Rapport de Recherche

Djamel Djenouri & Nadjib Badache

LSI-TR0404

April 2004

FACULTE ELECTRONIQUE & INFORMATIQUE
Département informatique
El Alia BP n°32 Bab Ezzouar 16111 Alger.
Tél / Fax : 213 (0) 21 24 79 17 - 24 76 07
A Gradual Solution to Detect Selfish Nodes in Mobile Ad hoc Networks

Djamel DJENOURI§, Nadjib BADACHE‡

§: Basic Software Laboratory, CERIST Center of Research, Algiers, Algeria  
E – mail: ddjenouri@mail.cerist.dz

‡: Computer Systems Laboratory, Computer Science Department, University  
of Science and technology, Algiers, Algeria  
E – mail: badache@wissal.dz

Abstract

In this paper we deal with an emergent security problem related to mobile ad hoc network (MANET) to which we present a gradual solution. This new problem is node selfishness on packet forwarding. The resource limitation of nodes forming the ad hoc network, particularly the energy limitation, along with the multi-hop nature of this network may cause a new phenomena which does not exist in the traditional networks. To save its energy, a node may behave selfishly, thereby it uses the forwarding service of other nodes, but it does not forward packets for them. This deviation from the correct behavior represents a potential threat against the service availability, which is one of the most important security requirements. Recently, some solutions have been proposed, however, almost all these solutions rely on the watchdog technique [1] which suffers from many problems. To mitigate these problems, we propose a new technique, we call Two hops acknowledgment, whose performance is improved gradually.

Key words: mobile ad hoc networks, security, selfishness, packet forwarding, energy consumption, power control

1. Introduction

An ad hoc network is a temporary infrastructureless network formed dynamically by mobile nodes without the need of any existing centralized administration. The absence of the central infrastructure imposes new challenges. The services ensured by this central infrastructure in traditional networks, such as packet forwarding, must now be ensured by the mobile nodes themselves, which renders the cooperation among nodes an essential requirement.

In some MANETs applications, such as the battlefield or the rescue operations, all the nodes have a common goal and their applications belong to a single authority. For this reason, the nodes are cooperative by nature. However, in many civilian applications, such as networks of cars and provision of communication facilities in remote areas, the nodes typically do not belong to a single authority and they do not pursue a common goal. In such networks, forwarding packets for other nodes is not in the direct interest of any node, so there is no good reason to trust nodes and assume that they always cooperate. Indeed, nodes try to save their precious resources, i.e memory, CPU cycles, bandwidth, and especially the battery power. Recent studies show that most of the nodes energy in MANETs is likely to be devoted to forward packets in behalf of other nodes. For instance, Levente Buttyan and jean-pierre Hubaux simulation studies [2] show that when the average number of hops from a source to a destination is around 5, then almost 80% of the transmission energy will be devoted to packet forwarding.

To take this constraint in charge many power aware protocols have been proposed such as [3, 4], but these protocols do not eliminate the problem due to the complex nature of the network, the users are then permanently anxious about their limited batteries. This paranoia may lead the nodes to
misbehave and tend to be *selfish*, a selfish node regarding the packet forwarding process is a node which takes advantage of the forwarding service and ask others to forward its own packets, but it does not participate in this service. Some solutions have been Recently proposed, however, almost all these solutions rely on the watchdog technique [1] which contains many problems that will be presented latter. To mitigate these problems we propose a new technique which we call *two hops acknowledgment*. In this paper we present a gradual solution based on a new technique to detect the selfish node. The rest of this paper is organized as follows: In the next section we present the related work, and we briefly present the watchdog technique in section 3. After that, the requirements of our solution and a solution overview are respectively presented in sections 4 and 5. Section 6 is devoted to the detailed presentation of our protocol. Finally, section 7 concludes the paper and summarizes the future works.

2. Related work

The emergent problem of selfishness in MANETs has recently received attention among researchers, and some solutions have been proposed. In our previous work [5] we have surveyed these solutions and classified them into two main categories; reactive solutions that aim at detecting the misbehavior when it appears in the network, and preventive solutions which try to inhibit the misbehavior either by motivating nodes to cooperate or by taking measures to prevent packets from being dropped. To the best of our knowledge, Sergio et al are the first who dealt with the problem of selfishness on packet forwarding in MANETs. In [1], they define two techniques called *watchdog* and *pathrather*, the former is to identify misbehaving nodes whereas the latter helps the routing protocol to avoid routing through these nodes. These techniques are used along with DSR [6] to built a misbehavior mitigating routing protocol. The watchdog was used by almost all the subsequent proposed reactive solutions, nevertheless, this technique has many drawbacks, as it will be shown later.

In [7] Yang et al describe a unified network layer solution to protect both routing and data forwarding in the context of AODV. Threshold cryptography based signature [8] and the watchdog technique [1] are at the core of this solution. Nodes in a neighborhood mutually accord participation admissions, and nodes without up-to-date admissions are excluded from any network service. Each node has a token issued by at least k neighbors, this token allows it to participate in the network operations, these neighbors collaboratively monitor it to detect any misbehavior. The token has a period of expiration whose value depends on how long the node has been behaving well, the node renews (updates) the token upon its expiration. This solution prevents nodes which have less than k neighbors to communicate, moreover, it inherits all the watchdog’s drawbacks.

Pietro Michiardi and Refik Molva [9] suggest a generic reputation-based mechanism, namely CORE (Collaborative Reputation Mechanism to enforce node cooperation in MANETs), supposed to be easily integrated with any network function. In this mechanism, each node keeps track of other entities collaboration through both direct observations and information provided from other nodes. In this solution only the positive observations (of well-behavior) are propagated but not the negative ones. The purpose is to provide robustness to the solution and prevent the vulnerability of false accusations propagation which can cause DOS (Denial Of Service) attacks. This approach reduces the potential of learning from observations made by others which can decrease the efficiency of misbehavior detections in the network. Since the solution relies on the watchdog technique, all the watchdog’s drawbacks remains unresolved with this solution.

Another reputation-based solution is proposed by Sonja Buchegger and Jean-Yves Le Boudec [10], they propose a protocol called CONFIDANT (Cooperation Of Nodes Fairness in Dynamic Ad hoc Networks), it relies on the DSR [6] routing protocol which is used as benchmark in their GloMosim-based simulation study to evaluate the new DSR fortified by CONFIDANT.
Unlike the previous reputation based solution (CORE), with CONFIDANT, reliable negative information are propagated beyond the neighborhood. To mitigate the vulnerability to DOS attacks by propagating false accusations, the trust manager is proposed along with the rate function that assigns different weights to each behavior detection type, such that more importance is given to local observations when computing reputation rating.

The simulation results show a significant improvement in term of goodput compared to the standard DSR when this latter is fortified with CONFIDANT. But, all the watchdog’s drawbacks remain untreated in this solution, since the confidant’s monitor component fully relies on this technique.

In the framework of the Terminodes Project [11], Levente Buttyan and Jean-Pierre Hubaux [12] propose a preventive economic-based approach which stimulates the nodes to cooperate, this solution is modeled and analyzed in [2]. They introduce what they call virtual currency or nuglets, along with mechanisms for charging/rewarding service usage/provision. The main idea of this technique is that nodes which use a service must pay for it (in nuglets) to nodes that provide the service. This makes nuglets indispensable for using the network, thus, each node is interested in increasing its stock of nuglets. A way to achieve this is to provide services to other nodes. We think this solution includes some disadvantages. If a well-behaved node is not asked to route enough packets, it cannot send enough packets. Moreover, this technique does not prevent a node with enough nuglets to misbehave, especially if it has not enough packets to send.

Another preventive mechanism is the game theory based approach. In [13] Vikram Srinivasan et al propose a solution to stimulate cooperation based on this approach. In this solution nodes are allowed to refuse the participation in the data forwarding process. This can present a potential risk of the service unavailability. Furthermore, it trusts the ACKs of intermediate nodes, a selfish node may also be reneging and agree to participate in a session and forward packets in order to give impression that it executes accurately the protocol but will not forward packets when it receive them. Therefore, reactive solution is required to resolve this problem.

In [14] Panagiotis Papadimitratos and Zygmunt J. Haas present the SMTP protocol, it prevents the selfishness effects (packets lost) by dispersing packets, and detects it by employing end-to-end feedbacks. These later allow the detection of the routes containing selfish nodes but fails to detect these selfish nodes.

3. Watchdog in a nutshell

Watchdog method is a basic technique on which many further solutions rely. It aims to detect misbehaving nodes that do not forward packets by monitoring neighbors in the promiscuous mode. A node A transmits a packet to B to forward it to C. A monitors B’s forwarding by using the promiscuous mode and listening to all the packets sent in its neighborhood. The watchdog is implemented at A by maintaining a buffer of recently sent packets and comparing each overheard packet with the packets in the buffer to see if there is a match. If so, the packet in the buffer is removed and forgotten by the watchdog, since it has been forwarded on. If a packet has remained in the buffer for longer than a given timeout, the watchdog increments a failure tally for the node responsible for forwarding the packet. If the tally exceeds a certain threshold, the monitor (node A) determines that the node is misbehaving and sends a message to the source notifying it of the misbehaving node. The watchdog technique relies on the assumption that each transmission can be overheard by all neighbors, if no collusion takes place. This is not inevitably correct when the transmission power is not constant, for instance, this may appear (transmission power is not constant) when the power control technique [3] is used by the routing protocol, like in the recently proposed power-aware ones, such as [4, 3].
In this case we remark a serious problem when using the watchdog, namely the possibility of false
detections (false positives in Intrusion Detection Systems terminology). Assume three nodes A, B and
C such that A monitors B’s forwarding to C and B uses controlled power, we also assume the required
power from B to C to be less than the one needed to reach A from B, thereby the packets sent from
B to C will not be received at A. The node A may accuse wrongly B as misbehavior even though it
forwards packets to C. Hence the watchdog fails when the power control technique is employed, the
aim of our proposal is to overcome this problem.

Moreover, this technique cannot detect the misbehavior in many cases. In [5], we have presented
and analyzed all these cases. For space limitation, we just cite them:

1. Partial dropping: node B can circumvent the watchdog by dropping packets at a lower rate than
   the watchdog’s configured minimum misbehavior threshold.
2. Receiver collusion: after a collusion at node C, B could skip retransmitting the packet without
   being detected by A
3. False misbehavior: A node may falsely report other innocent nodes in its neighborhood as
   misbehaving to avoid getting packets to forward.
4. Insufficient transmission power: B can control its transmission power to circumvent the watchdog.
   If A is closer to B than C, then B could attempt to save its energy consumed for forwarding packets
   by adjusting its transmission power such that the power is strong enough to be overheard by the
   previous node (A) but too weak to be received by the true recipient (C).
5. Cooperator misbehavior: B and C could collude to cause mischief. In this case, B forwards a
   packet to C but does not report to A when C drops the packet.

4. Requirements

Our solution is based on the following assumption:

- A security association between each pair of nodes, that is each node has a public key PK, and
  a private (secret) key SK. The PK of each node is known by each node participating in the
  network, whereas the SK is kept secret by each node and can not be broken. This requires a key
  distribution mechanism, which is out of the scope of this paper, anyway a mechanism like [15] or
  [16] can be used.
- Robustness at low layers: the MAC and physical layers are assumed to be robust and tamper
  resistance, this can be ensured by the hardware and the Operating system of each node. However,
  the upper layer including the network layer may be tampered by a selfish or a malicious. That is
  the operations of the lower layers cannot be modified by any node, but this does not mean that
  a node cannot send a falsified packet.
- A source routing protocol is used

5. Solution overview

In this paper we define a new approach and a gradual solution to mitigate the watchdog technique
problem. Each node monitors the forwarding of each packet it sends. To explain the concepts in all the rest of this
paper we suppose without loss of generality that A sends packets to B and monitors its forwarding to C.
We define a new kind of feedbacks that we call *two hops ACK*, it is an ACK that travels two hops, node C acknowledges packets sent from A via node B, this ACK is performed at the MAC level and cannot be tampered. For this purpose, we use the following strategy:

Node A generates a random number and encrypts it with C’s PK then appends it in the packet’s header as well as A’s address. When C receives the packet it gets the number back, decrypts it using its SK, encrypts it using A’s PK and puts it in a two hops ACK. This packet is sent back to A via B, A decrypts the random number and checks if the number within the packet matches with the one it has generated to validate the B’s forwarding regarding the appropriate packet. However, if B does not forward the packet, A will not receive the two hops ACK then it will be able to detect this misbehavior after a time out.

The encrypted random number ensures that B cannot escape from forwarding the packet and tamper A by sending a falsified ACK without relying the packet.

To ensure robustness, we propose that the sending of two hops ACKs is provided implicitly upon the reception of the packet at the MAC layer, that is node C cannot ignore the reception of a packet by not sending an ACK. However, if the node C behave selfishly, and does neither forward the packet nor send the two hops ACK back to A, B could be supposed by A to not forward the packet even if it actually does. Indeed, by implementing the two hops ACK sending at the MAC level, we can ensure the robustness.

As we will see in the next section, acknowledging each packet by a two hops ACK requires an important overhead, we will gradually decrease this overhead, firstly by employing the ordinary MAC ACKs, and secondly by aggregating each set of n two hops ACKs in one.

### 6. The protocol

#### 6.1. Solution 1

**6.1.1. Solution description**

Here we describe the algorithm executed at the monitor of each node i, we note the following operations:

- $R_{Key}$: encrypting R with Key,
- $R^{Key}$: decrypting R with Key,

we also note $P_X$ the public key of node X and $S_X$ the secret key of node X.

The node i keeps the buffer Wait2HopsACK, each entry of which corresponds to a packet whose forwarding is monitored by i. The entry contains the address of the forwarder node and the random number generated by i for the packet. Moreover, for each node j, i keeps a rating on packets it detects their drop at j. If the j’s rating exceeds a given threshold, j will be considered misbehaving. The solution is composed of two parts, the first one is located at the network layer and can be viewed as a sub layer at the bottom of this layer, whereas the second one is located in the MAC layer and is a sub layer at the top of this latter. Figure 1 illustrates this framework.

Each node, except the destination is monitored by the previous node. To monitor its successor, each node i adds the random number it generates for each packet encrypted with its successor’s successor public key along with i’s address to each packet it receives from the routing protocol, and maintains the generated random number as well as the monitored node (i’s successor) address in an entry within Wait2HopsACK. When a packet is received from another node X, i’s MAC component automatically generates and sends X back a two hops ACK after encrypting and decrypting again the random number as described previously. The Network layer component removes the appropriate entry upon the reception of the two hops ACK, and as a timeout is associated to each entry, the lack of the two hops ACK after during the timeout results in the increasing of the rating regarding the appropriate forwarder node.
A gradual solution to detect selfish nodes in Mobile Ad hoc Networks

Figure 1: Solution one framework

Algorithm 1 and 2 describes respectively the network component and the MAC component of this solution.

6.1.2. Discussion

If no packet can be lost due to channel conditions or nodes mobility, this solution allows to detect any misbehavior even if the power control technique is employed, unlike the watchdog. In practice, however, the lack of ACK does not mean directly B’s misbehavior. Hence, like in [1], A does not accuse directly B but it records ratings and use threshold as it is shown in the algorithm. Unlike the end to end ACK, this new mechanism allows to detect the misbehaving node. Moreover, with this mechanism, the cases 2 and 4 of the watchdog are detected, nevertheless, this solution requires an important overhead. If we assume the average path length is H, the communication complexity is: $O(2^*(H-1))$ two hops ACK transmissions for each packet.

Algorithm 1 Network module of solution 1

When receive a packet D from the routing protocol to send to node X (X either the next hop or the destination and i is either the source or a forwarding node):

if (X ≠ D’s destination) then
  R = a generated random number
  Y = X’s successor in the source route
  append (R; i) to D’s header
  add(R, X) to the buffer Wait2HopsACK
end if
send D to X

When receive a packet D from the MAC protocol sent by X:

if X ≠ D’s source then
  remove the random number generated by X’s predecessor from the header along with the corresponding node address
end if
send the packet to the network layer protocol

When receive a two hops ACK packet TwoHopsACK from the MAC layer component

$R’ = TwoHopsACK.Rand^{t’}$
if (R’, TwoHopsACK.sender) ∈ Wait2HopsACK then
  remove (R’, TwoHopsACK.sender) from Wait2HopsACK
end if

When a timeout out of a Wait2HopsACK entry (R, X) is exceeded
increment the rating regarding node X
if X’s rating > threshold then
  consider X as a misbehavior
end if
Algorithm 2 MAC layer located component of solution 1

when receive a packet D sent by X from the MAC protocol
if X ≠ D's source then
   Y = X's predecessor in the source route
   Get the random number R generated by y
   \( R' = R^y \)
   \( R'' = R''^y \)
   construct a two hops ACK packet TwoHopsACK
   TwoHopsACK.Rand = R''
   TwoHopsACK.sender = i
   TwoHopsACK.dest = Y
   send two hops ACK to X
end if
pass the packet up to the network component

when receive a two hops ACK packet TwoHopsACK
if TwoHopsACK.dest ≠ i then
   TwoHopsACK.sender = i
   forward TwoHopsACK to TwoHopsACK.dest
else
   pass the packet up to the network layer component
end if

6.2. Solution 2:

6.2.1. Solution description

To decrease the overhead, we propose that node C use the MAC ordinary ACK where it integrates the two hops feedback to acknowledge the B's forwarding. Hence, unlike in the previous solution, the two hops forwarding costs only one extra forwarding from B to A, instead of two. For this purpose, a little extension is required for the MAC header and the MAC ACK packet to include the two hops ACK information. We propose to extend the ACK packet with the following fields too:

TwoHopsACK: A flag indicating if the ACK contains a two hops ACK.
Rand: it contains the random number
SecondDest: the destination of the two hops ACK
We also add the following fields to the packet’s MAC header:
TwoHopsACK: A flag indicating if the packet requires a 2 hops ACK
Rand: It contains the random number
TwoHopsSrc: The node to which a two hops ACK is required.

There is no change required to the network component, nevertheless the MAC component needs to be changed. The packets provided by the physical layer have to be caught and treated by our protocol before sending any ACK back, hence in the scheme of this solution the MAC component can be viewed as a component within the MAC protocol instead of being as a sub layer over it, like shown in figure 2. Upon the reception of a packet sent by X, node i encapsulates the required information to acknowledge X forwarding at X’s predecessor. When X receives this ordinary ACK, it checks the TwoHopsACK flag and realizes that the packet contains a two hops feedback, it consequently constructs a two hops ACK and sends it back to its predecessor, the MAC component of this latter passes the two hops ACK to the network component to validate X’s forwarding and remove the appropriate entry from the buffer. The Algorithm 3 describes the MAC component of this solution

6.2.2. Discussion

The complexity of the previous solution is reduced and divided by 2, since half of the two hops ACK separate transmissions in the previous solution are eliminated in this one, i.e the communication complexity is \( O((H-1) 2) \) two hops ACKs transmissions for each packet.
Algorithm 3 MAC component of solution 2

When receive a packet $D$ sent by $X$

if $(D.MACHeader.TwoHopsACK == true)$ then

$R = D.MACHeader.Rand$\text{\textsuperscript{2}}$

$R' = R \oplus D.MACHeader.TwoHopsSrc$

construct an ACK packet $ACKpack$

$ACKpack.TwoHopsACK = true$

$ACKpack.Rand = R'$

$ACKpack.SecondDest = D.MACHeader.TwoHopsSrc$

the other ACK’s fields have to be filled out by the MAC protocol

send the ACK to $X$

else

send an ordinary ACK if the packet requires an ACK

end if

pass the packet up to the network component after doing the handling required by the MAC protocol (moving the MAC header, frames defragmentation, etc)

When receive packet $D$ from the network layer

Do the required processing required by the MAC protocol (fragmentation, making the MAC header, etc)

if $(D’s source \neq i)$ then

$D.MACHeader.TwoHopsACK = true$

$D.MACHeader.Rand = \text{the random number generated and encrypted by the network component}$

$D.MACHeader.TwoHopsSrc = i’s$ predecessor

else

$D.TwoHopsACK = false$

end if

forward the packet

When receive an ACK packet $ACKpack$

if $(ACKpack.TwoHopsACK == true)$ then

construct a two hops ACK packet $TwoHopsACK$

$TwoHopsACK.Rand = ACKpack.Rand$

$TwoHopsACK.dest = ACKpack.SecondDest$

$TwoHopsACK.sender = I$

send two hops ACK to $ACKpack.SecondDest$

end if

do the handling required by the MAC protocol

When receive a two hops ACK packet $TwoHopsACK$

pass the packet up to the network layer component

Figure 2: Solution 2 framework
6.3. Solution 3

6.3.1. Principle

The aim of this solution is to more decrease the overhead. In the previous solution, the ordinary MAC ACK was used to carry the two hops ACK in the first hop which reduced the solution 1 overhead as much as the half, anyway an extra ACK is required on the second hop for each packet, which is somewhat important. Now, we propose to aggregate each n two hops ACK on just one, that is the monitoring node asks a two hops ACK for each n successive packets, which we call a block of n packets. This solution has the same framework as the previous one, the algorithm, however, is different.

6.3.2. Network component

Each node i keeps two rating regarding each other node X, the first one counts the number of packets lost at the node X, and the second one counts the number of two hops ACK waited but not sent from X, thereby a threshold regarding each rating is required. i maintains a threshold for each rating, as well a table n\text{count}[X][Y] which safeguards the number of packets sent to X to forward to Y waiting for a two hops ACK, an entry in the table is required for each \{(X,Y) \setminus i\} monitors X forwarding to Y.

The Wait2HopsACK entry is extended, and it includes the number of packets (n) in addition to the random number and the forwarder address. The packet header is also extended to include the lastpacket flag which indicates that the packet is the last one among the block of n packets to be acknowledged by the two hops ACK, therefor each time i sends n packets, it activates the lastpacket flag of the n\text{th} packet. To tolerate the lost of the packet containing lastpacket flag, the random number is included in each packet and not only the n\text{th} one, thereby the MAC component will be able to partially acknowledge the block of n packets in spite of the n\text{th} packet lost, as we will see later. When i receives a two hops ACK, it removes the appropriate entry like in the previous solution, there is extra computing, however, since this ACK is for n packets and not only one. The two hops ACK carries the number of packets well received which may be less than n, in this case, i decreases the packet lost rating by the number of packets lost, i.e (n - the number of packet acknowledged). When the timeout of a Wait2HopsACK buffer entry is triggered, i increments the ACK missed rating as well as the packet lost rating regarding the corresponding node. Algorithm 4 describes this Network layer component.

6.3.3. MAC component

An extra field is added to both the two hops ACK and the ordinary ACK. This field indicates the number of packets received among the block of n packets sent. We call this field NbrPackets, i’s MAC component maintains two tables, LastRand for each two hops neighbor and NbrPack for each neighbor. LastRand[x] contains the last random number sent by x as a second hop neighbor, whereas NbrPack[x] the number of packets received from x and not acknowledged, it is initialized to 0. We point out that a two hops neighbor of the node i is the node located at two hops from i, i.e a neighbor’s neighbor. NbrPack[x] is incremented each time i receives a packet from X and it is re-initialized from a block reception to another, when the node receives the last packet of a block (the one containing the lastpacket flag activated) or when it receives a packet with a random number different from the current value of LastRand[x]. It implicitly includes the two hops acknowledgment information in the ACK as in the previous solution, along with the number of packets received for the last block sent included in NbrPack[x]. Thereby, the algorithm takes into account the possibility of losing the critical last packet in a block.

Except on the handling of the first event, this component does not differ too much from the corresponding one in the previous solution. Algorithm 5 clarifies this component.
Algorithm 4 Network layer component of solution 3

When receive a packets D from the routing protocol to send to X
  if (X ≠ D’s destination) then
    Y = X’s successor in the source route
    if (\( n_{\text{count}}[X][Y] = 0 \)) then
      \( R = \) a generated random number
    end if
    \( n_{\text{count}}[X][Y] = (n_{\text{count}}[X][Y] + 1) \) mode n
    if (D is the last packet to X) then
      \( n_{\text{count}}[X][Y] = 0 \)
    end if
    if \( n_{\text{count}}[X][Y] = 0 \) then
      lastpacket=true
    else
      lastpacket=false
    end if
    append \((R, PY, i, \text{lastpacket})\) to D’s header
    add \((R, n_{\text{count}}[X][Y], X)\) to the bufer Wait2HopsACK
  end if
  send the D to X

When receive a packet D from the MAC protocol sent by X
  if (X ≠ D’s source) then
    remove the random number generated by X’s predecessor from the header along with the corresponding: node address, number of packets to acknowledge, ACKFlag
  end if
  send the packet to the network layer protocol

When receive a two hops ACK packet TwoHopsACK from the MAC layer component
  \( R’ = \text{TwoHopsACK.Random} \)
  if \((R’, \text{TwoHopsACK.sender}) \in \text{Wait2HopsACK}\) then
    remove \((R’, \text{TwoHopsACK.sender}, n)\) from Wait2HopsACK
    increment the packet lost rating regarding TwoHopsACK.sender by \( n\text{-TwoHopsACK.NbrPackets} \)
    if X’s packet lost rating > packet lost threshold then
      consider X as a misbehavior
    end if
  end if
  \end if

When a timeout out of a Wait2HopsACK entry \((R, n, X)\) is exceeded
  increment the ACK missed rating regarding node X
  increment the packet lost rating regarding X by \( n\)
  if X’s packet lost rating > packet threshold OR X’s ACK missed rating > ACK missed threshold then
    consider X as a misbehavior
  end if


Algorithm 5 MAC component of solution 3

When receive a packet D sent by X

if (D.MACHeader.TwoHopsACK == true and x ≠ D’s source) then
  y=D.MACHeader.TwoHopsSrc
  if (NbrPack[y]==0) then
    LastRand[y]=D.MACHeader.Rand
  end if
  if (D.MACHeader.Rand== LastRand[y] or D.MACHeader.LastPacket == true) then
    if (D.MACHeader.LastPacket == true and D.MACHeader.Rand == LastRand[y]) then
      NbrPack[y] = NbrPack[y]+1
      R = LastRand[y]*R₀
      ACKpack.TwoHopsACK = true
      ACKpack.Rand= R
      ACKpack.SecondDest=y
      ACKpack.NbrPackets = NbrPack[y]
    else
      NbrPack[y]=1
    end if
    LastRand[y]=D.MACHeader.Rand
    the other ACK’s fields have to be filled out by the MAC protocol
    send the ACK to X
  else
    NbrPack[y]=NbrPack[y]+1
  end if
else
  send an ordinary ACK if the packet requires an ACK
end if

pass the packet up to the network component after doing the handling required by the MAC protocol (moving the MAC header, frames de/fragmentation, etc)

When receive packet D from the network layer

idem to solution 2, just add the instruction: D.MACHeader.LastPacket = D.IPHeader.lastpacket anywhere within the if(D’s source ≠ i) statement

When receive an ACK packet ACKpack

idem to solution 2, just add the instruction: TwoHopsACK.NbrPackets = ACKpack.NbrPackets anywhere within the if(ACKpack.TwoHopsACK == true) statement

When receive a two hops ACK packet TwoHopsACK

quite idem to solution 2 (pass the packet up to the network layer component)
A gradual solution to detect selfish nodes in Mobile Ad hoc Networks

6.3.4. Discussion

The complexity of this solution is \( n \) times less as much as the previous one, that is \( O \left( \frac{(H-1)}{n} \right) \) two hops ACK transmissions. This has no effect on the latency, since packets are relayed as in the previous solution. The two hops ACK is, however, delayed which may delay a little bit the detection of the misbehavior. We think this is meaningless, as the detection of the misbehavior requires many packets lost detections and may take an important time. The cost of this improvement is the fields added to the packets, and the extra memory required to hold data structures at the nodes. However, The fields added to the packets are limited, even negligible, the data structures added to the nodes are also limited and linearly proportional to the nodes’ communication.

7. Conclusions and perspectives

In this paper we have presented a new technique which allows to detect the misbehaving node that fails or is unwilling to forward packets, this technique is applicable even though the power control technique is used. It also allows to detect the misbehaving nodes in many cases when the watchdog fail to do so. As we have seen our first proposed solution have an important overhead, therefore, we have gradually improved it and decreased the required overhead.

In our further research we plan to proof the correctness of our protocol, to conduct an analytic study, and to evaluate its performance by simulation. We also plan to complete the proposal by given rigorous definitions to the threshold used, to define actions that have to be taken when a node is accused as a selfish, and to define mechanism allowing nodes to exchange their knowledge regarding nodes that behave selfishly.

References


