Cooperation in ad hoc networks: Enhancing the virtual currency based models

Michaël Hauspie, Isabelle Simplot-Ryl

Cooperation in ad hoc networks: Enhancing the virtual currency based models

Michaël Hauspie
INRIA Futurs/L.I.F.L. CNRS UMR 8022
Université de Lille I, Cité Scientifique
F-59655 Villeneuve d'Ascq Cedex, France
Email: hauspie@lifl.fr

Isabelle Simplot-Ryl
L.I.F.L. CNRS UMR 8022
Université de Lille I, Cité Scientifique
F-59655 Villeneuve d'Ascq Cedex, France
Email: ryl@lifl.fr

***Abstract*— Cooperation of nodes is a main issue of civilian applications of mobile ad hoc networks. More precisely, mobile nodes with few resources may try to maximize the benefits they get from the network without participating to its services (*e.g.* without forwarding packets of other nodes). A solution to this problem is to use virtual currency mechanisms: nodes have to pay to send packets and they are rewarded when they forward packets of other nodes. Several papers of the literature have shown that such models enforce individual node cooperation. We address in this paper two main issues of these models that have been left out: the computation by the sender of the price it will be charged and the possibility to reward nodes depending on their charge or resources. We show that it leads to a better load balancing of the network.**

I. Introduction

An ad hoc network is a collection of wireless mobile hosts forming a spontaneous network without the aid of any fixed infrastructure. They have potential application in civilian and military environments such as disaster relief, conference, wireless office, and battlefield. Ad hoc sensor networks for monitoring environment are also being deployed.

In an ad hoc network a message sent by a node reaches all its neighboring nodes that are located at distances up to the transmission radius. A widely accepted basic graph-theoretical model for ad hoc networks is the unit graph model, defined in the following way. Two nodes A and B in the network are neighbors (thus joined by an edge) if and only if the Euclidean distance between their coordinates in the network is at most R, where R is the transmission radius which is equal for all nodes in the network. Due to the limited transmission radius, the routes between two nodes are usually created through several hops.

Most protocols developed for ad hoc networks usually consider that nodes are cooperative. Indeed, in military or emergency applications such collaboration can be assumed. However, ad hoc networks also have a great potential in civilian applications (for extending wireless internet connectivity for example) where all nodes typically do not belong to a single authority. In such context, malicious nodes could try to cheat the network. A main cheat action would be to not participate to routing by not forwarding any packet. This cheating form is called selfishness, where nodes try to maximize their own welfare for example to save some battery power. Even non malicious nodes may yield to temptation to avoid cooperation since the cost of the participation to network functionalities may be very high for individual nodes: a simple estimation says than when the average number of hops from source to destination is 5 in an ad-hoc network then 80% of the energy spent by a node to send packets is dedicated to packet forwarding for other nodes.

The first work to point out the network service degradations caused by selfish nodes was [1]. This paper shows that 10 to 40 % of misbehaving nodes cause 16 to 32 % of degradation of the average throughput of the network. The watchdog approach proposed in this paper allows route discovery protocols to avoid selfish (or other kinds of misbehaving nodes) nodes and thus to increase the network throughput in a relevant manner. Anyway, the goal of this work was not to enforce nodes to cooperate. Several algorithms exist in the literature to prevent such selfishness, mostly based on reputation mechanism and virtual currency mechanism. The idea is most of the time either to reward cooperating nodes or to punish misbehaving nodes.

Virtual currency mechanisms are more robust to attacks. Moreover, they could easily be adapted to network using directional antennas whereas reputation mechanisms are mostly based on monitoring communication of other nodes. Thus, we focus on virtual currency mechanisms and more precisely on enhancing two well-known protocols: the nuggets [2] and Sprite [3]. These protocols have been shown to be efficient in enforcing node cooperation and robust to many attacks. Anyway, in our opinion, two main issues of these protocols have not yet been achieve: the algorithm lacks the computation of the exact cost the sender has to pay to send a packet, and the evaluation of the impact of a variable cost on the network. The variable cost of the forwarding service is left out in most of the papers where each node is paid the same amount of money regardless of its state (energy, memory, network use...), or it requires a large amount of knowledge of the network and thus are only adapted for very small ones like in [4].

In this paper, we propose to enhance the virtual currency based model by introducing the possibility for each intermediate node to announce the price it wants to be rewarded. Moreover, we introduce the use of a multiple routes discovery

protocol so that the sender can select the cheapest route from a set of possible paths leading to the destination. We propose an evaluation of the impact of this choice to the load of the network.

The paper is organized as follows. Section II describes the nugget and the Sprite models. Section III introduces the route cost computation in the route discovery protocols to pre-calculate the cost of a route. Section IV introduces the use of a route selecting mechanism based on the route cost and discusses the multicast case. Section V gives an evaluation of our propositions. At last, we conclude in Section VI.

II. Existing models

We first present two main virtual currency models of the literature that we use for discussion. In both algorithms, the idea is to apply economical models to the network communication. Each communication has a cost, nodes have to pay to send/receive packets, and nodes that forward other nodes' packets earn some virtual currency. Nodes are thus enforced to cooperate to the packet forwarding to be able to send their own packets.

A. The nugget model

The nugget model has been introduced by Buttyán and Hubaux in [2]. In this model, the virtual currency is called the nuggets. As most of the works on nuggets, we will focus on the packet purse model (PPM) which avoids network overloading and which suppresses the risk for intermediate nodes to lose nuggets. The only drawback of this model is that it does not work for multicast. In the PPM model, a node that wants to send a packet has to load it with nuggets to pay its transport. Then, each forwarding node uses some nuggets of the packet to pay itself before forwarding it. If a packet has not enough nuggets to reach the destination, it is discarded. This model has been shown to enforce node cooperation, see for example [5] for an evaluation.

The model is simple, but the difficulty is to avoid nugget tampering. Nodes may be tamped to forge nuggets, to reuse them, to take nuggets from the packet without forwarding it, and so on. Thus, the problem of respecting the forwarding protocol is replaced by the problem of respecting the nugget protocol. The solution proposed by Buttyán and Hubaux is to introduce a hardware tamper-proof module, called the security module, that is in charge of the nugget management. A new header is added to packets and the action of the security module occurs between the MAC layer and the Network Layer in each direction.

The protocol is based on some hypotheses:

- the security modules are tamper-proof,
- there exists a public key infrastructure (*i.e.* security modules can authenticate each others),
- the neighbors change slowly: a node can keep a view of neighborhood with some "Hello" protocol.

Each security module has a unique identifier, a public and a private key, a nugget counter, and a list of neighbors (obtained thanks to the "Hello" protocol). Moreover, the security module shares a secret with each neighbor in order to secure communications with this neighbor. Thus, the crypt header secures the nugget management.

The security headers are used for one hop communications. A security header starts with the identifier of the sender's security module. It contains the identity of the next hop security module, the nuggets of the packet and a sequence number shared by these two security modules to avoid packet replays (this counter is initialized with a random value during the "Hello" protocol). Another counter is added to the header to ensure that a node that temporally stops to cooperate will pay it later. As the model is supposed to be used in the context of multi-directional antennas, the same header contains the acknowledgement of the packet for the security module of the previous hop. As security modules do not trust their hosts, they only increase their own nugget counters when receiving the ACK for a packet. The protocol could be adapted for directional antennas using a separated ACK message.

As conclusion, the nugget model has been shown to be very efficient in enforcing cooperation of nodes. Nevertheless, some points have to be improved:

- the PPM model does not work for multicast,
- the way of choosing the price to pay to intermediate nodes is left out in most of the papers.

The last point is – to our mind – a relevant issue. First for technical reasons since the source node needs to know the needed number of nuggets otherwise the packet may be discarded or the excess of nuggets may be lost. Second, as mentioned in a lot of papers, the price could depend of several factors as for example the battery level of the forwarding nodes. We advocate that such consideration would increase the lifetime of the network.

B. Sprite

In [3], the authors propose Sprite, a cheat proof algorithm that was designed to get rid of the need of a tamper-proof security module. As seen before, if virtual money is included in the message, we need the routing algorithm to be run by a trusted hardware. Sprite does not include money in the message and lays on a central entity, which acts as a banking service. This entity is called the Credit Clearance Service (CCS) and is in charge of managing the nodes' money. Sprite relies on the fact that each node could have access to a fast connection so that it can communicate with the CCS at a low cost, and on a public key infrastructure. This connection does not need to be always available but just as often as needed for the node to get enough virtual money to send its message until the next connection to the CCS.

When a node wants to send a message m, it sends the message, the path, a sequence number and a digital signature (RSA for example) on these data. Each node of the path is able to check if the message is valid using the public key of the source and a record of sequence numbers. If it decides to forward the packet, it keeps a receipt of the message. So that the receipt size keeps small, it consists of a message digest, which is function of the message, the path, the sequence

number and the signature. Using this information, the CCS is able to check if the receipt corresponds to a valid message. Then, the CCS rewards the forwarding node and charges the sender.

Sprite also avoids collusions and false receipts. If a node colludes with the sender, it could be paid a behind-the-scene price by the sender and thus save money for the sender by not submitting its receipt. To counter this, the CCS charges the sender an extra amount of money if the destination does not report the receipt of the message. To prevent false receipts to be made, the amount of money given to a node that reports a receipt is greatly reduced if the destination does not report the corresponding receipt.

In the multicast case (or in the route discovery case, which can be viewed as a special case of multicast), nodes store the route request as a receipt. If they decide to forward the route request, hence if they decide to be a valid forwarder for the route, they append their own address to the route request and sign the extended message. When the CCS computes payment, it rejects a route request if any of its signatures is invalid. Furthermore, if the route request is part of another route request submitted by the same node, the former one is rejected. Finally, the CCS computes a tree based on the accepted route requests. The sender shall pay more money to the nodes that are non-leaf of the tree than to the nodes that are leaves. An even smaller amount of money is paid for nodes that are not included in the tree.

The main advantage of the sprite algorithm is that it does not rely on a tamper-proof hardware and thus can be used in a more general case. On the other hand, this algorithm has two main drawbacks. First, to motivate the nodes to send receipts, the algorithm takes as hypothesis that the nodes can have access to a fast and cheap connection to the CCS. Second, they only use static costs, which are identical for every node of the network. In practice, forwarding a message will not cost the same thing for all nodes. Moreover, the cost of forwarding for a single node will not be the same at different times because of energy or memory needs for example.

III. Route cost computation

The computation of the cost of the forwarding service is left out in most of the papers dealing with virtual currency. It is often considered as a constant (*e.g.* in the simulation of [5] for example, each node forwarding a packet is given one nuggets but the sender pay a constant number of nuggets, or in [3] each intermediate node is paid a constant α except if the message does not reach its destination) even if the possibility of using factors like the remaining battery power or the charge to compute the price of a service is a major argument for the use of virtual currency mechanisms. To our knowledge, the paper which is the closest to what we propose is [6]. In this paper, each node has a battery utility function and a nugget utility function. When receiving a packet, a node compute a payoff depending on these two functions and on the number of nuggets it is offered to forward the packet. If the payoff is positive, it forwards the packet else it drops it. Two models are presented, one with a fixed per hop charge and one with auctions. In both model, the number of nuggets that a source loads in a packet results from an estimation: a local forwarding cost (the number of nuggets the source would need to forward the packet in the first model and the result of auctions in the second one) and the estimated number of hops toward the destination. This estimation does not ensure that the packet will reach the destination.

The problem for the sender to know the exact price of the forwarding (on the total path) is a “technical” point which seems to us to be very relevant: in an heterogeneous network, the prices required by different nodes to forward the same packet may be very different depending of their current state. In the nugget model, if a packet does not contain enough nuggets, it can be dropped and if it contains too many nuggets, they are lost. In Sprite, as the payment is done after the arrival of the message to its destination, knowing the price of the route allows the sender to be sure that he has enough money to pay all the forwarders. Indeed, if the sender cannot afford the cost of its message, other nodes will not be paid for their action and thus the CCS could blacklist the sender for example. Knowing the cost of a forward by advance give the opportunity to the sender to avoid bad action against it due to its lack of money.

We propose to combine the virtual currency mechanisms with the route discovery protocol. Thus, a node that has a route towards a destination will know the exact price it will be charged for each packet sent on this route.

A. Route price discovering

The idea is to use the route discovery protocol to compute the cost of sending a packet through the discovered route (from now on we call this “the route cost”). Each node registered in the route will announce the price it wants to be paid. This price may depend on the battery level of the node, its load, or any other factor. We suppose the existence of a mechanism allowing each node to evaluate the price it wants to be paid to forward a packet (different nodes may use different price function).

We consider the DSR (Dynamic Source Routing) protocol [7] as a basis of our work but other route discovery algorithms could be adapted the same way. Let us recall the basic idea of this protocol that is sufficient to explain our ideas:

- when a node is looking for a route, it sends a ROUTE_REQUEST,
- when a node, which is not the destination, receives a ROUTE_REQUEST message it has not received before, it adds itself to the route and resends the message,
- when the destination node receives the message, it sends a ROUTE_REPLY message containing the route. As we consider a network with symmetric links, we can consider that the ROUTE_REPLY uses the discovered route in the reverse direction.

We simply propose to add the route cost with the route returned to a node that has initiated a route discovery. The cost is computed in the ROUTE_REPLY to be included in real routes and not in all the ROUTE_REQUEST messages. Thus:

- we include a cost counter in the ROUTE_REPLY message that accumulates the cost of the route, and a timeout that corresponds to the lifetime of the proposed price. As we cannot assume the presence of a global clock, the timeout is given in number of forwarded packets on this route,
- each node that receives the ROUTE_REPLY, increases the counter by the price it wants to be paid, and computes the minimum of the timeout of the ROUTE_REPLY and the timeout it proposes before re-sending the message,
- when the source node receives the ROUTE_REPLY message, it contains the route, and the route cost valid for a given number of packets.

Each node has to keep the price it has proposed for a given route and the validity counter that will be decreased each time a packet is forwarded on this route.

For example in Figure 1, `S` sends a ROUTE_REQUEST message to discover a route to `D`. Due to the broadcast, `D` receives two possible routes `SAB` or `SCEFG` and chooses the shortest one. Its sends a ROUTE_REPLY message containing the route on the reverse route with a route cost and a timeout initialized to `0`. Then, each node of the route adds to the cost the price it asks to forward the messages: `B` proposes `5` for `10` packets and `A` proposes `3` for `9` packets thus it forwards the route with a timeout of `9`. The node `S` now knows the exact price it will have to pay for each packet to `D`.

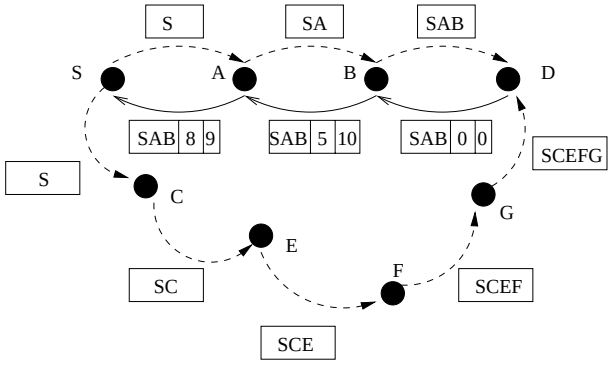


Fig. 1. Inclusion of the route cost in the route.

The over cost of new information is very low: only two counters are added to the ROUTE_REPLY message. The problem is now to avoid nodes to cheat: nodes have to be paid the price they have announced.

B. Implementation on the nugget protocol

It is necessary to avoid nodes to take more nuggets from a packet that they have announced. We propose to include all the necessary manipulations in the security modules for two reasons:

1) to avoid nodes to cheat,
2) because the nugget counter is encapsulated in the security module so all the nugget manipulations have to be made by the security modules.

The difference between our solution and the original algorithm is the addition of a security module header to the route discovery messages too. This header is used in the ROUTE_REPLY messages: it only contains the counter representing the route cost and the timeout. (Let us recall that nodes are not charged nor rewarded for route discovery messages in the nugget model). Then:

- when a node generates a ROUTE_REPLY message, it adds a security module header containing its identifier, the identifier of the next security module in the route, the route cost, the timeout initialized to 0, and a checksum (CS),
- each security module adds the cost it wants to be paid in the security module header and computes the new timeout before sending the packet to the next node in the route,
- each security module maintains a cost table containing the costs it has proposed for routes associated to timeouts. As long as the route is active or the timeout is not exceeded, the security module forwards packets for the same cost,
- when a node receives a packet that it has to forward, it checks in its cost table if there exists a valid cost for this route, then, it takes the announced number of nuggets out of the purse and sends the packet to the next node. Otherwise, it drops the packet and sends a ROUTE_TIMEOUT message to the source.

Let us notice that this solution does not imply any modification of the DSR protocol.

C. Implementation on the Sprite protocol

As we could not rely on a security module, adding variable cost in the Sprite protocol must be done with care. We must add a mechanism which ensures that the price proposed by a node to the sender will be the same price that this node will request to the CCS.

We propose the following protocol:

- when replying to a route request, each node n_i forwarding the reply will add a couple $((p_i, nbp_i), s_i)$ to the reply where p_i is the price proposed by n_i for forwarding each message belonging to this route. The node also give a number of packet nbp_i which represents the validity period for this price. So that the sender can verify that the price has been given by the node n_i (and thus not forged by another node in order to cheat the sender), n_i also computes a digital signature s_i on (p_i, nbp_i),
- when sending a receipt to the CCS, a node n_i also sends $((p_i, nbp_i), s_i)$ so that the CCS knows how much n_i must be paid,
- when the sender gets a connection to the CCS, it submits a new kind of receipt which embeds a couple $((p_i, nbp_i), s_i)$ for each node n_i in the path. This set is used by the CCS as a validation of the request of the other nodes. As the node that proposed it has signed each price, the CCS can verify that the sender tells the truth. If the sender does not send this receipt, the CCS trusts the receipt submitted by the forwarding nodes. Thus, if

the sender does not want to be cheated, it must send the receipt.

All other mechanism of Sprite remains to keep the protocol cheat proof. The α value, which is a constant value in the original protocol, is replaced by the price proposed by each forwarding node.

IV. Using cost information in routing

The above-proposed solution allows the sender to know the exact cost it will be charged to send a packet before sending it. It also allows the forwarders to be paid according to what their action will really cost them, thus motivating them to participate to the network life. The problem left by such protocol is that the sender does not have the choice of which route to take: using protocol like DSR leads to choose the shortest path or path on other criteria but these algorithms are most of the time determinist. Indeed, the route returned by the route reply is not necessarily the cheapest one. To let the sender choose route, we can use a multiple routes discovery protocol. Then, the sender knows more than one route to the destination and thus can select the path regarding its own criteria, for example the cheapest one.

Moreover, this kind of protocol could take into account that some nodes of the networks are over used. Indeed, if those nodes propose an expensive cost because of their network load, they are not likely to be used as forwarder and therefore, see their overhead lowered. As both protocols (nuggets and sprite) use route discovery, they can easily be adapted for using a multiple routes discovery algorithm. We propose in this section a solution, which allows the sender to choose between several routes and to select the cheapest one. We advocate that this solution increases the network lifetime: the network load is better balanced; nodes having more battery will be more solicited.

A. Finding multiple paths to destination

To discover multiple routes, we use a protocol that we presented in [8]. The protocol is based on the blind flooding protocol [9] and is improved by two mechanisms to counter the main drawbacks of this naive protocol for our purpose.

Blind flooding protocol is known to generate few paths because each node is allowed to forward a packet once and only once and thus disabling potential better routes. To solve this issue, we allow each node to forward the same packet more than once and thus potentially enable new routes. To limit the overhead generated by such a flooding, the forwarding nodes are selected so that they are close to the source and the destination.

The protocol works in two steps. First, the source sends a route request to the destination. When the destination receives the request packet, it forges a reply packet by including its distance to the source, a set of sequence numbers and other typical broadcast informations such as hop count, `TTL` and so on.

Let $d(s,d)$ be the distance between the source and the destination, h the number of hops followed by the packet and $d(s,u)$ the distance between the client and the node u. Every node u receiving a reply runs the following algorithm:

1) if the node has already forwarded a packet with the same sequence number, the packet is discarded,
2) if h + $d(s,u) > d(s,d)$ + k (where k is a protocol's parameter), the packet is discarded,
3) if the h is equal to half the distance between the source and the destination, the node selects a random sequence number in the set of sequence numbers included in the packet and rebroadcast the packet using this sequence number.

The first rule of the protocol is similar to the blind flooding. The second one is used to *target* the source with the broadcast. For this rule, we need to know the distance between every node and the source node. This information can be obtained from the request packet if the source sends it using an optimized broadcast [10], [11], [12]. Each node receiving the request – and deciding to forward it – remembers its distance to the source.

The third rule is used to generate a bit more packets and thus, more paths. As the destination sets how many sequence numbers to use, it can control the overhead generated by the reply. This overhead can also be controlled by changing the value of k. If k is large, there are more paths and paths are longer. But, on the other hand, there are more nodes that forward the reply and thus, more overhead.

With our sequence number change policy and our limited rebroadcast, we are able to compute an almost large paths set by generating a small overhead.

More details on this multipath discovering protocol can be found in [8].

B. Algorithm

We use exactly the same idea than the one developed in section III-A in the case of DSR: each node forwarding the packet inserts the amount of virtual money it wants to be rewarded to forward the messages along this path. When receiving all the paths, the sender is given a large amount of choices so that it can select the best route for its needs (cheapest, shortest...). Each intermediate node adds a local timeout on each route in its cost table to be able to erase routes that will not be chosen.

If we come back to the example of Figure 2, the node `S` will now receive the first route with a cost of `8` but also the second one which is longest but which costs only `4` as seen in Figure 2.

The advantage of this solution is to let nodes decide the price they require at a given moment to forward a packet. Each sender is able to select the route it will use to optimize one of the factors: the route cost, the route length, or the route timeout. A node that has a lot of charge can increase the price it wants to be rewarded at each new path request. Then, source nodes may be tempted to choose longer but cheaper paths and thus use forwarding nodes that are not overloaded. We show in the following section that this solution increase the total lifetime of the network.

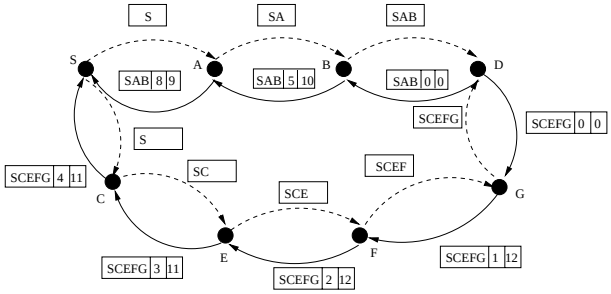


Fig. 2. Multiple paths including cost.

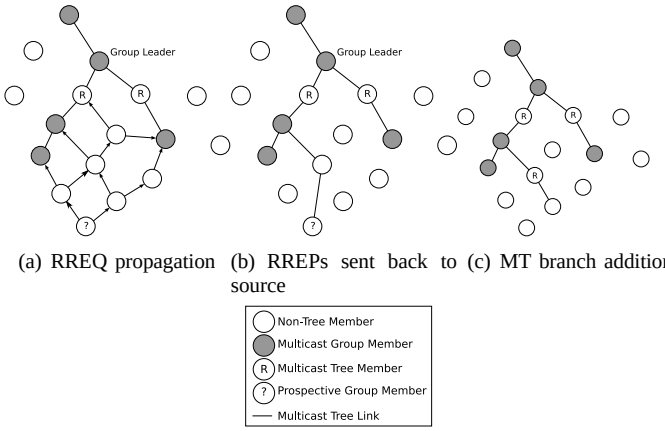
C. Using cost information for multicast routing

Computing the route cost before sending any data messages allows us to use the same virtual currency mechanism for multicast algorithms.

The easiest case is the Sprite algorithm, as the authors describe a multicast extension to their protocol in [3]. The extension they propose is totally compatible to what we propose to add to the algorithm in section III-C. Thus, we can use our extension of Sprite for multicast operations.

For the nuggets algorithm, the modification is also almost straightforward. We just need a multicast protocol. Let us consider for example the MAODV protocol [13].

1) Original algorithm: The MAODV algorithm is based on a multicast tree (MT), which is built using a route request (RREQ), route reply (RREP) scheme. When a node wishes to join a multicast group, it sends a *join* RREQ message to the group. This *join* RREQ can be either unicast or broadcast, depending on the informations available to the prospective node at the time of its request. Figure 3(a) illustrates the propagation of the RREQ message if broadcast is used.



(a) RREQ propagation (b) RREPs sent back to source (c) MT branch addition

Fig. 3. MAODV Multicast join operation.

Any node forming the actual multicast tree, which receives the *join* RREQ, sends back a RREP to the node that wants to join the group using the classical AODV unicast algorithm (see Figure 3(b)).

Upon receiving all the RREPs, the prospective node can select the shortest path to one of the member of the multicast tree. It then sends a multicast route activation message (MACT) back to the node that has sent the RREP. Each node on the path from the prospective node to the multicast tree member becomes part of the multicast tree and will be used to forward messages sent to the multicast group.

2) Sending a message to the multicast group: To send a message to a multicast group, a node must send its message to a member of the group and then, this member will use the multicast tree to forward the message to the other members of the group. Then, the total number of nuggets that a node will have to pay to send a message to the multicast group is:

$$C = C_r + C_{MT} + C_m,$$

where C_r is the cost of unicasting the message to the nearest multicast tree member and C_{MT} the cost of the multicast tree (*i.e.* the sum of the nuggets that each multicast tree member which is not a leaf will claim for forwarding a message to the multicast tree). C_m is the number of nuggets asked by the first multicast tree member reached. This cost is equal to 0 if the first node reached is not a leaf (as it is already included in C_{MT}).

Each node belonging to the multicast tree keeps the cost C_{MT}. When a node wishes to send a packet to multicast group, it sends a RREQ packet to the multicast tree. The modification of the previous protocol lies in the RREP packet. Each node initiating such a packet must add a security header exactly as it did in the unicast case. The route cost is initialized to C_{MT} and the timeout to T_{MT} (which is the timeout of the multicast tree). Each node which forwards the packet adds the value it wants to be paid to the route cost and computes the minimum of T_{MT} and of the timeout it proposes.

When receiving the different possible RREPs, the prospective node can send a message to the multicast group with the right number of nuggets in the purse, and choose the route to the multicast tree depending on the cost.

3) Computing C_{MT} and T_{MT}: In order to know the cost of a multicast routing so that a node can provide the right number of nuggets, the multicast tree members need to keep the values of C_{MT} and T_{MT}. Intuitively, C_{MT} is the sum of the cost that nodes of the tree that are not leaves want to be rewarded and T_{MT} is the minimum of the corresponding timeouts of these costs. The cost C_{MT} can be modified by two events: either a new node joins the multicast group or T_{MT} is reached.

a) When a node N joins the multicast group,: it reaches the nearest tree member M (*i.e.* the node which sends the RREP to it). Each node on the path from N to M adds its cost in the packet as well as the number of packets it will forward at this cost. Then, M can calculate the new cost C_{MT} as well as the new number of packets this cost is valid for:

$$C_{MT}^{new} = C_{MT}^{old} + C_r + C_m,$$

where C_r is the cost added by the nodes on the path from N to M, and C_m the number of nuggets asked by M (this cost is equal to 0 if M was not a leaf before as it is already

included in C_{MT}). The new timeout of the multicast tree is

$$T_{MT}^{new} = min(T_{MT}^{old}, T_r, T_m),$$

where T_r is the minimum provided by the nodes on the path between N and M and T_M is the timeout provided by M if it was a leaf. When the new values are computed, M multicasts them to the tree.

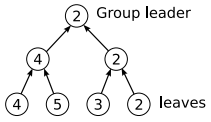


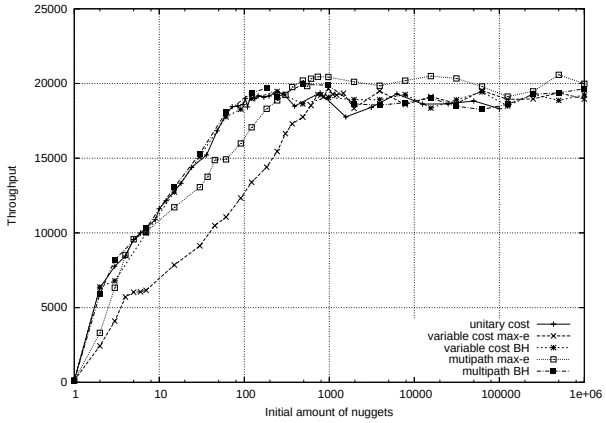
Fig. 4. Propagation of $T_{MT}(l_i)$ in the multicast tree.

b) If the maximum number of packets (T_{MT}) has been sent,: each leave l_i of the multicast tree sends its new couple $(C(l_i), T_{MT}(l_i))$ to its parent in the tree ($C(l_i)$ is the cost asked by node l_i to forward one packet to the multicast group). The parent can then compute the new cost by adding its children costs and the minimum number of packets, and sends them to its parent recursively until the group leader is reached (see Figure 4). At last, the group leader can decide what is the new couple (C_{MT}, T_{MT}) for the tree and sends it back to the tree members.

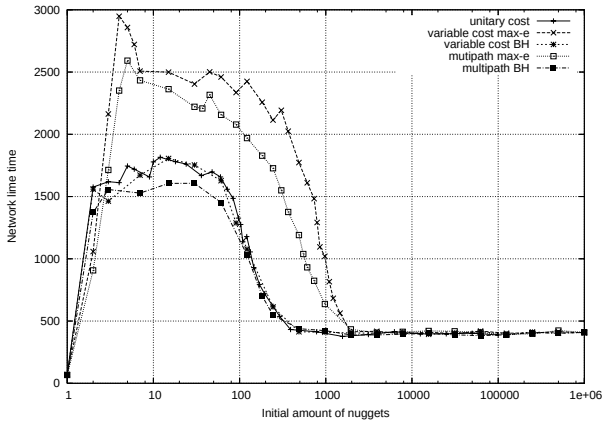
V. Performance evaluation

In this section, we give evaluations of the algorithms we propose on the nugget protocol. For each graph, the number of experiments depends on the convergence of results: we run sequences of 100 experiments until the evolution of the average is less than 1%.

We have experimented the algorithms on a uniform network, which is the worst case for the multipath solution. The simulated networks are composed of 200 nodes that are placed uniformly on a 650m $\times$ 650m square. The communication range of nodes is 100m and they start with full battery level that allows them to send 1000 data packets. The experiment stops when the nodes do not have nuggets anymore or when the network is not connected anymore (there are at least two connected components). We simulate 5 algorithms: shortest path with a cost of one nugget per hop (`unitary cost`), shortest path with a variable cost which is proportional to the energy spent since the network has started (`variable cost max-e`), shortest path with a variable cost which is computed using the functions and initial conditions given in [6] (`variable cost BH`), and the two last ones with the multipath algorithm (`multipath max-e`, `multipath BH`). We can see on Figure 5(a) that the total throughput of the network admits few variations. Anyway, the solution with multipath is always better. The main point to notice is that the total throughput mainly depends on the initial number of nuggets: if this number is very high, there is no nugget famine and each node can send its packets at the maximal rate. Anyway the goal of the nugget model is to enforce node cooperation by making nodes greedy of nuggets: if they have enough nuggets for their own packets, they will not be enforced to cooperate. Figure 5(b) shows the lifetime of the network in the same experiments: if the initial number of nuggets is greater than 1000 the lifetime of the network is constant, this corresponds to the case when nodes are never starving of nuggets. For example, with unitary cost, the average cost of a route is 3 so starting with 3000 nuggets ensures each node to be able to send its own packets with its battery power without any cooperation. If we look at Figure 6, which is the parametric throughput versus lifetime graph when initial nugget number varies, we see that `variable max-e` always dominates the `unitary cost`. We can notice that parameters given in [6] lead to a behavior very similar to the `unitary cost`.



(a) Throughtput



(b) Lifetime

Fig. 5. Throutput and lifetime vs initial number of nuggets.

In Figure 7, we can see the number of inactive nodes (starving of nuggets or battery less) depending on time for variable cost protocols with DSR route discovery and multipath route discovery. We can observe that for same performances in terms of total throughput and network lifetime, we have less inactive nodes in the multipath solution; it reflects the fact that the traffic is well balanced in the multipath solution.

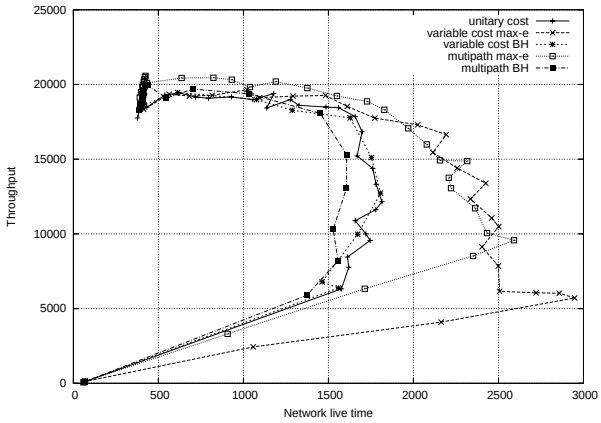


Fig. 6. Parametric Throughput vs Lifetime.

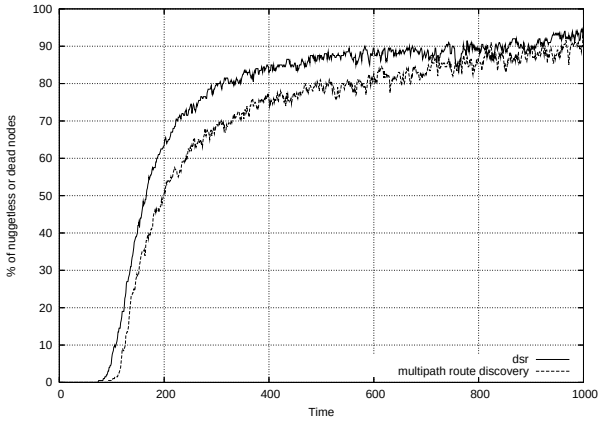


Fig. 7. Surviving nodes.

VI. Conclusion

We have given some improvements to the virtual currency models that help to make the protocol realistic and efficient.

We also give an algorithm that makes them work for the multicast. Moreover, we show that the use of variable costs combined with a multipath route discovery algorithm leads to a better load balancing of the network.

Acknowledgment

This work was partially supported by a grant from CPER Nord-Pas-de-Calais/FEDER TAC MOSAÏQUE and CNRS National platform RECAP.

References

[1] S. Marti, T. J. Giuli, K. Lai, and M. Baker, "Mitigating routing misbehavior in mobile ad hoc networks." in *MOBICOM*, 2000, pp. 255–265.

[2] L. Buttyán and J.-P. Hubaux, "Enforcing service availability in mobile ad-hoc WANs," in *Proc. of the 1st ACM Interational Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2000)*. Boston, Massachusetts: ACM, 2000, pp. 87–96.

[3] S. Zhong, J. Chen, and Y. R. Yang, "Sprite: A simple, cheat-proof, credit-based system for mobile ad-hoc networks," in *Proc. of 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2003)*. San Francisco, CA, USA: IEEE, 2003.

[4] J. Crowcroft, R. Gibbens, F. Kelly, and S. Östring, "Modeling incentives for collaboration in mobile ad hoc networks," in *Proceedings of WiOpt 2003*, Sophia Antipolis, France, March 2003.

[5] L. Buttyán and J.-P. Hubaux, "Stimulating cooperation in self-organizing mobile ad hoc networks." *Mobile Networks and Applicaions (MONET)*, vol. 8, no. 5, pp. 579–592, 2003.

[6] ——, "Nuglets: a virtual currency to stimulate cooperation in self-organized mobile ad hoc networks," EPFL, Tech. Rep. DSC/2001.001, January 2001.

[7] D. B. Johnson and D. A. Maltz, "Dynamic source routing in ad hoc wireless networks," in *Mobile Computing*, T. Imielinski and H. F. Korth, Eds. Kluwer Academic, 1996, vol. 353.

[8] M. Hauspie, D. Simplot-Ryl, and J. Carle, "Partition detection in mobile ad-hoc networks using disjoint paths set," in *Proceedings of the 1st International Workshop on Objects models and Multimedia technologies (OMMT'03)*, Geneva, Switzerland, September 2003.

[9] Y.-C. Tseng, S.-Y. Ni, and Y.-S. Chen, "The broadcast storm problem in a mobile ad hoc network," *ACM Wireless Networks*, vol. 8, no. 2, pp. 152–167, March 2002.

[10] J. Cartigny, D. Simplot, and J. Carle, "Stochastic flooding broadcast protocols in mobile wireless networks," *submitted*, 2002.

[11] A. Laouiti, A. Qayyum, and L. Viennot, "Multipoint relaying: An efficient technique for flooding in mobile wireless networks," *35th Annual Hawaii International Conference on System Sciences (HICSS'2001)*, 2001.

[12] J. Wu and F. Dai, "Broadcasting in ad hoc networks based on self-pruning," in *Proc. of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, San Francisco, 2003.

[13] E. Royer and C. Perkins, "Multicast operation of the ad hoc on-demand distance vector routing protocol," in *Proceedings of the 5th International Conference on Mobile Computing and Networking (MobiCom99)*, Seattle, USA, August 1999.