Implementation and Testing of LOADng Protocol on Contiki OS

Team Member:
Haimo Bai
Jiahao Liang
Zhikun Liu

Contents

Implementation and Testing of LOADng Protocol on Contiki OS

Team members: Haimo Bai, Jiahao Liang, Zhikun Liu

Abstract—**In this project, our team propose is building, simulating and testing "LLN On-demand Ad hoc Distance-vector Routing Protocol – Next Generation" (LOADng) protocol, which is a router protocol, derived from AODV. The operating system we used is Contiki. The routing protocol we focused on is WSN protocol, which is different from other type of networks. We wrote and rebuilt two modules which are route module and the route-discovery module in order to make LOADng implementation. LOADng protocol which is based on the paper "Efficient Data Acquisition in Sensor Networks: Introducing (the) LOADng Collection Tree Protocol" [1].**

Index Terms—**Contiki, Cooja, LOADng, router, Algorithm, message, IPv6, address**

I. Introduction

LOADng stands for "LLN On-demand Ad hoc Distance-vector Routing Protocol – Next Generation"[5], which is an on-demand routing protocol for sensor networks whose objectives are to discover bi-directional paths [4]. There will be three different types of packets generated by each LOADng Router in this project which are RREQ (Router forwards Route Request), RREP (Route Reply), and RERR (Route Error). Compare to AODV (Ad hoc On-Demand Distance Vector), LOADng simplifies AODV in a number of ways for use in lossy, low-power and constrained environments [6]. LOADng also reduces the incurred of overhead by RREQ generation and flooding; reduces the address length: from full 16 octet IPv6 addresses to shorter 1 and 2 octet addresses; A LOADng router does not keep a list of precursor, which caused when forwarding of a data packet to the recoded next hop on the path to the destination fails, the RERR message will be sent only to the originator of that data packet [2]. On the other hand, control messages can include a set of type-Length-Value (TLV) elements, permitting flexible protocol extensions to be developed, and making possible to adapt the protocol to specific application needs [7].

LOADng is layer-agnostic, which means this protocol can be used at layer 2 as a mesh under routing protocol or at layer 3 as a route over protocol [8].

As a reactive protocol, routes are created only if there is path towards destination and there is data to be sent. If there is traffic using the path, the routes will be maintained [3]. When a LOADng router sends a data packet to a LOADng router for which it has no matching entry in its Routing Set, it begins the LOADng route discovery procedure. Then, a packet which called RREQ generated and flooded in the network. During this process, the nodes router receiving such RREQ install or update the routes towards RREQ originator if corresponds [9]. When the destination receives the RREQ, it will generate a RREP packet towards RREQ originator along the reverse route. When the originator which send the RREQ receives the correspondent RREP successfully, the route will be installed in the Routing Set.

Inside of the route discovery procedure, our project only allowed destination to respond to a RREQ in order to eliminates the need for Destination Sequence Numbers (DSN), so that the size of the message can be reduced compared to other protocols. In this way, no gratuitous RREP are not sent whilst loop freedom is retained [4]. When a route towards destination is formed, a LOADng router will only ensure the next hop to forward the packet towards the final destinations, but not maintain a precursor list. When forwarding a data packet to the next hop towards destination fails, RERR (error packet) from a route is sent only to the originator of that data packet [6].

For the address lengths part, we supported different length of address, which includes those of IPv6. The only constraint is that in a given network each device has a unique address, and that all addresses of the network are of the same length [4]. There is one or more interfaces in LOADng router, each of the interface configured with at least one network address.

Compare to other routing protocols such as RPL, LOADng supports per-path maintenance. In other words, path maintenance do not need global topology recalculation.

II. Definitions

A. Terminology

LOADng Router: A router implementing the LOADng routing protocol

Destination: The address of a router or host

Originator: The address of a router

Forward Route: An established route will send the data packets to the Destination which from the Originator [3]

Reverse Route: A route from the Destination to the Originator set up when a LOADng Router forwards RREQ messages [4]

B. *Parameters*

RREQ_RETRIES: is the maximum number of times a particular router can attempt to discover a route to a specific destination before declaring it unsuccessful

R_TIME: is the minimum time a route should be kept active after being created or refreshed

BLACKLIST_TIME: is the time during a LOADng router's neighbor should be kept in the blacklist after being added [4]

III. Structure

LOADng routing protocol uses the packets which described in the following subsections:

A. *LOADng Packet Format*

The packet format for all LOADng packets are using the same structure, which include the type field, the tlv block, and the message. (Table 1) The general format for all packets, generated, forwarded and processed by this specification, is as follows:

```
<packet> := <type>
            <tlv-block>
            <message>
```

Field	Size	Description
<type>	4 bit	Encodes the type of message in the <message> field
<tlv-block>	variable	Contains the required TLVs
<message>	Variable	Contains the RREQ, RREP, RERR message according to the <type> field

Table 1: LOADng packet format

1. TLV Block

The TLV Block contains zero or more Type-Length-Value elements (TLVs). A TLV allows the association of an arbitrary attribute with a packet. The attribute value is made up from an integer number of consecutive octets. Different attributes will have different types; the unknown attributes can be skipped when parsing [5]. (Details shows in table 2)

```
<tlv-block> := <tlv-count>
(<tlv-type><tlv-flags><tlv-value>)*
```

Field	Size	Description
<tlv-count>	4 bit	Unsigned integer field, specifying the number of TLVs included
<tlv-type>	4 bit	Unsigned integer field, specify the type of the TLV
<tlv-flags>	4 bit	Specify the interpretation of the remainder
<tlv-length>	8 bit	Unsigned integer field, specifying the length of the <tlv-value> field
<tlv-value>	tlv-length	A field of length <length> octets

Table2: TLV block

2. Message Format

There are three types included in our LOADng message which are RREQ, RREP, and RERR (Table 3)

LOADng message	<type>
RREQ	0
RREP	1
RERR	2

Table 3: Encoding for the <type> field

2.1 RREQ and RREP Message Format

RREQ Messages: the packets are generated by a router of LOADng, when it has data packets to deliver to a destination for which it has no matching entry in the Routing Set. The RREQ is transmitted to all reachable neighbor routers of LOADng, the packet can be transmitted by simply broadcast or other flooding techniques [8].

RREP Messages: means Route Reply Packets, which are generated by the destination node of a RREQ packet in response to such RREQ.

```
<message> := <flags>
             <addr_length>
             <cost_type>
             <weak_links>
             <rreq_ID>
             <route_cost>
             <destination>
             <originator>
```

Note: the format of RREQ and RREP message is identical.

Details:

- <flags> is a 4 bit unsigned integer field and specifies the interpretation of the remainder of the message.
- <addr_length> is a 4 bit unsigned integer field, encoding the length of <destination> and <originator>. [the length of an address in octets - 1] *Note: the actual length of address should be calculated by <addr_length> + 1*
- <cost_type> is a 4 bit unsigned integer field and specifies how the value of the <route-cost> field is calculated
- <weak_links> is a 4 bit unsigned integer field and specifies the total number of weak links on the routing path from the originator to the destination
- <rreq_ID> is an 8 bit unsigned integer field and specifies the cost of the routing path from the originator to the destination
- <destination> is an identifier with length equal to address_length, specifying the address of the destination, to which a route is sought
- <originator> is an identifier with length to address-length, specifying the address of the originator, which has initiated route discovery for the destination[5]

RREQ and RREP initial value shows in Table 4:

Field	Size/bits	Initial value
<flags>	4	Bit 0 is ACK-REQUIRED flag, set to 1 if RREP-ACK message is required. Bits 1 to 3 are RESERVED
<addr_length>	4	Length of the address – 1, in octets
<weak_links>	4	0
<rreq_ID>	8	Next LOADng router unused RREQ_ID
<destination>	addr_length + 1	LOADng router address, destination of the RREQ
<originator>	addr_length + 1	LOADng router address, generating the RREQ

Table 4: RREQ/RREP fields and initial values[4]

2.2 RERR Message Format

RERR Message: A Route Error packet is generated by a LOADng router after detected an error for route, the route repair mechanism is not attempted or not successful [4].

A route error occurs when a link has broken and it is failed to forward a data packet towards its destination. A RERR packet is then unicast to the source of the undelivered data packet.

In other words, if a link breakage is detected, an intermediate LOADng router MAY attempt to repair the route locally, by issuing a RREQ and awaiting an RREP. This RREQ is generated with the intermediate LOADng router's address, and with the same constraints on RREQ generation [7].

```
<message> := <error_code>
             <addr-length>
             <source>
             <destination>
```

Details:

- <error_code> is a 4 bit unsigned integer field and specifies the reason for the error message being generated
- <addr_length> is a 4 bit unsigned integer field, encoding the length of <destination> and <originator>. [the length of an address in octets - 1] *Note: the actual length of address should be calculated by <addr_length> + 1*
- <source> is an identifier with length equal to address-length, specifying the source address of a data packet, for which delivery to <destination> failed.
- <destination> is an identifier with length equal to address-length, specifying the address of the destination[5]

RERR Generation: if route repair is unsuccessful or not attempted, then a LOADng router will generate a RERR packet, and it will send this RERR along the reverse path to the source of the data packet for which delivery was unsuccessful [4].

Field	Size/bits	Description
<error_code>	4	Error code specifies the type of error
<addr_length>	4	The length of the address
<source>	addr_length + 1	The source address from the unsuccessfully delivered data packet
<destination>	addr_length + 1	The destination address from the unsuccessfully delivered data packet

B. *Information Base*

Information base sets which includes the Routing Set, Route Request Set and the Destination Address Set for each LOADng router maintains protocol state. These information sets are used to facilitate description of message generation, forwarding and processing rules.

1. Path Set

This set will record the next hop along a path to each destination, for which such a path is known. Routing set stores the routing information for all reachable nodes [6]. It contains: R_dest_addr, R_next_addr, R_dist, R_valid_time.

Where:

- R_dest_addr: the address of the destination.
- R_next_addr: the address of the "next hop" on the selected path to the destination
- R_dist: the distance of the selected path to the destination with address R_dest_addr
- R_valid_time: the time until the information contained in that tuple is considered to be valid

2. Route Request Set

A LOADng router's Route Request Set records information about issued route discovery processes. It contains: D_SRC_ADDR, D_RREQ_ID.

Where:

- D_SRC_ADDR: the address of the LOADng router, it will generating a RREQ in order to initiate the route discovery process
- D_RREQ_ID: a sequence number, uniquely identifying each RREQ issued by a given LOADng router

3. Destination Address Set

This set will record the destination address for a LOADng router, which generates RREPs in response to received RREQ messages.

C. *Route Discover*

For AODV, during Route Discovery, RREQ messages are flooded through the network, indicating the address to which a route is sought. Upon receiving an RREQ, the router which will take care of the address listed in the RREQ will respond with an REP, the router which will take care of the address listed in the RREQ will respond with an RREP, sent to the source of the RREQ in unicast [7].

As a reactive light-weight derivative of AODV, LOADng targets routing in low power and constrained environment [2]. For example: in figure 1, Router S initiates a route discovery to Router D, while A and B have already available routes to D. On receiving the broadcast RREQ from S, A identifies that it already has a valid route to D, and so proceeds to unicast, rather than broadcast, the RREQ via the recorded path to D [7].

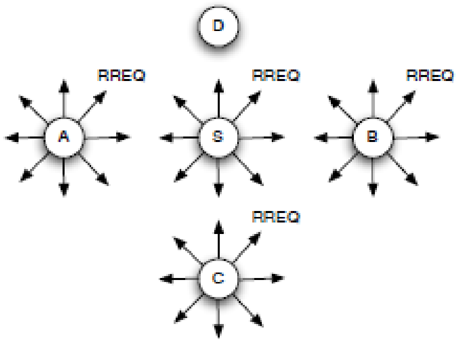


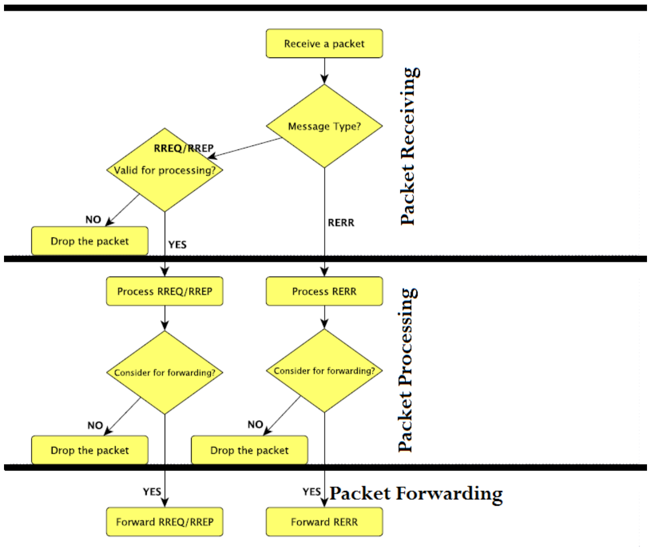
Figure 1: Route Discovery. S initiates an RREQ for D.A,B and C already know routes to D.[1]

Thus, compared to AODV, LOADng prohibits intermediate RREPs, in order to reduce the size of control messages [5]. Besides, LOADng protocol reduce the state and processing required in routers in the network.

IV. ALGORITHM

The main algorithm of LOADng protocol is: one certain LOADng router can obtain routes by sending RREQ packets, and Routing table will be modified during receiving packets. Figure 2 shows the LOADng algorithm flowchart which includes three phases: packet receiving, packet processing and packet forwarding.

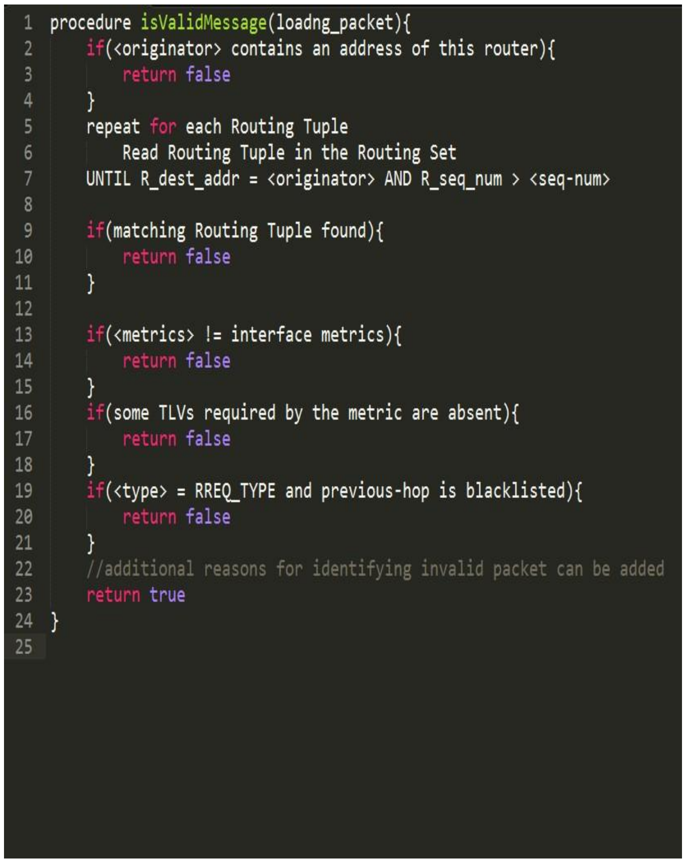
Figure 2: LOADng Algorithm Flowchart [6]



A. RREQ and RREP messages

RREQ and RREP messages are identical in structure and have the processing steps similarly.

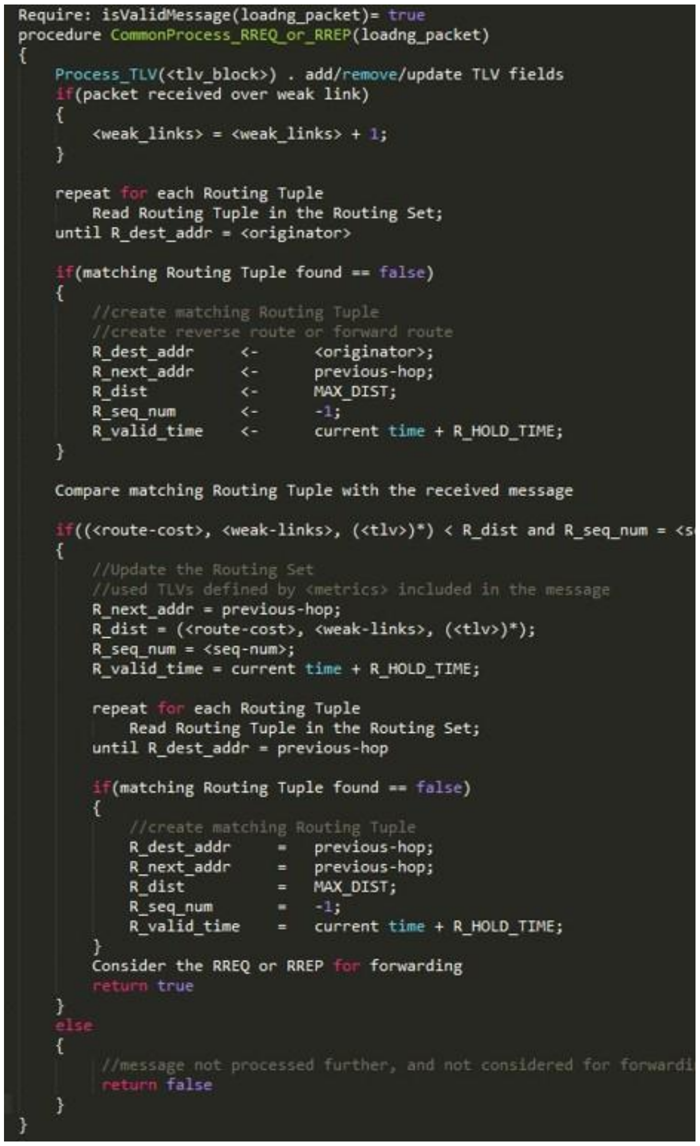
1. Identifying valid RREQ and RREP messages
   When the LOADng router received a message (RREQ/RREP), it will firstly check if the message is invalid for processing is shown in figure 3



```
procedure isValidMessage(loadng_packet){
    if(<originator> contains an address of this router){
        return false
    }
    repeat for each Routing Tuple
        Read Routing Tuple in the Routing Set
    UNTIL R_dest_addr = <originator> AND R_seq_num > <seq-num>

    if(matching Routing Tuple found){
        return false
    }

    if(<metrics> != interface metrics){
        return false
    }
    if(some TLVs required by the metric are absent){
        return false
    }
    if(<type> = RREQ_TYPE and previous-hop is blacklisted){
        return false
    }
    //additional reasons for identifying invalid packet can be added
    return true
}
```

Figure 3 Check the message is valid or not

2. RREQ and RREP Message Processing
   The common processing algorithm for RREQ and RREP packets valid for processing is shown in figure 4

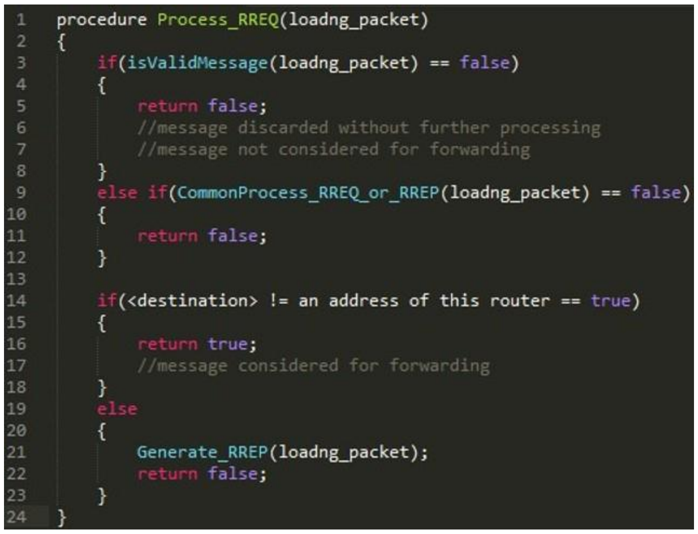


```
Require: isValidMessage(loadng_packet)= true
procedure CommonProcess_RREQ_or_RREP(loadng_packet)
{
    Process_TLV(<tlv_block>) . add/remove/update TLV fields
    if(packet received over weak link)
    {
        <weak_links> = <weak_links> + 1;
    }

    repeat for each Routing Tuple
        Read Routing Tuple in the Routing Set;
    until R_dest_addr = <originator>

    if(matching Routing Tuple found == false)
    {
        //create matching Routing Tuple
        //create reverse route or forward route
        R_dest_addr     <-      <originator>;
        R_next_addr     <-      previous-hop;
        R_dist          <-      MAX_DIST;
        R_seq_num       <-      -1;
        R_valid_time    <-      current time + R_HOLD_TIME;
    }

    Compare matching Routing Tuple with the received message

    if((<route-cost>, <weak-links>, (<tlv>)*) < R_dist and R_seq_num = <s
    {
        //Update the Routing Set
        //used TLVs defined by <metrics> included in the message
        R_next_addr = previous-hop;
        R_dist = (<route-cost>, <weak-links>, (<tlv>)*);
        R_seq_num = <seq-num>;
        R_valid_time = current time + R_HOLD_TIME;

        repeat for each Routing Tuple
            Read Routing Tuple in the Routing Set;
        until R_dest_addr = previous-hop

        if(matching Routing Tuple found == false)
        {
            //create matching Routing Tuple
            R_dest_addr     =   previous-hop;
            R_next_addr     =   previous-hop;
            R_dist          =   MAX_DIST;
            R_seq_num       =   -1;
            R_valid_time    =   current time + R_HOLD_TIME;
        }
        Consider the RREQ or RREP for forwarding
        return true
    }
    else
    {
         //message not processed further, and not considered for forwardi
         return false
    }
}
```

Figure 4 Common Processing Algorithm

3. RREQ Message Processing

When a LOADng router is receiving a RREQ message, it will process the message, according to Figure 5

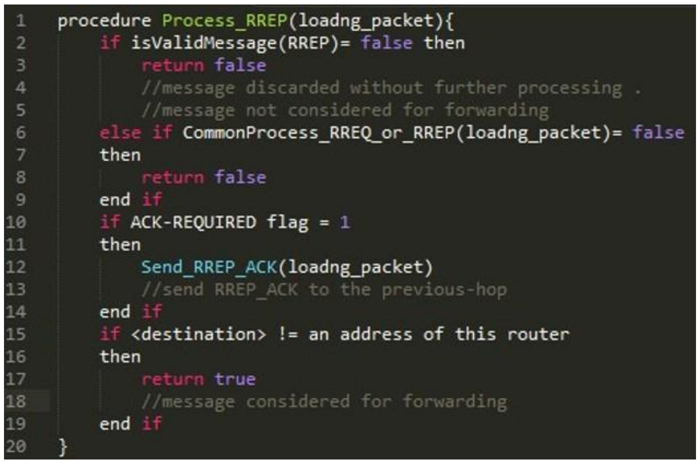


```
procedure Process_RREQ(loadng_packet)
{
    if(isValidMessage(loadng_packet) == false)
    {
        return false;
        //message discarded without further processing
        //message not considered for forwarding
    }
    else if(CommonProcess_RREQ_or_RREP(loadng_packet) == false)
    {
        return false;
    }

    if(<destination> != an address of this router == true)
    {
        return true;
        //message considered for forwarding
    }
    else
    {
        Generate_RREP(loadng_packet);
        return false;
    }
}
```

Figure 5 RREQ Message Processing

4. RREP Message Processing

When a LOADng router is receiving a RREP message, it will process the message, according to Figure 6

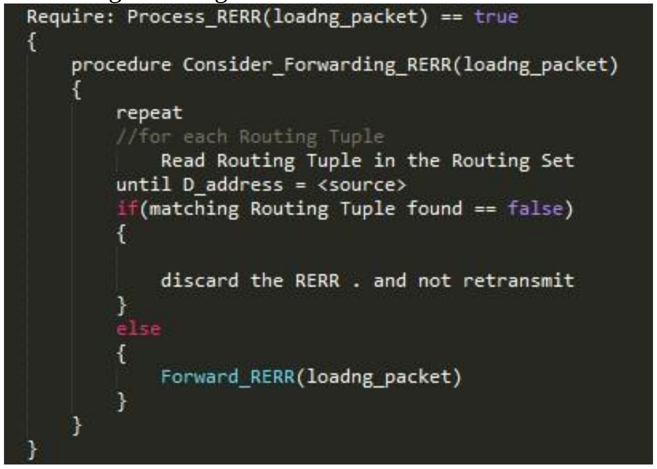


```
procedure Process_RREP(loadng_packet){
    if isValidMessage(RREP)= false then
        return false
        //message discarded without further processing .
        //message not considered for forwarding
    else if CommonProcess_RREQ_or_RREP(loadng_packet)= false
    then
        return false
    end if
    if ACK-REQUIRED flag = 1
    then
        Send_RREP_ACK(loadng_packet)
        //send RREP_ACK to the previous-hop
    end if
    if <destination> != an address of this router
    then
        return true
        //message considered for forwarding
    end if
}
```

Figure 6 RREP Message Processing

5. RERR Message Processing

When a LOADng router is receiving a RERR, it will update its Routing Set as figure 7 shows



```
Require: Process_RERR(loadng_packet) == true
{
    procedure Consider_Forwarding_RERR(loadng_packet)
    {
        repeat
        //for each Routing Tuple
            Read Routing Tuple in the Routing Set
        until D_address = <source>
        if(matching Routing Tuple found == false)
        {

            discard the RERR . and not retransmit
        }
        else
        {
            Forward_RERR(loadng_packet)
        }
    }
}
```

Figure 7 RERR Message Processing

B. LOADng Packet Forwarding

After message processing, and if the processing function returns TRUE value, which means it still continues being processed, then the message will be considered to be forwarded.

1. RREQ Forwarding

RREQ message is considered for forwarding, which will be updated as follows:

```
Require: Process RREQ(packet) == true
{
    consider_forwarding_rreq(packet)
    //update <route_cost>
    Updata_Route_Cost(interface,
<metrics>)
    //UpdateRouteCost updates <route-cost>
field according to the cost associated
with the interface over which the RREQ is
transmitted, and according to the
specification of the <metrics> included in
the RREQ
      Forward_RREQ(packet)
}
```

2. RREP Forwarding

RREP message is considered for forwarding, which will be updated as follows:

```
Require: Process RREP(packet) == true
{                   consider_forwarding_r
rep(packet)
    //update <route_cost>
    Updata_Route_Cost(interface,
<metrics>)
    //Update_Route_Cost updates <route-cost>
field according to the cost associated with
the interface over which the RREQ is
transmitted, and according to the
specification of the <metrics> included in
the RREP
      Forward_RREP(packet)
//this function unicasts the RREP message to
the next hop towards the destination
indicated in the RREP
}
```

3. RERR Forwarding

A RERR is destined for the LOADng router ultimately, which has the address from the <source> field, in its Destination Address Set. A RERR message is considered for forwarding as follows:

```
Require: Process RERR(packet) == true
{
    consider_forwarding_rerr(packet)
    //update <route_cost>
    Repeat
        Read routing tuple in the
routing set
    Until D_address = <source>
    If no matching Routing Tuple found
    Then
        Discard the RERR
    Else
        Forward_rerr(packet)
    //the function Forward_rerr unicasts the
RERR message to the next hop towards the
<source> indicated in the RERR
    End if
}
```

V. LOADng Implementation

A. Contiki OS

The LOADng routing protocol of this project will be implemented to run as a routing protocol in Contiki OS.

Contiki OS which is an open source operating system written in C for network system with a particular focus on low-power wireless Internet of Things (IoT) devices [10].

Besides, Contiki OS which is also networked embedded systems, whose propose is helping designing for networking applications.[5] Contiki can support standard OS features such as timers, clock, threads, and a file system. It includes an IPv4/IPv6 stack, TCP and UDP support [2].

Note: for routing part in Contiki OS, which will use AODV as a default routing protocol.

B. Simulation and Testing

1. Simulation Setup

We use simulation supplication Cooja, which comes with Contiki, to test and simulate our implementation.

We simulates with random topologies with 5, 15, 25 nodes respectively. Each node except node 1 intends to send a packet to node 1 at various time. We count the time it needs to discover an available route.

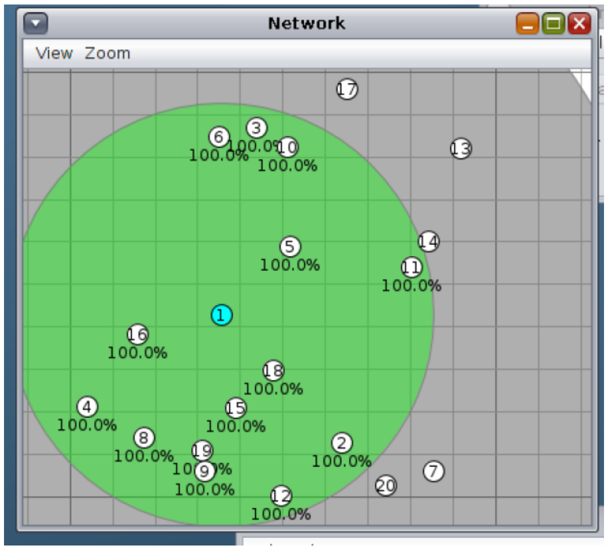


Figure 8 Sample topology with 25 nodes

2. Simulation Result

# of Nodes	Avg. discover time (s)	# of samples
5	0.36	100
15	4.37	100

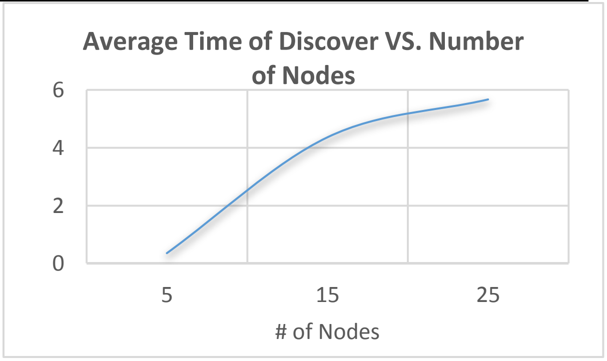


Figure 9 Simulation Result

25	5.68	100

The result from Figure 9 shows that the time it need for LOADng to discover a route highly depends on the complexity of the topology. The dramatic decrease of discover time for 5-node shows, LOADng runs efficiently if the distance between originator and destination is low (i.e 1 or 2 hops). Also, it will take lower than 0.5 sec to finish the discover process which shows that LOADng is suitable the network with mobility. As number of nodes increases, too many packets flooding in the network degrades the performance of LOAGng.

Consequently, we recommend deploying LOADng as the routing protocol in a sparse network with high mobility.

VI. Conclusion

In this project, LOADng has been implemented for use in Contiki OS successfully. Besides, this project tested LOADng and we are able to discover routes with high and success rate on discovering optimal paths. Compare to other protocol such as AODV, a LOADng router does not maintain a precursor list, it also optimized flooding. In one words, LOADng is a more "mobility" protocol.

VII. Acknowledgement

We would like to thank Professor Bhaskar Krishnamachari for his expert advice and encouragement throughout this difficult project, and gave us many helpful instructions and suggestions for our project during this semester so that we can finally finish the term project.

VIII. Contributors

This specification is the result of the joint efforts of the following contributors – listed alphabetically.

- Haimo Bai, <haimobai@usc.edu>
- Jiahao Liang, <jiahaol@usc.edu>
- Zhikun Liu, <zhikunli@usc.edu>

IX. References

[1] Bas, A , "Expanding Ring Search for Route Discovery in LOADng Routing Protocol Protocol - Next Generation (LOADng)". RetrievedSeptember , 2012 Available

[2] "Contiki Operating System". Retrieved September, 2012 Available

[3] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. Experimental PFC 3561, Jul. 2003

[4] I.F Akyildiz, Weilian Su, Y. Sankarasubramaniam, and E, Cayirci, A servey on sensor networks. *Communications Magazinc, IEEE*, 40(8):102-114, aug 2002.

[5] Martinez, A ,Implementation and Testing of LOADng: a Routing Protocol for WSN. RetrievedFebruary , 2013 Available.

[6] T. Clausen, A. C. de Verdiere, J. Yi, A. Niktash, Y. Igarashi, H. Satoh, and U. Herberg, "The lln on-demand ad hoc distance-vector routing protocol - next generation," The Internet Engineering Task Force, April 2012, Internet Draft, work in progress, draftclausen-lln-loadng.

[7] T. Clausen, A. C. de Verdiere, J. Yi, A. Niktash, Y. Igarashi, H. Satoh, and U. Herberg, "Efficient Data Acquisition in Sensor Networks: Introducing (the) LOADng Collection Tree Protocol" The Internet Engineering Task Force, April 2012, Internet Draft, work in progress, draftclausen-lln-loadng

[8] T. Clausen and Ulrich Herberg. A Comparative Performance Study of the Routing Protocols LOAD and RPL with Bi-Directional Traffic in Low-power and Lossy Networks (LLN). INRIA, June 2011.

[9] T. Clausen and Ulrich Herberg. Study of Multipoint-to-Point and Broadcast Traffic Performance in RPL. INRIA, April 2011.

[10] *"The ESB Embedded Sensor Board,"* http://www.sics.se/adam/contiki