Appendix A

Pseudocode

A.1 Network Setup Phase

```
/* This code is initiated at the beginning of the network setup phase */

Network-Setup ()
    {Let T_ic be a pre-defined value indicating the information collection time

    If (the current node is the PCA)
        {Broadcast NETSET-Packet
        While (T_ic-Elapsed= False)
            {If (packet is received)
                If (NIN-Packet) and (not duplicate packet) and (from authorized node)
                    Allow node to participate in the network and store the needed information
                Else
                    Discard the packet
            } /* while */
        Divide the network into multiple virtual zones
        Calculate probability of each node to be a LCA to a given zone side of the zone it resides in
        Assign four LCAs for each zone
        Unicast a NROLE-Packet to each participating node
        /* The PCA now becomes an ordinary node */
        } /* If the current node is the PCA */

    Else
        /* Non-PCA */
        {Setup=False
        /* Wait till a NETSET packet is received from PCA */
        While not (Setup )
            {If (packet is received)
                If (NETSET-Packet) and (from PCA)
                    Setup=True
                Else
                    Discard the packet
            }
        /* Upon receiving the NETSET packet from PCA */
        Record the IP address of the predecessor node
        /* Continue broadcasting the packet*/
        Broadcast the NETSET-Packet
        Reply with a NIN-Packet to the PCA
        While (Setup)
            {If (NIN-Packet is received)
                If (not duplicate packet) and (from authorized node)
                    {Encrypt the packet using CK
                    /* Continue sending the packet through the reverse path to PCA */
                    Forward the NIN-Packet back to the predecessor from which NETSET-Packet was received
                    }
                Else
                    Discard the packet
            If (NROLE-Packet is received)
                If (from PCA) and (not duplicate packet)
                    If (current node is the destination)
                        {Setup=False
                        Store needed information in the NROLE-Packet}
                    Else
                        Forward the NROLE-Packet to the next hop in the source route
                Else
                    Discard the packet
            } /* While */
        /* Node is now able to participate in the routing activities. At the same time, it is still able to forward
           NROLE packets for other nodes joining the network */
        } /* Non-PCA */
    } /* Network-Setup ()*/
```

A.2 Code Executed At All Nodes after Finishing Network Setup

```
/* This code is executed at all nodes after finishing network setup */

Pos-Needed = False              /* Set to True when the position of a destination is needed */
Route-Needed = False            /* Set to True when a route to a destination is needed */
Pos-known = False               /* Set to True when the position of a destination is known */
Let T_cu be a pre-defined value indicating the certificate update time
Let T_rd and T_pd be pre-defined values indicating route discovery time and position discovery time respectively
Let D_mov be a pre-defined distance that a node moves before informing its zone LCAs about its new position

If (current node is LCA)
    {LCA-Election = False           /* Set to True when a new LCA is needed to replace the current one*/
    Failed-LCA-Election= False      /* Set to True when a new LCA is needed to replace a failed one */
    For each node i in this zone
        Cert-Needed [i] = False     /* Set to True when a certificate update to a node is needed */
    }
/* This code is executed continuously */
While (True)
    {/************************************ LCA duties ************************************/
    If (the current node is a LCA)
        LCA-Duties ()

    /********************************** Updating certificates **********************************/
    If (T_cu -Elapsed)
        If (the current node is a LCA)
            Send-ACREQ (Current-LCA)
        Else
            Send a CREQ-Packet to the nearest LCA to itself using restricted directional flooding

    /********************************** Nodes movements **********************************/
    If (the current node has moved D_mov from its last known position)
        If (the current node is a LCA)
            Send a ULPOS-Packet to nodes in its zone and UALPOS-Packet to its adjacent LCA
        Else
            Send a UNPOS-Packet to the nearest LCA to itself using restricted directional flooding

    /******************************** Having data packets to be sent ********************************/
    If (the current node has a DATA-Packet to be sent)
         If (route exists in the routing table)
            Send the DATA-Packet
        Else
            {Add DATA-Packet to Buffer
            If (Pos-known = True)
                If (Route-Needed = False)
                    /* Use destination position to begin route instantiation */
                    {Route-Needed = True
                     RDP-No=0
                     T_rd-Elapsed = True}
            Else
                If (Pos-Needed = False)
                    {Pos-Needed = True
                     PDP-No=0
                     T_pd-Elapsed = True}
            }

    /******************************** Position and route discovery ********************************/
    If (Pos-Needed = True)
        Find-Pos ()
    If (Route-Needed = True)
        Find-Route ()
    If there is a broken link
        Send an ERR-Packet towards the source of the route

    /******************************** Handling received packets ********************************/
    If (Packet is received)
        Handle-Packet (Packet)
    } /* While (True) */
```

A.2.1 *LCA*-Duties Function

```
/* This code is executed by the LCA nodes */

LCA-Duties ()
    {Let T_ls be a pre-defined value indicating the LCA synchronization time
    Let T_ac be a pre-defined value indicating the acceptance of certificate time
    Let T_sl be a pre-defined value indicating the LCA serving as LCA time
    Let D_sid be a pre-defined distance that a LCA is allowed to be from the zone side middle point
    Let T_pc be a pre-defined value indicating the probability packets collection time

    /************************** Updating certificates of the nodes **************************/
    For each node i in this zone
        If (Cert-Needed [i] = True) and (T_ac-Elapsed)
            {Cert-Needed [i] = false
            If (Number-of-received-ACREP-packets [i] >=3)
                If (current node is the Source)
                    {Issue its self a node and LCA certificates}
                Else
                    {unicast a CREP-Packet back along the reverse path to the source of the CREQ-Packet
                    unicast a NCERT-Packet to other LCAs}
            }

    /****************************** LCAs' synchronization ******************************/
    If (T_ls-Elapsed) and (current node turn to send the CLSYN-Packet)
        Send a CLSYN-Packet to other LCAs in the network

    /************************** Current LCA movement and periodic election*****************/
    If (current node decided to leave its zone) or (T_sl-Elapsed) or
       (its distance from the middle point of the zone side > D_sid)
        {Send a NLCAE-Packet to nodes in its zone to start a new LCA election process to the specified zone
        side
        LCA-Election = True
        Max-Prob=Current-LCA-Prob}

        If (LCA-Election = True) and (T_pc-Elapsed)
        {Elect-New-LCA ()
        LCA-Election = False
        /* The LCA now becomes an ordinary node */}

    /********************************* Other LCAs failure *********************************/
    /* The voluntary LCA takes the responsibility of electing a new LCA*/
    If (Authentication table has an expired LCA certificate) and (LCA index== current node index -1)
        {/* Inform nodes in the zone about the failure and ask them to calculate
            their probability to replace the failed LCA */
        Broadcast FLCA-Packet specifying the zone side for which the failed LCA was responsible for
        Send FALCA-Packet to the adjacent LCA of the failed one
        Failed-LCA-Election = True
         Max-Prob=Current-LCA-Prob
        /* The voluntary LCA will temporarily play the role of the failed LCA till electing the new one */}

    If (Failed-LCA-Election = True) and (T_pc-Elapsed)
        {Elect-New-LCA()
        Notify-Node-Sudden-Failure (Failed-LCA-IP-Address, Failed-LCA-Public-Key)
        Failed-LCA-Election = False}

    /****************************** Non-LCA nodes failure ******************************/
    /* The LCA that had issued the last certificate for the failed node will inform other LCAs about the failure*/
    If (Authentication table has an expired node certificate) and
       (current LCA is the one that issued the last certificate for the failed node )
        Notify-Node-Sudden-Failure (IP-Address, Public-Key)

    /************************ Last node in the zone decides to leave it ************************/
    /* Note: this node plays the role of the four LCAs */
     If (current node is the last node in the zone) and (decides to leave the zone)
        Send an EZONE-Packet to adjacent LCA in the zone it is leaving to

    } /* LCA-Duties () */
```

A.2.2 Find-Pos Function

```
/* This code is executed when the position of a destination is needed */

Find-Pos ()
    {Let T_pd be a pre-defined value indicating the position discovery time
    If (T_pd-Elapsed)
        If (PDP-No!=2)
            {Send a PDP-Packet to the nearest LCA using restricted directional flooding
            PDP-No= PDP-No + 1
            T_pd-Elapsed=False}
        Else
            {Pos-Needed = False
            Broadcast RDP-Packet}
    } /* Find-Pos () */
```

A.2.3 Find-Route Function

```
/* This code is executed when a route to the destination is needed */

Find-Route ()
    {Let T_rd be a pre-defined value indicating the route discovery time
    If (T_rd-Elapsed)
        If (RDP-No!=2)
            {Send a RDP-Packet to the intended destination using restricted directional flooding
            RDP-No= RDP-No + 1
            T_rd-Elapsed =False}
        Else
            {Route-Needed = False
            Broadcast RDP-Packet}
    } /* Find-Route () */
```

A.2.4 Handle-Packet Function

```
/* This code is executed upon receiving a packet */
Handle-Packet (Packet)
    {Switch (Packet-Identifier)
        {/********** Updating certificates and synchronization *********/
        Case CREQ: Handle-CREQ (Packet)
        Case ACREQ: Handle-ACREQ (Packet)
        Case CREP: Handle-CREP (Packet)
        Case ACREP: Handle-ACREP (Packet)
        Case CLSYN: Handle-CLSYN (Packet)
        Case NCERT: Handle-NCERT (Packet)
        /****************** Nodes movement and failure *************/
        Case UNPOS: Handle-UNPOS (Packet)
        Case DNODE: Handle-DNODE (Packet)
        Case NNODE: Handle-NNODE (Packet)
        Case NZONE: Handle-NZONE (Packet)
        Case NLCAE: Handle-NLCAE (Packet)
        Case FLCA: Handle-NLCAE (Packet)
        Case FALCA: Handle-Adj-LCA-Packet (Packet)
        Case NALCA: Handle-Adj-LCA-Packet (Packet)
        Case UALPOS: Handle-Adj-LCA-Packet (Packet)
        Case NPROB: Handle-NPROB (Packet)
        Case FNODE: Handle-FNODE (Packet)
        Case ULPOS: Handle-Zone-Packet(Packet)
        Case FLCA : Handle-Zone-Packet(Packet)
        Case NLCA : Handle-Zone-Packet(Packet)
        /***************** Position and route discovery ***************/
        Case PDP: Handle-PDP (Packet)
        Case PREP: Handle-PREP (Packet)
        Case RDP: Handle-RDP (Packet)
        Case RREP: Handle-RREP (Packet)
        Case ERR: Handle-ERR (Packet)
        /****************** Data transmission **********************/
        Case DATA: Handle-DATA (Packet)
        } /* Switch (packet) */
} /* Handle-Packet */
```

A.3 Validation Functions

A.3.1 Validate-Initiator Function

```
Boolean Validate-Initiator (Packet)
    {If (the Packet initiator has a valid signature and a fresh node certificate)
        Return (True)
     Else
        Return (False)
    }
```

A.3.2 Validate-Predecessor Function

```
Boolean Validate-Predecessor (Packet)
    {If (the Packet predecessor has a valid signature and a fresh node certificate)
        Return (True)
    Else
        Return (False)
    }
```

A.3.3 Validate-*LCA*-Initiator Function

```
Boolean Validate-LCA-Initiator (Packet)
    {If (the Packet initiator has a fresh zone LCA certificate for its zone)
        Return (True)
    Else
        Return (False)
     }
```

A.4 Packet Forwarding Techniques

A.4.1 Restricted Directional Flooding

```
/* This function is called when the current node is an intermediate
   node for a packet sent using restricted directional flooding */

Restricted-Directional-Flooding (Packet)
    {If (the current node is the first node that receives this Packet)
        If (Validate-Initiator (Packet))
            {Record the IP address of the predecessor node
            Add a new field indicating the distance from itself to the destination
            Sign the contents of the Packet and append own certificate
            Broadcast the Packet}
        Else
            Discard the Packet
    Else
        {/* Validate the signatures for both the Packet initiator and the neighbour it received the Packet
            from and continue broadcasting the Packet only if the current node is closer to destination */
        If (Validate-Initiator (Packet)) and (Validate-Predecessor (Packet)) and
           (the distance between itself and the destination < the recorded distance)
            {Record the IP address of the predecessor node
            Change the distance value in the Packet
            Remove predecessor's certificate and signature
            Sign the contents of the Packet and append own certificate
            Broadcast the Packet}
        Else
            Discard the Packet
        } /* Not the first node that receives this Packet */
} /* Restricted-Directional-Flooding */
```

A.4.2 Source Routing

```
/* This function is called when the current node is an intermediate node for a packet sent using source routing */

Source-Routing (Packet)
    {If (the current node is the first node in the source route)
        If (Validate-Initiator (Packet))
            {Record the IP address of the predecessor node
            Sign the contents of the Packet and append own certificate
            Forward the packet to the next hop in the source route}
        Else
            Discard the Packet
    Else
        {If (Validate-Initiator (Packet)) and (Validate-Predecessor (Packet)) and
            (Forwarded from previous hop in source route)
           {Record the IP address of the predecessor node
            Remove predecessor's certificate and signature
            Sign the contents of the Packet and append own certificate
            Forward the packet to the next hop in the source route}
        Else
            Discard the Packet
        } /* Not the first node in the source route */
    }/* Source-Routing */
```

A.4.3 Forwarding *LCA-LCA* Communication by A *LCA*

```
/* The following code is executed by LCA upon deciding to forward a LCA-LCA packet it receives. All is a
Boolean value indicating whether to send the packet to all LCAs in the zone or only those have adjacent LCAs */

Continue-LCA-LCA-Communication (LCA-LCA-Packet, All)
    {If (zone of the LCA-LCA-Packet initiator = zone of the current node)
        /* The packet is from the same zone */
        {If (the adjacent LCA is within the transmission range of this LCA)
            Send a LCA-LCA-Packet to adjacent LCA using 1-hop unicast.
        Else
            Send a LCA-LCA-Packet to adjacent LCA using restricted directional flooding
        }
    Else
        If (zone of the LCA-LCA-Packet initiator = adjacent zone)
            If (All=True)
                Send the LCA-LCA-Packet to all LCAs in the current zone using source routing
            Else
                Use source routing to send a LCA-LCA-Packet to other LCAs
                in the current zone those have adjacent LCAs
        Else
                Discard the packet
    } /* Continue-LCA-LCA-Communication */
```

A.4.4 Forwarding *LCA-LCA* Communication by Non-*LCA* Node

```
/* The following code is executed upon receiving a LCA-LCA packet by intermediate nodes */

Intermediate-LCA-LCA-Communication (LCA-LCA-Packet)
    {If (zone of the LCA-LCA-Packet initiator = zone of the current node)
        /* Packet is sent between LCAs inside the same zone */
        Source-Routing (LCA-LCA-Packet)
    Else
        /* Packet is sent between different zones*/
        If (LCA-LCA-Packet is not a 1-hop unicast packet)
            Restricted-Directional-Flooding (LCA-LCA-Packet)
        Else
            Discard the packet
    } /* Intermediate-LCA-LCA-Communication */
```

A.4.5 Reverse-Path

```
/* The following code is executed upon receiving a reply packet by intermediate nodes */

Reverse-Path (Reply-Packet, Original-Request-Packet-Predecessor)
    {If (the current node is the first node that receives this Packet)
        If (Validate-Initiator (Packet))
            {Sign the contents of the Packet and append own certificate
            /* Continue sending the packet through the reverse path */
            Forward the packet back to the Original-Request-Packet-Predecessor}
        Else
            Discard the Packet

    Else
        {If (Validate-Initiator (Packet)) and (Validate-Predecessor (Packet))
            {Remove predecessor's certificate and signature
            Sign the contents of the Packet and append own certificate
            Forward the packet back to the Original-Request-Packet-Predecessor}
        Else
            Discard the Packet
        } /* Not the first node that receives this Packet */
    } /* Reverse-Path */
```

A.4.6 Handle A Packet Sent To Nodes Residing Currently In A Specific Zone

```
/* The following code is executed upon receiving a packet sent to nodes in a specific zone */

Handle-Zone-Packet (Packet)
  {If (the current node is not in the corresponding zone) and (not duplicate packet) and
      (Validate-Initiator (Packet)) and (Validate-Predecessor (Packet)) and
      (Validate-LCA-Initiator (Packet))
          Store needed information and continue broadcasting the Packet
    Else
      Discard the packet
  } /* Handle-Zone-Packet (Packet)*/
```

A.4.7 Handle A Packet Sent Between Adjacent *LCA*s

```
/* The following code is executed upon receiving a packet sent between adjacent zones */

Handle-Adj-LCA-Packet (Packet)
    {If (not duplicate packet) and (current node is the destination) and
        (Validate-Initiator (Packet)) and (Validate-Predecessor (Packet)) and
        (Validate-LCA-Initiator (Packet))
        Store needed information

    Else
        If (not duplicate packet) and (current node is not the destination) and
           (Validate-Initiator (Packet)) and (Validate-Predecessor (Packet)) and
           (Packet is not a 1-hop unicast packet)
             Restricted-Directional-Flooding (Packet)
        Else
            Discard the packet
    } /* Handle-Adj-LCA-Packet (Packet) */
```

A.5 Network Maintenance

A.5.1 Updating Certificates

A.5.1.1 Code Executed Upon Receiving *CREQ* Packet

```
/* The following code is executed by the nearest LCA upon receiving a CREQ packet */

Handle-CREQ (CREQ-Packet)
    {If (not duplicate packet) and (current node is not the destination)
        /* The current node is an intermediate node for a CREQ packet*/
        Restricted-Directional-Flooding (CREQ-Packet)

    Else
        /* The current node is the nearest LCA */
        If (not duplicate packet) and (current node is the destination) and
           (Validate-Initiator (CREQ-Packet)) and (Validate-Predecessor (CREQ-Packet))
             Send-ACREQ (Node-Requesting-Certificate)
        Else
            Discard the packet
    } /* Handle-CREQ (CREQ-Packet) */
```

A.5.1.2 Code Executed To Send *ACREQ* Packet

```
/* The following code is executed by a LCA to send a ACREQ packet to other LCAs in the current zone to ask
   them whether to update a given certificate or not*/

Send-ACREQ (i)
    {
    Cert-Needed [i] = True
    Send an ACREQ-Packet to each LCA in its zone using source routing

    /* Its own acceptance */
    Number-of-received-ACREP-packets [i] = 1
    } /* Send-ACREQ (i) */
```

A.5.1.3 Code Executed Upon Receiving *ACREQ* Packet

```
/* The following code is executed upon receiving a ACREQ packet */

Handle-ACREQ (ACREQ-Packet)
    {If (not duplicate packet) and (current node is not the destination)
        /* The current node is an intermediate node for a ACREQ packet */
        Source-Routing (ACREQ-Packet)

    Else
        /* The current node is the destination LCA */
        If (not duplicate packet) and (current node is the destination) and
           (Validate-Initiator (ACREQ-Packet)) and
           (Validate-Predecessor (ACREQ-Packet)) and (Validate-LCA-Initiator (ACREQ-Packet)) and
           (the initiator of the CREQ-Packet worth updating the certificate)
             Unicast ACREP-Packet through the reverse path of the ACREQ-Packet
        Else
            Discard the packet
    } /* Handle-ACREQ (ACREQ-Packet) */
```

A.5.1.4 Code Executed Upon Receiving *ACREP* Packet

```
/* The following code is executed upon receiving a ACREP packet */

Handle-ACREP (ACREP-Packet)
    {If (not duplicate packet) and (current node is not the destination)
        /* The current node is an intermediate node for a ACREP packet */
        Reverse-Path (ACREP-Packet, the predecessor from which the ACREQ-Packet was received)

    Else
        /* The current node is the destination LCA */
        If (not duplicate packet) and (current node is the destination) and
           (Validate-Initiator (ACREP-Packet)) and
           (Validate-Predecessor (ACREP-Packet)) and (Validate-LCA-Initiator (ACREP-Packet))
             Number-of-received-ACREP-packets [i] = Number-of-received-ACREP-packets [i] +1
        Else
             Discard the packet
    } /* Handle-ACREP (ACREP-Packet) */
```

A.5.1.5 Code Executed Upon Receiving *CREP* Packet

```
/* The following code is executed upon receiving a CREP packet */

Handle-CREP (CREP-Packet)
    {If (not duplicate packet) and (current node is not the destination)
        /* The current node is an intermediate node for a CREP packet */
        Reverse-Path (CREP-Packet, the predecessor from which the CREQ-Packet was received)

    Else
        /* The current node is the destination */
        If (not duplicate packet) and (current node is the destination) and
           (Validate-Initiator (CREP-Packet)) and
           (Validate-Predecessor (CREP-Packet)) and (Validate-LCA-Initiator (CREP-Packet))
             Record the updated certificate
        Else
             Discard the packet
    } /* Handle-CREP (CREP-Packet) */
```

A.5.1.6 Code Executed Upon Receiving *NCERT* Packet

```
/* The following code is executed upon receiving a NCERT packet */

Handle-NCERT (NCERT-Packet)
    {If (not duplicate packet) and (current node is not the destination)
        /* The current node is an intermediate node for a NCERT packet */
        Source-Routing (NCERT-Packet)

    Else   /* The current node is the destination LCA */
        If (not duplicate packet) and (current node is the destination) and
           (Validate-Initiator (NCERT-Packet)) and
           (Validate-Predecessor (NCERT-Packet)) and (Validate-LCA-Initiator (NCERT-Packet))
             Record the updated certificate
        Else
             Discard the packet
    } /* Handle-NCERT (NCERT-Packet) */
```

A.5.2 *LCA*s Synchronization

A.5.2.1 Code Executed Upon Receiving *CLSYN* Packet

```
/* This code is executed upon receiving a CLSYN packet initiated by one of the LCAs in the network */

Handle-CLSYN (CLSYN-Packet)
  {If (not duplicate packet) and (current node is not the destination)
     /* The current node is an intermediate node for a CLSYN packet */
     Intermediate-LCA-LCA-Communication (CLSYN-Packet)
  Else
     /* The current node is the destination LCA */
     If (not duplicate packet) and (current node is the destination) and (Validate-Initiator (CLSYN -Packet)) and
        (Validate-Predecessor (CLSYN-Packet)) and (Validate-LCA-Initiator (CLSYN-Packet))
          {If (zone of the CLSYN-Packet initiator = zone of the current node) or
              (zone of the CLSYN-Packet initiator = adjacent zone)
              {Store system current time
              /* Send the packet to all LCAs in the zone */
              Continue-LCA-LCA-Communication (CLSYN-Packet, True)}
          Else
              Discard the packet}
     Else
          Discard the packet
  } /* Handle-CLSYN (CLSYN-Packet) */
```

A.5.3 Nodes Movement and Failure

A.5.3.1 Code Executed Upon Receiving *UNPOS* Packet

```
/* The following code is executed upon receiving a UNPOS packet */

Handle-UNPOS (UNPOS-Packet)
    {If (source is non-LCA node)                    /*source is the moving node itself*/
        Handle-Node-LCA-UNPOS (UNPOS-Packet)
    Else
        Handle-LCA-LCA-UNPOS (UNPOS-Packet)
    } /* Handle-UNPOS (UNPOS-Packet)*/
```

A.5.3.2 Code Executed Upon Receiving *UNPOS* Packet Initiated by Moving Node

```
/* The following code is executed upon receiving a UNPOS packet initiated by the moving node itself */

Handle-Node-LCA-UNPOS (UNPOS-Packet)
  {If (not duplicate packet) and (current node is not the destination)
     /* The current node is an intermediate node for a UNPOS packet*/
     Restricted-Directional-Flooding (UNPOS -Packet)
  Else
     /* The current node is the nearest LCA */
     If (not duplicate packet) and (current node is the destination) and
        (Validate-Initiator (UNPOS-Packet)) and (Validate-Predecessor (UNPOS-Packet))
          {If (the node leaves to one of the neighbouring zones)  /* Whether 4-neighbouring  or D-neighbouring
          */
              {If (the node leaves to an empty zone)
                   Node-Enter-Empty-Zone ()
              Else
                   If (the adjacent LCA is within the transmission range)
                        Send a DNODE-Packet to adjacent LCA using 1-hop unicast
                   Else
                        Send a DNODE-Packet to adjacent LCA using restricted directional flooding
              Remove the node's information from own tables}
          Else
              Store new position
          Use source routing to send a UNPOS-Packet to other LCAs in its zone
          } /* If the current node is the nearest LCA */
     Else
          Discard the packet
```

A.5.3.3 Code Executed Upon Receiving *UNPOS* Packet Initiated by Nearest *LCA*

```
/* The following code is executed upon receiving a UNPOS packet initiated by the nearest LCA */

Handle-LCA-LCA-UNPOS (UNPOS-Packet)
{If (not duplicate packet) and (current node is not the destination)
    /* The current node is an intermediate node for an LCA-LCA UNPOS packet */
    Intermediate-LCA-LCA-Communication (UNPOS-Packet)

Else
     /* The current node is the destination */
    If (not duplicate packet) and (current node is the destination) and (Validate-Initiator (UNPOS -Packet)) and
      (Validate-Predecessor (UNPOS-Packet)) and (Validate-LCA-Initiator (UNPOS-Packet)) and
      (zone of the UNPOS-Packet initiator = zone of the current node)
        {If (the node leaves to one of the neighbouring zones)
            Remove the node's information from own tables
        Else
            Store new position

        } /* The current node is the destination */
    Else
        Discard the packet
} /* Handle- LCA-LCA-UNPOS (UNPOS-Packet)*/
```

A.5.3.4 Code Executed Upon Receiving *DNODE* Packet

```
/* The following code is executed upon receiving a DNODE packet */

Handle-DNODE (DNODE-Packet)
{If (not duplicate packet) and (current node is not the destination)
    /* The current node is an intermediate node for a DNODE packet */
    Intermediate-LCA-LCA-Communication (DNODE-Packet)

Else
    /* The current node is the destination LCA */
    If (not duplicate packet) and (current node is the destination) and (Validate-Initiator (DNODE-Packet)) and
      (Validate-Predecessor (DNODE-Packet)) and (Validate-LCA-Initiator (DNODE-Packet))
        {If (zone of the DNODE-Packet initiator = adjacent zone)
            {If (the new position of the node is inside this zone)
                {Store needed information
                Send an NNODE-Packet to each LCA in its zone using source routing
                Send an NZONE-Packet to the departing node using source routing}
            Else  /* The new position of the node is in the D-neighbouring zone */
                Resend a DNODE-Packet to the LCA adjacent to the new D-neighbouring zone
            } /* Packet is from adjacent zone */

        Else
            /* The packet is from the same zone */
            If (zone of the DNODE-Packet initiator = zone of the current node)
                {If (the adjacent LCA is within the transmission range of this LCA)
                    Send a DNODE-Packet to adjacent LCA using 1-hop unicast.
                Else
                    Send a DNODE-Packet to adjacent LCA using restricted directional flooding
                }
            Else
                Discard the packet
        } /* If the current node is the destination LCA */
    Else
        Discard the packet
} /* Handle-DNODE (DNODE-Packet) */
```

A.5.3.5 Code Executed Upon Receiving *NNODE* Packet

```
/* The following code is executed upon receiving a NNODE packet */

Handle-NNODE (NNODE-Packet)
  {If (not duplicate packet) and (current node is not the destination)
      /* The current node is an intermediate node for a NNODE packet */
      Source-Routing (NNODE-Packet)
  Else
      /* The current node is the destination LCA*/
      If (not duplicate packet) and (current node is the destination) and (Validate-Initiator (NNODE-Packet)) and
         (Validate-Predecessor (NNODE-Packet)) and (Validate-LCA-Initiator (NNODE-Packet))
            Record needed information about the new node
      Else
            Discard the packet
  } /* Handle-NNODE (NNODE-Packet) */
```

A.5.3.6 Code Executed Upon Receiving *NZONE* Packet

```
/* The following code is executed upon receiving a NZONE packet */

Handle-NZONE (NZONE-Packet)
  {If (not duplicate packet) and (current node is not the destination)
      /* The current node is an intermediate node for a NZONE packet */
      Source-Routing (NZONE-Packet)
  Else
      /* The current node is the departing node */
      If (not duplicate packet) and (current node is the destination) and (Validate-Initiator (NZONE-Packet)) and
         (Validate-Predecessor (NZONE-Packet)) and (Validate-LCA-Initiator (NZONE-Packet))
            Record needed information about the new zone
      Else
            Discard the packet
  } /* Handle-NZONE (NZONE-Packet)*/
```

A.5.3.7 Code Executed Upon Receiving *NLCAE* (or *FLCA*) Packet

```
/* The following code is executed upon receiving a NLCAE (or FLCA) packet */

Handle-NLCAE (NLCAE-Packet)
  {If (the current node (n) is inside the corresponding zone) and
      (not duplicate packet) and (Validate-Initiator (NLCAE-Packet)) and
       (Validate-Predecessor (NLCAE-Packet)) and (Validate-LCA-Initiator (NLCAE-Packet))
      {Continue broadcasting the NLCAE-Packet
      Let z and s be the zone number and zone side number specified in the NLCAE-Packet
      If (node is close to the zone side) and (node is moving outside the zone)
          Let probability ProbL_nzs = 0
      Else
          {Let W_D = W_S=0.25, W_B =0.2, W_C = W_M = W_F =0.1
          Let S_max and B_max be the maximum possible node movement speed and battery life time respectively
          Let C_max and M_max be the maximum possible node CPU power and memory capacity respectively
          Let D_max be the maximum possible distance between a node and the middle point of the zone side
          Assign S_n and B_n to the current node's movement speed and remaining battery life time respectively
          Assign C_n and M_n to the current node's CPU power and memory capacity respectively

          /* Distance between current node's position (x_n, y_n) and middle point of the zone side (xm_zs, ym_zs) */
          Calculate D_nzs = ((x_n – xm_zs)^2 + (y_n – ym_zs)^2)^1/2
          /* the nodes that have recently played the role of a LCA will be avoided */
          Calculate LCAF_n as the fraction of time during which it serves as a LCA within the last Ns seconds

          Calculate probability ProbL_nzs = W_D * (1- D_nzs /D_max) + W_S * (1-S_n/S_max) + W_B * (B_n/B_max) +
                                            W_C * (C_n/C_max) + W_M * (M_n/M_max) + W_F * (LCAF_n)
          Send calculated probability via NPROB-Packet, through reverse path to the corresponding LCA
          } /* Else */
      } / * If (the current node (n) is inside the corresponding zone) … */
  Else
      Discard the packet
  } /* Handle-NLCAE (NLCAE-Packet) */
```

A.5.3.8 Code Executed Upon Receiving *NPROB* Packet

```
/* The following code is executed upon receiving a NPROB packet */

Handle-NPROB (NPROB-Packet)
  {If (not duplicate packet) and (current node is not the destination)
      /* The current node is an intermediate node for a NPROB packet */
      Reverse-Path (NPROB-Packet, the predecessor from which the NLCAE-Packet was received)
  Else
      /* The current node is the destination LCA */
      If (not duplicate packet) and (current node is the destination) and (Validate-Initiator (NPROB-Packet)) and
         (Validate-Predecessor (NPROB-Packet)) and (node probability in the NPROB-Packet) > max-prob)
           Store information of this node as the new LCA
      Else
           Discard the packet
  } /* Handle-ACREP (ACREP-Packet) */
```

A.5.3.9 Code Executed Upon Electing A New *LCA*

```
/* The following code is executed upon electing a new LCA*/

Elect-New-LCA ()
    {Broadcast a NLCA-Packet so that all nodes inside that zone know the address and position of the new LCA
    Send NALCA-Packet to the adjacent LCA in the neighbouring zone
    Transfer the needed information to the new LCA
    } /* Elect-New-LCA () */
```

A.5.3.10 Code Executed To Notify Nodes Failure

```
/* To enable the failed node to join the network from any zone after repairing,
   node's IP address and public key will be sent to all LCAs in the network */

Notify-Node-Sudden-Failure (IP-Address, Public-Key)
    {Send a FNODE-Packet, using source routing, to LCAs in the current
     zone containing failed node's IP-Address and Public-Key
    }
```

A.5.3.11 Code Executed Upon Receiving *FNODE* Packet

```
/* This code is executed upon receiving a FNODE packet initiated by the LCA that had issued the last
    certificate for a failed node, as well as by the voluntary LCA in the case of LCA sudden failure */

Handle-FNODE (FNODE-Packet)
  {If (not duplicate packet) and (current node is not the destination)
      /* The current node is an intermediate node for a FNODE packet */
      Intermediate-LCA-LCA-Communication (FNODE-Packet)

  Else
      /* The current node is the destination LCA */
      If (not duplicate packet) and (current node is the destination) and (Validate-Initiator (FNODE-Packet)) and
        (Validate-Predecessor (FNODE-Packet)) and (Validate-LCA-Initiator (FNODE-Packet))
          {If (zone of the FNODE-Packet initiator = zone of the current node) or
               (zone of the FNODE-Packet initiator = adjacent zone)
               {Store needed information
               /* Send the packet to all LCAs in the zone */
               Continue-LCA-LCA-Communication (FNODE-Packet, True)}
          Else
               Discard the packet
          }
      Else
           Discard the packet
  } /* Handle-FNODE (FNODE-Packet) */
```

A.6 Authenticated Location Service

A.6.1 Code Executed Upon Receiving *PREQ* Packet

```
/* This code is executed upon receiving a PDP packet */

Handle-PDP (PDP-Packet)
    {If (source is non-LCA node)
        Handle-Internal-PDP (PDP-Packet)
    Else
        Handle-External-PDP (PDP-Packet)
    } /* Handle-PDP (PDP-Packet) */
```

A.6.2 Code Executed Upon Receiving Internal *PREQ* Packet

```
/* Before beginning the route discovery the source should know the destination's position. The source (S)
  sends an internal PDP packet to the nearest LCA in its zone using restricted directional flooding to ask the
  LCA about the position of the destination (D). This code is executed upon receiving an internal PDP packet */

Handle-Internal-PDP (PDP-Packet)
    {If (not duplicate packet) and (current node is not the destination)
        /* The current node is an intermediate node for an internal PDP packet */
        Restricted-Directional-Flooding (PDP-Packet)
    Else
        /* The current node is the destination (nearest LCA) */
        If (not duplicate packet) and (current node is the destination) and
           (Validate-Initiator (PDP-Packet)) and (Validate-Predecessor (PDP-Packet))
            {If (the destination is in the same zone of the source)
                Unicast a PREP-Packet back along the reverse path to the source of the PDP-Packet
            Else
                If (the required destination is in the non-existing list)
                  Send a packet to source informing it that required destination is not available in network now
                Else
                  Use source routing to send a PDP-Packet to other LCAs in its zone those have adjacent LCAs
            } /* The current node is the destination (nearest LCA) */
        Else
            Discard the packet
    } /* Handle-Internal-PDP (PDP-Packet) */
```

A.6.3 Code Executed Upon Receiving External *PREQ* Packet

```
/* This code is executed upon receiving an external PDP packet sent between LCAs of different zones when the
destination is not in the same zone as that of the source */

Handle-External-PDP (PDP-Packet)
    {If (not duplicate packet) and (current node is not the destination)
        /* The current node is an intermediate node for an external PDP packet */
        Intermediate-LCA-LCA-Communication (PDP-Packet)
    Else
        /* The current node is the destination */
        /*the source is a LCA node*/
        If (not duplicate packet) and (current node is the destination) and (Validate-Initiator (PDP-Packet)) and
           (Validate-Predecessor (PDP-Packet)) and (Validate-LCA-Initiator (PDP-Packet))
            {If ((zone of the PDP-Packet initiator = zone of the current node) or
                 (zone of the PDP-Packet initiator = adjacent zone))
                If (source is in this zone)
                    unicast a PREP-Packet back along the reverse path to the source of the PDP-Packet
                Else
                    /* Send only to LCAs those have adjacent LCAs */
                    Continue-LCA-LCA-Communication (PDP-Packet, False)
            } /* The current node is the destination */
        Else
            Discard the packet
    } /* Handle-External-PDP (PDP-Packet) */
```

A.6.4 Code Executed Upon Receiving *PREP* Packet

```
/* The following code is executed upon receiving a PREP packet */

Handle-PREP (PREP-Packet)
    {Let T_rd be a pre-defined value indicating route discovery time

    If (not duplicate packet) and (current node is not the destination)
        /* The current node is an intermediate node for a PREP packet */
        Reverse-Path (PREP-Packet, the predecessor from which the PDP-Packet was received)
    Else
        /* The current node is the destination */
        If (not duplicate packet) and (current node is the destination) and (Validate-Initiator (PREP-Packet)) and
           (Validate-Predecessor (PREP-Packet)) and (Validate-LCA-Initiator (PREP-Packet))
            /* Use the position of the destination to begin route instantiation */
            {Pos-Needed = False
             Pos-known = True
            Route-Needed = True
            RDP-No=0
            T_rd-Elapsed = True}
        Else
            Discard the packet
    } /* Handle-PREP (PREP-Packet) */
```

A.7 Route Discovery, Setup and Maintenance

A.7.1 Code Executed Upon Receiving *RDP* Packet

```
/* The source begins route instantiation to destination by sending a RDP packet. This is done using
   restricted directional flooding. The following code is executed upon receiving a RDP packet */

Handle-RDP (RDP-Packet)
    {If (not duplicate packet) and (current node is not the destination)
        /* The current node is an intermediate node for a RDP packet */
        Restricted-Directional-Flooding (RDP-Packet)
    Else
        /* The current node is the destination */
        If (not duplicate packet) and (current node is the destination) and
           (Validate-Initiator (RDP-Packet)) and (Validate-Predecessor (RDP-Packet))
            unicast a RREP-Packet back along the reverse path to the source of the RDP-Packet
        Else
            Discard the packet
    } /* Handle-RDP (RDP-Packet) */
```

A.7.2 Code Executed Upon Receiving *RREP* Packet

```
/* The following code is executed upon receiving a RREP packet */

Handle-RREP (RREP-Packet)
    {If (not duplicate packet) and (current node is not the destination)
        /* The current node is an intermediate node for a RREP packet */
        {Reverse-Path (RREP-Packet, the predecessor from which the RDP-Packet was received)
        Add route to routing table
        }
    Else
        /* The current node is the destination */
        If (not duplicate packet) and (current node is the destination) and
           (Validate-Initiator (RREP-Packet)) and (Validate-Predecessor (RREP-Packet))
            {Add route to routing table
            Route-Needed = False
            Send all DATA-Packets stored in Buffer
            }
        Else
            Discard the packet
    } /* Handle-RREP (RREP-Packet) */
```

A.7.3 Code Executed Upon Receiving *ERR* Packet

```
/* The following code is executed upon receiving an ERR packet */

Handle-ERR (ERR-Packet)
    {Let T_rd be a pre-defined value indicating route discovery time

    If (not duplicate packet) and (current node is not the source of the route)
        /* The current node is an intermediate node for a ERR packet */
        {Delete route from routing table
        Reverse-Path (ERR-Packet, the predecessor from which the RDP-Packet was received)
        }
    Else
        /* The current node is the source of the route */
        If (not duplicate packet) and (current node is the source of the route) and
           (Validate-Initiator (ERR-Packet)) and (Validate-Predecessor (ERR-Packet))
             {If (Route-Needed = False)
                  {Delete route from routing table

                   /* Begin a new route instantiation */
                  Route-Needed = True
                  RDP-No=0
                  T_rd-Elapsed = True}
             }
        Else
             Discard the packet
    } /* Handle-ERR (ERR-Packet) */
```

A.8 Data Transmission

A.8.1 Code Executed Upon Receiving *DATA* Packet

```
/* The following code is executed upon receiving a DATA packet */

Handle-DATA (DATA-Packet)
    {If (current node is not the destination)
        /* The current node is an intermediate node for a DATA packet */
        {If (route exists in the routing table)
             Send DATA-Packet to next-hop in the route
        Else
             Send an ERR-Packet towards the source of the route
        }
    Else
        /* The current node is the destination */
        Store the DATA-Packet
    } /* Handle-RDP (RDP-Packet) */
```

Appendix B

Introduction to GloMoSim*

This appendix presents some important files in *GloMoSim* simulator along with the supported nodes placements and mobility models. After that the needed steps to add a new routing protocol to *GloMoSim* simulator are discussed.

B.1 Structure of *GloMoSim* Directories

After installing the *GloMoSim* software, the *GloMoSim* directory contains the following subdirectories:

Table B.1: Structure of *GloMoSim* directories.

Directory	Content
/main	Contains the header and source files for node placements, mobility models, queuing functions used for buffering messages, message sending and retrieval as well as reading parameters from the input file.
/bin	Contains the executable and other configuration files such as config.in, app.config, mobility.in, nodes.in, glomo.state and routes.in. The contents of these files are explained in the following three sections.
/include	Contains the header files for defining the node structure (this structure includes all the information related to a particular node), message structure (generic message received/sent by any layer in *GloMoSim*), supported MAC layers, transport layer protocols, network routing protocols and radio types.
/application /transport /network /mac /radio	Contains the header and source files that define the protocols supported by the specific layer.
/java_gui	Contains the visualization tool files.
/scenarios	Contains sample scenarios.

B.2 *GloMoSim* Configuration File (config.in)

Before running the *GloMoSim*, there are many important parameters that must be specified in the configuration file (config.in). The following table lists some of these parameters.

* Information in this appendix is obtained by browsing *GloMoSim* directories and documentation files.

Table B.2: Important parameters of the config.in file.

Parameter	Description
SIMULATION-TIME	The maximum time elapsed during simulation (in nano-seconds, milli-seconds, seconds, minutes, hours or days).
SEED	Random number seed used to initialize part of the seed of various randomly generated numbers in the simulation. It can be used to vary the seed of the simulation to see the consistency of the results of the simulation.
TERRAIN-DIMENSIONS	The physical area in which the nodes are being simulated (in meter × meter).
POWER-RANGE	The transmission range of the nodes (in meters).
NUMBER-OF-NODES	The number of nodes being simulated.
NODE-PLACEMENT	The way the nodes are placed in the terrain, for example RANDOM, UNIFORM, GRID or nodes positions are read from NODE-PLACEMENT-FILE. These nodes placements are explained in *Section B.5*.
MOBILITY	Movement of nodes. Can be NONE, RANDOM-WAYPOINT or by TRACE. The details of these mobility models are given in *Section B.6*.
RADIO-FREQUENCY	Radio frequency in heltz.
RADIO-BANDWIDTH	Radio bandwidth in bits per second.
MAC-PROTOCOL	Defines the MAC layer used (802.11, MACA, CSMA and TSMA layers are supported).
NETWORK-PROTOCOL	Defines the network protocol to be used (the only choice available currently is IP).
ROUTING-PROTOCOL	The routing protocol to be used (the supported protocols are Bellman-Ford, AODV, DSR, LAR, WRP, Fisheye, ZRP and static route). With the possibility to add new user-defined routing protocols.
APP-CONFIG-FILE	Setup applications such as FTP, TELNET, FTP/GENERIC, CBR and HTTP. The file will need to contain parameters that will be used to determine connections and other characteristics of the particular application.
Parameters for reporting statistics (YES/NO selection)	App, Tcp, Udp, Routing, Network, Mac, Radio, Channel and Mobility.
GUI-OPTION	To enable visualizing the topology and simulation of the model.

B.3 *GloMoSim* Application File (app.conf)

The traffic generators currently available in *GloMoSim* are File Transfer Protocol (FTP), FTP/GENERIC, TELNET, Constant Bit Rate (CBR) and HTTP. The user must specify the needed traffic generator and configure its parameters in the app.conf file.

- FTP: FTP uses tcplib to simulate the file transfer protocol. In order to use FTP, the following format is needed:

  FTP <source> <destination> <number of items to be sent> <start time>

  For example "FTP 0 1 10 0S" means node 0 sends node 1 ten items at the start of the simulation, with the size of each item randomly determined by tcplib.

- FTP/GENERIC: FTP/GENERIC does not use tcplib to simulate file transfer. Instead, the client simply sends the data items to the server without the server sending any control information back to the client. In order to use FTP/GENERIC, the following format is needed:

  FTP/GENERIC <source> <destination> < number of items to be sent>

  <item size> <start time> <end time>

  For example "FTP/GENERIC 0 1 10 1460 0S 600S" means node 0 sends node 1 ten items of 1460B each at the start of the simulation up to 600 seconds into the simulation. If the ten items are sent before 600 seconds elapsed, no other items are sent.

- TELNET: TELNET uses tcplib to simulate the telnet protocol. In order to use TELNET, the following format is needed:

  TELNET <source> <destination> <session duration> <start time>

  "TELNET 0 1 100S 0S" means node 0 sends node 1 telnet traffic for a duration of 100 seconds at the start of the simulation.

- CBR: CBR simulates a constant bit rate generator. Its format is:

  CBR <source> <destination> <number of items to be sent> <item size>

  <interval between the items > <start time> <end time>

  For example "CBR 0 1 10 1460 1S 0S 600S" means that node 0 sends node 1 ten items of 1460B each at the start of the simulation up to 600 seconds into the

simulation. The interdeparture time for each item is 1 second. If the ten items are sent before 600 seconds elapsed, no other items are sent.

- HTTP: HTTP simulates single-TCP connection web servers and clients. The following format describes its use for servers:

  HTTPD <address of a node which will be serving Web pages >

  For HTTP clients, the following format is used:

  HTTP <address of the node on which this client resides > <num_of_server>

  <server_1> ... <server_n> <start> <thresh>

  Example:

  HTTPD 2

  HTTPD 5

  HTTPD 8

  HTTPD 11

  HTTP 1 3 2 5 11 10S 120S

  There are HTTP servers on nodes 2, 5, 8 and 11. There is an HTTP client on node 1. This client chooses between servers {2, 5, 11} only when requesting web pages. It begins browsing after 10 seconds of simulation time have passed, and will "think" (remain idle) for at most 2 minutes of simulation time, at a time.

B.4 Other Important Files (mobility.in, nodes.in, routes.in and glomo.state)

The following are some configuration files that the config.in file gets parameters from:

- Mobility.in: Used if a special movement is needed (other than available mobility models), this is done by specifying the time and the destination of each movement for each moving node.
- Nodes.in: Used if a specific initial node placement is desired (other than available node placements), this is done by specifying the *x*, *y*, and *z* coordinates for each node.
- Routes.in: Used for static routing if needed.

Glomo.state is produced by the simulation; *GloMoSim* writes the output of each execution to this file. It contains the statistics gathered from different layers for each node.

B.5 Available Nodes Placement Strategies

Nodes placement strategy represents the way the nodes are placed in the terrain. The nodes placement strategies available in *GloMoSim* are:

- RANDOM: Nodes are placed randomly within the physical terrain.
- UNIFORM: Based on the number of nodes in the simulation, the physical terrain is divided into a number of cells. Within each cell, a node is placed randomly.
- GRID: Node placement starts at (0, 0) and are placed in grid format with each node GRID-UNIT away from its neighbors. The number of nodes has to be square of an integer.
- FILE: Position of nodes is read from NODE-PLACEMENT-FILE.

B.6 Available Nodes Mobility Models

Movement of the nodes in *GloMoSim* can be NONE, RANDOM-WAYPOINT, or by TRACE.

- NONE: There is no movement of nodes in the model
- RANDOM-WAYPOINT: A node randomly selects a destination from the physical terrain. It moves in the direction of the destination in a speed uniformly chosen between MOBILITY-WP-MIN-SPEED and MOBILITY-WP-MAX-SPEED (meter/sec). After it reaches its destination, the node stays there for MOBILITY-WP-PAUSE time period.
- TRACE: Movement information is read from MOBILITY-TRACE-FILE.

B.7 Main Components of a New Routing Protocol

Assume that we want to design the *ARANz* routing protocol. We should add two new files to the network directory; the ARANz.H and ARANz.PC. ARANz.H file contains functions prototypes and variables definitions. These variables may include *RDP* fields, *RREP* fields, *ERR* fields as well as needed variables to collect statistics during the execution. ARANz.PC file contains functions definitions. There are four main functions that should be found in any routing protocol:

- Initialization function: Called at the beginning of the simulation to initialize the variables such as statistics, route table and *RDP* seen table.
- Packet handler function: Called when a packet is received from *MAC* layer. For example, it may contain calling to functions that handle *RDP*, *RREP* and *ERR* packets.
- Event handler function: Handles all the protocol events, such as removing an entry from the *RDP* seen table, removing the route that has not been used for awhile, and checking if *RREP* is received after sending *RDP*.
- Finalization function: Called at the end of the simulation to collect results and prints statistics collected during the execution to glomo.state file.

B.8 Adding a New Routing Protocol to *GloMoSim*

Assume that we have designed the *ARANz* routing protocol (ARANz.PC and ARANz.H) and stored it in the network directory. Also we have the variables routingProtocolChoice = ROUTING_PROTOCOL_ARANz and protocolString = ARANz. In addition, the user_nwip.pc contains the variables routingProtocolString = protocolString and routingProtocol = routingProtocolData. In order to inform *GloMoSim* about the new protocol, the following steps must be done:

- Adding the following line to nwcommon.h file:

```
/* protocol number for IP */
#define IPPROTO_FISHEYE 530
#define IPPROTO_AODV 123
#define IPPROTO_DSR 135
#define IPPROTO_ODMRP 145
#define IPPROTO_LAR1 110
#define IPPROTO_ZRP 133
#define IPPROTO_ARANz 188          ← HERE
#define NETWORK_UNREACHABLE -2
```

Figure B.1: Modifications in nwcommon.h file upon adding a new routing protocol.

- Adding the following line to network.h file:

```
typedef enum {
NETWORK_PROTOCOL_IP = 0,
ROUTING_PROTOCOL_AODV,
ROUTING_PROTOCOL_DSR,
ROUTING_PROTOCOL_LAR1,
ROUTING_PROTOCOL_ODMRP,
ROUTING_PROTOCOL_OSPF,
ROUTING_PROTOCOL_ZRP,
ROUTING_PROTOCOL_ALL,
ROUTING_PROTOCOL_ARANz,     ← HERE
ROUTING_PROTOCOL_NONE
} NetworkRoutingProtocolType;
```

Figure B.2: Modifications in network.h file upon adding a new routing protocol.

- Making the following changes on NetworkIpUserProtocolInit (…) function which is found in user_nwip.pc file:

```
#include "network.h"
#include "ip.h"
#include "nwip.h"
// The Ip Protocol is 0..255, and must be different than the GloMoSim values for IP protocols.
//"IPPROTO_" is the current prefix for these numbers in GloMoSim.
#define IPPROTO_ARANz 188                                                    ←HERE
// The GLOMOSIM protocol number is a "unsigned short" that must be unique within the layers.
   So pick a random one to get near zero probability of conflict.
#define ROUTING_PROTOCOL_ARANz 188                                           ←HERE
void NetworkIpUserProtocolInit(GlomoNode *node, const GlomoNodeInput *nodeInput,
const char* routingProtocolString,NetworkRoutingProtocolType* routingProtocolChoice, void**
routingProtocolData)
 {if (strcmp(routingProtocolString, "ARANz") == 0) {                         ←HERE
   *routingProtocolChoice = ROUTING_PROTOCOL_ARANz;                          ←HERE
   // Put function to create your datastructures and initialize your protocol here. For Example,
  // YourRoutingProtocolInit(node, nodeInput, (YourType**)routingProtocolData);
RoutingARANzlInit(node, nodeInput, (GlomoRoutingARANz**)routingProtocolData); ←HERE
  }
else {fprintf(stderr, "CONFIG.IN Error: Unknown Routing ProtocolType " "Type %s.\n",
         routingProtocolString);
        assert(FALSE); abort();  }
}
```

Figure B.3: Modifications in NetworkIpUserProtocolInit (…) function upon adding a new routing protocol.

- Performing the following modifications on NetworkIpUserHandleProtocolEvent (…) function which is found in user_nwip.pc file:

```
void NetworkIpUserHandleProtocolEvent(GlomoNode* node, Message* msg) {
   switch (msg->protocolType) {
   case ROUTING_PROTOCOL_ARANz: {                    ←HERE
   // Put function to handle GlomoSim Events for your protocol here.
   RoutingAARANzProtocolEventHandler(node, msg); }          ←HERE
  default: printf("GloMoSim: Network IP Protocol %d Not Handled\n", msg->protocolType);
          assert(FALSE); abort(); }/*switch*/
}
```

Figure B.4: Modifications in NetworkIpUserHandleProtocolEvent (…) function upon adding a new routing protocol.

- Making the following alterations on NetworkIpUserHandleProtocolPacket (…) function which is found in user_nwip.pc file:

```
void NetworkIpUserHandleProtocolPacket( GlomoNode* node, Message* msg,
unsigned char ipProtocol, NODE_ADDR sourceAddress, NODE_ADDR destinationAddress, int ttl)
{switch (ipProtocol) {
case IPPROTO_ARANz: {                                                    ←HERE
// Put function call to handle Ip protocol packets (with IP header already stripped).
// For Example YourIpPacketHandlingFunction(node, msg, sourceAddress, destinationAddress, ttl);
RoutingAranzHandleProtocolPacket(GlomoNode *node, Message *msg,          ←HERE
NODE_ADDR srcAddr, NODE-ADDR destAddr, int ttl);
break;}
default:  printf("Ip Protocol %d Not Handled\n", ipProtocol);
          assert(FALSE); abort();
}/*switch*/
}
```

Figure B.5: Modifications in NetworkIpUserHandleProtocolPacket (…) function upon adding a new routing protocol.

- Carrying out the following adjustments on NetworkIpUserProtocolFinalize (…) function which is found in user_nwip.pc file:

```
void NetworkIpUserProtocolFinalize( GlomoNode* node, int
userProtocolNumber)
{
switch (userProtocolNumber) {
case ROUTING_PROTOCOL_ARANz: {                          ←HERE
// Put function call to handle routing protocol finalization here:
// For Example YourFinalizationRoutine(node);
RoutingARANzFinalize(GlomoNode *node);                  ←HERE
}
default: printf("Glomosim Protocol %d Not Handled\n", userProtocolNumber);
          assert(FALSE); abort();
}/*switch*/
}
```

Figure B.6: Modifications in NetworkIpUserProtocolFinalize (…) function upon adding a new routing protocol.

- Defining the new protocol in config.in file.

```
ROUTING-PROTOCOL ARANz          ←HERE
#ROUTING-PROTOCOL BELLMANFORD
#ROUTING-PROTOCOL AODV
#ROUTING-PROTOCOL DSR
#ROUTING-PROTOCOL LAR1
#ROUTING-PROTOCOL WRP
#ROUTING-PROTOCOL FISHEYE
#ROUTING-PROTOCOL ZRP
#ZONE-RADIUS 2
```

Figure B.7: Defining a new routing protocol in config.in file.

After performing these steps, the new protocol *ARANz* can be tested by running the "c:\glomosim\bin\glomosim config.in" command.

B.9 *GUI* Visualization Tool

GUI is a Java-based visualization tool for wireless networks. It provides graphical simulations and helps to see node placement and packet flow. So users can use *GloMoSim* to simulate the model while use GUI to visualize the topology and simulation. To use GUI, the GUI-OPTION in the config.in file must be switch to 'YES'. After that user should run *GloMoSim* inside the visualization tool that can be found in java_gui directory. Then write a trace file and play it back.

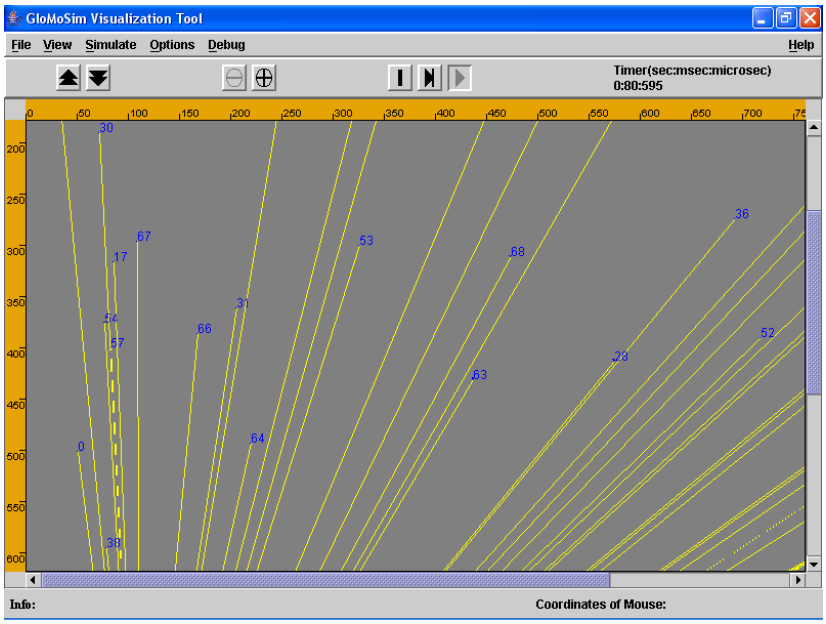


Figure B.8: GUI visualization tool.

Appendix C

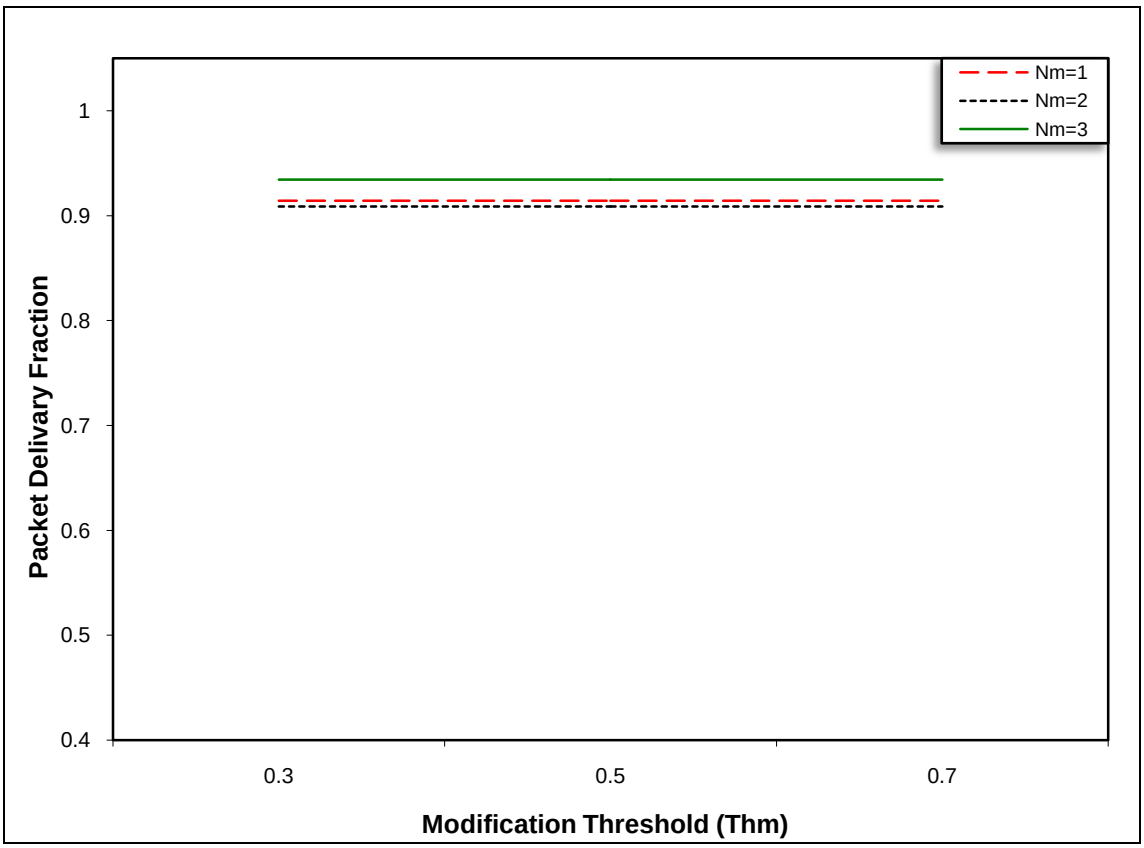
Specifying Constant Parameters Values for Studying Malicious Nodes Percentage Effect

In order to choose best values for number of *MNODE* packets that should be received by *LCA*s in a particular zone to consider a specific node as compromised one (*Nm*), modification threshold (*Thm*), dropping threshold (*Thd*) and fabrication threshold (*Thf*) some initial experiments have been conducted. A 2km×2km network that contains 240 nodes and divided into 4 zones is considered. Nodes are moving with a speed of 5m/s. Five CBR sessions were simulated in each run three of them are local and two are external. Simulations are run with 20% malicious node percentages.

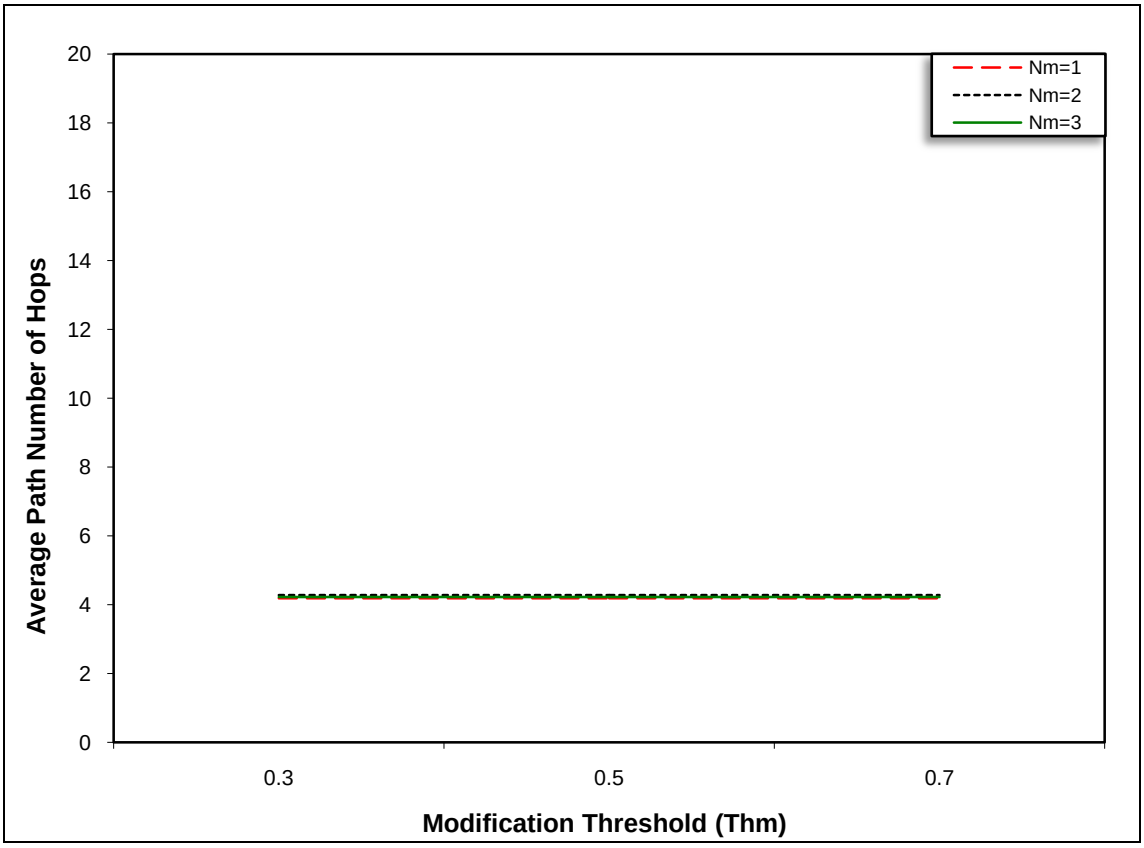
C.1 Specifying Constant Parameters for Studying Modification Attackers Effect

The first experiment is conducted to study the impact of *Nm* and *Thm* on discovering modification attackers. Simulations are run with setting *Nm* to 1, 2 and 3 also different values for *Thm* are considered which are 0.3, 0.5 and 0.7. Results are shown in Figure C.1. It is clear from the figure that, roughly speaking, none of the metrics is affected by changing *Thm* value.

Moreover, the lowest *MPR* and highest *CNP* are obtained with *Nm*=1 meaning that higher number of malicious nodes are discovered and identified as compromised while having minimum *PML* and *BML*. On the other hand, upon setting *Nm* to 2 and 3 nodes keep sending *MNODE* packets resulting in higher *PML* and *BML* while still not recognizing the malicious nodes as compromised, i.e., increased *MPR* and reduced *CNP*. Discovering higher number of malicious nodes results in a little bit decrease in *PDF* and increase in *PRL*, *BRL* and *ARL*. However these changes are outweighed with increasing the security. Hence values of *Nm* and *Thm* are set to 1 and 0.5 respectively upon simulating scenarios 1 and 5.

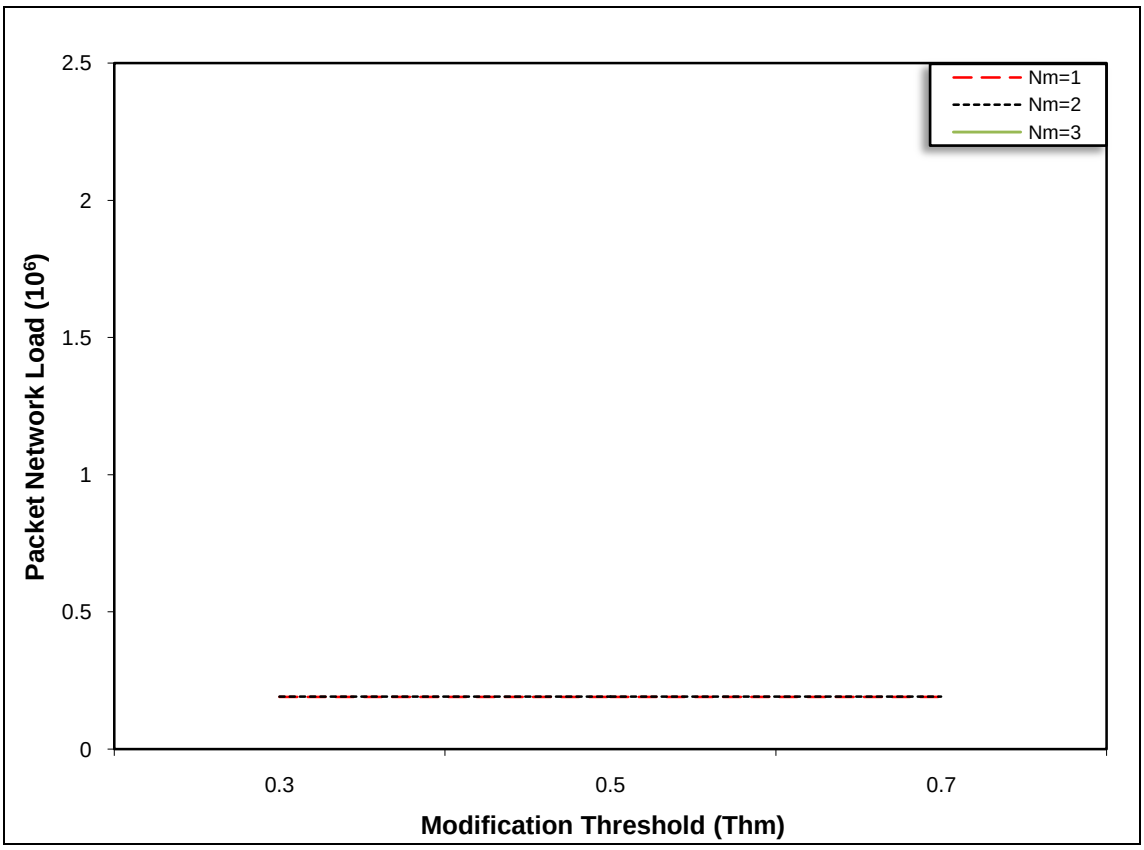


(a) Packet delivery fraction.

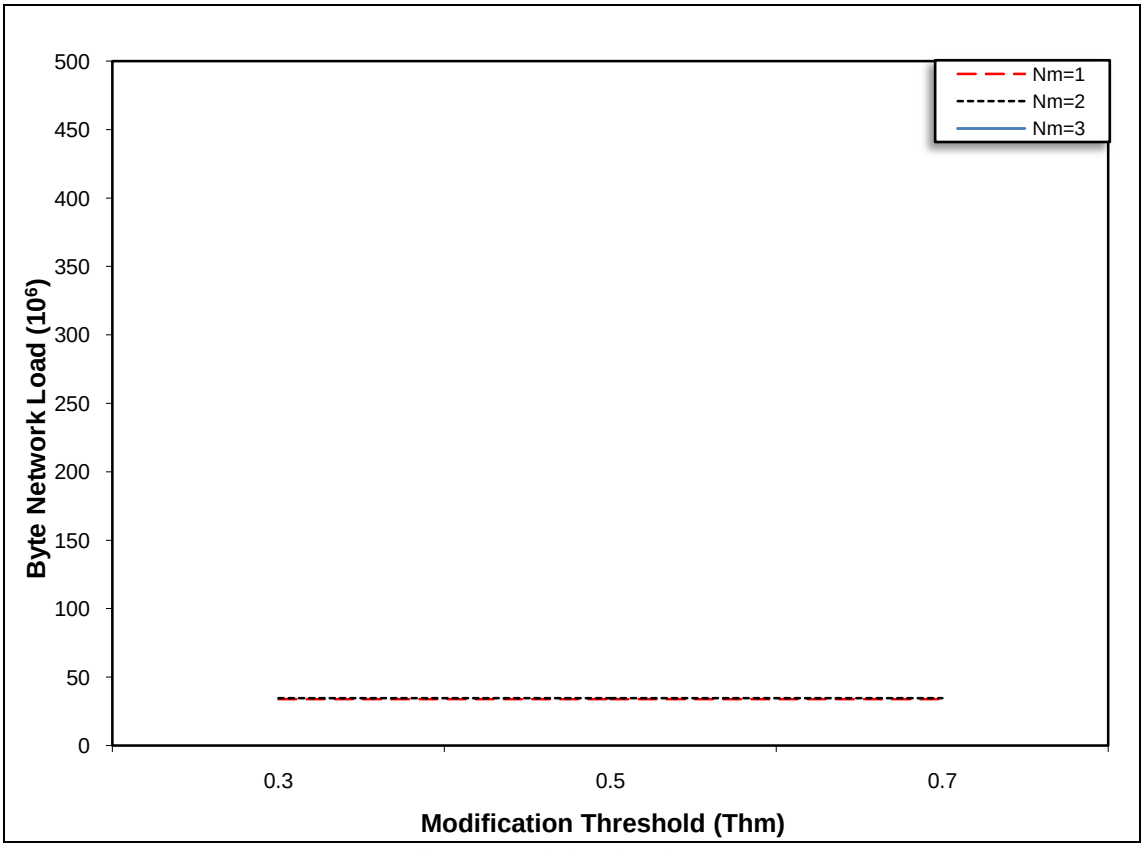


(b) Average path number of hops.

Figure C.1: Impact of *Nm* and *Thm* on discovering modification attackers.

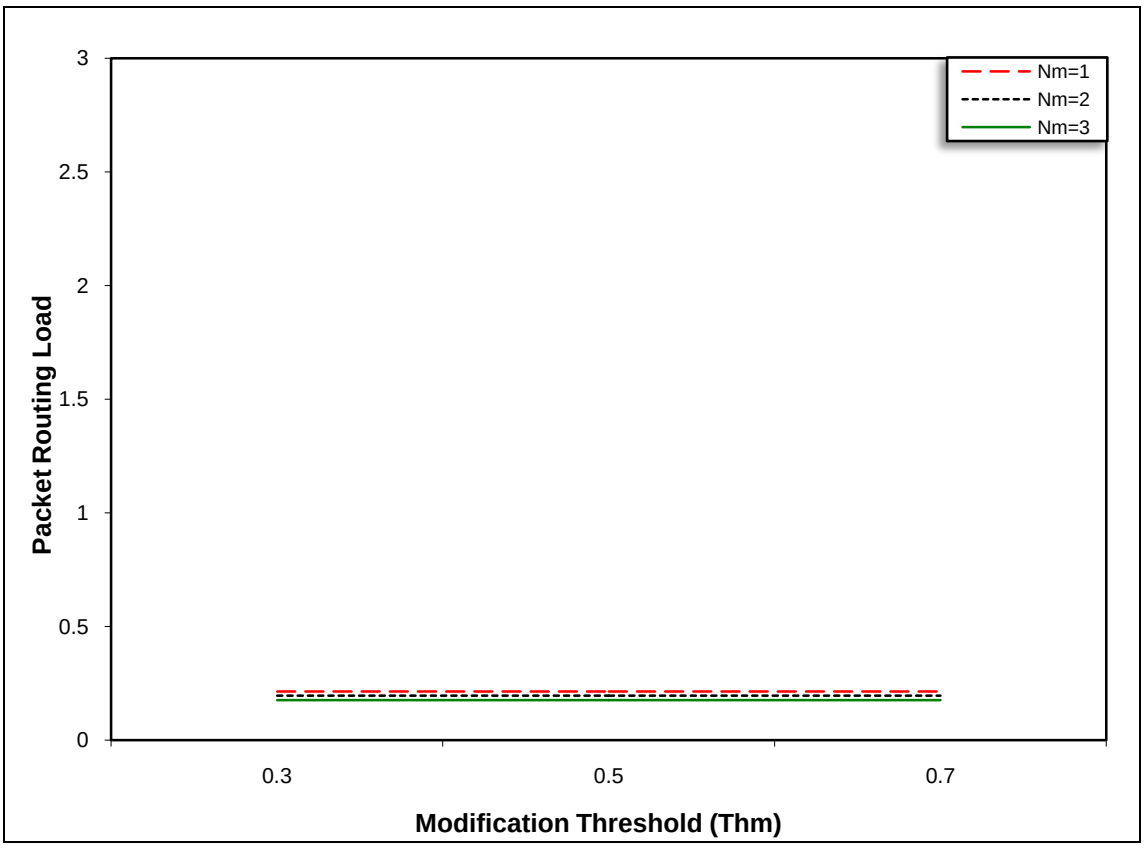


(c) Network load in packets.

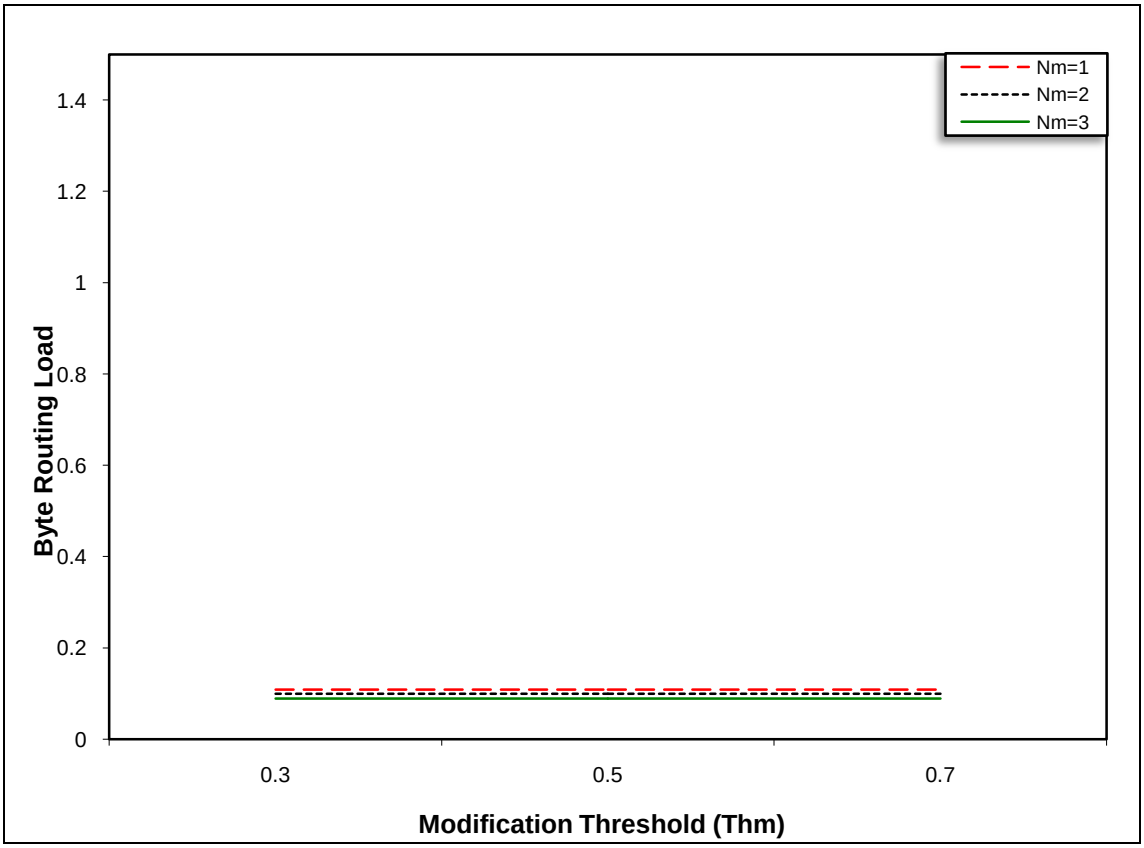


(d) Network load in bytes.

Figure C.1: Impact of *Nm* and *Thm* on discovering modification attackers.

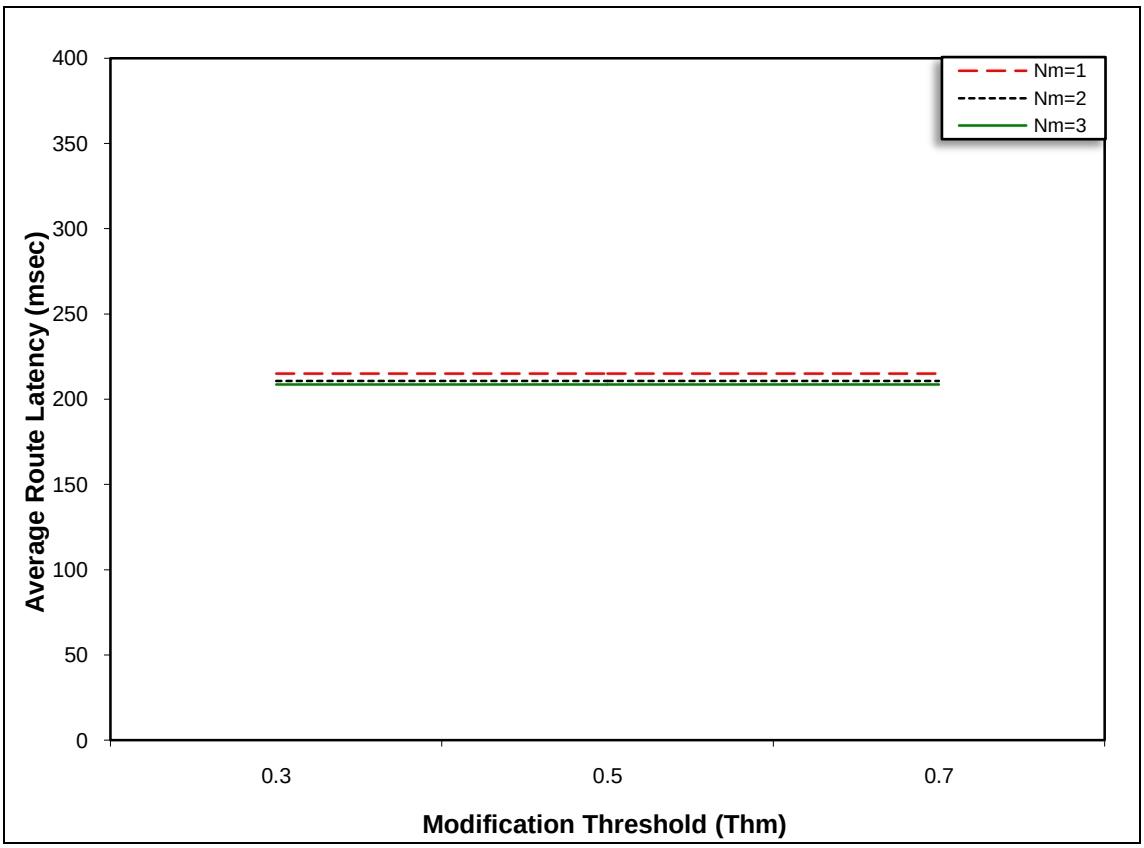


(e) Routing load in packets.

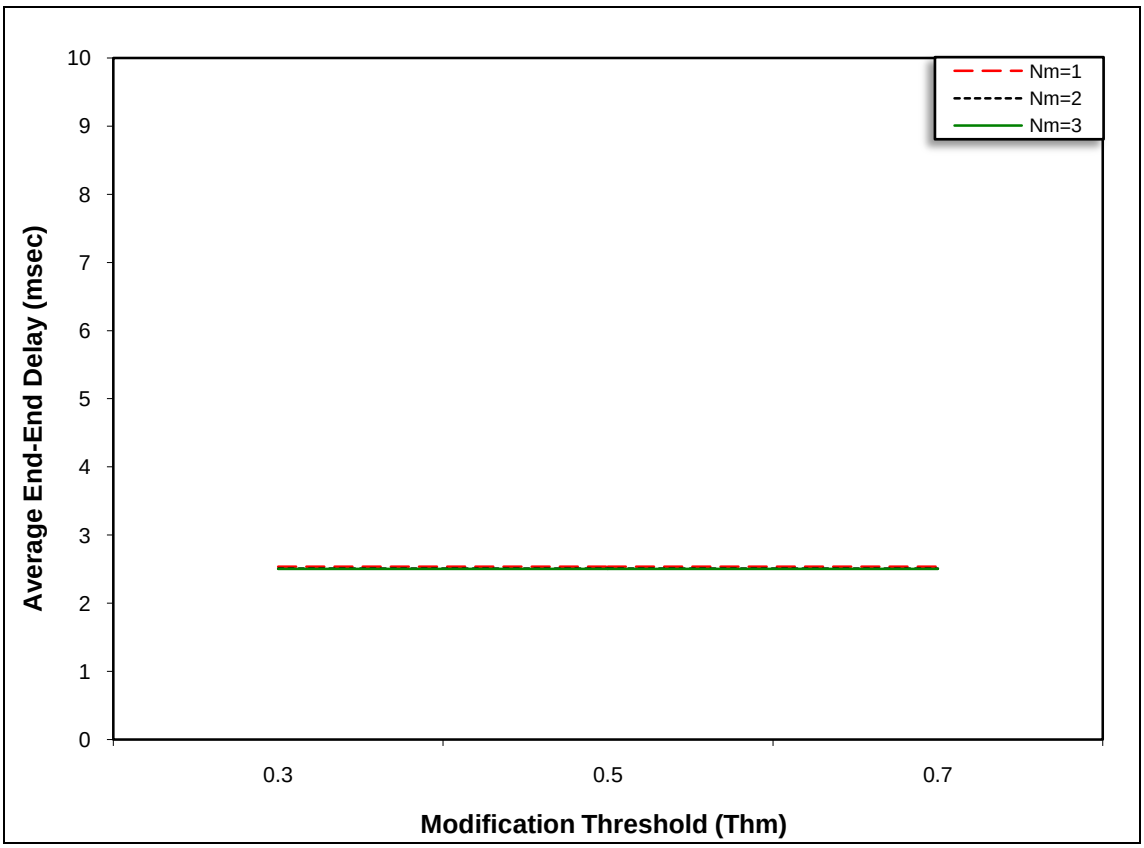


(f) Routing load in bytes.

Figure C.1: Impact of *Nm* and *Thm* on discovering modification attackers.

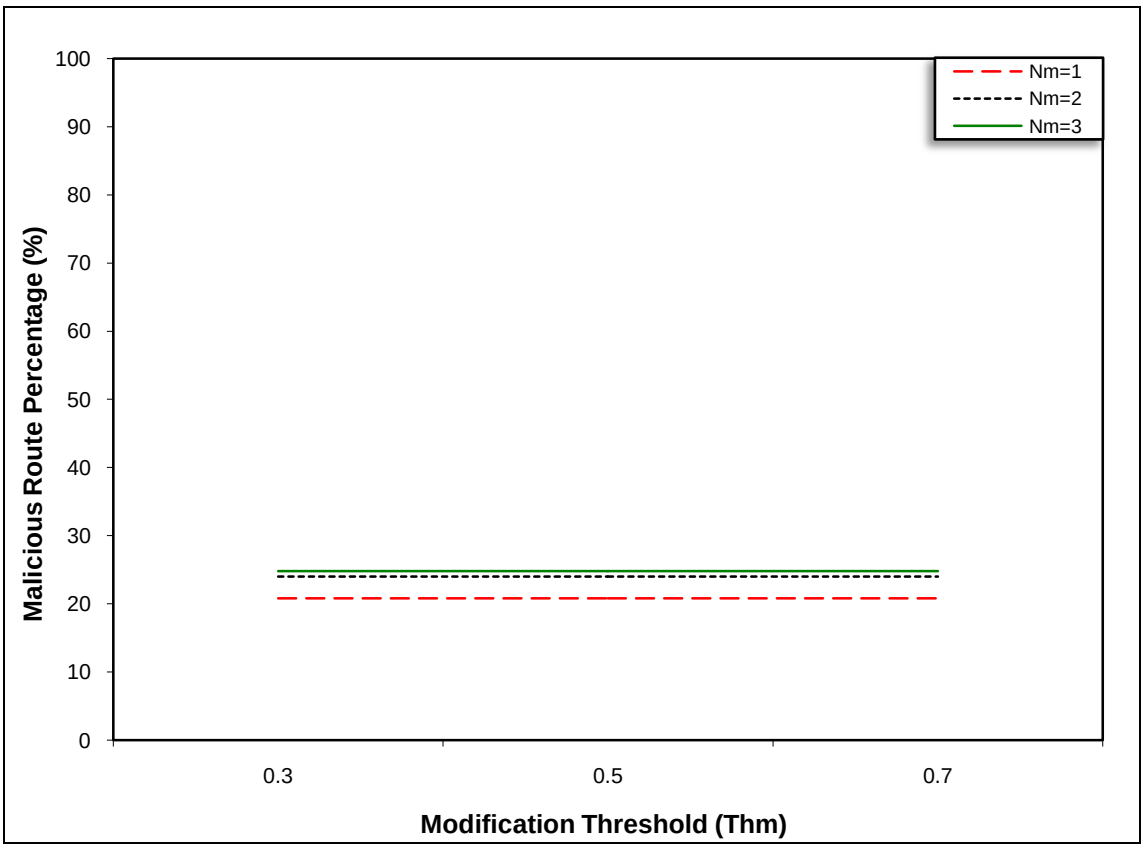


(g) Average route acquisition latency.

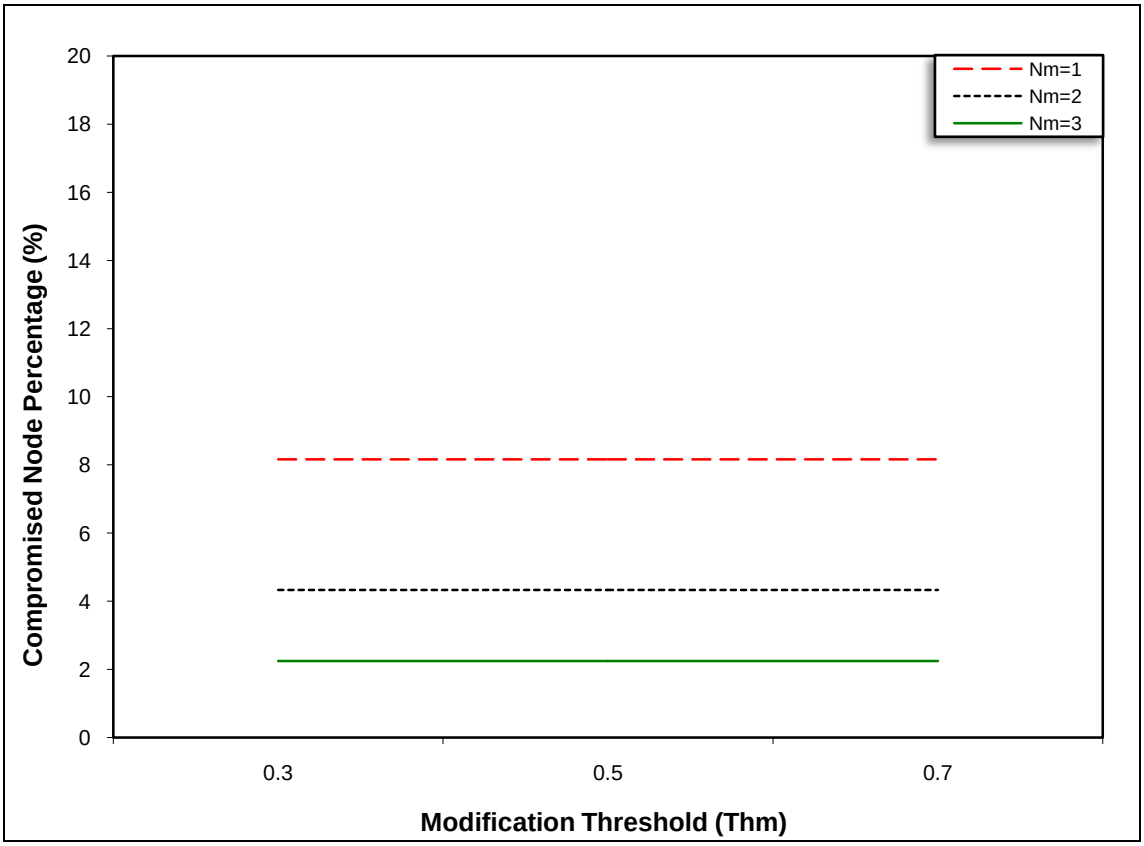


(h) Average end-to-end delay of data packets.

Figure C.1: Impact of *Nm* and *Thm* on discovering modification attackers.

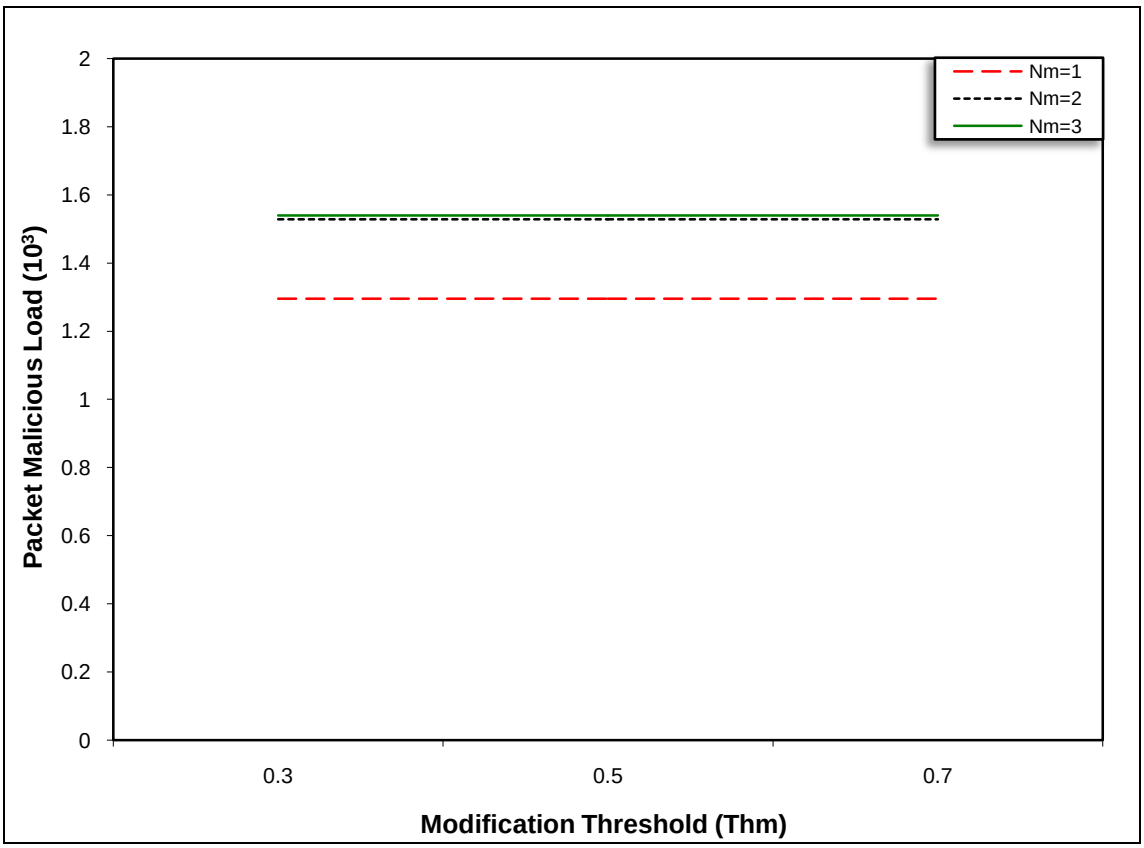


(i) Malicious route percentage.

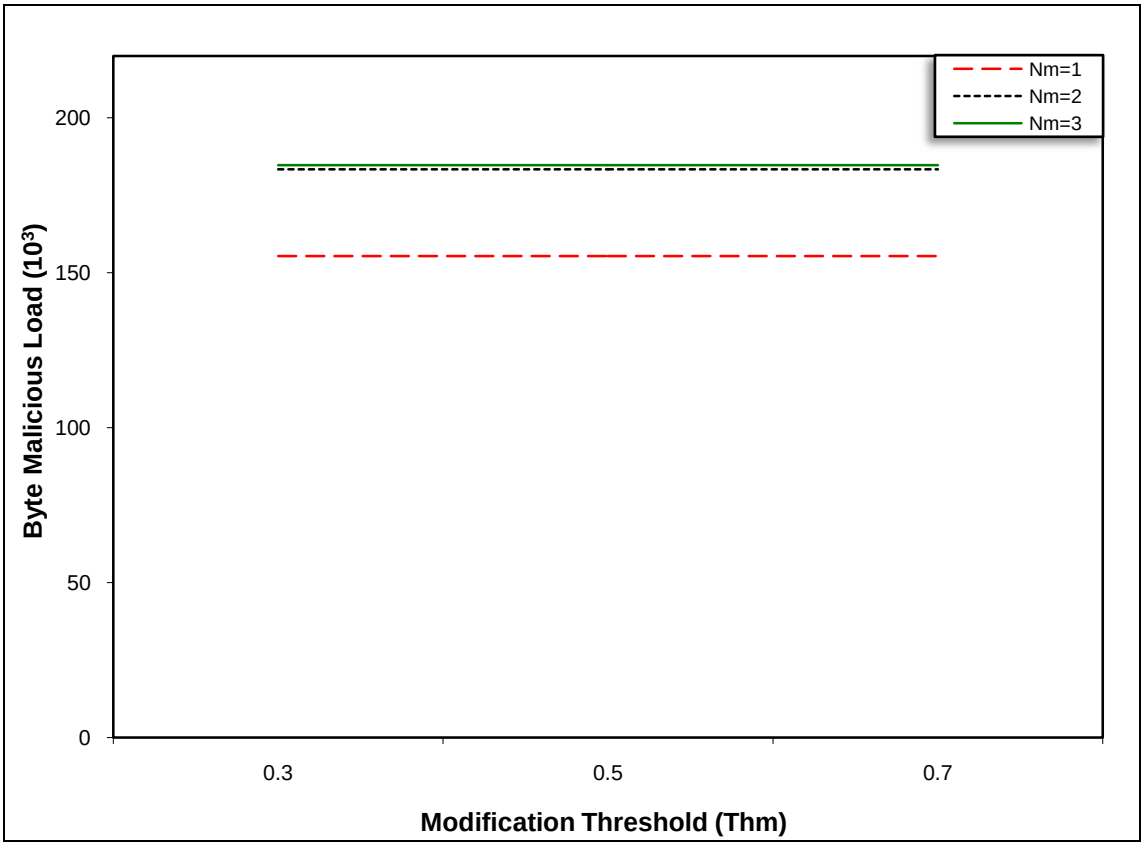


(j) Compromised node percentage.

Figure C.1: Impact of *Nm* and *Thm* on discovering modification attackers.



(k) Malicious load in packets.



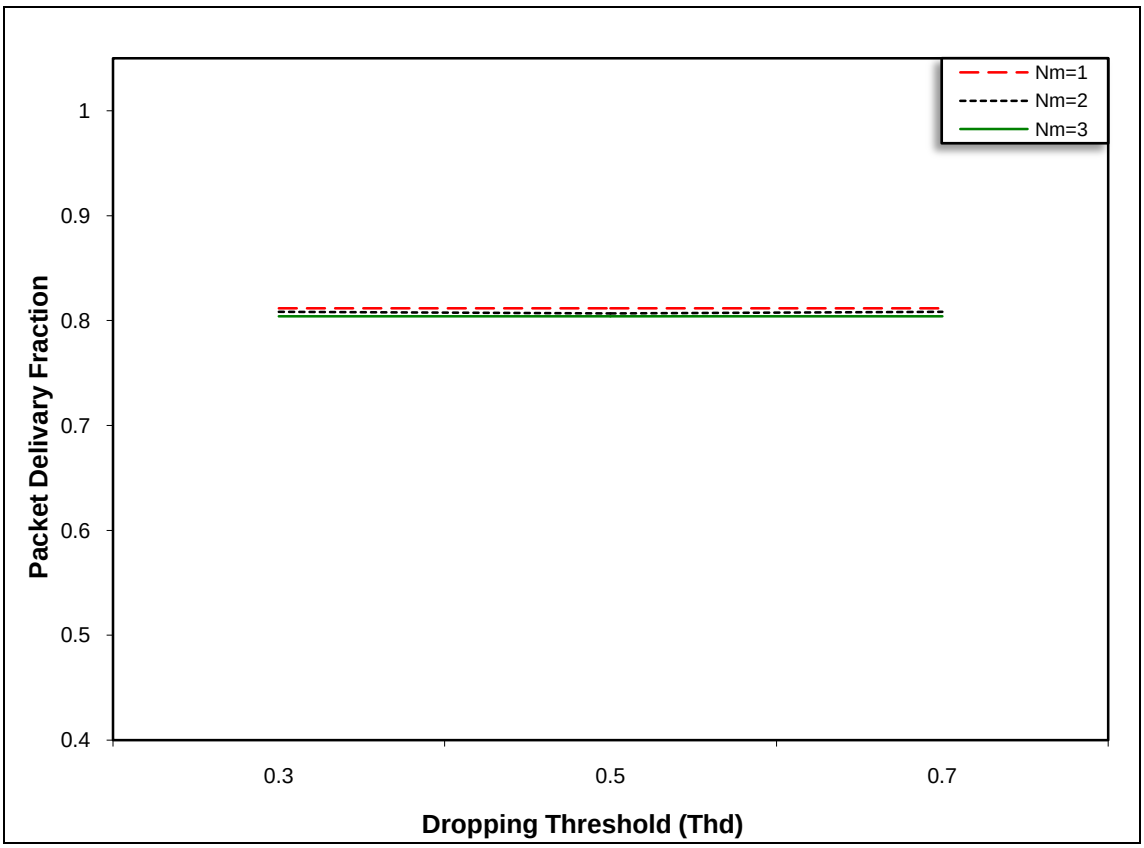
(l) Malicious load in bytes.

Figure C.1: Impact of *Nm* and *Thm* on discovering modification attackers.

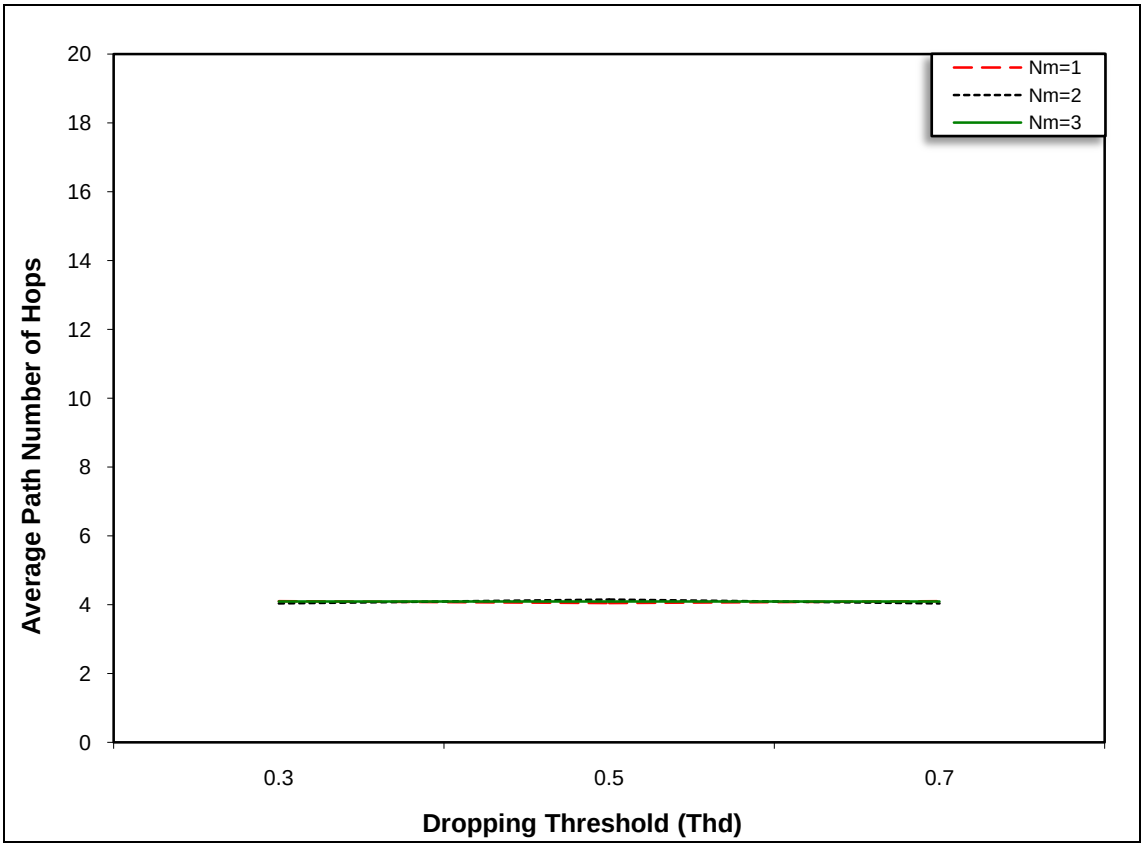
C.2 Specifying Constant Parameters for Studying Black Hole Attackers Effect

The second experiment studies the impact of *Nm* and *Thd* values on discovering black hole attackers. Simulations are run with assigning *Nm* to 1, 2 and 3 along with setting *Thd* to 0.3, 0.5 and 0.7. Figure C.2 shows that the studied metrics are roughly not affected by changing the value of *Thd*.

The minimum *PLP* and maximum *CNP* are obtained upon setting *Nm* to 1; indicating that larger number of black hole attackers are discovered and isolated from the network resulting in minimizing *PML* and *BML*. On the other hand, upon setting *Nm* to 2 or 3 nodes keep sending *MNODE* packets resulting in higher *PML* and *BML* while not distinguishing the malicious nodes, i.e., higher *PLP* and lower *CNP*. Discovering higher number of malicious nodes results in increasing *PRL* and *BRL* with keeping the minimum *ARL* and maximum *PDF*. Hence values of *Nm* and *Thm* are set to 1 and 0.5 respectively upon simulating scenario 2.

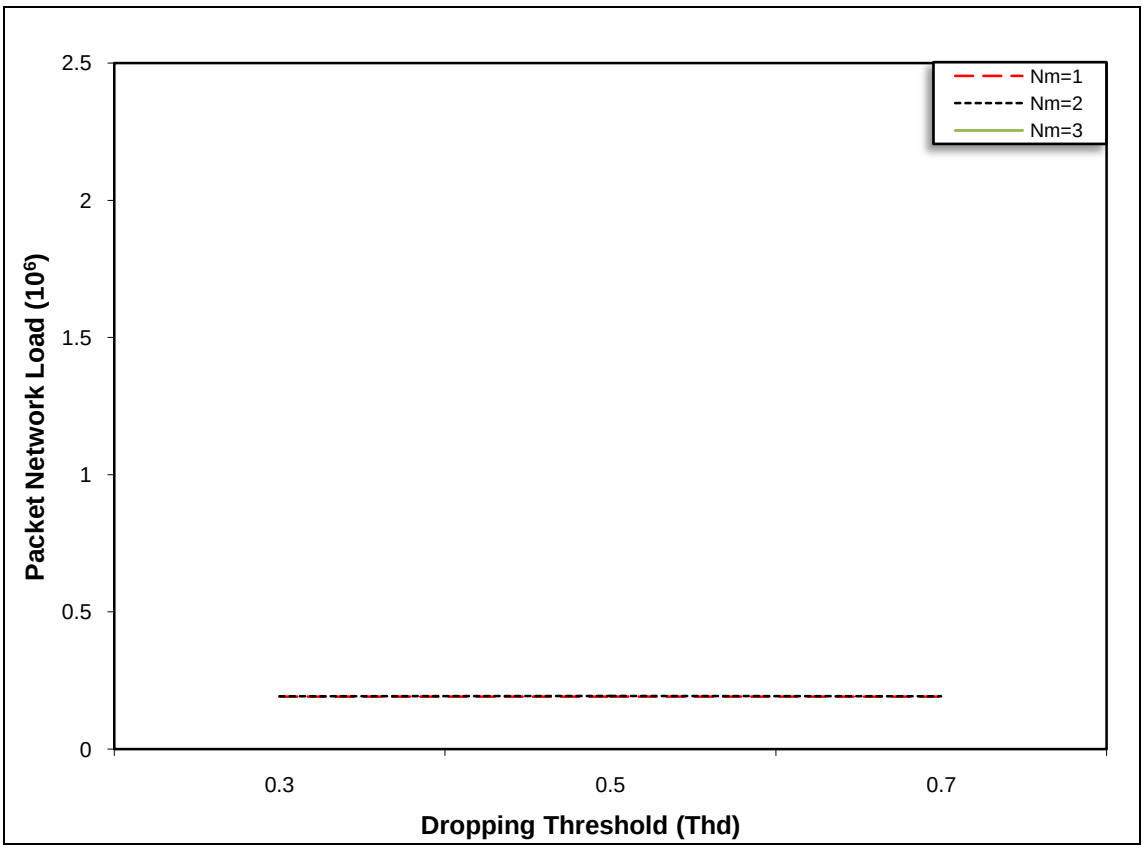


(a) Packet delivery fraction.

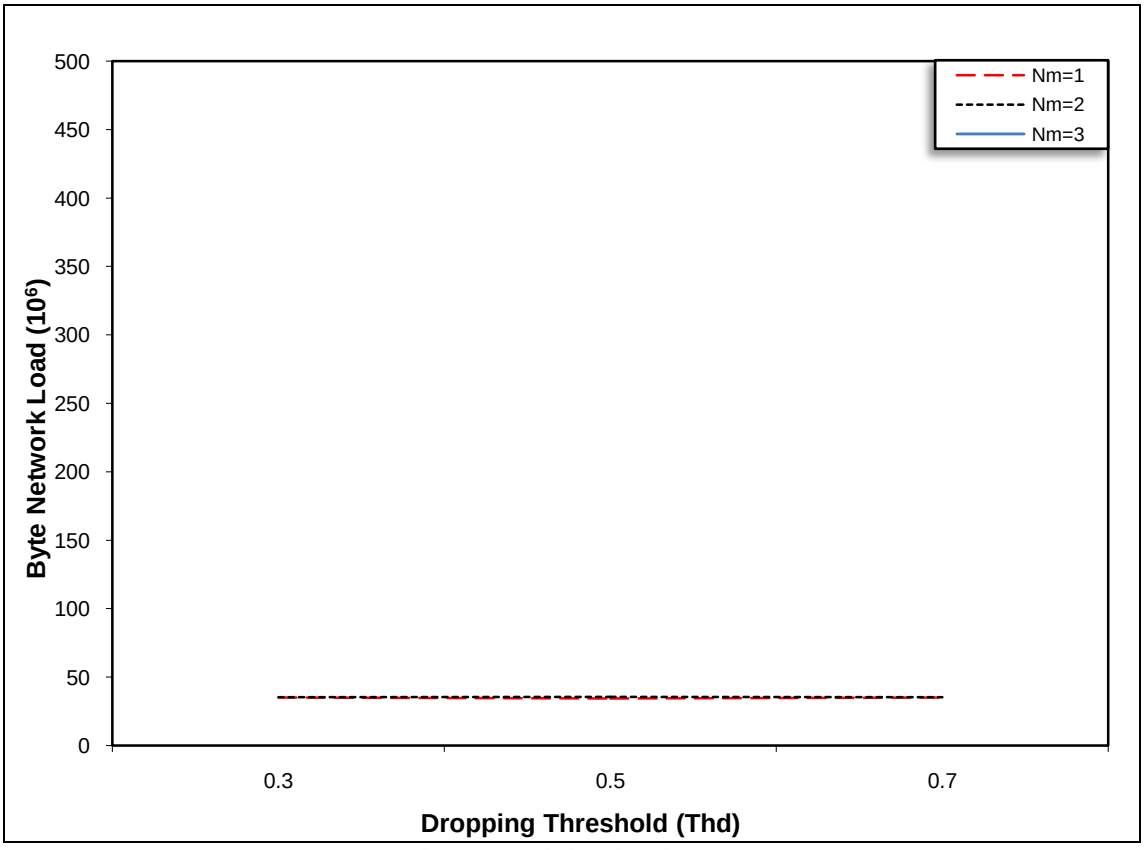


(b) Average path number of hops.

Figure C.2: Impact of *Nm* and *Thd* on discovering black hole attackers.

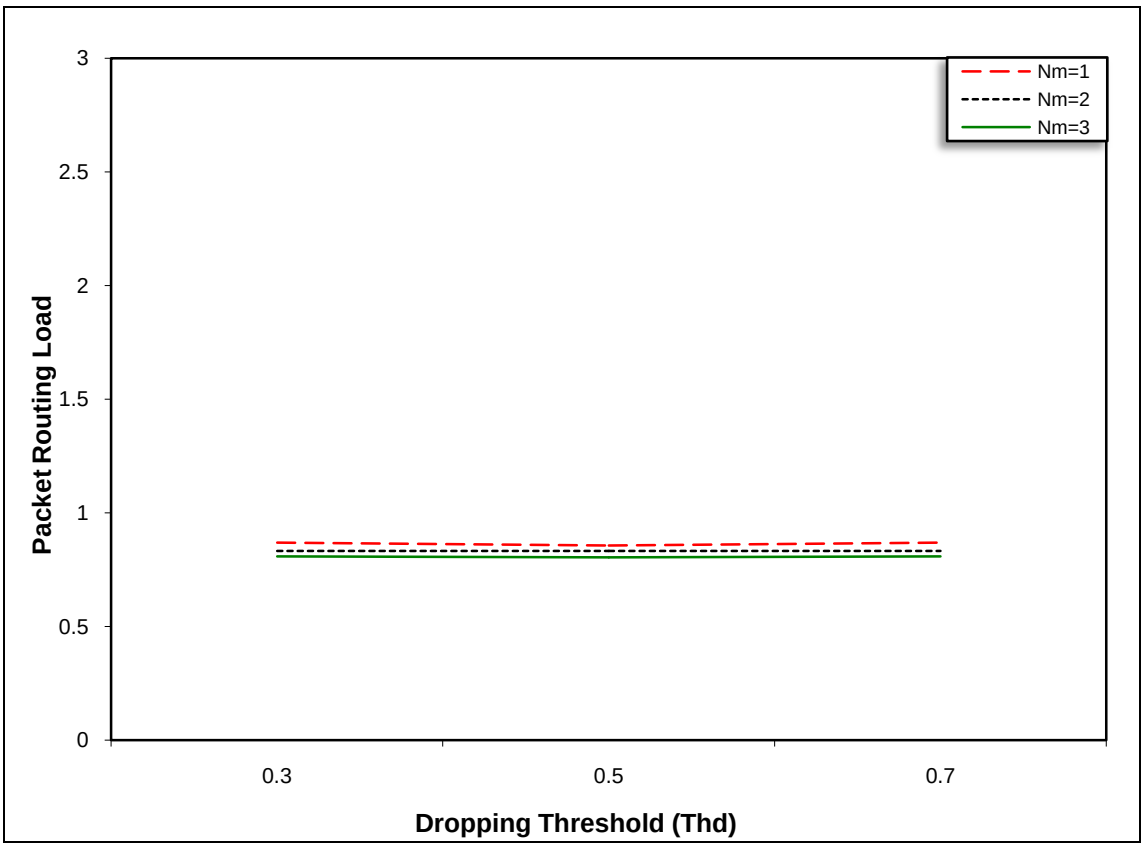


(c) Network load in packets.

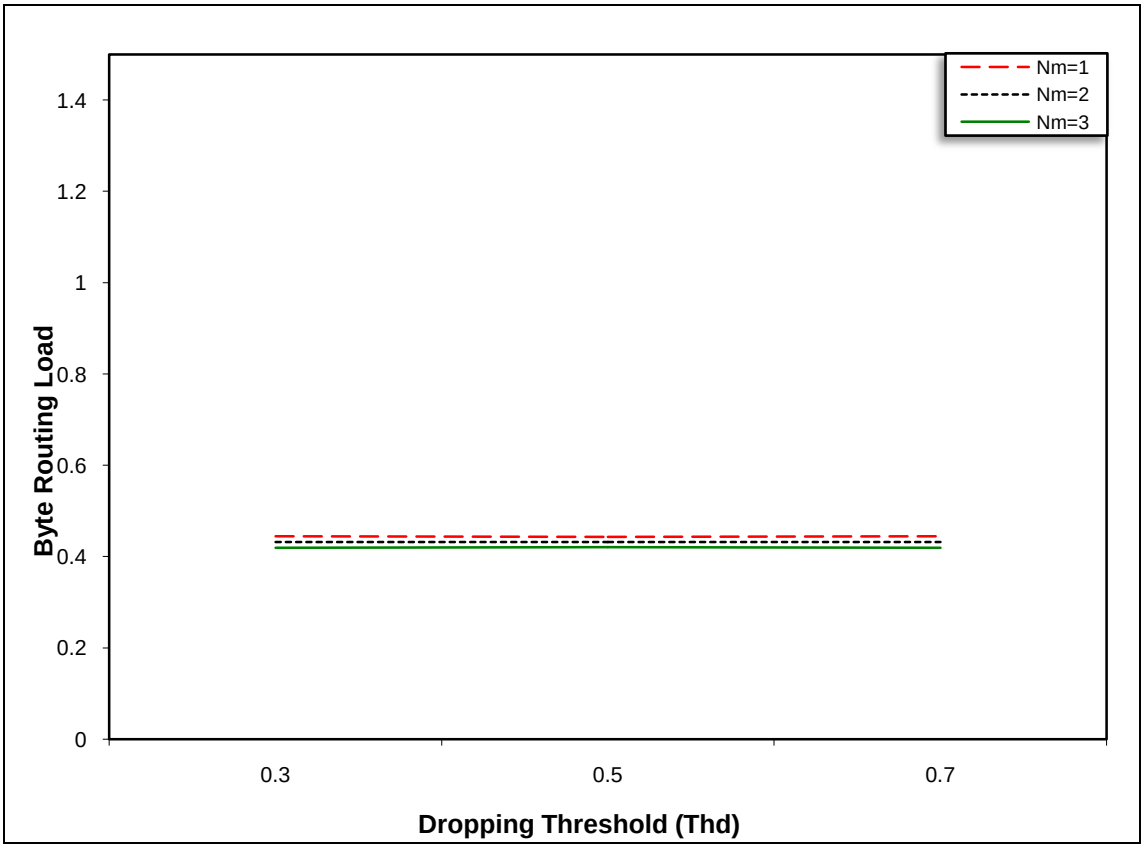


(d) Network load in bytes.

Figure C.2: Impact of *Nm* and *Thd* on discovering black hole attackers.

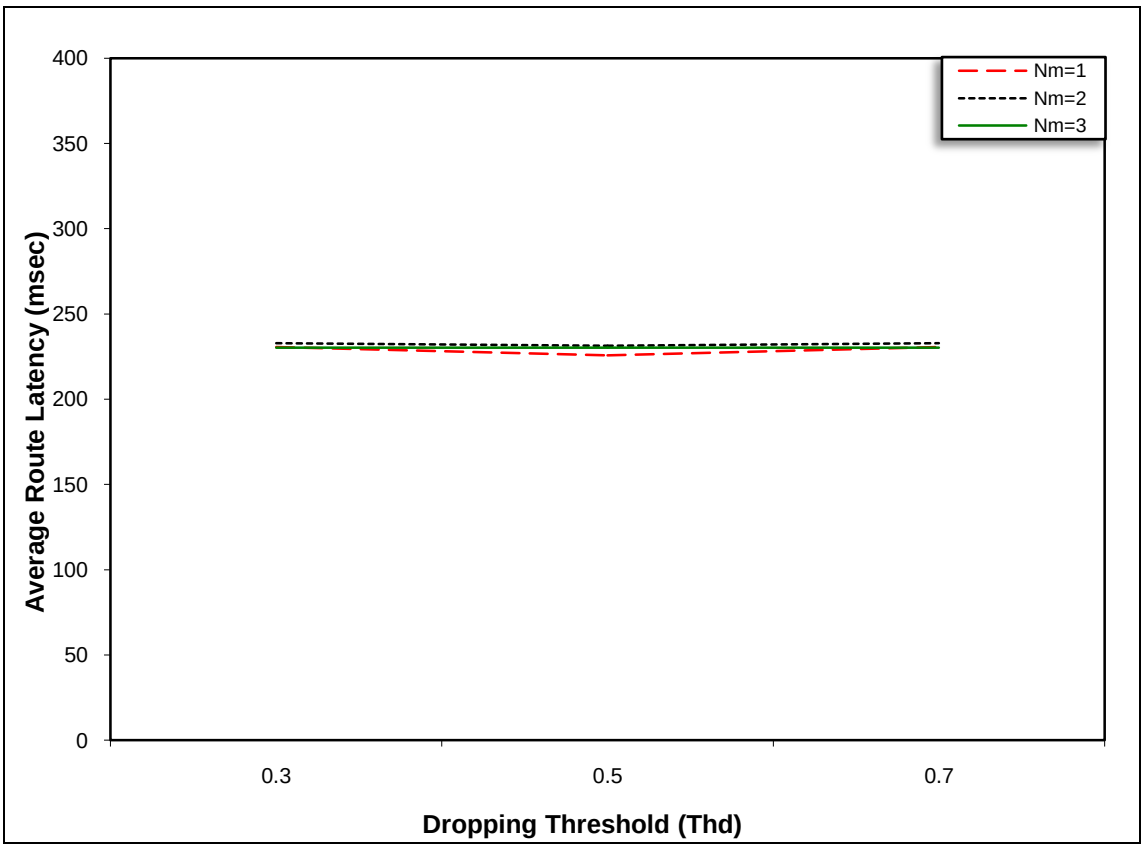


(e) Routing load in packets.

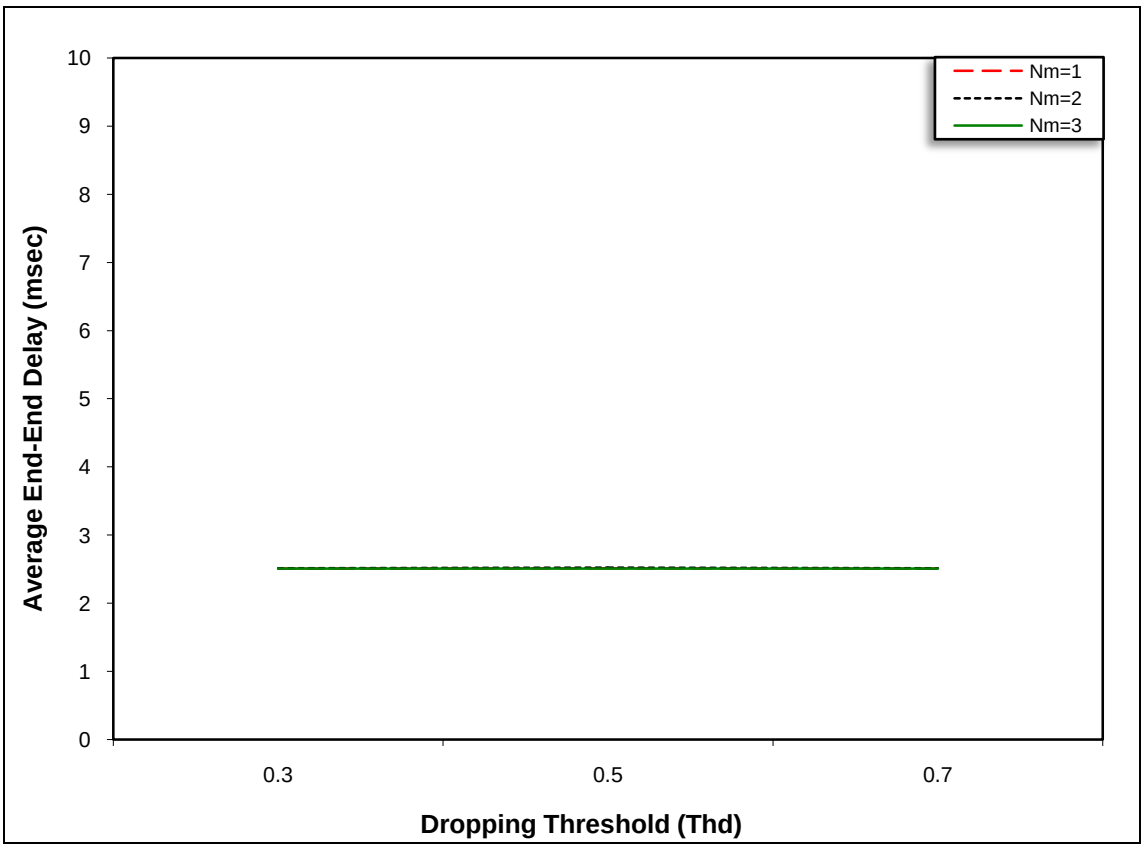


(f) Routing load in bytes.

Figure C.2: Impact of *Nm* and *Thd* on discovering black hole attackers.

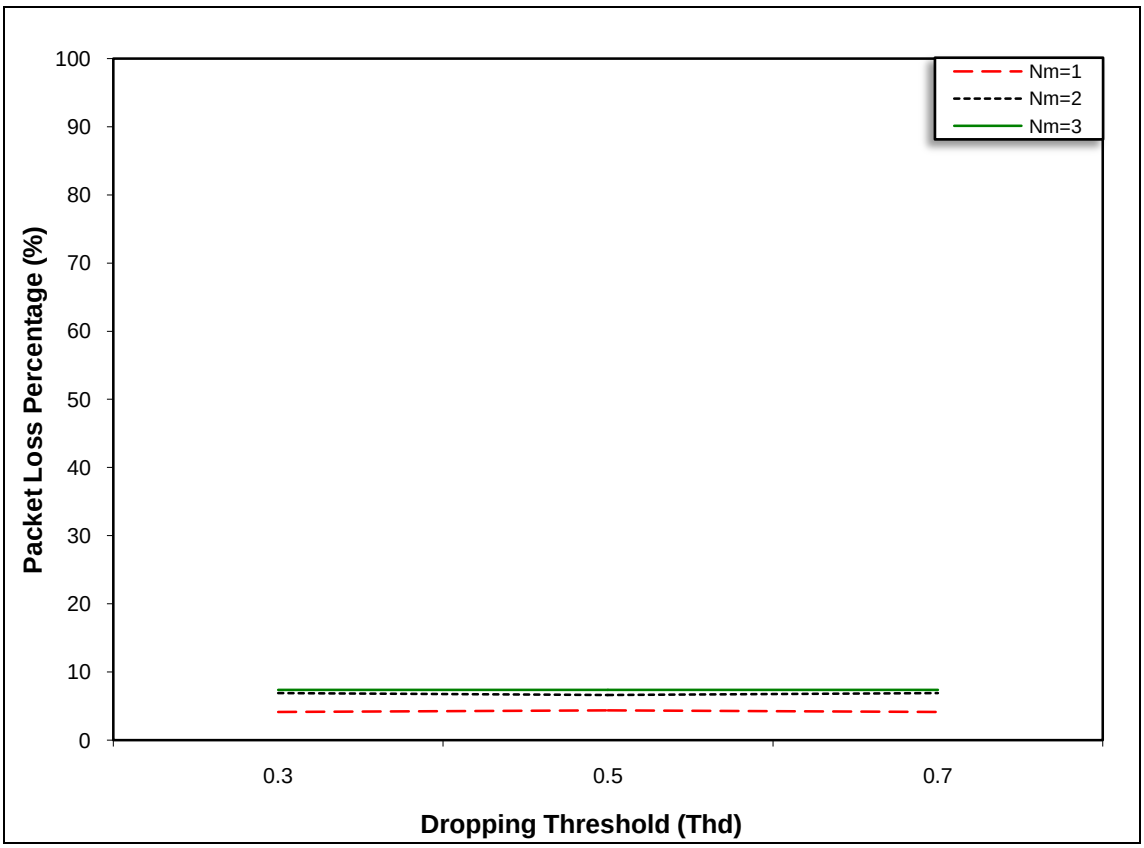


(g) Average route acquisition latency.

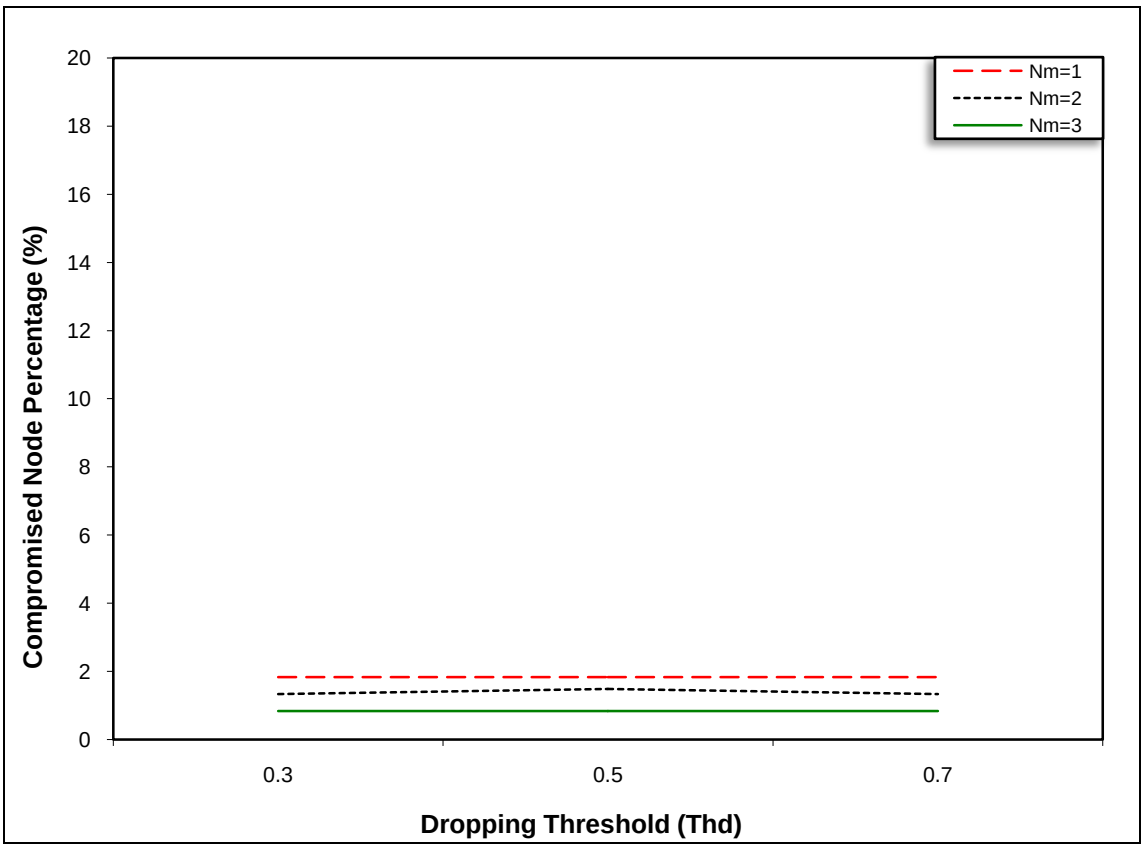


(h) Average end-to-end delay of data packets.

Figure C.2: Impact of *Nm* and *Thd* on discovering black hole attackers.

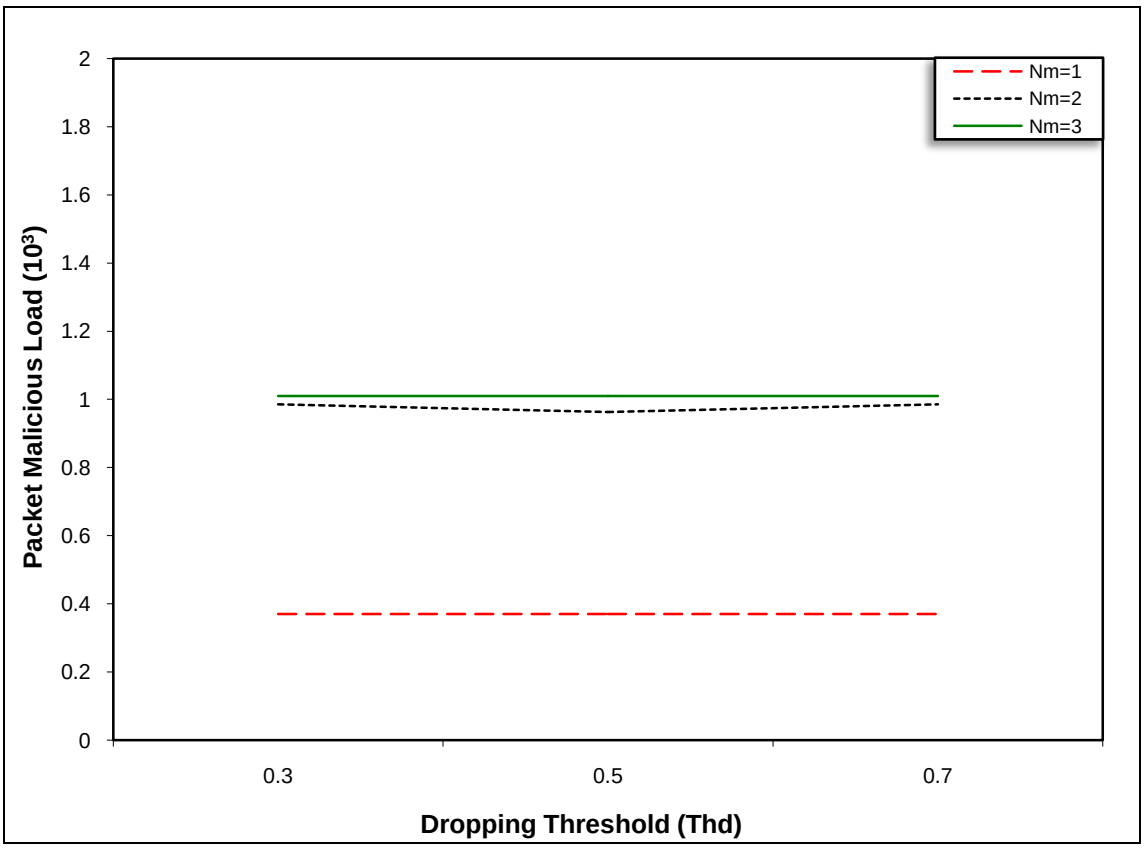


(i) Packet loss percentage.

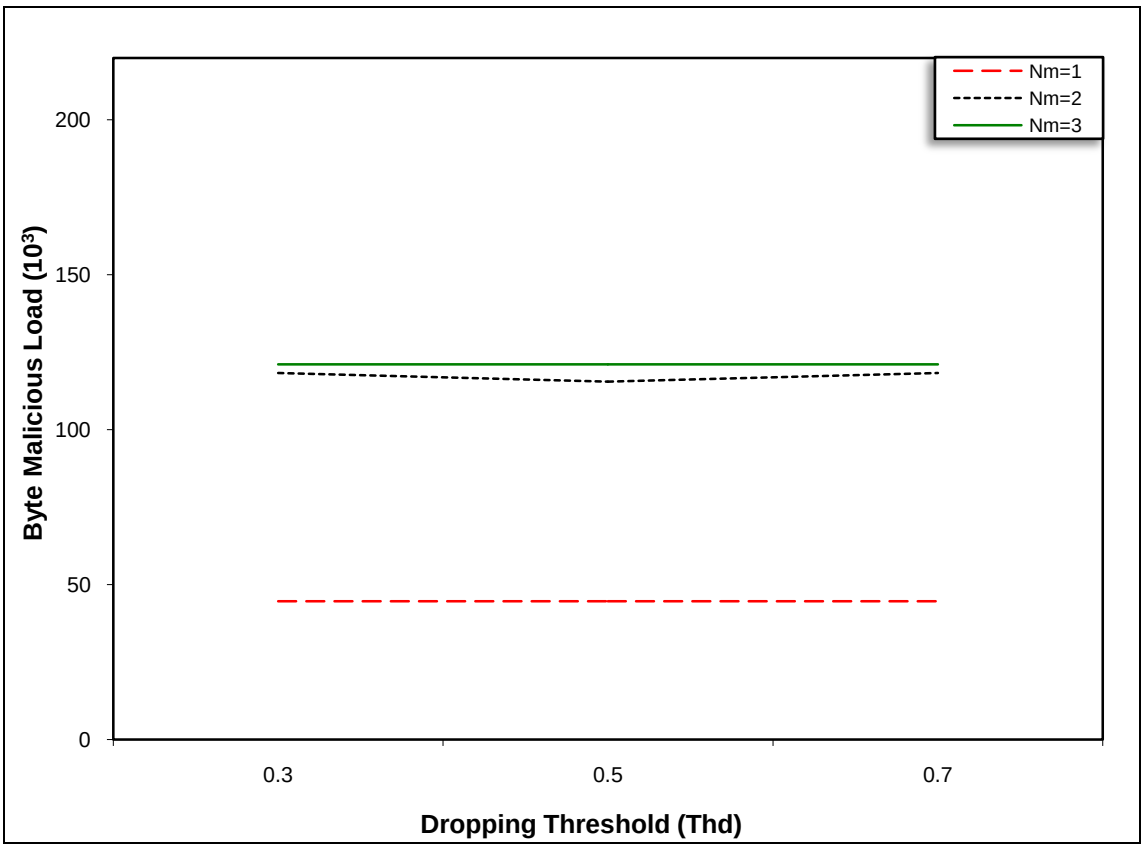


(j) Compromised node percentage.

Figure C.2: Impact of *Nm* and *Thd* on discovering black hole attackers.



(k) Malicious load in packets.

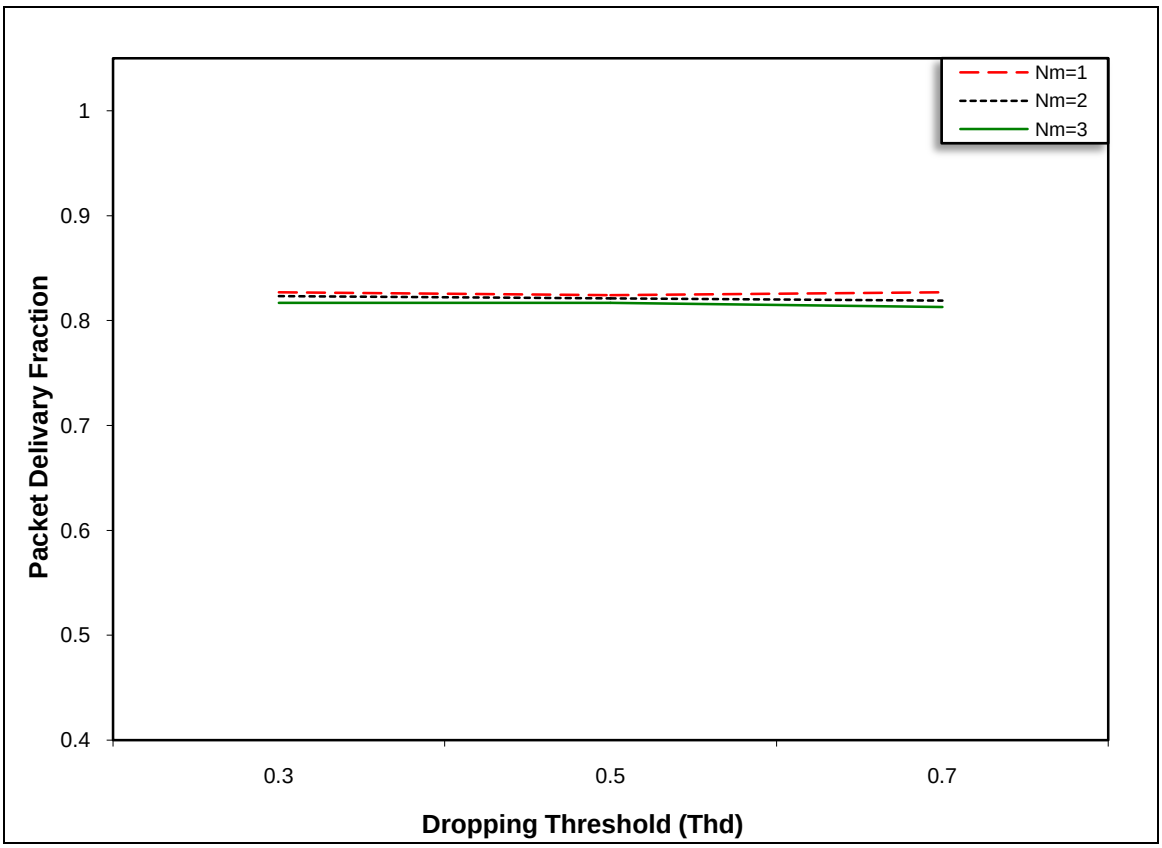


(l) Malicious load in bytes.

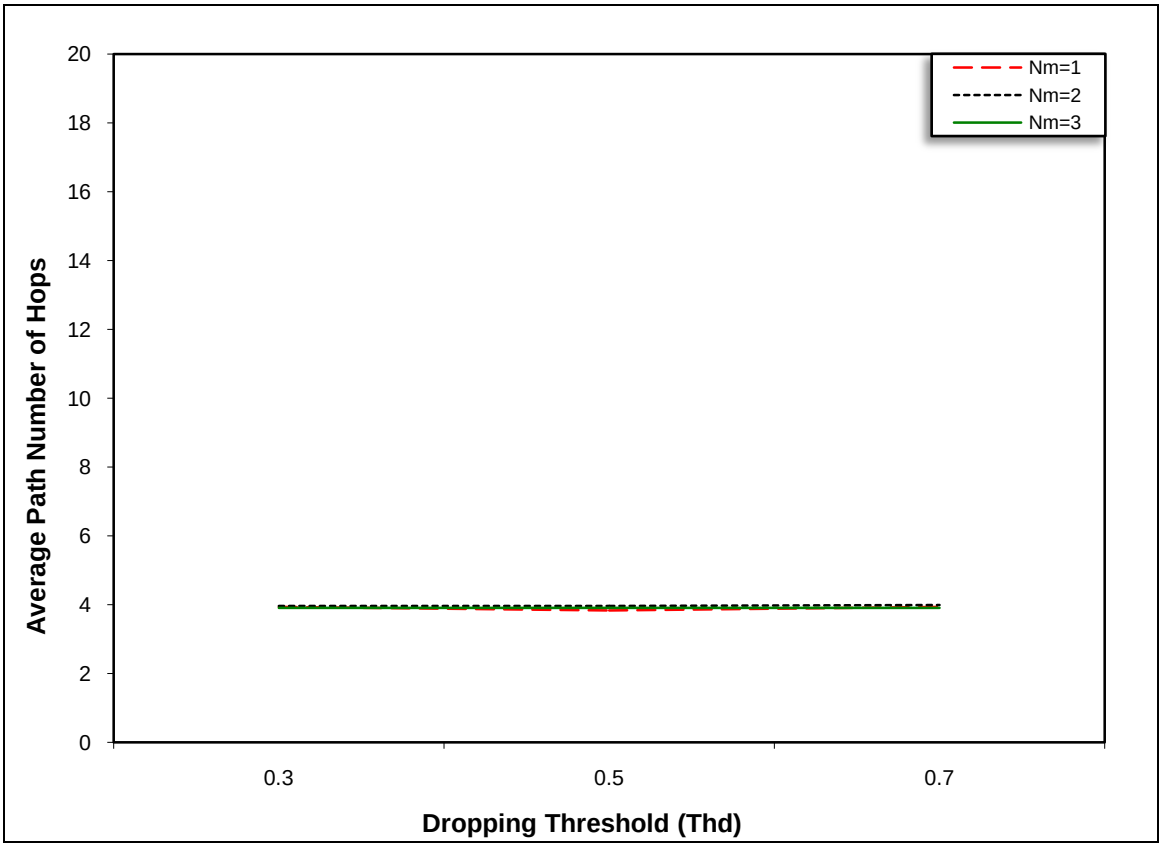
Figure C.2: Impact of *Nm* and *Thd* on discovering black hole attackers.

C.3 Specifying Constant Parameters for Studying Grey Hole Attackers Effect

The third experiment is carried out to study the effect of *Nm* and *Thd* on discovering grey hole attackers. *Nm* is assigned to 1, 2 and 3 and *Thd* is assigned to 0.3, 0.5 and 0.7. Results are shown in Figure C.3. The obtained results are similar to the results obtained in the previous section. None of the metrics is affected by changing *Thd* value. Also, the lowest *PLP* and highest *CNP* are obtained with *Nm*=1 meaning that higher number of grey hole attackers are identified as compromised while having minimum *PML* and *BML*. Discovering larger number of malicious nodes results in a little bit increase in *PRL* and *BRL* while keeping the maximum *PDF*. Consequently values of 1 and 0.5 are assigned to *Nm* and *Thm* respectively upon simulating scenarios 3 and 5.

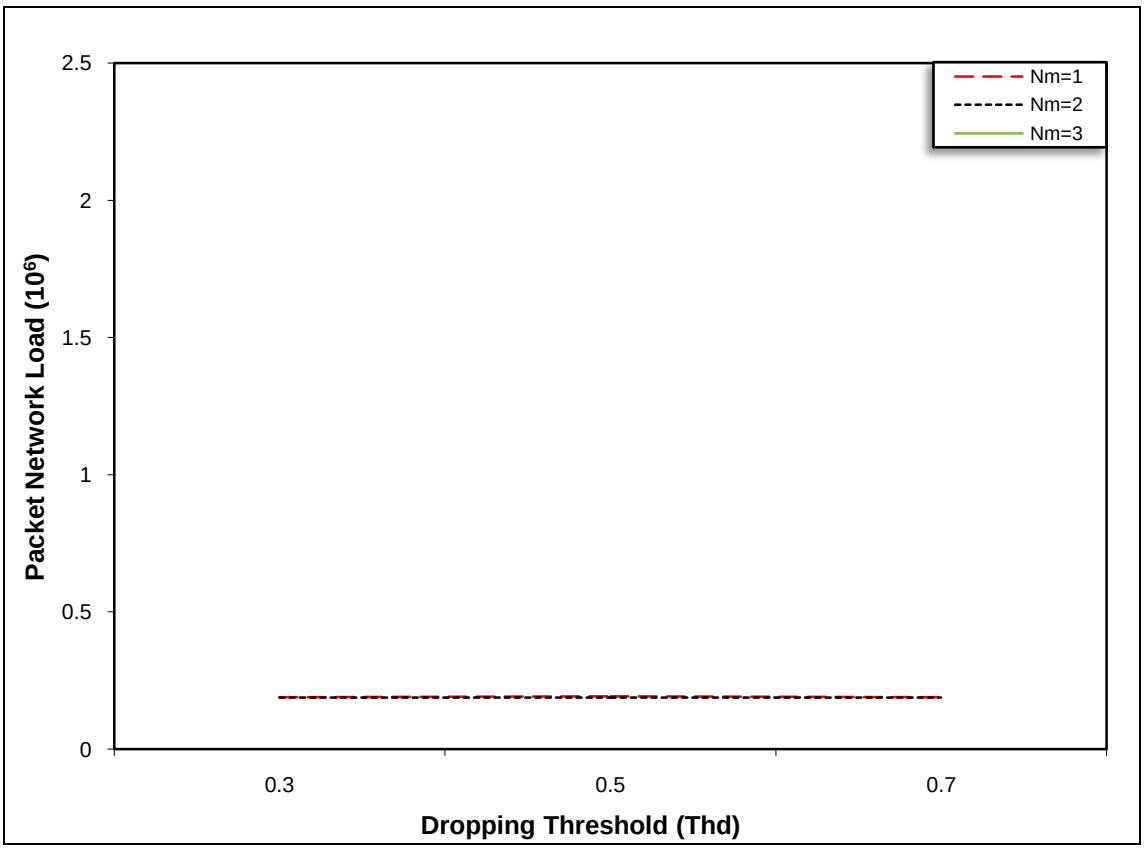


(a) Packet delivery fraction.

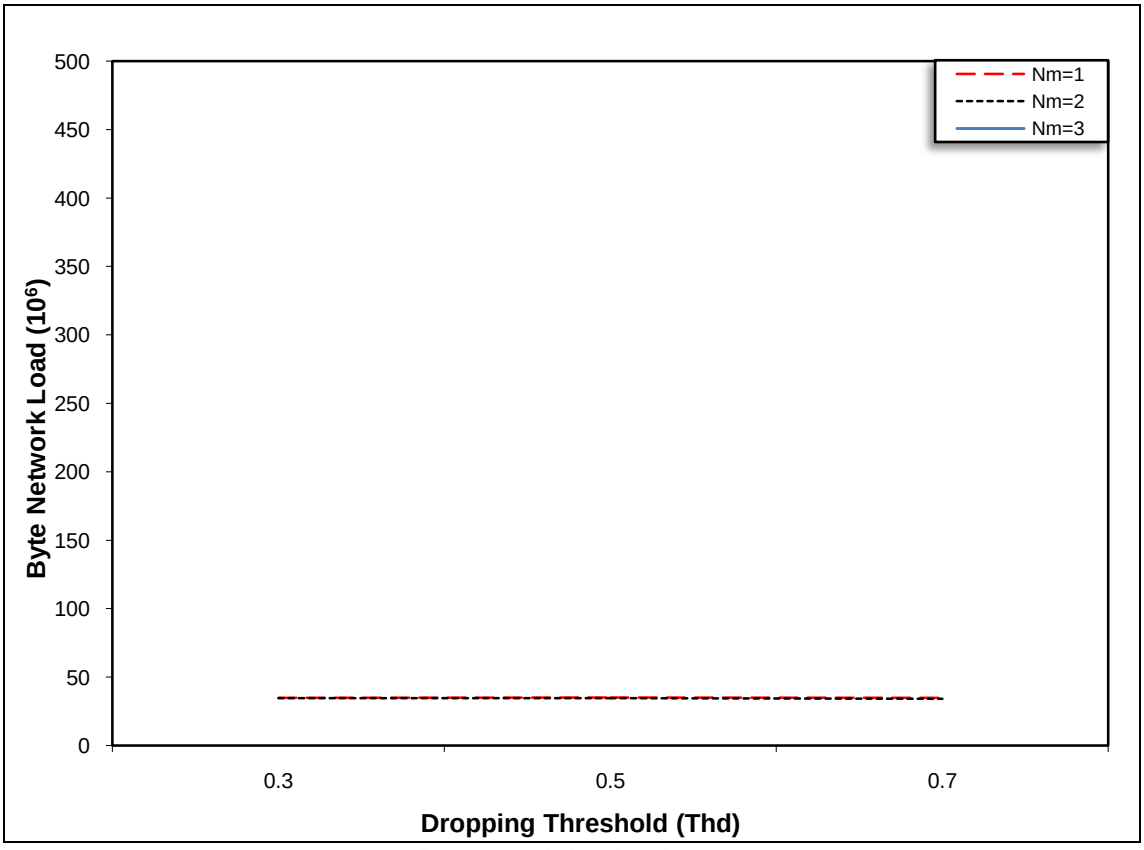


(b) Average path number of hops.

Figure C.3: Impact of *Nm* and *Thd* on discovering grey hole attackers.

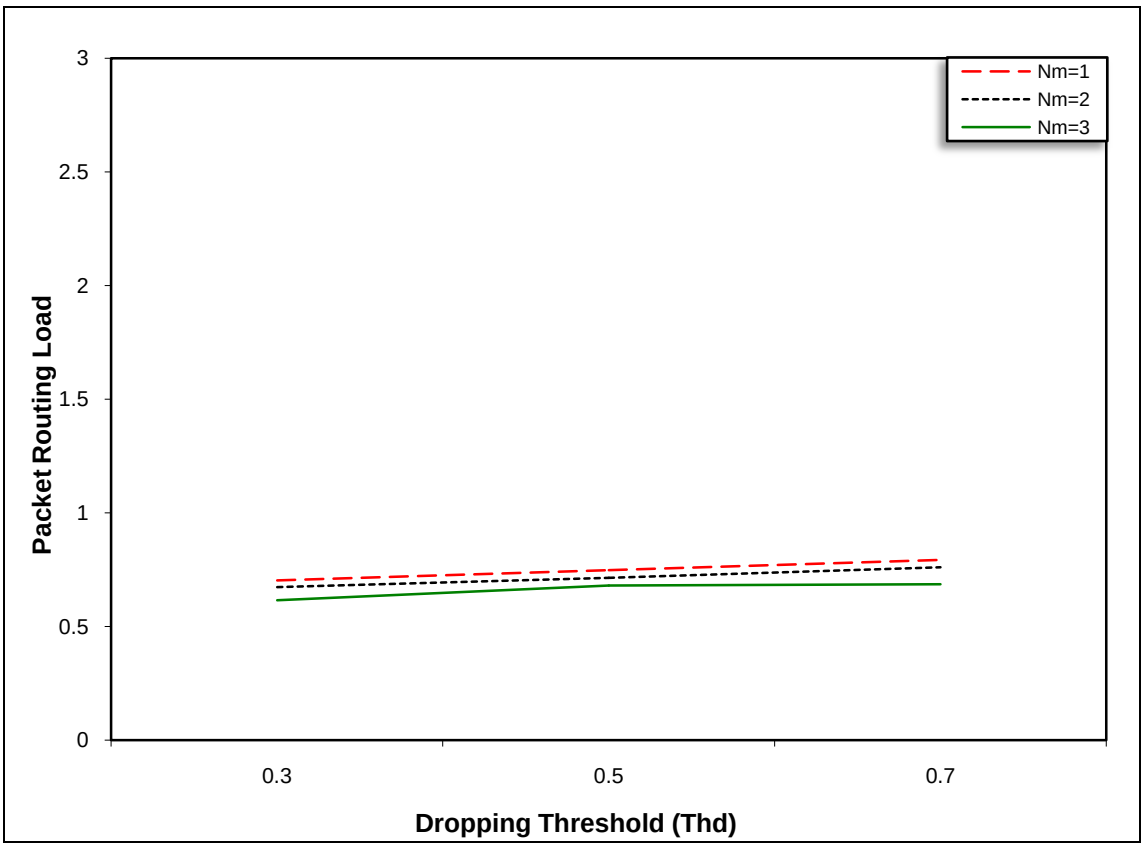


(c) Network load in packets.

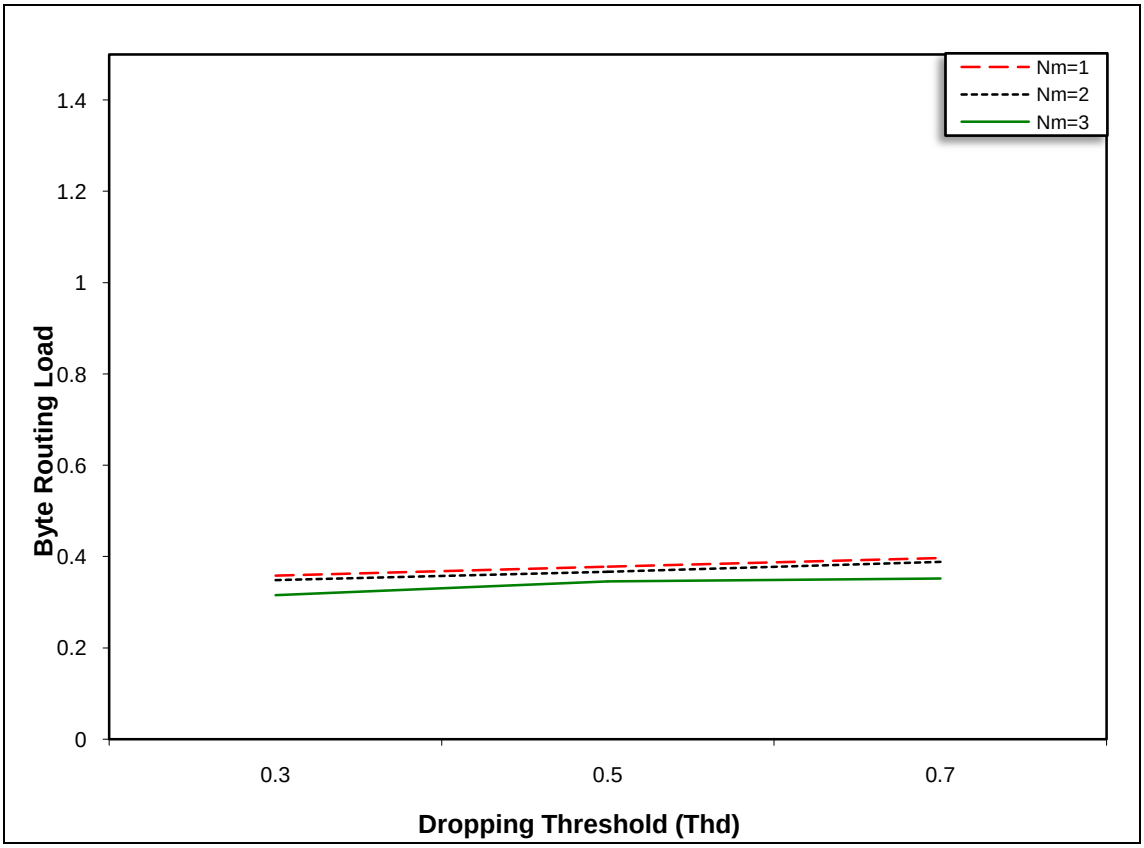


(d) Network load in bytes.

Figure C.3: Impact of *Nm* and *Thd* on discovering grey hole attackers.

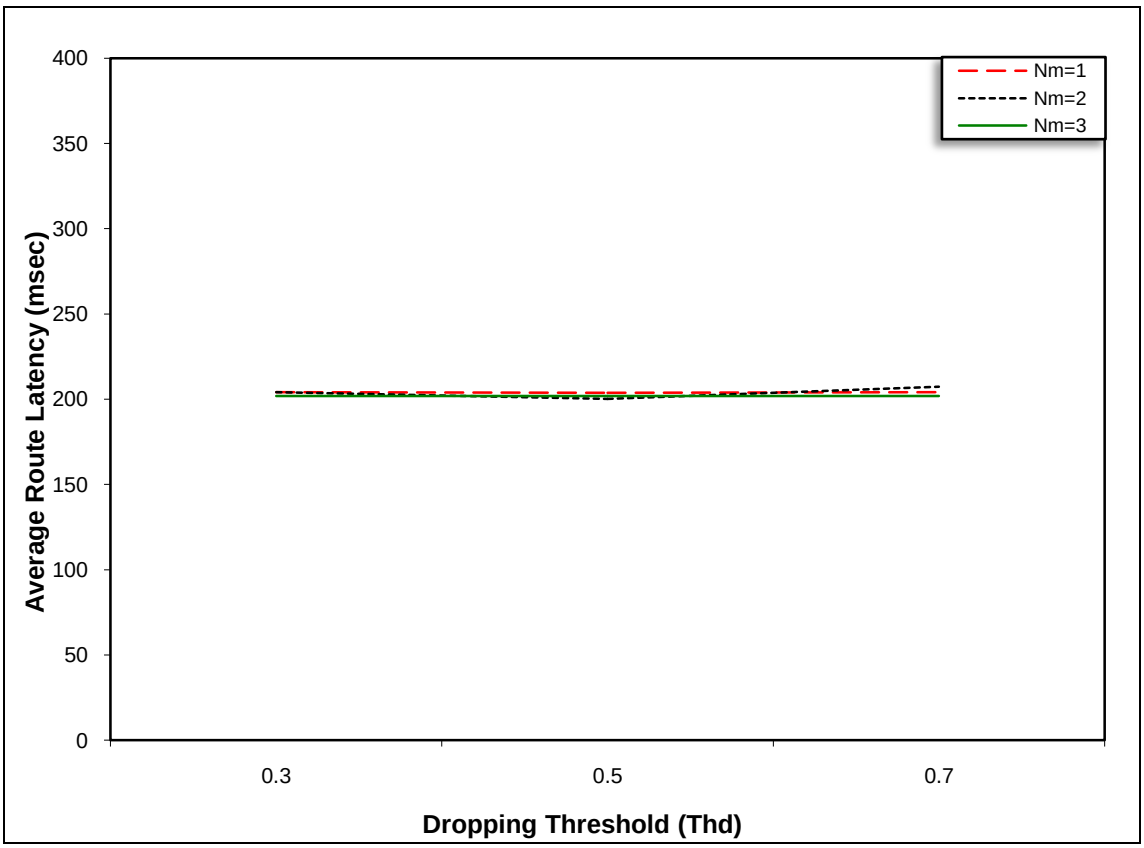


(e) Routing load in packets.

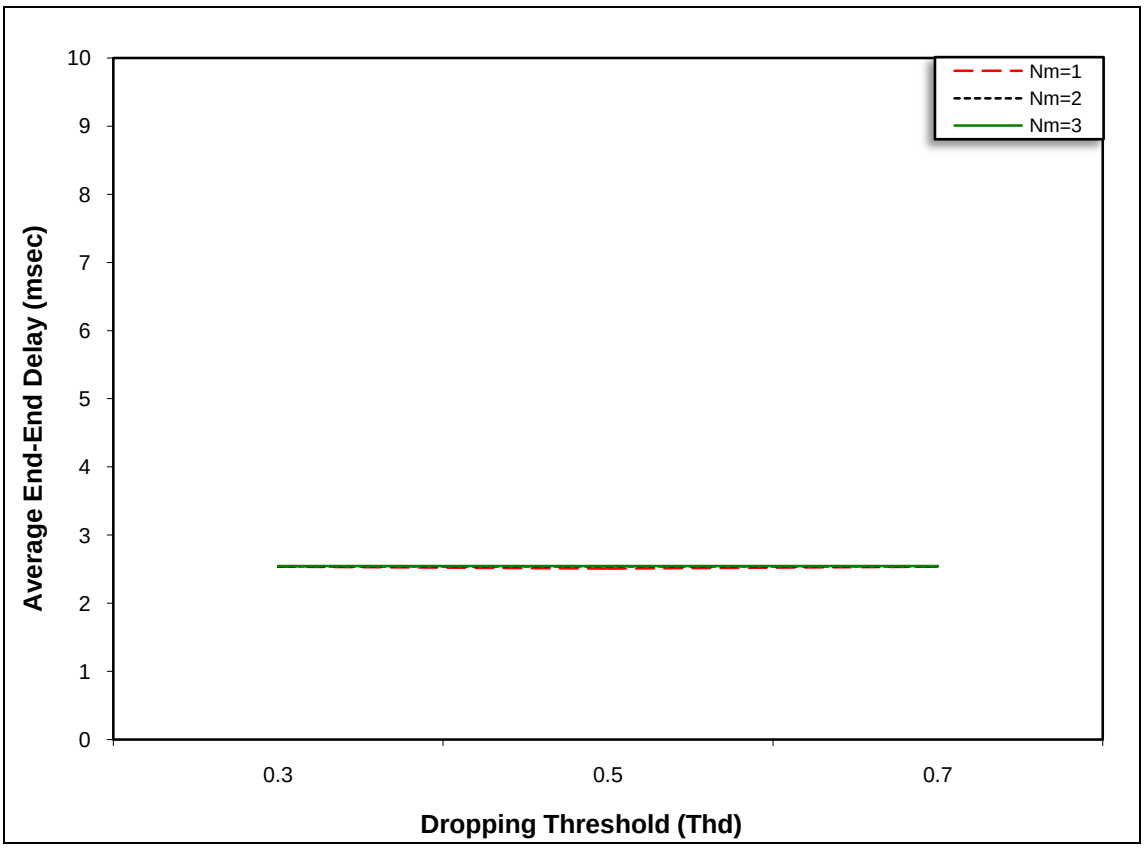


(f) Routing load in bytes.

Figure C.3: Impact of *Nm* and *Thd* on discovering grey hole attackers.

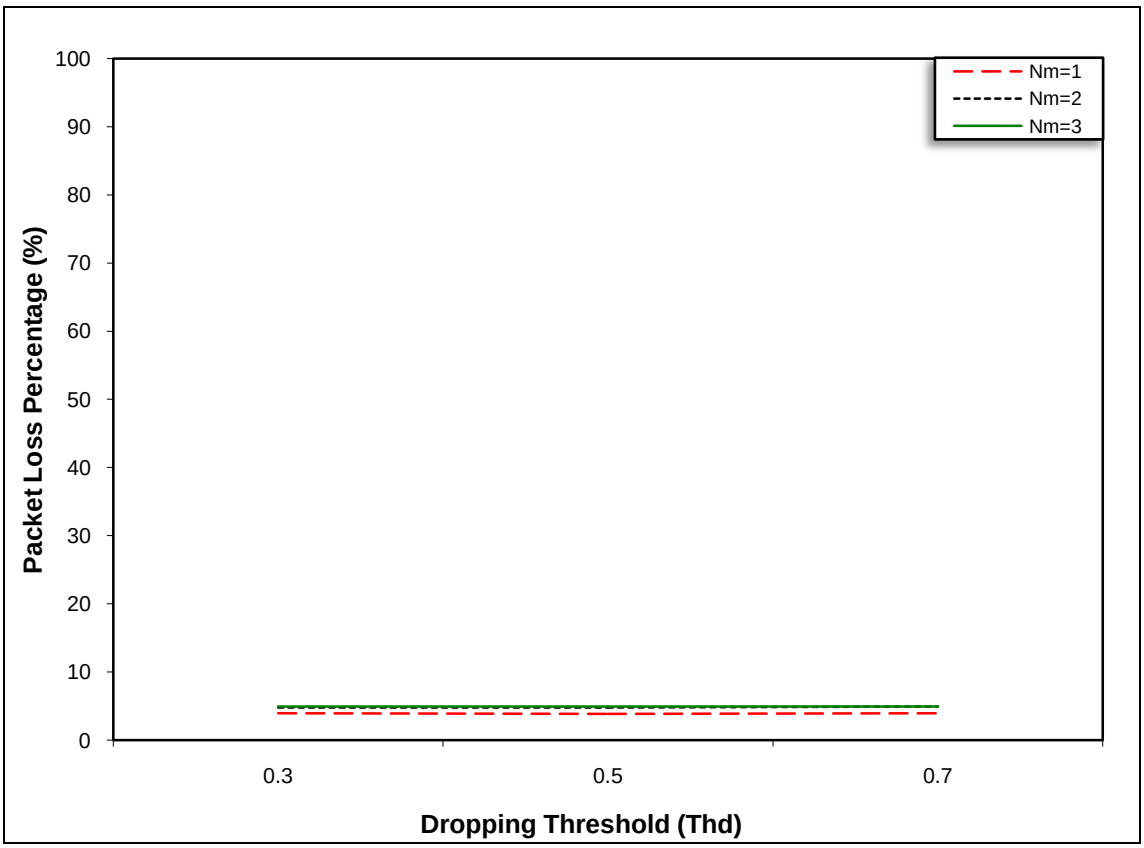


(g) Average route acquisition latency.

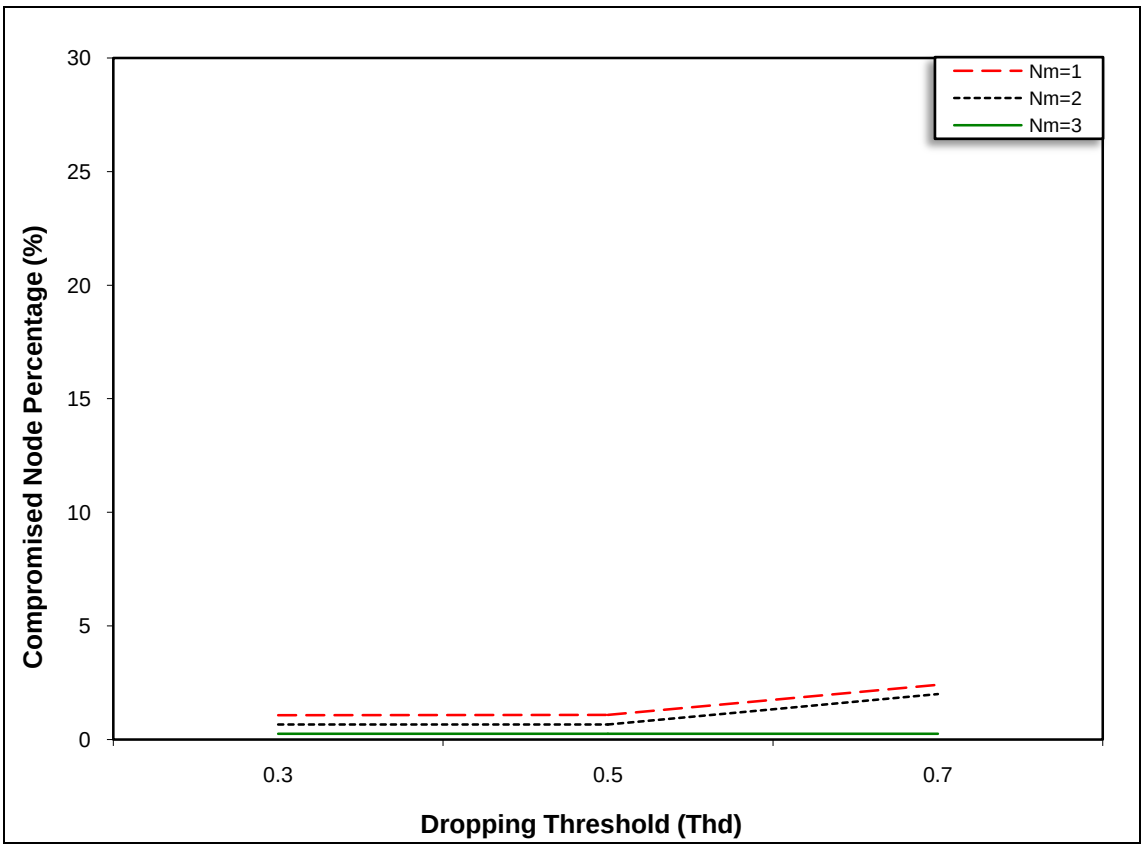


(h) Average end-to-end delay of data packets.

Figure C.3: Impact of *Nm* and *Thd* on discovering grey hole attackers.

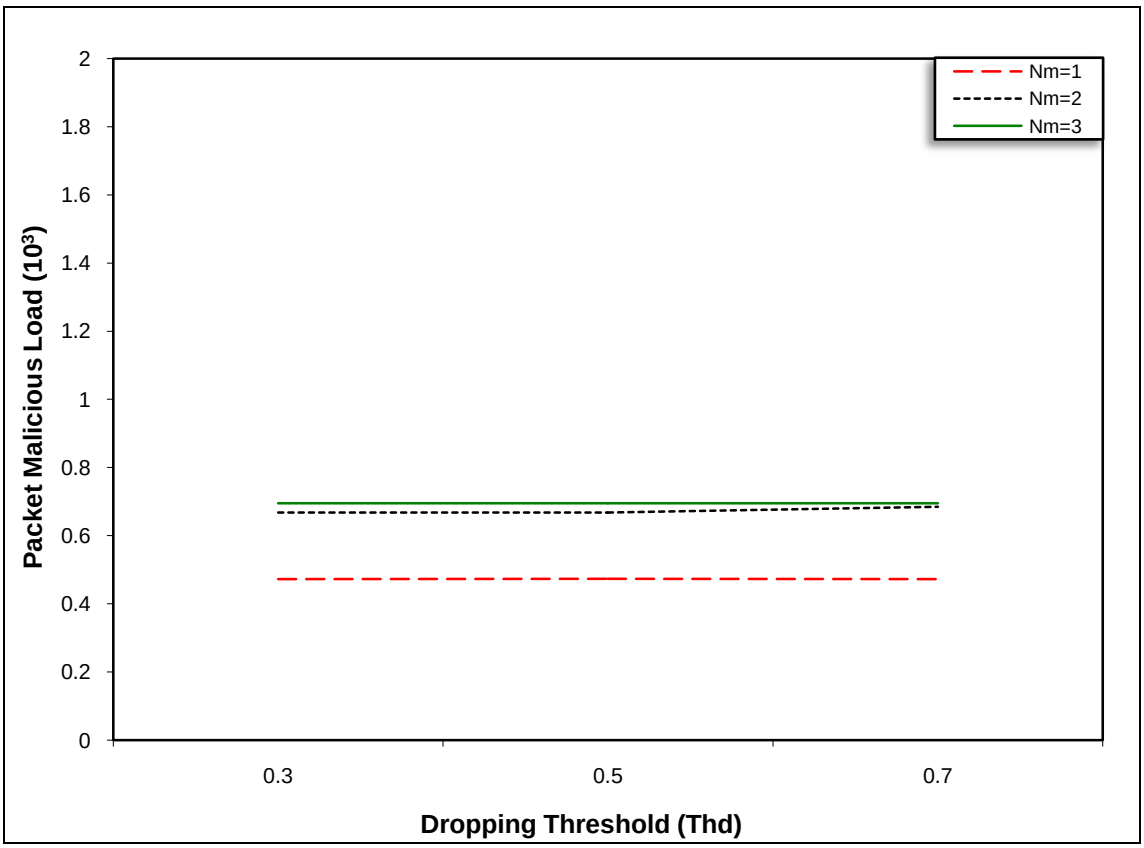


(i) Packet loss percentage.

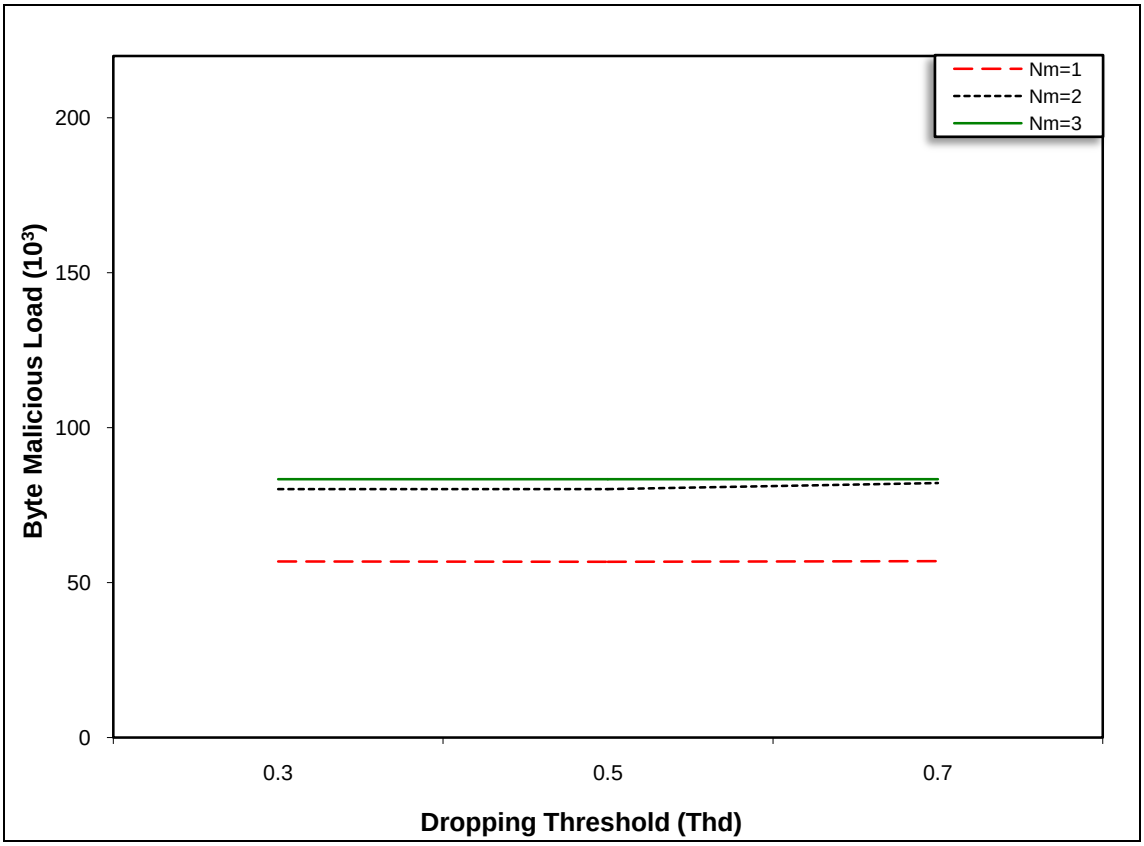


(j) Compromised node percentage.

Figure C.3: Impact of *Nm* and *Thd* on discovering grey hole attackers.



(k) Malicious load in packets.



(l) Malicious load in bytes.

Figure C.3: Impact of *Nm* and *Thd* on discovering grey hole attackers.

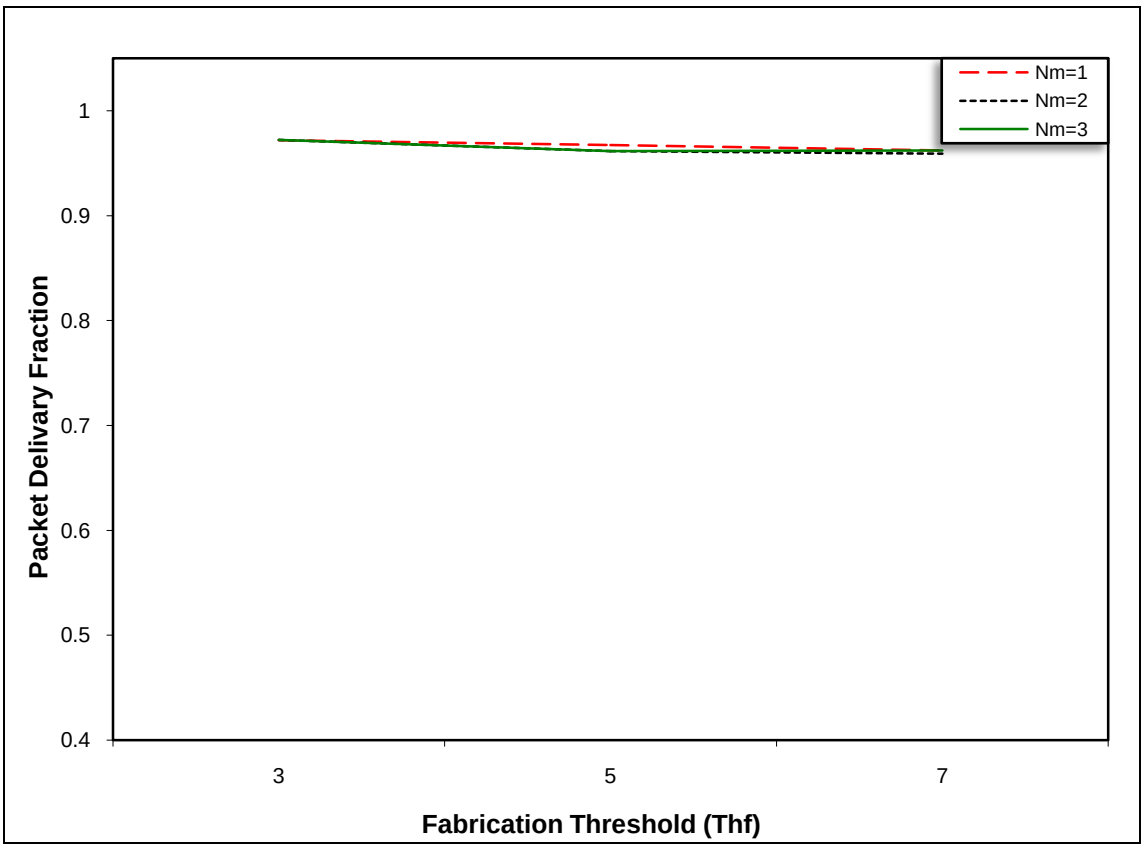
C.4 Specifying Constant Parameters for Studying Fabrication Attackers Effect

This experiment studies the effect of *Nm* and *Thf* on determining fabrication attackers. Different values for *Thf* are considered which are 3, 5 and 7 also *Nm* is set to 1, 2 and 3. Results are shown in Figure C.4.

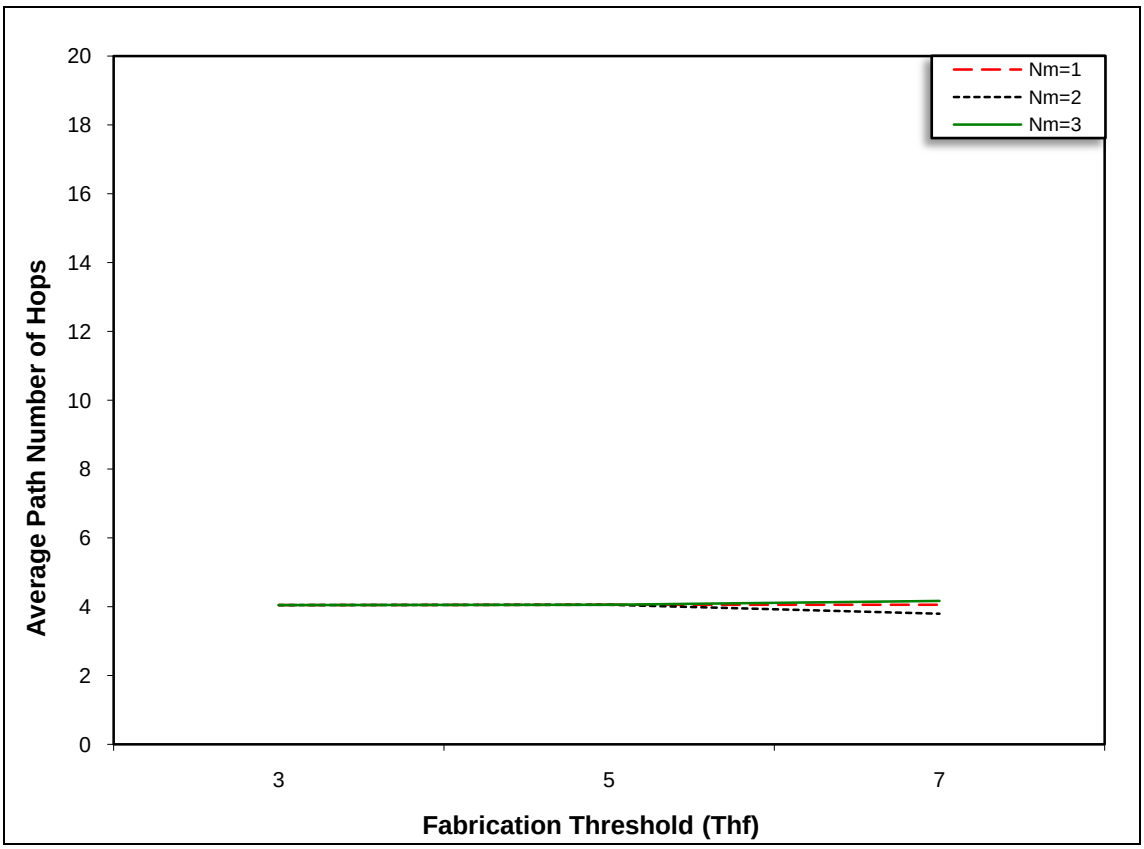
Minimum *FEP* and maximum *CNP* are obtained with *Nm*=1 whatever the value of *Thf*; this suggests that higher number of fabrication attackers are discovered and identified as compromised while having minimum *PML* and *BML*. On the other hand, upon setting *Nm* to 2 and 3 nodes keep sending *MNODE* packets resulting in higher *PML* and *BML* while still not recognizing the malicious nodes as compromised, i.e., increased *FEP* and reduced *CNP*. Discovering higher number of malicious nodes results in a little bit increase in *PRL* and *BRL* due to reinitiating *RDP* packets while keeping the minimum *ARL* and maximum *PDF*. Therefore value of *Nm* is chosen to be 1.

Considering *Nm=1*, it is clear from the figure that with *Thf*=3 the lowest *FEP* and highest *CNP* are obtained meaning that higher number of fabrication attackers are discovered. Excluding these malicious nodes from future rotes reduces the probability of reinitiating route request due to fabricated error packets; which results in increasing *PDF* and reducing *PRL, BRL, PML* and *BML*.

Hence values of *Nm* and *Thf* are set to 1 and 3 respectively upon simulating scenarios 4 and 5.

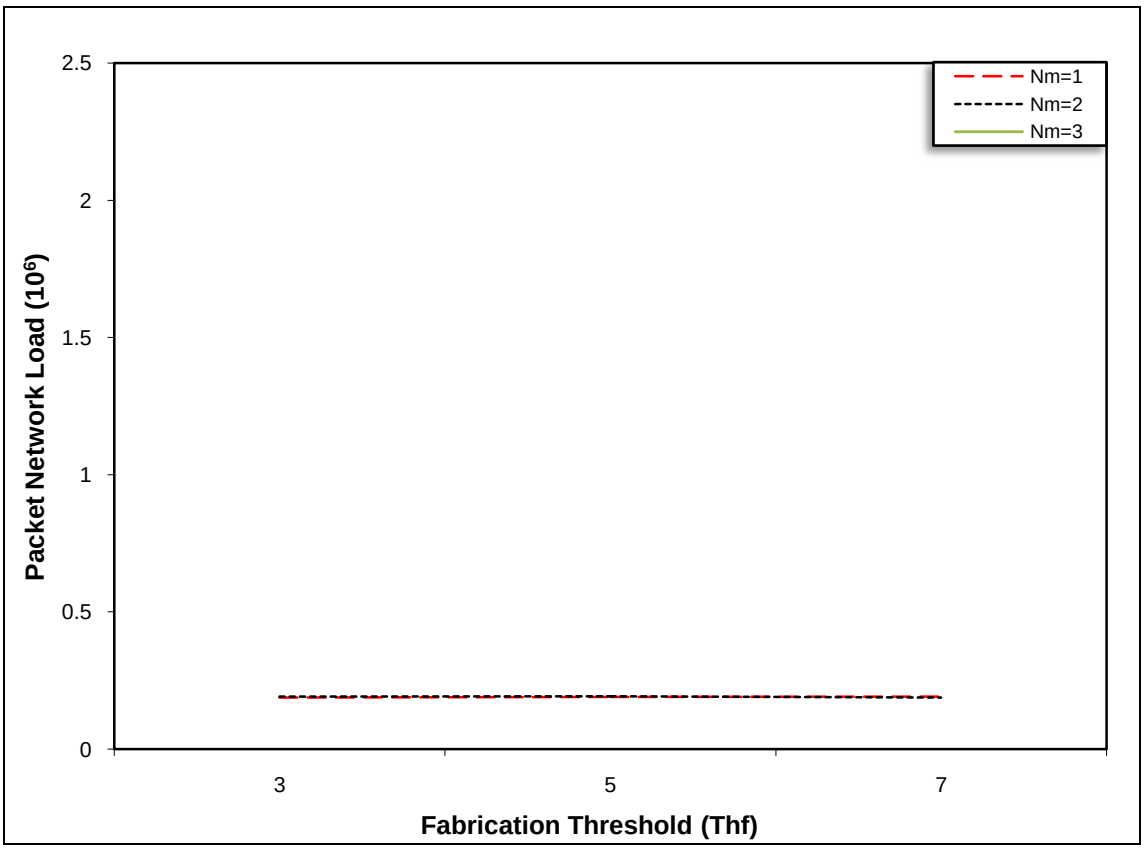


(a) Packet delivery fraction.

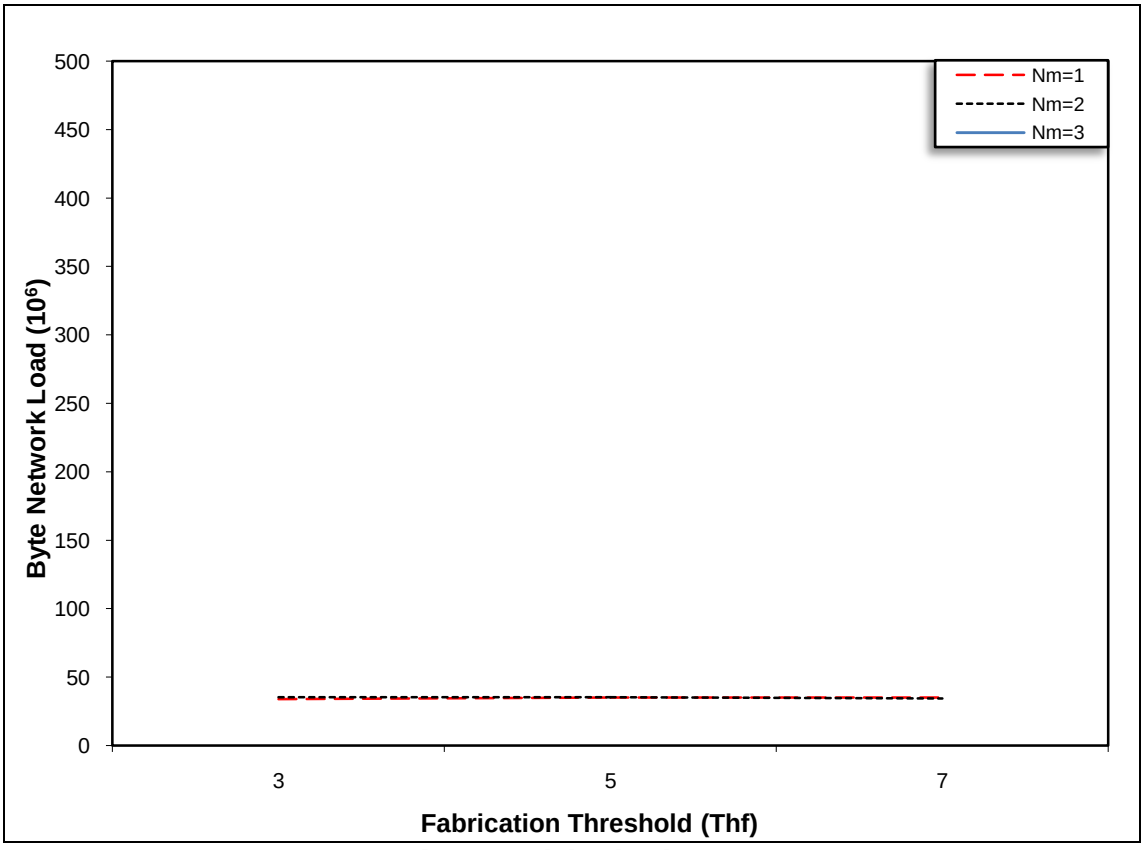


(b) Average path number of hops.

Figure C.4: Impact of *Nm* and *Thf* on discovering fabrication attackers.

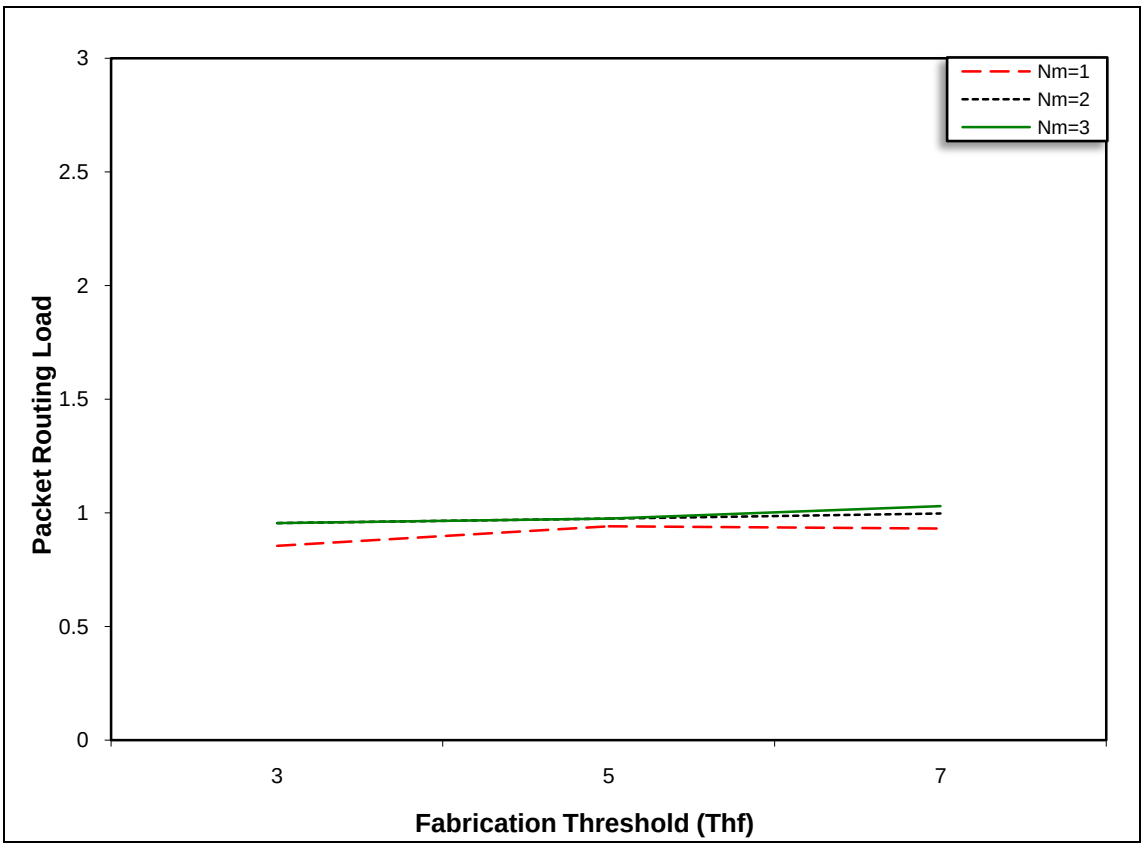


(c) Network load in packets.

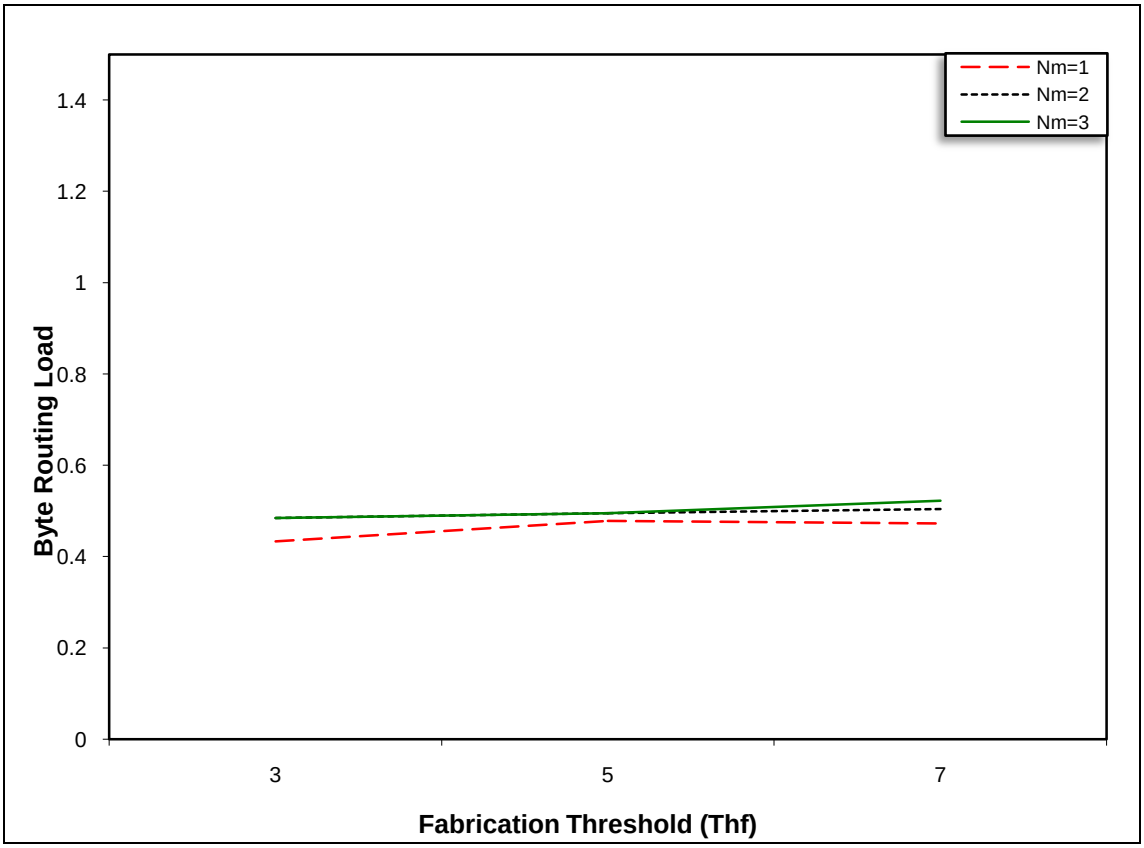


(d) Network load in bytes.

Figure C.4: Impact of *Nm* and *Thf* on discovering fabrication attackers.

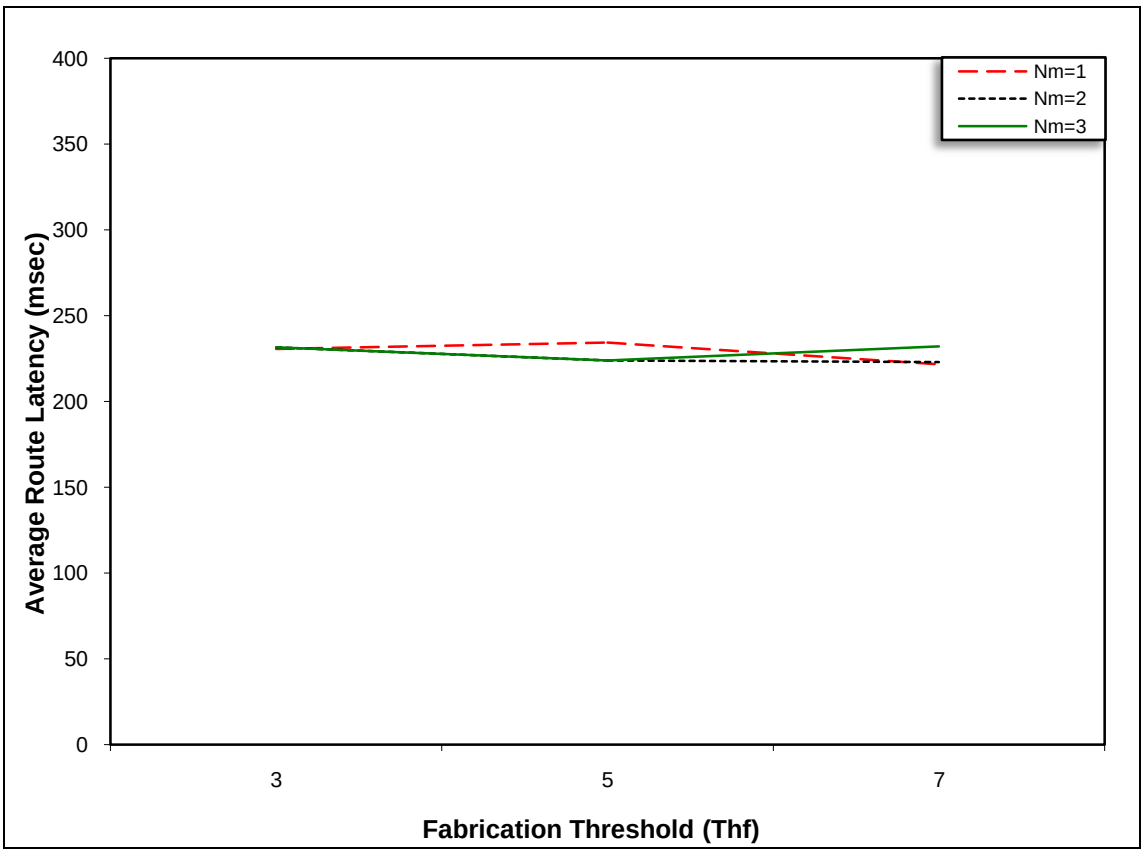


(e) Routing load in packets.

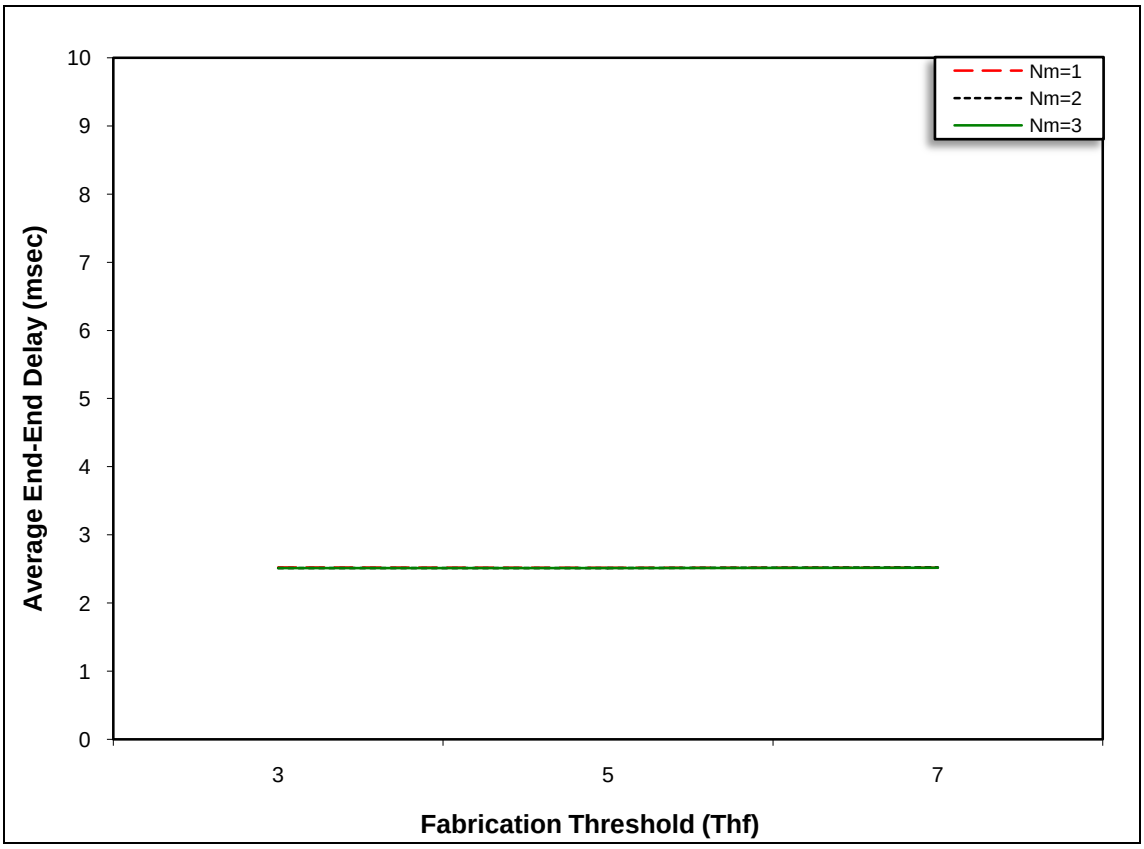


(f) Routing load in bytes.

Figure C.4: Impact of *Nm* and *Thf* on discovering fabrication attackers.

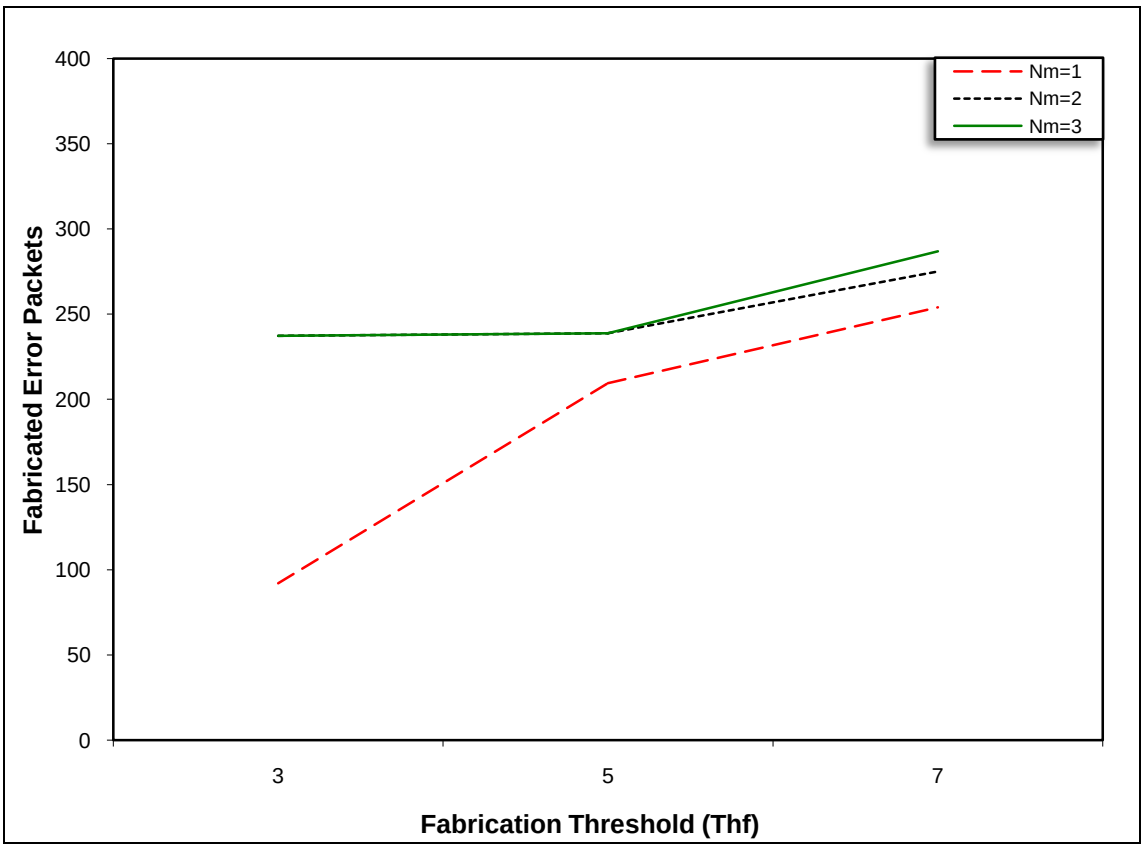


(g) Average route acquisition latency.

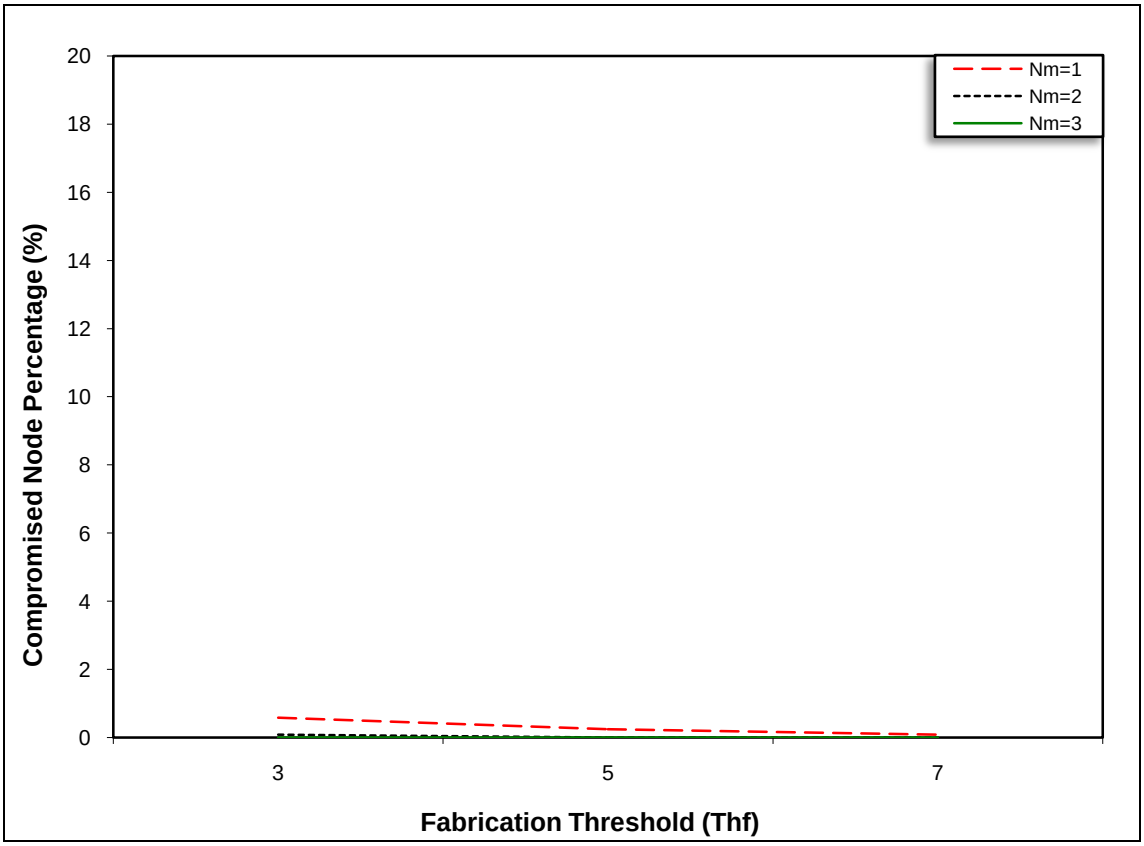


(h) Average end-to-end delay of data packets.

Figure C.4: Impact of *Nm* and *Thf* on discovering fabrication attackers.

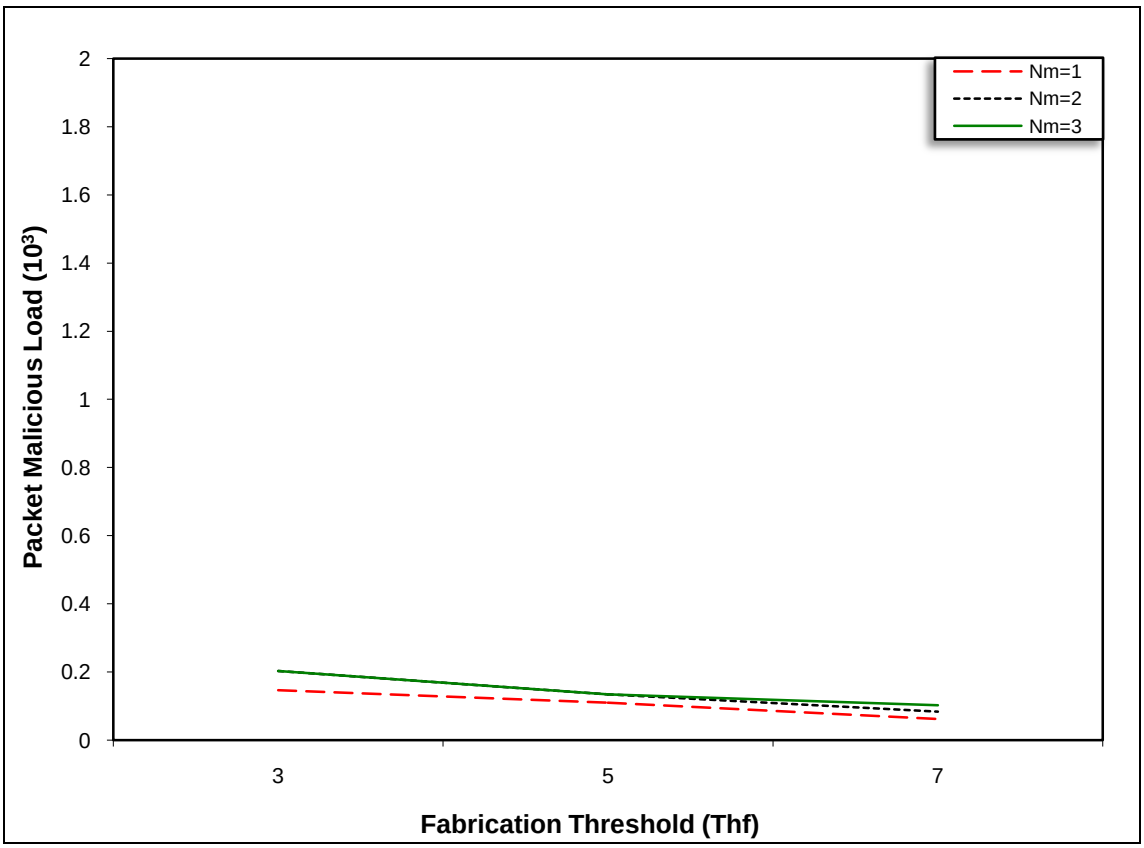


(i) Fabricated error packets.

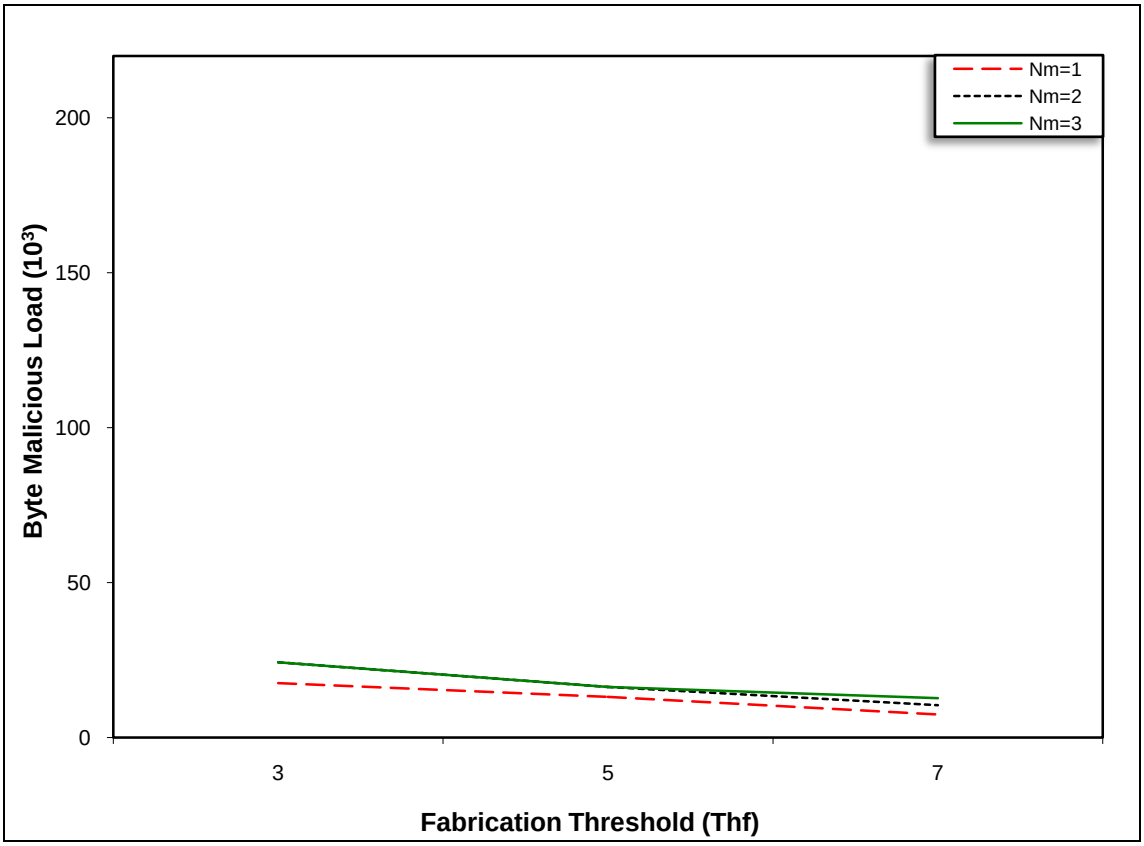


(j) Compromised node percentage.

Figure C.4: Impact of *Nm* and *Thf* on discovering fabrication attackers.



(k) Malicious load in packets.



(l) Malicious load in bytes.

Figure C.4: Impact of *Nm* and *Thf* on discovering fabrication attackers.