International Journal of Computer Applications (0975 – 8887)
Volume 37– No.2, January 2012

Formal Specifications of Trusted OLSR Protocol of Ad hoc Network in Z

Amandeep Verma
Punjabi University Regional Centre for IT & Mgmt., Mohali

Manpreet Singh Gujral
University College of Engineering, Punjabi University, Patiala

ABSTRACT

A Mobile Ad hoc Network is a network of mobile nodes operating in an infrastructure-less network. These nodes not have the defense rendered by firewalls in infrastructure-based networks. Trust oriented system aids to improvise this situation. So, the incorporation of trust in routing decisions yields a more secure and reliable framework for such type of networks. As if any suggested model does not achieve as projected, it is reduced quality of service. The validation before deployment of any model leads to a more stable model. A good way to confirm a model is to use formal specification and verification techniques. In the present study, with the aim to include trust component in conventional OLSR protocol of ad hoc network and also to rule out invalid actions, formal specifications of the various procedures of trusted OLSR Protocol are given using "Z" specification language. Z is a state-oriented formal specification language based on set theory and predicate logic. Z/EVES, a proof tool based on EVES and ZF set theory that supports the Z notation is used for the formal specifications.

Keywords

Ad hoc Network; Formal Specification; OLSR; Z Specification Language; Z Eves

1. INTRODUCTION

Nodes in an ad-hoc network, an infrastructure less network, are more prone to attacks as there is no line of defense offered by firewalls, which is available to the counterpart, infrastructure-based networks. The conventional cryptographic approaches do not prove an effectual measure to defend against threats from internal compromised or malicious nodes. The operating environment of ad-hoc networks depends exclusively on the collaboration among nodes to provide connectivity and communication routes. This naive dependency on intermediate nodes makes the ad-hoc network vulnerable to passive and active attacks by malicious nodes. To alleviate the effect of malicious nodes and to achieve higher levels of security and even reliability, it is worthwhile to appraise the trustworthiness of other nodes in network before any operation or interaction. In the dominion of network security, the perception of trust is as a relation among entities that are involved in protocols.

Trust and reputation have been suggested in the literature as a thriving security means for open environments such as the Internet, and substantial research has been made on modeling and administering trust and reputation. There is a widespread postulation in the routing protocols that all nodes are trustworthy and cooperative. Though, the actuality is unlike. Malicious nodes can make use of this assumption to corrupt the network. A bundle of attacks such as black hole, gray hole, flooding attack, Denial of Services may annihilate the network. Even though trust is extensively researched nowadays, even there is not a accord and systematic theory based on trust but it has affirmative effects on the security solutions for infrastructure-less networks. There might be some additional operating cost in terms of time and complexity by the inclusion of trustworthiness but it is marginal and that too at the cost of more security so this overhead can be disregarded.

The reliability of critical systems is a grave concern. Formal methods have demonstrated their appeal in extending the reliability of such systems in this regard. All of these are based on mathematical representation and analysis of systems. The prime advantages [5] of applying formal methods in designing a system are in the lessening of the figure of errors in systems. More efforts have to spend in the initial phases of software development in the case of formal specifications. This lessens requirement errors, as it causes a comprehensive analysis of the requirements. Incompleteness and inconsistencies can be revealed and resolved. Hence, the amount of rework due to requirements problems is reduced. Formal specification is an element of a entire compilation of practices that are known as 'formal methods'.

Keeping in mind all these concerns, in an attempt to include trust in the functioning of routing of OLSR protocol of ad hoc networks, formal specifications of the procedures of trusted OLSR protocol are presented in this paper. The novelty of the work is to give the abstract representation of the procedures with trust attribute base that are going to be used in making routing decisions and as well other decisions. This work is an integral part of the study to develop a trust oriented security framework for ad hoc networks. The intended readers are researchers interested in building abstract models and to use trust for security purposes.

The paper is organized as follows. The section 2 is in relation to the review of literature. The section 3 gives a general idea of the protocol under study and the specification language used. The section 4 lists down the specification(s). The last section 5 concludes the paper.

2. REVIEW OF LITERATURE

There are numerous routing protocols for ad hoc networks are in literature, and every protocol has some worth on one aspect or the other. With the aim of to decide on a routing protocol for building a trust oriented framework for ad hoc networks, a study comprising the performance analysis of routing protocols [13] namely – DSR, AODV, OLSR and GRP has been done for common applications: email and ftp in a simulated environment. The observation is that, the OLSR protocol has the better response over others.

A highly secure, save node's power and even the time for communication is less was devised [6], that use trust which is based upon a node has on its neighbor, different trust level defined and security is applied accordingly A study [1] demonstrates the involvement of trust in making routing decisions results higher output, less amount of malicious drops and higher packet delivery ratio. A comprehensive review of the impact of trust in ad hoc network was presented [11]. The literature supports and emphasizes the need of inclusion of trust in ad hoc environments

There are number of formal approaches and their applications in diverse areas for validation and for proving correctness the models in study. The formal verification techniques applicable to all areas of ad hoc network namely, authentication, access control, routing etc. An exhaustive review of the formal verification of adhoc network routing protocols [12] using the variety of formal specification languages, modeling techniques and verifying tools are quoted in early work. AVISPA, BAN Logic, Petri nets, SDL, SPIN, PROMELA and UPPAAL are some of the keywords of this study. The usage of SPIN, as model checker and PROMELA, as specification language found extensively, in studies related to ad hoc network.

To demonstrate the methodology for formal verification of routing protocols of ad hoc network [2], a case study of OLSR protocol using PROMELA specification language and SPIN model checker is depicted. The OLSR protocol is also a subject of case study in the description of testing methodology for an ad hoc routing protocol [9].

The Z notation is used as a formal technique for formal Verification of Route Request procedure for AODV Protocol. The formal specification is analyzed and validated using Z Eves tool. Z specification language is used to describe the component in the proposal of conceptual component model [4]. The Z language seems more abstract, reasoned to choice of specification language for the present study.

3. PRELIMINARIES

The protocol under study is OLSR and the specification language used for formal specification is the "Z" language. Outlines of these are as follows—

3.1 OLSR Protocol

Optimized Link State Routing (OLSR) [3] is proposed by IETF's MANET Group at 2003. Although, OLSR has newer and more precise narratives, with the purpose of to congregate our intents, agreed to its minimalism, we use the original version [3]. The Optimized Link State Routing Protocol (OLSR) is developed for mobile ad hoc networks.

It operates as a table driven, proactive protocol, i.e., exchanges topology information with other nodes of the network regularly. Each node selects a set of its neighbor nodes as "multipoint relays" (MPR). In OLSR, only nodes, selected as such MPRs are responsible for forwarding control traffic, intended for diffusion into the entire network. MPRs provide an efficient mechanism for flooding control traffic by reducing the number of transmissions required. A node selects MPRs from among its one hop neighbors with "symmetric", i.e., bi-directional, linkages. The MPR nodes are chosen among the 1-hop neighbors in such a way that they are the minimum set that covers all the 2-hop neighbors In OLSR, each node is injecting topological information into the network through the transmission of HELLO messages and, for some nodes, TC messages. If some nodes (malicious or malfunctioning) inject invalid network traffic, network integrity may be compromised.

3.2 The Z notation

The Z (pronounced Zed) language [8] [14] is a formal specification language named after Zermelo–Fraenkel set theory, is a formal specification language used for describing and modeling computing systems. "Z" language is a formal specification language, which is used for description and functions modeling of computer systems. This language enables to write formal specification of computer programs and to formulate evidences of system behavior.

Z specification language is based on the standard mathematical notation used in axiomatic set theory, lambda calculus, and first-order predicate logic. All expressions in Z notation are typed, thereby avoiding some of the paradoxes of naive set theory. Z has a number of language constructs including given type, abbreviation type, axiomatic definition, state and operation schema definitions, etc. but the basic unit of the Z specification for the components is a boxed notation called 'schema'.

3.3 Z/EVES

Z/EVES [10][15] use state-of-the-art formal methods techniques from Europe and North America, integrating a leading specification notation with a leading automated deduction capability. Z/EVES support almost the entire Z notation; only the unique existence quantifier for schema expressions is not yet supported. The Z/EVES prover provides powerful automated support with user commands for directing the prover. Z/EVES consist of two parts. First part is virtual server, which insures syntactic propriety revision and activities for execution of theorems and paragraphs logic validation. Second part is graphical interface, which enables work with specification.

4. THE SPECIFICATION(S)

In order to develop a trust oriented framework for ad hoc network using OLSR routing protocol the approach that we are using is depicted in the Figure 1. This paper is in the direction of building an abstract model of the protocol under study.

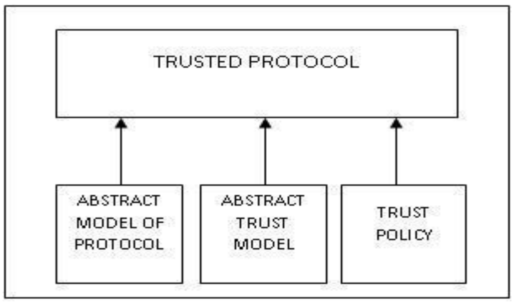

Figure 1: A General Trust Oriented Framework

4.1 Terminology

The terminology used in the specification of OLSR protocol [3] is as –

main_address	The main address of a node, which will be used in OLSR control traffic as the "originator address" of all messages emitted
neighbor node	A node X is a neighbor node of node Y if

	node Y can hear node X
2-hop neighbor	A node heard by a neighbor.
multipoint relay (MPR)	A node which is selected by its 1-hop neighbor, node X, to "re-transmit" all the broadcast messages that it receives from X, provided that the message is not a duplicate, and that the time to live field of the message is greater than one.
multipoint relay selector	A node which has selected its 1-hop neighbor, node X, as its multipoint relay, will be called a multipoint relay selector of node X

The information repositories record information about neighbors, 2-hop neighbors, MPRs and MPR selectors [3].

Neighbor Set	A node records a set of "neighbor tuples" (N_neighbor_main_addr, N_willingness), describing neighbors
2-hop Neighbor Set	A node records a set of "2-hop tuples" (N_neighbor_main_addr, N_2hop_addr, N_time), describing symmetric links between its neighbors and the symmetric 2-hop neighborhood.
MPR Set	A node maintains a set of neighbors which are selected as MPR. Their main addresses are listed in the MPR Set.
MPR Selector Set	A node records a set of MPR-selector tuples (MS_main_addr, MS_time), describing the neighbors which have selected this node as a MPR.

4.2 Assumptions

In order to simplify the implementation two assumptions regarding the OLSR protocols have been made.

- Every node has a single interface
- If the connection exists between any two node then it is of symmetric type

4.3 Formal Specifications

The identifier of an object in ad hoc network is denoted by *Node* as given below. The power of battery is dead, low or normal denoted by *Dead, Low and Normal* respectively [7].

[*Node*]
Power :: *Dead* *Low* *Normal* *Safe* *Full*

The Trust as in Figure 2 can be any value from 0 to 10 where 0 indicates complete distrust and 10 signifies the full trust in the node and the willingness of node whose specifications are given in Figure 3, depicts the willingness of node to participate in the various decisions or operations of the ad hoc network. The value 1 is the lower level and 7 is the upper level of willingness.

Trust
N_trust:

N_trust 10 *N_trust* 1

Figure 2: Trust Specifications

Willingness
N_willingness:

N_willingness 7 *N_willingness* 1

Figure 2: Willingness Specifications

The Neighbor is specified as a node and the Neighbor_tuple that forms an entry for the neighborset of the object has the neighbor, its willingness to participate and trust of the host on that neighbor as shown in Figure 4.

Neighbor
N_neighbor_main_addr: Node

Neighbor_tuple
N_neighbor_tuple: Neighbor *Willingness* *Trust*

Figure 4: Neighbor Set Specifications

The TwoHopNeighbor is specified as a node and the Twohop_tuple that forms an entry for the twohopneighborset of the object has the neighbor, twohopneighbor , time till its live, and the rust of the host on that twohop neighbor as shown in Figure 5.

TwoHopNeighbor
N_2hop_addr: Node

N_time :: *Live* *Expired*
Twohop_tuple
N_twohop_tuple: Neighbor *TwoHopNeighbor*
N_time *Trust*

Figure 5: TwoHop Neighborset Specifications

The specification of the MPRSet and MPRSelectorSet is in Figure 6. It gives the address of the node which is MPR and the trust of the object on that MPR. In MPRSelectorSet the address of the node which selects host as MPR and validity time is stored.

MPRSet
N_mpr_tuple: Node *Trust*

MPRSelectorSet
MS_main_addr: Node
MS_time:

Figure 6: MPRSet and MPRSelectorSet Specifications

The specification of the object with basic attributes is given in Figure 7. The id gives the main address of the Object. The willingness indicates about that node in the participation of communication. The other attributes neighborset, twohopneighborset, mprset and mprselectorset are information repositories of the object. The duplicateset, linkset, topologyset are the sets used for operation of the ad hoc network and routing table is used for making routing entries.

```
  Object
id: Node
battery: Power
willingness:
neighborset:    Neighbor_tuple
twohopneighborset:    Twohop_tuple
mprset:    MPRSet
mprselectorset:    MPRSelectorSet
duplicateset:    Duplicate_tuple
linkset:    Link_Tuples
topologyset:    Topology_Tuples
routingtable:    RoutingTable
```

Figure 7: An Object with attributes Specifications

To avoid re-processing of some messages, each node maintains a Duplicate Set. For such a message, a node records a "Duplicate Tuple", where D_addr is the originator address , D_seq_num is the message sequence number of the message, D_retransmitted is a boolean indicating whether the message has been already retransmitted and its formal specifications are given in Figure 8.

```
  Duplicate_addr
D_addr: Node

  Duplicate_seq_num
D_seq_num:

  Duplicate_retransmitted
D_retransmitted: Boolean

  Duplicate_time
D_time:

  Duplicate_tuple
duplicate_tuple: Duplicate_addr
Duplicate_seq_num    Duplicate_retransmitted
Duplicate_time
```

Figure 8: Duplicate Set Specifications

The specification for local link base that stores information about links to neighbors is as shown in Figure 9. The LocalAddress is the address of the local node NeighborAddress is the address of the neighbor and L_time to indicate time till it is linked and the Link_Tuples is the structure of the tuple forming a link set for the object.

```
LocalAddress
  L_local_iface_addr: Node

    NeighborAddress
  L_neighbor_iface_addr: Node

L_time ::    Linked     NotLinked
    Link_Tuples
  link_tuple: LocalAddress    NeighborAddress
  L_time
```

Figure 9: Link Set Specifications

The formal specifications of the OLSR packet and their attributes are given in the Figure 10. The IP and the UDP headers are not considered in the specification.

```
  Packet_length
packet_length:

  Packet_seq_num
packet_seq_num:

  Messages
messages: iseq MessagePacket

  Packet
packet: Packet_length     Packet_seq_num
Messages
```

Figure10: Packet Specifications

The specification of the procedure for processing the packet is shown in Figure 11. The packet is discarded if it is of short in length than the specified or if the trust on the packet is less as per the policy. In the present case, the trust value 5 is the at least requirement for the packet to process.

```
  PacketProcessing
host?: Object
packet?: IP_Packet
status!: Status_of_message_or_packet

if packet? . ip_packet . 2 . packet . 1 . packet_length
   16          trust_on_packet ďhost?ưʹ packet?Ơ    5
then status! = Discarded
else status! = Accepted
```

Figure 11: Processing of Packet

The specification of the message packet is given in Figure 12. The formal specifications of the components of the message packet are also shown.

```
   Message_type
  message_type: 1 .. 127

   Originator_address
  originator_address: Node

   Ttl
  ttl:

   Hop_count
  hop_count:

Message_size
  message_size:

   Message_seq_num
  message_seq_num:

   MessagePacket
  msg_packet_tuple: Message_type
  Originator_address     Ttl    Hop_count
       Message_size    Message_seq_num    Message
```

Figure 12: Message Specifications

The specification of the function to make decision to forward or discard a packet is shown in the Figure 13. The object and IP address are the inputs and the status is the output. Of this function

```
 forward_or_discard: Object    IP_Address
 Status_of_message_or_packet

   h: Object; n: IP_Address
        if    m: MPRSelectorSet
                m     h . mprselectorset    n . ip_address
=       m . MS_main_addr      true
      then forward_or_discard ďhư nƠ = Forwarded
      else forward_or_discard ďhư nƠ = Discarded
```

Figure 13: Function to forward or discard a message/packet

A packet consist number of messages may be of different types. The message processing and forwarding messages are two different actions, conditioned by different rules. Processing relates to using the content of the messages, while forwarding is related to retransmitting the same message for other nodes of the network. The formal specification of the procedure of default forwarding algorithm is presented in the Figure 14. The message is the part of the packet that is the payload of packet.

```
   MessageProcessing
  ΞDuplicate_tuple
  host?: Object
  sender_address?: IP_Address
  message?: MessagePacket
  status!: Status_of_message_or_packet
  dup_set:    Duplicate_tuple
  new_dup_tuple: Duplicate_tuple

  if message? . msg_packet_tuple . 4 . ttl    1        trust_on_message ďhost?ư message?Ơ    5
  then status! = Discarded else if   ds:    Duplicate_tuple          d: Duplicate_tuple      ds = host? . duplicateset
       d    host? . duplicateset      d . duplicate_tuple . 1 . D_addr
         message? . msg_packet_tuple . 3 .   originator_address     d . duplicate_tuple . 2 . D_seq_num
       message? . msg_packet_tuple . 7 . message_seq_num     d . duplicate_tuple . 3 . D_retransmitted = False      true
      then new_dup_tuple . duplicate_tuple . 1 . D_addr    = message? . msg_packet_tuple . 3 . originator_address
        new_dup_tuple . duplicate_tuple . 2 . D_seq_num    = message? . msg_packet_tuple . 7 . message_seq_num
     ďif forward_or_discard ďhost?ư sender_address?Ơ =   Forwarded  then new_dup_tuple . duplicate_tuple . 3 . D_retransmitted
=    True  else new_dup_tuple . duplicate_tuple . 3 . D_retransmitted  = FalseƠ
      new_dup_tuple . duplicate_tuple . 4 . D_time = 10       host? . duplicateset = host? . duplicateset       new_dup_tuple
            ďif forward_or_discard ďhost?ư sender_address?Ơ = Forwarded
            then message? . msg_packet_tuple . 4 . ttl  = message? . msg_packet_tuple . 4 . ttl - 1
            else status! = DiscardedƠ       ďif forward_or_discard ďhost?ư sender_address?Ơ = Forwarded
          then message? . msg_packet_tuple . 5 . hop_count   = message? . msg_packet_tuple . 5 . hop_count + 1
            else status! = DiscardedƠ   else status! = Discarded
```

Figure 14: MessageProcessing Specifications

The formal specification of the function to determine the trust on message is presented in the Figure 15. The message packet and the object are the inputs and the trust value is the output.

```
  trust_on_message: Object    MessagePacket

   h: Object; m: MessagePacket       temp:
            if    t: Neighbor_tuple      t    h . neighborset
         m . msg_packet_tuple . 3 . originator_address
 = t . N_neighbor_tuple . 1 . N_neighbor_main_addr
     true    temp = t .N_neighbor_tuple . 3 . N_trust
       then trust_on_message ďhư mƠ = temp
       else trust_on_message ďhư mƠ = 5
```

Figure 15: Function to find trust on message

This is the formal specification of the function to determine the trust on a packet is given in Figure 16. The object and the IP packet are the input arguments and trust as an integer value is the output of this function.

```
 trust_on_packet: Object    IP_Packet

   h: Object; ip: IP_Packet        temp:
      if   t: Neighbor_tuple      t    h . neighborset
      ip . ip_packet . 1 . ip_address
  = t . N_neighbor_tuple . 1 . N_neighbor_main_addr
         true    temp = t . N_neighbor_tuple . 3 . N_trust
         then trust_on_packet ďhư ipơ = temp
         else trust_on_packet ďhư ipơ = 5
```

Figure 16: Function to find trust on packet

The mechanism [3] is employed for populating the local link information and the neighborhood information base is by the periodic exchange of HELLO messages. Htime specifies the hello emission interval used by the node. The LinkType specify the possibilities for various types of link and similarly the NeighborType enumerate the possible values for that. The structure of the HelloMessage is specified as a tuple with Htime, willingness and LinkMessage. The specifications are as shown in following Figure 17.

```
   HelloMessageTime
 Htime:

LinkType ::    UNSPEC_LINK   ASYM_LINK
   SYM_LINK   LOST_LINK
NeighborType ::    SYM_NEIGH     MPR_NEIGH
   NOT_NEIGH
   LinkMessageSize
 linkmessagesize:

   LinkCode
 linkcode: NeighborType    LinkType

   Neighbors
 neighbor: Node    Trust

   LinkMessage
 link_message: LinkCode    LinkMessageSize
 Neighbors

   HelloMessage
 hello_message: HelloMessageTime    Willingness
 LinkMessage
```

Figure 17: HelloMessage Specifications

The formal specification for the generation of HelloMessage is depicted in the Figure 18. The host uses the information available in their local linkset and neighborset to populate the various fields of HelloMessage. The trust in neighbor(s) by host, listed in the HelloMessage, also get advertised with the address(es) of the neighbor(s) and this is the accumulation by the present study. Each HelloMessage generated is broadcast by the node to its neighbors. HelloMessage(s) must never be forwarded.

```
   HelloMessageGeneration
 host?: Object
 hello!: HelloMessage
 mpr_link:    Node
 neighbor_link:    Node

 if   lt: Link_Tuples     lt    host? . linkset    lt . link_tuple . 3 = Linked     true
 then hello! . hello_message . 3 . link_message . 1 . linkcode . 2 = SYM_LINK
 else hello! . hello_message . 3 . link_message . 1 . linkcode . 2 = LOST_LINK
   m: MPRSet     m    host? . mprset     mpr_link =   m . N_mpr_tuple . 1
      ďif   lt: Link_Tuples        lt    host? . linkset     lt . link_tuple . 2 . L_neighbor_iface_addr   mpr_link        true
  then hello! . hello_message . 3 . link_message . 1 . linkcode . 1  = MPR_NEIGH
       else ď  m: Neighbor_tuple      m     host? . neighborset     neighbor_link=   m . N_neighbor_tuple . 1 .
N_neighbor_main_addr
       ďif   lt: Link_Tuples        lt    host? . linkset     lt . link_tuple . 2 . L_neighbor_iface_addr   neighbor_link     true
     then hello! . hello_message . 3 . link_message . 1 . linkcode  . 1 = SYM_NEIGH
                 else hello! . hello_message . 3 . link_message . 1. linkcode    . 1 = NOT_NEIGHơơơ
   nt: Neighbor_tuple; n: Neighbors   nt    host? . neighborset   n . neighbor . 1 = nt . N_neighbor_tuple . 1 .
N_neighbor_main_addr
      n . neighbor . 2 . N_trust = nt . N_neighbor_tuple . 3 . N_trust      hello! . hello_message . 1 . Htime = htime?
         hello! . hello_message . 2 . N_willingness = 4     hello! . hello_message . 3 . link_message . 3 =   n
```

Figure 18: HelloMessageGeneration Specifications

A node process incoming Hello messages for the purpose of conducting link sensing, neighbor detection and MPRSelectorSet population. Link sensing populates the local link information base. Neighbor detection populates the neighbor information base and Twohop neighbor detection populates the twohop information base and both of them are populated through the periodic exchange of Hello Message. The processing specification of HelloMessage is shown in the Figure 19.

```
  HelloMessageProcessing
 msg?: MessagePacket
 host?: Object
 new_link: Link_Tuples
 new_neighbor: Neighbor_tuple
 hello: HelloMessage
 new_twohop_neighbor: Twohop_tuple

 if   lt: Link_Tuples
        lt    host? . linkset     lt . link_tuple . 2 . L_neighbor_iface_addr  = msg? . msg_packet_tuple . 3 . originator_address     true
 then ď  ns:    Neighbors     ď  t: Neighbors       ns = hello . hello_message . 3 . link_message . 3    t    ns     t  neighbor . 1     host? .
id
        new_twohop_neighbor . N_twohop_tuple . 1 . N_neighbor_main_addr  = msg? . msg_packet_tuple . 3 . originator_address
              new_twohop_neighbor . N_twohop_tuple . 2 . N_2hop_addr  = t . neighbor . 1
              new_twohop_neighbor . N_twohop_tuple . 4 . N_trust  = t . neighbor . 2 . N_trust
              new_twohop_neighbor . N_twohop_tuple . 3 = Liveơơ
       ďif hello . hello_message . 3 . link_message . 1 . linkcode . 1 = SYM_NEIGH
           hello . hello_message . 3 . link_message . 1 . linkcode . 1 = MPR_NEIGH
       then host? . twohopneighborset   = host? . twohopneighborset      new_twohop_neighbor
       else host? . twohopneighborset  = host? . twohopneighborset \   new_twohop_neighbor  ơ
 else host? . twohopneighborset = host? . twohopneighborset
 if   lt: Link_Tuples      lt    host? . linkset
   lt . link_tuple . 2 . L_neighbor_iface_addr     msg? . msg_packet_tuple . 3 . originator_address  true
         ďnew_link . link_tuple . 1 . L_local_iface_addr = host? . id
      new_link . link_tuple . 2 . L_neighbor_iface_addr = msg? . msg_packet_tuple . 3 . originator_address
          new_link . link_tuple . 3 = Linkedơ
        new_neighbor . N_neighbor_tuple . 1 . N_neighbor_main_addr = new_link . link_tuple . 2 . L_neighbor_iface_addr
       new_neighbor . N_neighbor_tuple . 2 . N_willingness = hello . hello_message . 2 . N_willingness
         new_neighbor . N_neighbor_tuple . 3 . N_trust = 5
 then host? . linkset = host? . linkset      new_link       host? . neighborset = host? . neighborset      new_neighbor
 else   lt: Link_Tuples      lt    host? . linkset     lt . link_tuple . 2 . L_neighbor_iface_addr
          = msg? . msg_packet_tuple . 3 . originator_address
         if hello . hello_message . 3 . link_message . 1 . linkcode . 2 = LOST_LINK
      then lt . link_tuple . 3 = NotLinked  else lt . link_tuple . 3 = Linked
```

Figure 19: HelloMessageProcessing Specifications

Any host while in operation may need to know the trust of other nodes in the network in order to populate their own entries in the neighborset and twohopneighborset for trust values. In order to accomplish this host sends a TREQMessage to other nodes. In response, other nodes generate TREPMessage with trust value if known otherwise with value 1 indicating unknown to it. The specification shown in Figure 20 for the TREQMessage with first node is address of the host and the other is node in question. In TREPMessage, first node is the address of the object recommending trust of the node specified by the second component with value given in third component.

```
  TREQMessage
 treq_message: Node    Node

  TREPMessage
 trep_message: Node    Node    Trust
```

Figure20: TREQ and TREP Message Specifications

The specification of the procedure to generate a message for the trust query is given in the Figure 21. The host is the Object asking for the trust value of the node_in_query to the other nodes in the network.

```
  TREQMessageGeneration
host?: Object
node_in_query?: Node
treqmessage!: TREQMessage

treqmessage! . treq_message . 1 = host? . id
   treqmessage! . treq_message . 2 = node_in_query?
```

Figure 21: Trust Request Message Generation Specifications

Whenever any node received a TREQMessage, it is processed with specification given in Figure 22 and generates the appropriate TREPMessage as per the specifications.

```
  TREQMessageProcessing
treqmessage?: TREQMessage
host?: Object
trepmessage!: TREPMessage
trust:

  ns:    Neighbor_tuple
        nt: Neighbor_tuple     ns = host? . neighborset    nt
   ns      if   n: Neighbor_tuple      n    ns    n .
N_neighbor_tuple . 1 . N_neighbor_main_addr
                    = treqmessage? . treq_message . 2      true
trust = n . N_neighbor_tuple . 3 .N_trust
    then trepmessage! . trep_message . 1 =host? . id
               trepmessage! . trep_message . 2 =
treqmessage? . treq_message . 2
      trepmessage! . trep_message . 3 . N_trust = trust
     else trepmessage! . trep_message . 3 .N_trust =  1
```

Figure 22: TREQ Processing Specifications

As whenever any node sends the TREQMessage to other objects in the network, in reponse, it gets a number of TREPMessages. In order to weight differently the recommendation provided by neighbors, twohopneighbors and other, they are categorized depending on the originator address of the TREPMessage. The specifications are shown in Figure 23.

```
  TREPMessageProcessing
host?: Object
trust_from_neighbors:
trust_from_twohop_neighbors:
trust_from_others:

  ns:    Neighbor_tuple; ts:    Twohop_tuple
        nt: Neighbor_tuple; tt: Twohop_tuple
          ns = host? . neighborset     nt    ns    ts = host? .
twohopneighborset      tt    ts      if   n:
Neighbor_tuple  n  ns
                   n . N_neighbor_tuple . 1 .
N_neighbor_main_addr = trepmessage? . trep_message .
1      true   then trust_from_neighbors =
trust_from_neighbors  +trepmessage? . trep_message . 3 .
N_trust   else if   t: Twohop_tuple      t    ts
                       t . N_twohop_tuple . 2 . N_2hop_addr
=trepmessage? . trep_message . 1      true   then
trust_from_twohop_neighbors
  = trust_from_twohop_neighbors  +   trepmessage? .
trep_message . 3 . N_trust  else
trust_from_others  = trust_from_others  + trepmessage?
.trep_message . 3 . N_trust
```

Figure 23: Trust Reply Message Processing Specifications

The MPR Selection procedure involves the procedures – Find Isolated Nodes, Other than Partia lMprs, Find Uncovered Twohops, Neighbor Covering, Find Neighbor Covering, Remove2hop Neighbors, Find Maximum. The Find Isolated Nodes given in Figure 24 searches the isolated twohops t, these are partial MPRs of the object. The invariants are i) the isolated object should not be connected to any other object other than the neighbor of given object.

```
  FindIsolatedNodes
host?: Object
isolated_2hop_nodes!:    Node
trust: Trust

  x:    Twohop_tuple        ob1: Object; ob2: Object; y:
Twohop_tuple; m: MPRSet      x = host? .
twohopneighborset     y    x    ďob1ư ob2Ơ   connection
   ob1 . id = y . N_twohop_tuple . 2 . N_2hop_addr
     m . N_mpr_tuple  = ďy . N_twohop_tuple . 1 .
N_neighbor_main_addrư trustƠ
     host? . mprset = host? . mprset      m
     isolated_2hop_nodes! =    y . N_twohop_tuple . 2 .
N_2hop_addr
```

Figure 24: Search of Isolated TwoHop Nodes

The OtherthanPartialMprs procedure shown in Figure 25 finds the neighbors that are not selected as MPRs by the FindIsolatedNodes procedure. The invariants are i) the others should be equal to the set difference between all neighbor identities and neighbor selected as MPR by the above procedure.

```
  OtherthanPartialMprs
 others!:    Node
 host?: Object

  x:    Neighbor_tuple; neighbor_ids:    Node; mpr_ids:
 Node; t:    MPRSet
         y: Neighbor_tuple; m: MPRSet
            x = host? . neighborset    y    x    t = host? .
 mprset    m    t     neighbor_ids =    y . N_neighbor_tuple .
 1 . N_neighbor_main_addr       mpr_ids =    m
 .N_mpr_tuple .1
           others! = neighbor_ids \ mpr_ids
```

Figure 25: Search Neighbors other than MPR

The procedure FindUncoveredTwohops of Figure 26 is to find the twohop nodes still not covered by any of the MPR in the MPRSet of the given object. The invariants is i) the set difference between the twohopneighborset and the set of tuples of twohopneighborset whose twohopneighbors are given by isolated_hops- one of the output of FindIsolatedNodes.

```
  FindUncoveredTwohops
 host?: Object
 isolated_2hops?:    Node
 temptwohopset!:    Twohop_tuple

  x:    Twohop_tuple
    y:Twohop_tuple  x=host?. twohopneighborset    y    x
     y . N_twohop_tuple . 2 . N_2hop_addr
 isolated_2hops?
          temptwohopset! = host?. twohopneighborset \
 y
```

Figure 26: Search TwoHop Neighbors not covered by MPR

The specification of the tuple for NeighborCovering is given in Figure 27. The components are the identity of the neighbor, the number of twohops covered by that neighbor and the set of identities of twohop neighbors covered by it.

```
  NeighborCovering
 covered_by_neighbors: Node            Node
```

Figure 27: Structure of Neighbor Covering

The procedure FindNeighborCovering given in Figure 28 is to build the neighbor covering by neighbors. The purpose of the procedure is to find the neighbor covering of uncovered twohops given by the above procedure.

```
  FindNeighborCovering
 host?: Object
 neighbor_covering!: NeighborCovering

  x:    Twohop_tuple; z: Neighbor
    y: Twohop_tuple    x = host? . twohopneighborset
y  x
           z      y . N_twohop_tuple . 1        z = y .
 N_twohop_tuple . 1      neighbor_covering! .
 covered_by_neighbors
  = ďz . N_neighbor_main_addrư
          #    y . N_twohop_tuple . 2 . N_2hop_addr  ư
            y . N_twohop_tuple . 2 . N_2hop_addr  Ơ
```

Figure 28: Serach Covering by Neighbor

The procedure given below in Figure 29, Remove2hopNeighbors is to find the twohopneighborset without the twohop neighbors that are covered by any of the entry of the MPRSet of the given object.

```
  Remove2hopNeighbors
 host?: Object
 modi_2hop_set?:    Twohop_tuple
 reduced_2hop_set!:    Twohop_tuple
 twohop_nodes?:    Node

  x, z:    Twohop_tuple
        y: Node; t: Twohop_tuple      x = modi_2hop_set?
          x    host? . twohopneighborset     y
 twohop_nodes?     t . N_twohop_tuple . 2 . N_2hop_addr
   twohop_nodes?      z =    t        reduced_2hop_set! = x \
z
```

Figure 29: Removal of TwoHop Neighbors

The procedure FindMaximum given in Figure 30, is to find the neighbor with maximum covering of twohop neighbors and then update the MPRSet of the object This procedure gets repeated till all twohop nodes get covered by any of the selected MPR.

```
  FindMaximum
 host?: Object
 covering_neighbors?:    NeighborCovering
 twohop_covered!:    Node
 reduced_covering_neighbors!:    NeighborCovering
 trust: Trust

  y: NeighborCovering
        x: NeighborCovering; m: MPRSet
      x    covering_neighbors?      y      covering_neighbors?
    x      y     x . covered_by_neighbors.2
           y. covered_by_neighbors . 2
     m .N_mpr_tuple = ďx . covered_by_neighbors . 1ư
trustƠ
      twohop_covered! = x . covered_by_neighbors . 3
           host? . mprset = host? . mprset       m
    reduced_covering_neighbors!=covering_neighbors?\  x
```

Figure30: Select MPR from Neighbor Covering Tuples

The topology information is dispersed through the network. The information given by the link sensing and neighbor detection is disseminated to the entire network through this and it is used to construct routes [3]. The formal specification for the structure of topology tuple is given in Figure 31.

```
  TopologyDestinationAddress
 T_dest_addr: Node
  DestinationLastAddress
 T_last_addr: Node
  TopologySequenceNumber
 T_seq_num:
  TopologyTime
 T_time: 1 .. 10
  Topology_Tuples
 topology_tuple: TopologyDestinationAddress
 DestinationLastAddress
    TopologySequenceNumber    TopologyTime    Trust
```

Figure 31: Topology Tuple Specifications

The specification of the Topology Control Message is presented in Figure 32. A TC message is sent by a node in the network to declare a set of links, called advertised link set which MUST include at least the links to all nodes of its MPR Selector set. This is sent as the data-portion of the general message format with the "Message Type" set to TC_MESSAGE. A sequence number is associated with the advertised neighbor set.

```
  ANSN
ansn:

  AdvertisedNeighbor
advertised_neighbor: Node    Trust

  AdvertisedNeighborSet
advertised_neighbor_tuple:    AdvertisedNeighbor

  TopologyControlMessage
tc_message: ANSN    AdvertisedNeighborSet
```

Figure 32: TC Message Specifications

TC messages are broadcast and retransmitted by the MPRs in order to diffuse the messages in the entire network. The formal specification of the updation of topology set is given in Figure 33. TC messages MUST be forwarded according to the "default forwarding algorithm".

```
Status_tc_message ::    process     discard
    TCMessageProcess
  host?: Object
  msg?: MessagePacket
  tcmsg: TopologyControlMessage
  status!: Status_tc_message
  new_topology_tuple: Topology_Tuples

    ts:    Topology_Tuples        tc: Topology_Tuples; an: AdvertisedNeighbor
         ts = host? . topologyset      tc    ts     an    tcmsg . tc_message . 2 . advertised_neighbor_tuple
         if tc . topology_tuple . 2 . T_last_addr = msg? . msg_packet_tuple . 3 . originator_address
            tc . topology_tuple . 3 . T_seq_num     tcmsg . tc_message . 1 . ansn   then status! = discard
       else if tc . topology_tuple . 2 . T_last_addr  = msg? . msg_packet_tuple . 3 . originator_address
                tc . topology_tuple . 3 . T_seq_num     tcmsg . tc_message . 1 . ansn
           then host? . topologyset = host? . topologyset \    tc
           else if tc . topology_tuple . 2 . T_last_addr = msg? . msg_packet_tuple . 3 . originator_address
                   tc . topology_tuple . 1 . T_dest_addr  = an . advertised_neighbor . 1
        then tc . topology_tuple . 4 . T_time = 10
        else if tc . topology_tuple . 2 . T_last_addr   = msg? . msg_packet_tuple . 3  . originator_address
                  tc . topology_tuple . 1 . T_dest_addr  = an . advertised_neighbor . 1
                   tc . topology_tuple . 5 . N_trust     an . advertised_neighbor . 2 . N_trust
        then tc . topology_tuple . 5 . N_trust  = an . advertised_neighbor . 2 . N_trust
                 else new_topology_tuple . topology_tuple . 1 . T_dest_addr = an . advertised_neighbor . 1
             new_topology_tuple . topology_tuple . 2 . T_last_addr = msg? . msg_packet_tuple . 3   . originator_address
               new_topology_tuple . topology_tuple . 3 . T_seq_num = tcmsg . tc_message . 1 . ansn
               new_topology_tuple . topology_tuple . 4  . T_time = 10    new_topology_tuple . topology_tuple . 5  . N_trust
             = an . advertised_neighbor . 2 . N_trust      host? . topologyset   = host? . topologyset       new_topology_tuple
```

Figure 33: Formal Specifications of TC Message Processing

In order to build the topology information base, each node, which has been selected as MPR, broadcasts Topology Control (TC) messages. TC messages are flooded to all nodes in the network and take advantage of MPRs. MPRs enable a better scalability in the distribution of topology information. The formal specification of the procedure of generating topology control message is shown in Figure 34.

```
   TCMessageGeneration
host?: Object
an_sn?:
tcmsg!: TopologyControlMessage
adv_neigh: AdvertisedNeighbor
adv_neigh_set:    AdvertisedNeighbor

  ns:    Neighbor_tuple       n: Neighbor_tuple     ns =
host? . neighborset    n    ns
```

```
          ɗadv_neigh . advertised_neighbor . 1 = n .
N_neighbor_tuple . 1 . N_neighbor_main_addr
          adv_neigh . advertised_neighbor . 2 . N_trust
= n . N_neighbor_tuple . 3 . N_trust
          adv_neigh_set =   adv_neigh  ơ     tcmsg! .
tc_message . 1 . ansn = an_sn? + 1
          tcmsg! . tc_message . 2 .
advertised_neighbor_tuple  = adv_neigh_set
```

Figure 34: TC Message Generation Specifications

Each node maintains a routing table which allows it to route data, destined for the other nodes in the network. The routing table is based on the information contained in the link set and the topology set[3]. Each entry in the table consists of R_dest_addr, R_next_addr, R_dist. Such entry specifies that the node identified by R_dest_addr is estimated to be R_dist hops away from the local node, that the symmetric neighbor node with interface address R_next_addr is the next hop node in the route to R_dest_addr. The specification of the Route Table is shown in the following Figure 35.

```
  Routing_destination
 R_dest_addr: Node

  Routing_next
 R_next_addr: Node

  Routing_distance
 R_dist:

  RoutingTable
 routing_table_tuple: Routing_destination
Routing_next      Routing_distance      Trust
```

Figure 35: Formal Specifications of TC Message Processing

The routing table is recalculated in case of neighbor appearance or loss, when a 2-hop tuple is created or removed, when a topology tuple is created or removed or when multiple interface association information changes [3]. The update of this routing information does not generate or trigger any messages to be transmitted, neither in the network, nor in the 1-hop neighborhood. The formal specification of the routing table procedure is presented in Figure 36.

```
RoutingTableCalculation
 host?: Object
 new_entry: RoutingTable

  rt:    RoutingTable        lt: Link_Tuples      lt    host? . linkset    rt = host? . routingtable
       new_entry . routing_table_tuple . 1 . R_dest_addr= lt . link_tuple . 2 . L_neighbor_iface_addr
       new_entry . routing_table_tuple . 2 . R_next_addr = lt . link_tuple . 2 . L_neighbor_iface_addr
        new_entry . routing_table_tuple . 3 . R_dist = 1     new_entry . routing_table_tuple . 4 . N_trust = lt . link_tuple . 4 .
N_trust
           host? . routingtable = host? . routingtable       new_entry
  tt: Twohop_tuple      tt     host? . twohopneighborset
      new_entry . routing_table_tuple . 1 . R_dest_addr= tt . N_twohop_tuple . 2 . N_2hop_addr
    new_entry . routing_table_tuple . 2 . R_next_addr = tt . N_twohop_tuple . 1 . N_neighbor_main_addr
    new_entry . routing_table_tuple . 3 . R_dist = 2     new_entry . routing_table_tuple . 4 . N_trust= tt . N_twohop_tuple . 4 .
N_trust
        host? . routingtable = host? . routingtable       new_entry
  te: Topology_Tuples; rt: RoutingTable; h:          te     host? . topologyset    rt     host? . routingtable
      te . topology_tuple . 1 . T_dest_addr    rt . routing_table_tuple . 1 . R_dest_addr
      te . topology_tuple . 2 . T_last_addr = rt . routing_table_tuple . 1 . R_dest_addr
      rt . routing_table_tuple . 3 . R_dist = h
      new_entry . routing_table_tuple . 1 . R_dest_addr = te . topology_tuple . 1 . T_dest_addr
    new_entry . routing_table_tuple . 2 . R_next_addr = te . topology_tuple . 2 . T_last_addr
     new_entry . routing_table_tuple . 3 . R_dist = h + 1     new_entry . routing_table_tuple . 4 . N_trust = te . topology_tuple . 5 .
 N_trust    host? . routingtable = host? . routingtable       new_entry
```

Figure 36: Formal Specifications of Routing Table Calculation

5. CONCLUSIONS

The formal specifications of the structures/classes and the procedures for a trust oriented OLSR protocol of ad hoc network are presented in the paper. The inclusion of trust in the specifications of the protocols is the contribution of this paper. The trust value is used for packet processing, message processing, routing decisions. The invariants are used rather than exhaustive functional analysis. These invariants, represented in the form of logical formulas, are checked in order to find any violation in their behavior. In this approach, invariants are checked that describe properties in order to identify behaviors that violate them. The future work is to compare the approach adopted in this paper with the other formal approaches.

6. REFERENCES

[1] Bhalaji, N., Sivaramkrishnan, A. R., Banerjee, S., Sundar, V., and Shanmugam, A. “Trust Enhanced Dynamic Source Routing Protocol for Adhoc Networks” *World Academy of Science, Engineering and Technology,* pp. 1074-1079, 2009

[2] Camara, D., Loureiro, A. A. F., and Filali F., “Methodology for Formal Verification of Routing Protocols for Ad Hoc Wireless Networks”, *In Proceedings of the IEEE Conference on Global Communications,* pp. 705 – 709. 2007.

[3] Clausen, T. Ed., Jacquet, P. Ed., “Optimized Link State Routing Protocol (OLSR)”, *IETF INTERNET DRAFT*, RFC 3626, 2003.

[4] Fu, L., Sun, G. and Chen J. "An Approach for Component-Based Software Development" *International Forum on Information Technology and Applications*. Vol. 1, pp. 22-25, 2010.

[5] Ian Sommerville, "Software engineering", Addison Wesley 7th edition 2004

[6] Nekkanti, R. K. and Lee, Chung-wei. "Trust Based Adaptive on Demand Ad Hoc Routing Protocol" *In Proceedings of the ACM Southeast Regional Conference*, pp. 88-93, 2004

[7] Shakeel A., Ramani, A. K. and Nazir A. Z. "Verifying Route Request Procedure of AODV Using Graph Theory and Formal Methods" *International Journal On Applications Of Graph Theory In Wireless Ad Hoc Networks And Sensor Network*. Vol. 2, No. 2, pp. 1-13. 2011.

[8] Spivey J M, "The Z Notation: A Reference Manual," *Prentice Hall*, 1989.

[9] Stéphane, M. and Faitha, Z., "Testing Methodology for an Ad Hoc Routing Protocol", *Proceedings of the ACM international workshop on Performance monitoring, measurement, and evaluation of heterogeneous wireless and wired networks*, pp. 48-55, 2006.

[10] Švec, J. and Zahradník, J., "Formal Specification in "Z" Language By Software Z/Eves", *International Journal on Advances in Electrical and Electronic Engineering*, pp. 166-168.

[11] Verma, A. and Gujral, M. S., **"**Impact of Trust Usage in Routing, Authentication and Access Control of Adhoc Network", *International Journal of Advance in Communication Engineering*, Vol. 2, No.1, pp. 1-7, 2010

[12] Verma, A., "Formal Verification of Ad Hoc Network Routing Protocols", *International Journal of Advanced Research in Computer Science*", Vol 2, No. 4, pp. 526 - 530, July-August 2011.

[13] Verma, A. and Gujral, M S , "Performance Analysis of Routing Protocols for Ad hoc Networks", *International Journal of Computer Science and Emerging Technologies*, , Vol 2, No. 4, pp. 484 – 487 , August 2011

[14] Woodcock, J. and Davies, J., "UsingZ: Specification, Refinement and Proof", www.cs.cmu.edu/~15819/zedbook.pdf

[15] Z/EVES Reference manual, available at: *http://www.oracanada.com/pub/doc/97-5493-03d.pdf*