. Conrad, Development of a portable XBee C library and RSSI triangulation localization framework, HONET 2014, IEEE

[9] Shue, S.; Conrad, J.M., "A survey of robotic applications in wireless sensor networks," Southeastcon, 2013 Proceedings of IEEE , vol., no., pp.1,5, 4-7 April 2013