Stimulating Cooperation in Self-Organizing Mobile Ad Hoc Networks*

Levente Buttyán and Jean-Pierre Hubaux
EPFL-DSC-ICA†

July 31, 2001

Abstract

In military and rescue applications of mobile ad hoc networks, all the nodes belong to the same authority; therefore, they are motivated to cooperate in order to support the basic functions of the network. In this paper, we consider the case when each node is its own authority and tries to maximize the benefits it gets from the network. More precisely, we assume that the nodes are not willing to forward packets for the benefit of other nodes. This problem may arise in civilian applications of mobile ad hoc networks. In order to stimulate the nodes for packet forwarding, we propose a simple mechanism based on a counter in each node. We study the behavior of the proposed mechanism analytically and by means of simulations, and detail the way in which it could be protected against misuse.

1 Introduction

A mobile ad hoc network is a wireless multi-hop network formed by a set of mobile nodes in a self-organizing way without relying on any established infrastructure. Due to the absence of infrastructure, in a mobile ad hoc network, all networking functions must be performed by the nodes themselves. For instance, packets sent between two distant nodes are expected to be forwarded by intermediate nodes [7, 13].

The above mentioned operating principle renders cooperation among nodes an essential requirement in a mobile ad hoc network. By cooperation, we mean that the nodes perform networking functions for the benefit of other nodes. Lack of cooperation may have fatal effects on the performance of the network. As an example, let us consider Figure 1, which illustrates that the throughput of the network decreases dramatically as the fraction of the nodes that deny packet forwarding increases. The different curves belong to networks of different sizes (100, 200, 300, and 400 nodes) but with the same node density. The figure also shows that larger networks are more sensitive to this kind of non-cooperative behavior of the nodes. These results are based on our own simulations described in detail in Section 4, but similar results have been presented in [11] as well.

So far, applications of mobile ad hoc networks have been envisioned mainly for crisis situations (e.g., in the battlefield or in rescue operations). In these applications, all the nodes

*Technical Report No. DSC/2001/046, Swiss Federal Institute of Technology – Lausanne

†Institute for Computer Communications and Applications, Swiss Federal Institute of Technology – Lausanne, e-mail: {Levente.Buttyan, Jean-Pierre.Hubaux}@epfl.ch

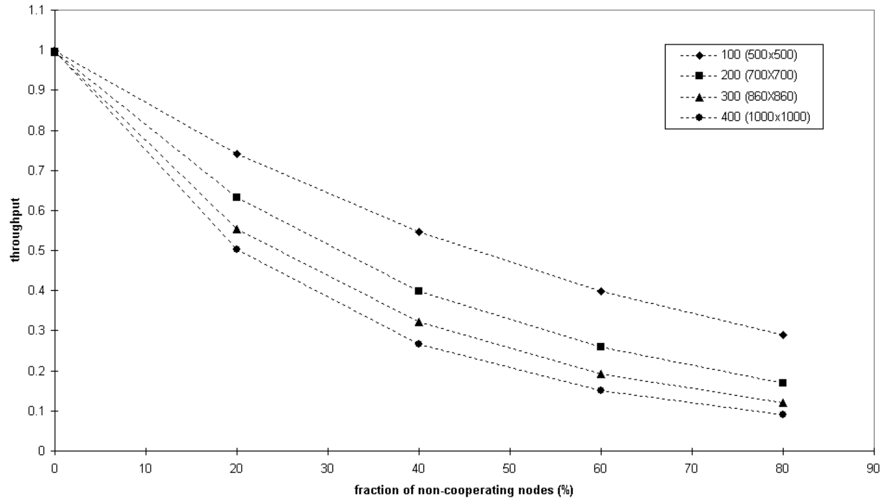

Figure 1: The effect of non-cooperating nodes on the throughput of the network

of the network belong to a single authority (e.g., a single platoon or rescue team) and have a common goal. For this reason, the nodes are naturally motivated to cooperate.

However, with the progress of technology, it will soon be possible to deploy mobile ad hoc networks for civilian applications as well. Examples include networks of cars and provision of communication facilities in remote areas. In these networks, the nodes typically do not belong to a single authority and they do not pursue a common goal. In addition, these networks could be larger and could have a longer lifetime, and they could be completely *self-organizing*, meaning that the network would run solely by the operation of the end-users. In such networks, there is no good reason to assume that the nodes cooperate. Indeed, the contrary is true: in order to save battery power, the nodes tend to be "selfish".

As a motivating example, let us consider packet forwarding again: Even in a small ad hoc network, most of the energy of a given node is likely to be devoted to forwarding packets for the benefit of other nodes. For instance, if the average number of hops from source to destination is around 5, then approximately 80% of the energy devoted to sending packets will be consumed by packet forwarding. Hence, turning the forwarding function off would very noticeably extend the battery life of a node, and increase its overall availability for its user.

In this paper, we address the problem of stimulating cooperation in self-organizing, mobile ad hoc networks for civilian applications. We assume that each node belongs to a different authority, its user, which has full control over the node. In particular, the user can tamper with the software and the hardware of the node, and modify its behavior in order to better adapt it to her own goals (e.g., to save battery power). We understand that regular users usually do not have the required level of knowledge and skills to modify their nodes. Nevertheless, our assumption is still reasonable, because criminal organizations can have enough interest and resources to reverse engineer a node and sell tampered nodes with modified behavior on a large scale. The experience of cellular networks shows that as soon as the nodes are under the control of the end-users, there is a strong temptation to alter their behavior in one way or another.

Our approach to alleviate this problem is based on a trusted and tamper resistant hardware module, called *security module*, in each node and cryptographic protection of packets. As opposed to the node itself, the security module cannot be tampered with by the user. One can think of the security module as a smart card (similar to the SIM card in GSM phones) or as a tamper resistant security co-processor. Our design ensures that while the user can still modify the behavior of the node (but not the security module), she cannot gain any advantages by doing so. Thus, tampering with nodes is uninteresting, and should happen only rarely.

We focus on the stimulation of packet forwarding, which is a fundamental networking function that the nodes should perform in a mobile ad hoc network. In a nutshell, we propose a protocol that requires the node to pass each packet (generated as well as received for forwarding) to its security module. The security module maintains a counter, which is decreased when the node wants to send a packet as originator, and increased when the node forwards a packet. The value of the counter must remain positive, which means that if the node wants to send its own packets, then it must forward packets for the benefit of other nodes. The counter is protected from illegitimate manipulation by the tamper resistance of the security module.

Besides stimulating packet forwarding, our mechanism encourages the users to keep their nodes turned on and to refrain from sending a large amount of packets to distant destinations. The latter property is particularly desirable, because, as mentioned in [8], the available bandwidth per node declines as the number of nodes increases (assuming that the traffic does not exhibit locality properties).

The present proposal was developed in the framework of the Terminodes Project [4, 9] (www.terminodes.org). However, it is generic; in particular, it could work in conjunction with any routing algorithm.

The outline of the paper is the following: In Section 2, we describe the proposed mechanism to stimulate packet forwarding, and study its behavior through the analysis of a simple model. In Section 3, we detail the ways in which the proposed mechanism could be protected against misuse. In Section 4, we describe our simulation settings, and the results that we obtained. Finally, in Section 5, we report on some related work, and in Section 6, we conclude the paper.

2 Stimulation mechanism

We assume that each node has a counter, called *credit counter*, and the following rules are enforced:

1. When the node wants to send one of its own packets, the number n of intermediate nodes that are needed to reach the destination is estimated. If the credit counter of the node is greater than or equal to n, then the node can send its packet, and the credit counter is decreased by n. Otherwise, the node cannot send its packet, and the credit counter is not modified.

2. When the node forwards a packet for the benefit of other nodes, the credit counter is increased by one.

Let us consider now the following model, the analysis of which will give an insight into the operation of the above mechanism. A node has two incoming and two outgoing flows of packets

(Figure 2). The incoming flow IN_o represents the packets that are generated by the node itself. We call these packets *own packets*. The other incoming flow IN_f represents the packets that are received for forwarding. We call these packets *forwarding packets*. The packets that the node receives as destination are not represented in the model. Each incoming packet (own as well as forwarding) is either sent or dropped. The outgoing flow OUT represents the packets that are sent by the node. This flow consists of two components OUT_o and OUT_f, where OUT_o represents the own packets that are sent and OUT_f stands for the forwarded packets. The other outgoing flow DRP represents the packets that are dropped. Similarly to OUT, this flow consists of two components too: DRP_o and DRP_f, representing dropped own and forwarding packets, respectively.

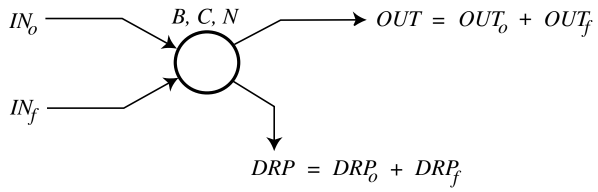

Figure 2: Model of a single node

The current state of the node is described by two variables b and c, where b is the remaining battery of the node and c stands for the value of its credit counter. More precisely, we interpret b as the number of packets that the node can send using its remaining energy. The initial values of b and c are denoted by B and C, respectively. To keep the model simple, we assume that when the node sends an own packet, c is decreased by an integer constant $N > 1$, which represents the estimated number of intermediate nodes that are needed to reach the destination. Since c must remain positive, the node can send its own packet only if $c \geq N$ holds. When the node sends a packet that was received for forwarding, c is increased by one. In addition, each time the node sends a packet (own as well as forwarding), b is decreased by one. When b reaches 0 (i.e., when the battery is drained out), the node stops its operation. We assume that the initial number C of credits is not enough to drain the battery out by sending only own packets (i.e., $C/N < B$).

Let us denote the number of own and forwarding packets sent during the whole lifetime of the node by out_o and out_f, respectively. Selfishness of the node could be represented by the goal of maximizing out_o subject to the following conditions:

$$out_o,\ out_f \geq 0 \tag{1}$$
$$N out_o - out_f \leq C \tag{2}$$
$$out_o + out_f = B \tag{3}$$

Condition (1) is trivial. Condition (2) describes the requirement that the number $N out_o$ of credits spent by the node cannot exceed the number out_f of credits earned plus the initial value C of the credit counter. Finally, Condition (3) represents the fact that the initial energy of the node must be shared between sending own packets and sending forwarding packets.

Figure 3 illustrates the conditions graphically. It is easy to see that the maximum of out_o is $\frac{B+C}{N+1}$. It can also be seen that in order to reach this maximum out_f must be $\frac{NB-C}{N+1}$. Thus, the node must forward this number of packets for the benefit of other nodes if it wants to

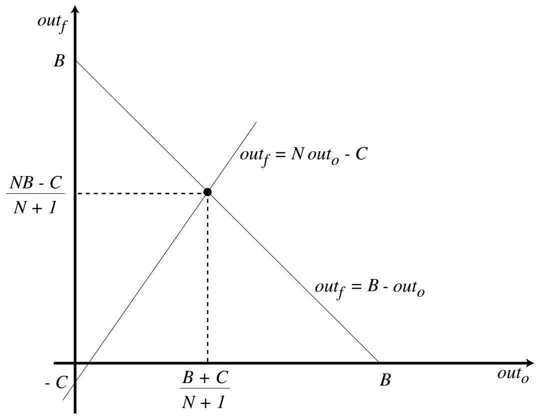

Figure 3: Maximizing out_o

maximize its own benefit. If there was no credit counter and an enforcing mechanism that does not allow the node to send an own packet when it does not have enough credits, then Condition (2) would be missing, and the maximum of out_o would be B. This means that the node would maximize its own benefit by dropping all packets received for forwarding.

In principle, the node can always reach $out_o = \frac{B+C}{N+1}$: When it runs out of credits, it can simply buffer its own packets until it forwards enough packets and earns enough credits to send them. However, this works only if the buffer is large enough and no delay constraint is imposed on the packets. In real-time applications, sending a packet that has spent too much time in the buffer may be useless, which means that the node must drop some of its own packets. It can still reach $out_o = \frac{B+C}{N+1}$, but it is now important how many own packets it must drop meanwhile.

In order to study this situation, we extend our model in the following way: We assume that the node generates own packets with a constant average rate r_o, and receives packets for forwarding with a constant average rate r_f. We denote the time when the battery is drained out by t_{end}. Note that t_{end} is not a constant, since the time when the battery is drained out depends on the behavior of the node. Furthermore, we assume that there is no buffering of own packets, which means that an own packet that cannot be sent immediately (due to the low value of the credit counter) must be dropped.

Selfishness of the node could now be represented by the goal of maximizing out_o and, at the same time, maximizing $z_o = \frac{out_o}{r_o t_{end}}$ (which is equivalent to minimizing the number of own packets dropped) subject to the following conditions:

$$out_o,\ out_f \geq 0 \quad (4)$$
$$out_o \leq r_o t_{end} \quad (5)$$
$$out_f \leq r_f t_{end} \quad (6)$$
$$N out_o - out_f \leq C \quad (7)$$
$$out_o + out_f = B \quad (8)$$

Using $out_f = B - out_o$ from Condition (8), we can reduce the number of unknowns and obtain the following set of conditions:

$$out_o \geq 0 \quad (9)$$

$$out_o \leq B \quad (10)$$

$$t_{end} \geq \frac{out_o}{r_o} \quad (11)$$

$$t_{end} \geq -\frac{out_o}{r_f} + \frac{B}{r_f} \quad (12)$$

$$out_o \leq \frac{B+C}{N+1} \quad (13)$$

Conditions (9-13) determine the feasible region, on which we have to maximize out_o and z_o. This is illustrated in Figure 4. As we have already seen, the maximum of out_o is $\frac{B+C}{N+1}$. Note that $\frac{B+C}{N+1}$ is always less than B, because we assumed that $C/N < B$. Different values of z_o are represented by lines with different slopes all going through the (0,0) point. In order to find the maximum of z_o, we have to find the line with the smallest slope that still intersects the feasible region.

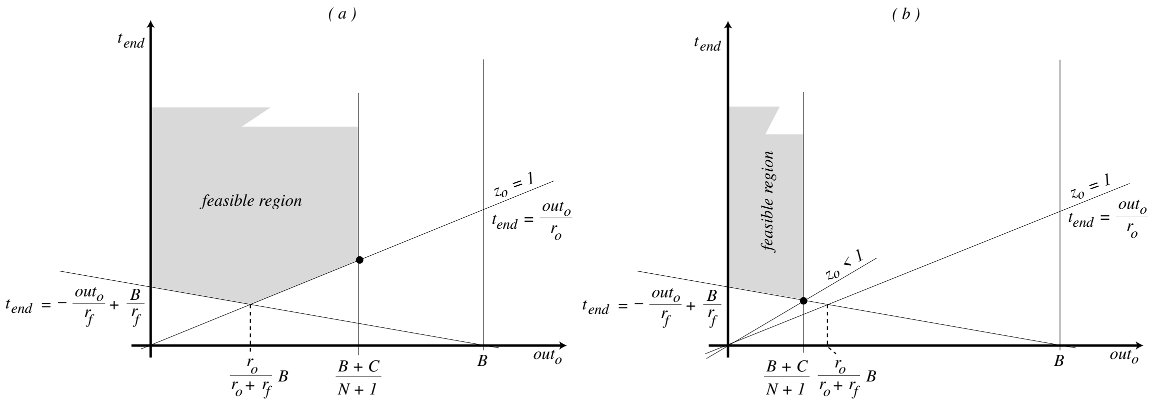

Figure 4: Maximizing out_o and $z_o = \frac{out_o}{r_o t_{end}}$

Depending on the ratio r_f/r_o of the rates, we can distinguish the following two cases (Figure 4 parts (a) and (b)):

1. If $\frac{r_f}{r_o} \geq \frac{NB-C}{B+C}$ (i.e., $\frac{B+C}{N+1} \geq \frac{r_o}{r_o+r_f}B$) then the maximum of z_o is 1. Because of Condition (11), this is the best that can be achieved. This means that in this case, the node does not have to drop any of its own packets.

2. If $\frac{r_f}{r_o} < \frac{NB-C}{B+C}$ (i.e., $\frac{B+C}{N+1} < \frac{r_o}{r_o+r_f}B$), then the maximum of z_o is $\frac{r_f}{r_o}\frac{B+C}{NB-C} < 1$. This means that in this case, the node must drop some of its own packets.

Intuitively, the difference between the two cases above can be explained as follows: In case 1, packets for forwarding arrive with high enough a rate to cover the expenses of sending own packets. On the other hand, in case 2, the arrival rate of forwarding packets is too low, and the node cannot earn enough credits to send all of its own packets even if it forwards all packets received for forwarding.

The above analysis shows *what* the node can achieve in terms of maximizing its own benefit. However, it does not shed light on *how* the node should actually behave in order to reach this theoretical optimum. It seems reasonable that the node should always send its own packets whenever this is possible (i.e., whenever it has enough credits to do so). But how should the node decide whether to forward or to drop a packet received for forwarding?

In order to get an insight into this question, let us consider the following four forwarding rules, where f denotes the number of forwarding packets sent so far:

Rule 1: **if** $f < \frac{NB-C}{N+1}$ **then** forward
else drop

Rule 2: **if** $f < \frac{NB-C}{N+1}$ **then**
  if $c \leq C$ **then** forward
  else forward with probability C/c or drop with probability $1 - C/c$
else drop

Rule 3: **if** $f < \frac{NB-C}{N+1}$ **then**
  if $c \leq C$ **then** forward
  else drop
else drop

Rule 4: **if** $f < \frac{NB-C}{N+1}$ **then**
  if $c \leq C$ **then** forward with probability $1 - c/C$ or drop with probability c/C
  else drop
else drop

In all four rules, packets are dropped after the threshold $f = \frac{NB-C}{N+1}$ has been reached. The reason is that in this case, it is not necessary to forward more packets, because the node has enough credits to drain its battery out by sending only its own packets. The four rules differ in what happens before this threshold is reached. In Rule 1, packets are always forwarded. In the other rules, the forwarding decision depends on the current value c of the credit counter. In Rule 2, packets are forwarded for sure if $c \leq C$, and with decreasing probability as c increases if $c > C$. In Rule 3, packets are forwarded for sure if $c \leq C$, and they are always dropped if $c > C$. In Rule 4, packets are forwarded with decreasing probability as c increases if $c \leq C$, and they are always dropped if $c > C$. Clearly, the most cooperative rule is Rule 1. Rules 2, 3, and 4 are less cooperative, in this order.

We studied the performance of the rules by means of simulation. We implemented the above described model of a single node in plain C++ language. In our simulations, we set the values of the parameters as follows: $B = 100000$, $C = 100$, $N = 5$. Both the own packets and the packets for forwarding were generated according to a Poisson process. The average generation rate of the own packets were 0.2 packets per second, and we varied the average generation rate of forwarding packets between 0.6 and 1.6 packets per second with a step size of 0.2 (i.e., we varied r_f/r_o between 3 and 8 with a step size of 1, in order to obtain some results for the $\frac{r_f}{r_o} < \frac{NB-C}{B+C} \approx 5$ case as well as for the $\frac{r_f}{r_o} \geq \frac{NB-C}{B+C}$ case). The simulations lasted until the node drained its battery out (i.e., 100000 packets were sent). We run the simulation 8 times for every configuration and took the average of the results obtained. Each rule reached $out_f = 16683 = \left\lfloor \frac{B+C}{N+1} \right\rfloor$ in every run of the simulation. The values obtained for z_o are depicted in Figure 5.

It can be seen that Rule 4 achieves the worst performance as it is the furthest from the theoretical optimum. The first three rules perform almost equally well when $r_f/r_o < 5$ and $r_f/r_o > 5$. However, a remarkable difference appears among the rules when $r_f/r_o = 5 = N$.

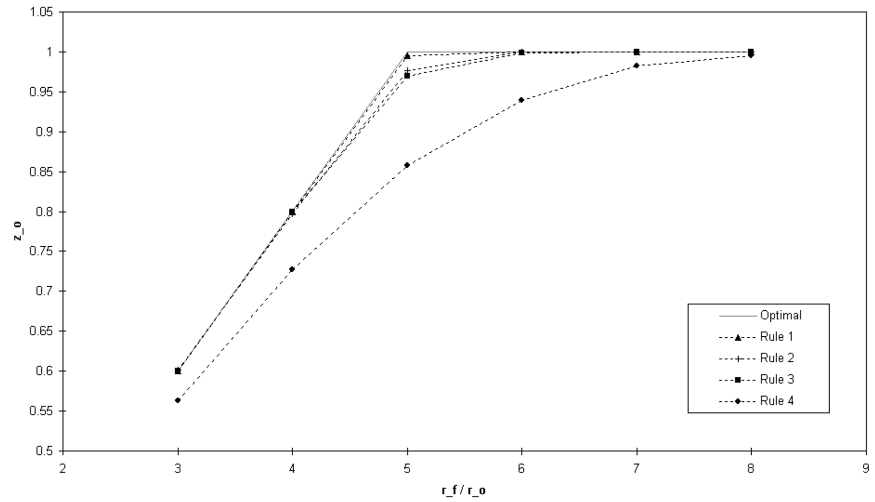

Figure 5: Comparison of the forwarding rules

Interestingly enough, the results show that the most cooperative the rule is the best the performance that it achieves. This means that if the node wants to maximize out_o and z_o at the same time, then the best forwarding rule is Rule 1 (i.e., to always forward).

Figure 6 is meant to provide an intuitive explanation for this phenomenon. Parts (a) and (b) of the figure illustrate the operation of Rules 1 and 3, respectively, when $\frac{r_f}{r_o} \geq \frac{NB-C}{B+C}$. The figure should be interpreted in the following way: Let us assume that time is divided into small time slots. Each small grey rectangle in the figure represents the set of possible points that the node can potentially reach in a given time slot assuming that it is in the bottom-left corner of the rectangle at the beginning of that time slot. Therefore, the ratio of the sides of the rectangles is r_f/r_o. The arrows show which points are actually reached by the node when Rules 1 and 3 are used. The dark vertical bars represent the amount of dropped forwarding packets in the time slots.

It can be seen that by using Rule 1, the node tends to get further from the edge of the feasible region that is represented by the $out_f = Nout_o - C$ line. This means that the node has usually more credits in reserve when Rule 1 is used. This property turns out to be advantageous when the ratio r_f/r_o is close to N. The reason is that, due to the random manner in which the packets arrive, there is always a small fluctuation in the ratio of the number of forwarding packets to the number of own packets. On average, this ratio is equal to r_f/r_o, but sometimes it can be less. If this happens and r_f/r_o is close to N, then the node does not receive enough forwarding packets to cover the cost of sending its own packets. In this case, it must use the credits that it has in reserve. By increasing the credit reserve, Rule 1 decreases the probability of temporarily running out of credits and dropping own packets.

3 Protection

Clearly, the stimulating mechanism described in the previous section must be secured and protected against various attacks. For instance, one has to prevent the user of the node from

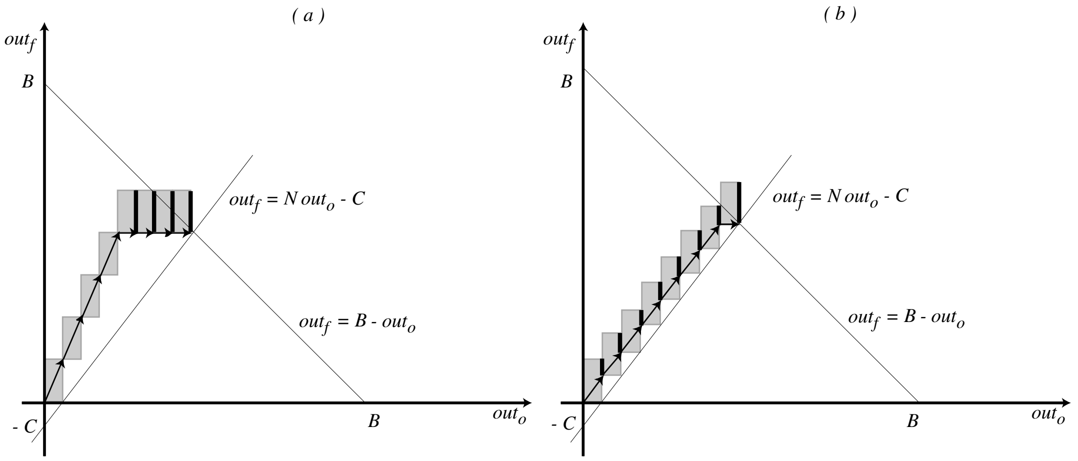

Figure 6: Operation of Rule 1 (a) and Rule 3 (b) when $\frac{r_f}{r_o} \geq \frac{NB-C}{B+C}$

manipulating (typically increasing) her credit counter in an illegitimate way. In addition, one has to ensure that the credit counter is increased only if a forwarding packet has indeed been forwarded. We address these and similar issues in this section.

3.1 Tamper resistant security module

In order to prevent the user from illegitimately increasing its own credit counter, we require that the credit counter is maintained by a trusted and tamper resistant hardware module in each node. We call this module *security module*. One can imagine a security module as a smart card (similar to the SIM card in GSM phones) or as a tamper resistant security co-processor. For more information on tamper resistant modules, we refer to [14, 2].

We assume that the security modules are manufactured by a limited number of trusted manufacturers. Furthermore, since the security module is tamper resistant, its behavior cannot be modified. Therefore, security modules are trusted for always functioning correctly.

Our design approach is to put the critical functions in the security module, and the rest of the functions in the node itself. Of course, the functions that are not placed in the security module can be tampered with, and thus, the behavior of the node can still be modified. However, our design ensures that *no advantages can be gained* by tampering with the unprotected functions, and therefore, the user of the node will not be interested in this activity.

3.2 Public-key infrastructure

We assume that each security module has a private key and a corresponding public key [12]. The private key is stored in the security module and kept secret. The public key is certified by the manufacturer of the security module and the certificate is stored in the security module. In addition, we assume that the manufacturers cross-certify the public keys of each other, and each security module stores the public-key certificates of all manufacturers issued by the manufacturer of the security module. Finally, we assume that each security module stores an authentic copy of the public key of its manufacturer, which is loaded in the module securely at manufacturing time. Note that storing all these certificates is feasible, because we limit the number of manufacturers.

In this system, each security module can easily obtain the authentic public key of any other security module in the network. Let us suppose, for instance, that A wants to obtain the public key of B. B can simply send its public-key certificate to A, who can verify it with the public key of the manufacturer of B. A possesses an authentic copy of this public key, since it stores the authentic public-key certificates of all manufacturers issued by its own manufacturer.

Our system is a rather pragmatic solution for the reliable distribution of public keys, and we had to limit the number of manufacturers in order for it to work. The design of a general purpose public-key infrastructure for large, self-organizing ad hoc networks is a challenging problem that is beyond the scope of this paper. An approach towards the solution of this problem is described in [10].

3.3 Security associations

When two nodes become neighbors, their security modules establish a security association. If this fails, the security modules do not consider each other neighbors.

A security association between two neighboring security modules A and B is represented, at A's side, by

- the unique identifier of B;
- the unique identifier of the node that hosts B,
- a symmetric session key k_{AB};
- two sequence numbers $c_{A\to B}$ and $c_{A\leftarrow B}$, which are called *sending* and *receiving sequence numbers*, respectively; and
- a counter $pc_{B@A}$, which is called *pending credit counter.*

At B's side, the same association is represented by

- the unique identifier of A;
- the unique identifier of the node that hosts A;
- the session key k_{AB};
- a sending sequence number $c_{B\to A}$ and a receiving sequence number $c_{B\leftarrow A}$, such that initially, $c_{B\to A} > c_{A\leftarrow B}$ and $c_{B\leftarrow A} < c_{A\to B}$; and
- a pending credit counter $pc_{A@B}$.

The session key k_{AB} is used to compute a message authentication code, which protects the integrity and ensures the authenticity of the packets sent between the nodes of A and B, but k_{AB} can also be used to provide other security functions (e.g., link-by-link encryption of the content of the packets). The sequence numbers are used to detect replayed packets. The pending credit counter $pc_{B@A}$ is used to accumulate credits at A that are due to B. Similarly, $pc_{A@B}$ counts the credits at B that are due to A. The way in which the session key, the sequence numbers, and the pending credit counters are used will be explained in more detail in the next subsection, where we present the envisioned packet forwarding protocol.

The security associations between the security modules are established using some public-key cryptographic protocol, which is executed through the nodes that host the security modules. The security modules obtain each other's public key according to the model of the above described public-key infrastructure.

3.4 Packet forwarding protocol

The packet forwarding protocol described in this subsection assumes that the security module runs the routing algorithm used in the network.

If a node P has an own packet to send, it must first pass the packet to its security module A. A estimates the number n of intermediate nodes needed to reach the destination. Precise estimation of this number is not so critical. If the value of the credit counter maintained by A is less than n, then A rejects the packet. Otherwise the credit counter is decreased by n, and the protocol continues.

Using the routing algorithm, A determines the next intermediate security module B toward the destination, and retrieves the security association that corresponds to B from its internal database. Then, it takes the session key k_{AB} and the sending sequence number $c_{A \to B}$, and generates a *security header* for the packet, which contains A, B, $c_{A \to B}$, and the output $h(k_{AB};\ A,\ B,\ c_{A \to B},\ \mathit{packet})$ of a publicly known keyed cryptographic hash function h. After this computation, $c_{A \to B}$ is increased by one.

Finally, A outputs the security header and the identifier of the next intermediate node Q (obtained from the data that represents the security association between A and B), and P can send the packet together with the security header to Q.

Now, let us assume that node Q received a packet with a security header for forwarding from node P. If Q wants to forward the packet in order to earn a credit, then Q must pass the packet with the attached security header to its security module B. B takes the identifier of the security module A that generated the security header from the header itself, and retrieves the corresponding security association from its internal database. Then, it verifies if the sending sequence number in the security header is greater than its receiving sequence number $c_{B \leftarrow A}$. If this is the case, then the packet is not a replay. Then, it verifies the received value of the keyed cryptographic hash function h. If the value is correct, then it accepts the packet, and updates $c_{B \leftarrow A}$ to the value of the sequence number received in the security header.

If the node that hosts A (known to B from the data that represents the security association between B and A) is not the originator of the packet (i.e., if it is an intermediate node), then B increases the pending credit counter $pc_{A@B}$ by one. Finally, B determines the next intermediate security module towards the destination, and generates a new security header for the packet, much in the same way as described earlier, using the security association that corresponds to the next intermediate security module.

3.5 Credit synchronization protocol

As it can be seen from the description of the packet forwarding protocol, when an intermediate node forwards a packet, its credit counter is not increased immediately. Instead, the security module of the next node increases the pending credit counter that it maintains for the first node. For clearing, the security modules regularly run a credit synchronization protocol, in which they transfer the pending credits, and reset the pending credit counters to 0. This mechanism ensures that the node is rewarded for the packet forwarding only if it really forwarded the packet.

It may happen that the nodes move out of each other's power range by the next time their security modules want to run the credit synchronization protocol. If this happens, the pending credit counters are reset to 0, and the pending credits are lost. Therefore, the mechanism does not guarantee that the node receives its credit for every forwarded packet. We will study the consequences of this in the next section.

3.6 Robustness

The protection mechanism described above is robust and resists against various attacks. The credit counter is protected from illegitimate manipulations by the tamper resistance of the security module. A security header is attached to each packet, which contains a message authentication code that protects the integrity of the packet and the data in the security header. This is important, because the security modules manipulate the credit counters based on the data received in the security header. Replay of packets is prevented by the use of an ever increasing sequence number. Moreover, the node is rewarded for packet forwarding only if it really forwarded a packet.

We should mention, however, that there is a subtle attack that our scheme may not always prevent in its current form. It is possible to construct a fake node that has two or more security modules. Such a node could bounce a packet back and forth between its security modules, and earn credits without actually transmitting anything. The full understanding of this attack and the design of the proper countermeasure are on our research agenda. However, we can already make the following observations: First, this attack would not always work, since routing is performed by the security modules, which means that the next intermediate security module is determined by the security module and not the node. In other words, the security module may output a security header for the packet that will not be accepted by the other security module of the node. To avoid this, the node may falsify routing information that is exchanged between the security modules, but this can be prevented by using appropriate cryptographic techniques. Second, such a fake node would be more expensive than a normal one, since it has two or more legitimate security modules. Whether the benefit obtained by using such a fake node is worth the increased cost is an open question.

3.7 Overhead

We must admit that our protection mechanism adds some computational overhead to the system, which is mainly related to the use of cryptographic operations. This issue has two aspects: cryptographic operations need energy and time to be performed. Regarding energy consumption, we note that the energy required to perform computation is negligible when compared to the energy required to perform transmission. As mentioned in [15], when two nodes communicate in BPSK at 1 GHz carrier frequency with 10^{-6} error probability, have an antenna elevation of 1/2 wavelength with fourth-power distance loss, Raleigh fading, and an ideal receiver, then the energy cost of transmitting 1 Kb over a distance of 100 meters is approximately 3 joules. By contrast, a general purpose processor with 100 MIPS/W power could efficiently execute 300 million instructions for the same amount of energy. This means that the execution of our cryptographic operations have a negligible energy cost when compared to the transmission cost.

Regarding time, we note that the only time critical operations are the generation and the verification of the security header for every packet and for every hop. However, these require only cryptographic hash function computations, which can be done very efficiently. Moreover, the security header is processed by the security module; to some extent, this can be accomplished in parallel with the processing performed by the main processor of the node.

Another issue is the communication overhead, which is due to the establishment of the security associations, the size of the security header, and the periodic execution of the credit synchronization protocol. In order to reduce this overhead, the establishment of the security associations could be integrated with the neighbor discovery protocol that the nodes usually have to run anyhow in mobile ad hoc networks, and the credit synchronization interval should be appropriately chosen. Finally, assuming that the identifiers of the security modules are 8 bytes long, the sequence numbers are 4 bytes long, and the output of the cryptographic hash function used is 16 bytes long (e.g., if MD5 [12] is used), we get that the security header is 36 bytes long. Considering the usual size of payloads in wireless networks, this seems to be an acceptable overhead.

4 Simulations

In Section 2, we studied the proposed stimulation mechanism through the analysis of a simplified model, and showed convincing arguments that it indeed stimulates packet forwarding in that model. In order to study the proposed stimulation mechanism in a more general setting, which is closer to the reality of mobile ad hoc networks, we conducted simulations of a full network written in plain C++ language. In this section, we describe our simulator, and the results that we obtained.

4.1 Simulation description

The simulated networks are composed of 100 nodes that are placed randomly (uniformly) on a 500 m $\times$ 500 m rectangle. Each node has the same power range of 120 m. The nodes move according to the *random waypoint* mobility model [5]. In this model, the node randomly chooses a destination point in space and moves towards this point with a randomly chosen constant speed. When it reaches the chosen destination, it stops and waits there for a randomly chosen time. Then, it chooses a new destination and speed, and starts to move again. These steps are repeated until the end of the simulation. In our simulations, the nodes choose their speed between 1 m/s and 3 m/s uniformly. The pause time is generated according to the exponential distribution. The average pause time is 60 s.

We do not use any particular MAC layer algorithm. Instead, we model the MAC layer by randomly choosing the packet transmission time between neighbors for each packet and for each hop. The average packet transmission time between neighbors is 10 ms. Packet transmission errors occur with 0.1 probability. If an error occurred, the packet is re-transmitted after a 1 s timeout. When the node is busy with packet transmission, it can still receive packets, which are placed in a buffer, and served when the previous packet transmission is finished.

For routing, we use a *geodesic packet forwarding* algorithm developed within the context of the Terminodes Project, and described in [3]. However, we considerably simplified the original algorithm in order to ease the implementation of its simulator. This does not affect our results, since we are not interested in the performance of the packet forwarding algorithm itself. The simplified geodesic packet forwarding algorithm works in the following way: We assume that each node knows its own geographic position and the geographic positions of its neighbors. Furthermore, the source of a packet knows the current geographic position of the destination. The way in which this information is obtained is not simulated. Before sending the packet, the source puts the coordinates of the destination in the header of the packet. Then, it determines which of its neighbors is the closest to the destination, and sends the packet to this neighbor. When a node receives a packet for forwarding, it first verifies if the destination is its neighbor. If this is the case, then it forwards the packet to the destination. Otherwise, it determines which of its neighbors is the closest to the destination, and sends the packet to this neighbor. This is possible, because the packet header contains the believed coordinates of the destination. If the forwarding node does not have any neighbor that is closer to the destination than the node itself, then the packet is dropped[1]. In our simulations, because of the rather high density and the rather low speed mobility of the nodes, packet drops of this kind almost never happened .

Energy consumption of the nodes is not simulated. For this reason, the size of the packets is not important for us. Therefore, we assume that each packet has the same size, and we focus only on the number of packets that are generated, sent, forwarded, and delivered.

Each node generates packets according to a Poisson process. The destination of each packet is chosen randomly (uniformly). In our reference simulation, the average packet generation rate was 0.2 pkt/s, but we also ran simulations with average packet generation rates of 0.5 and 0.8 pkt/s.

The initial value C of the credit counter of each node is 100. When a node i sends an own packet to a node d that is not the neighbor of i, the credit counter of i is decreased by n. Unlike in the simple model of Section 2, n is not a constant, but computed according to the following formula:

$$n = \left\lceil \frac{distance(i,d)}{power\ range} \right\rceil - 1$$

This gives a lower bound on the number of intermediate nodes needed to reach the destination. When a node forwards a packet, its pending credit counter at the next node is increased by one. In our reference simulation, the credits of each node are synchronized in every 5 s, but we also ran simulations with credit synchronization intervals of 10, 15, and 20 s.

We always ran 8 simulations for a given simulation setting, and considered the average of the obtained values for each observed variable. In each run, 2 hours of network operation were simulated.

We listed the values of the main simulation parameters for an overview in Table 1.

4.2 Simulation results

4.2.1 Comparison of forwarding rules

In the first set of simulations, our goal was to determine which of Rule 1, Rule 2, or Rule 3 is the most beneficial for the nodes in terms of maximizing z_o. We did not use Rule 4, because it performed much worse than the other three rules in the single node model of Section 2.

[1] This simplification is true only in our simulation setting. The complete geodesic packet forwarding algorithm described in [3] can cope with such a situation.

Parameter	Value
Space	500 m $\times$ 500 m
Number of nodes	100
Power range	120 m
Mobility model	random waypoint
Speed	1 m/s – 3 m/s
Average pause time	60 s
Packet generation rate	0.2 (0.5, 0.8) pkt/s
Choice of destination	random
Routing	geodesic packet forwarding
Initial number of credits (C)	100
Credit synchronization interval	5 (10, 15, 20) s
Simulation time	7200 s

Table 1: Value of the main simulation parameters

Since battery usage is not taken into consideration in our simulations, we had to modify the rules as follows:

Rule 1': always forward

Rule 2': **if** $c \leq C$ **then** forward
else forward with probability C/c or drop with probability $1 - C/c$

Rule 3': **if** $c \leq C$ **then** forward
else drop

Our approach to determine which of these rules is the best was the following: We set 90% of the nodes to use a given rule (we call this the *majority rule*), and the remaining 10% of the nodes to use first Rule 1', then Rule 2', and finally Rule 3'. We observed the average value of z_o that the 10% of the nodes could achieve in each cases. We repeated the above experiment for packet generation rates of 0.2, 0.5, and 0.8 pkt/s. The results are depicted in Figures 7, 8, and 9.

Remarkably, Rule 1' performed the best in every case. This means that the 10% deviating nodes achieve the highest z_o (i.e., drop the smallest portion of their own packets) when they use Rule 1', no matter whether the 90% of the nodes use Rule 1', Rule 2', or Rule 3'. Furthermore, this is true for every packet generation rate that we have simulated. Therefore, our conclusion is that the proposed stimulation mechanism indeed stimulates packet forwarding, and not only in the simple model of Section 2, but in a much more general setting too.

4.2.2 The effect of less cooperative nodes on the throughput of the network

In the second set of simulations, our goal was to study the effect of less cooperative nodes on the throughput of the network when the proposed stimulation mechanism is used. As opposed to the simulation that yielded Figure 1, in which we assumed that some nodes fully deny packet forwarding, here we assumed that some nodes use the least cooperative forwarding rule (i.e., Rule 3'). The rational is that full denial of packet forwarding quickly

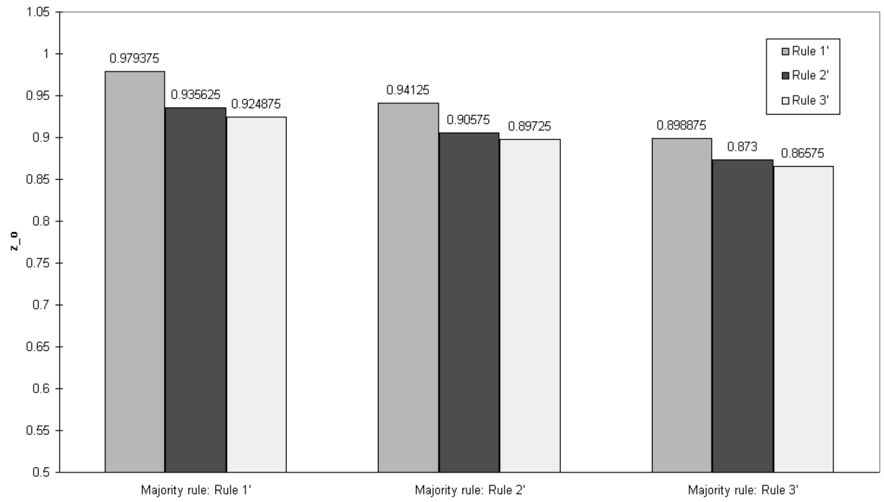

Figure 7: Comparison of the forwarding rules when the packet generation rate is 0.2 pkt/s

results in running out of credits, and thus, dropping own packets, and therefore, it is not beneficial at all for the nodes. On the other hand, Rule 3' can be viewed as a trade-off, where the node can send a large portion of its own packets, and it forwards only a small number of packets that is necessary to cover its expenses.

In this experiment, our approach was the following: We first set all the nodes to cooperate (i.e., to use Rule 1'), and then progressively increased the fraction of less cooperative nodes (i.e., the fraction of nodes that use Rule 3'). In order for the results to be comparable with the results shown in Figure 1, we ran simulations with networks of 100, 200, 300, and 400 nodes but with the same node density. We observed the cumulative throughput of the network, which is defined as the ratio of the total number of packets delivered to the total number of packets sent. The results are shown in Figure 10.

It can be seen that the throughput of the network decreases as the fraction of less cooperative nodes increases, but far less dramatically than in Figure 1. Even if all the nodes use Rule 3', the throughput is around 0.9.

The value of this experiment is that it shows that the network can tolerate less cooperative nodes quite well. A node may tend to be less cooperative, when it is about to run out of battery. In this case, it may not be beneficial to use Rule 1', and in this way, increase the credit reserve, because those credits cannot be used if the battery becomes empty. Therefore, the node may decide to use a less cooperative forwarding rule, or even to drop all forwarding packets. However, we note that the battery can usually be reloaded, and the accumulated credits can be used again. For this reason, it is not clear at all whether using a less cooperative rule when running out of battery is a good strategy or not. Nevertheless, the results of the above experiment show that the network would be able to cope with this situation.

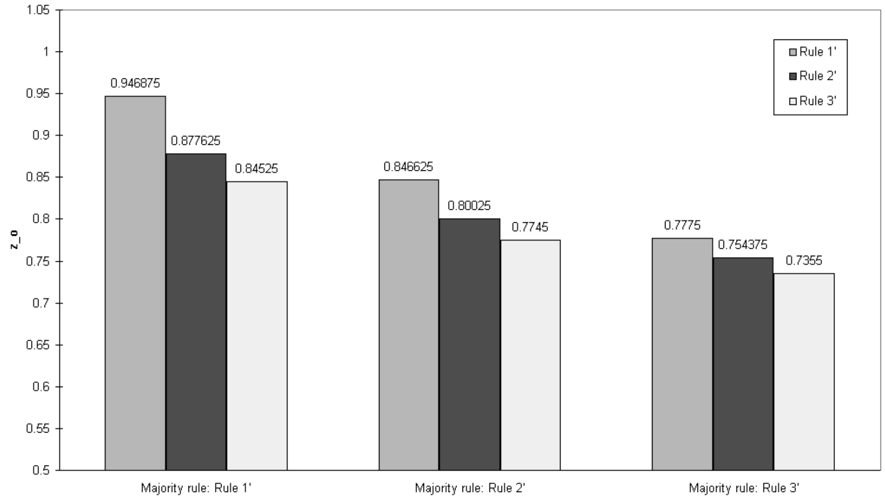

Figure 8: Comparison of the forwarding rules when the packet generation rate is 0.5 pkt/s

4.2.3 Variation of the average credit level in the network

In a third set of simulations, our goal was to study how the average credit level in the network is effected by the number of less cooperative nodes and by the size of the credit synchronization interval. To this end, we observed how the average credit level in the network varies in time as we increase the fraction of less cooperative nodes and as we increase the size of the credit synchronization interval. The results are shown in Figures 11 and 12.

When most of the nodes are cooperative, the average credit level in the network shows an increasing tendency. This is because the formula that we use to determine the number of intermediate nodes needed to reach a given destination under-estimates the actual number. This means that if a packet is delivered, then the joint credit income of the intermediate nodes is usually higher than the expenses of the source of the packet. Furthermore, when more nodes use Rule 1', packets are delivered with a higher probability, and thus, the average credit level increases more rapidly.

When less cooperative nodes are in majority, the average credit level in the network decreases. However, this decrease slows down, and after some time, it stops, and the average credit level becomes constant. The intuitive explanation is the following: When the nodes use Rule 3', their forwarding decisions depend on the current value of their credit counters. At the beginning, the average credit level is high, and packets are often dropped before they reach their destinations. This results in a decrease of the average credit level in the network. At the same time, the probability of dropping a packet due to the usage of Rule 3' also decreases, since the nodes have less credits in general, and they are more willing to forward. Therefore, more and more packets are delivered, and the decrease of the average credit level slows down. After some time, the decreasing effect of using Rule 3' (i.e., dropping packets) and the increasing effect of under-estimating the actual number of intermediate nodes needed to reach a given destination equalize each other, and the system attains an equilibrium. The fact that this equilibrium is below the initial value $C = 100$ of the credit counters explains why

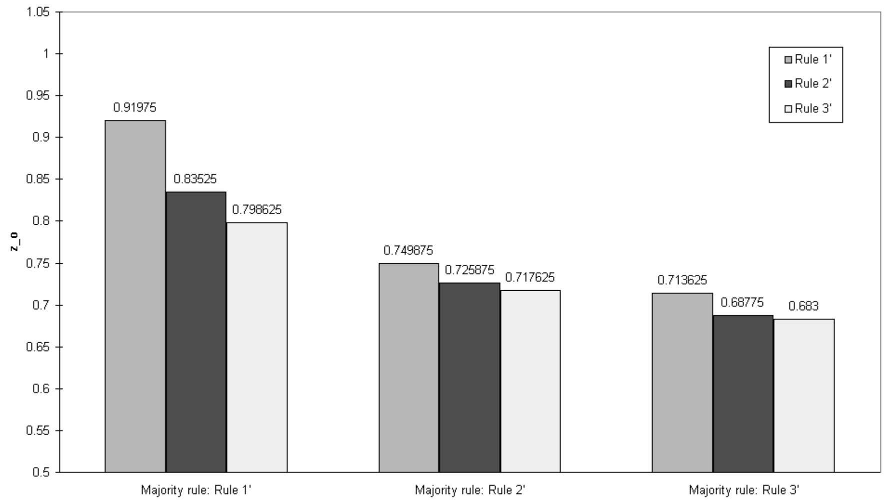

Figure 9: Comparison of the forwarding rules when the packet generation rate is 0.8 pkt/s

the throughput of the network is around 0.9 even if all the nodes use Rule 3' (see Figure 10). The reason is that in the equilibrium, most of the nodes have less than C credits (note that none of the nodes has more than C credits because of Rule 3'), and therefore, most of the nodes are willing to forward.

The effect of the credit synchronization interval on the average credit level in the network is not surprising: The larger the credit synchronization interval is, the slower the increase of the average credit level in the network is. Moreover, when the credit synchronization interval is 20 s, the average credit level continuously decreases in time. The reason is that when the credit synchronization interval is large, the probability that the neighbors of a node move away by the time of the next run of the credit synchronization protocol is high, and thus, the number of credits lost in the system is also high.

If mobility exhibits some locality properties, then this problem can be alleviated by slightly modifying the credit synchronization protocol, and letting the security module keep the accumulated pending credits for a given neighboring node in memory (until this memory is not needed for other purposes) even if that node has moved away and is not a neighbor anymore. In this case, because of the locality of mobility, nodes that were neighbors in the past may become neighbors again with a higher probability, which means that there are good chances that the pending credits can be cleared some time in the near future.

In any case, the size of the credit synchronization interval must be carefully chosen. If it is too small, then the credit synchronization protocol is run too often, which leads to a considerable overhead. However, if it is too large, then the average credit level in the network may become too low. Therefore, one has to find an appropriate trade-off.

An approach to limit the variation of the average credit level in the network would be to reset the credit counter to a reference value regularly. For instance, it could be reset each time the battery is reloaded. However, the security module, which maintains the credit counter, may not have reliable information about the battery reload events. On the other hand, since it maintains the credit counter, it can pretty well estimate the number of packets

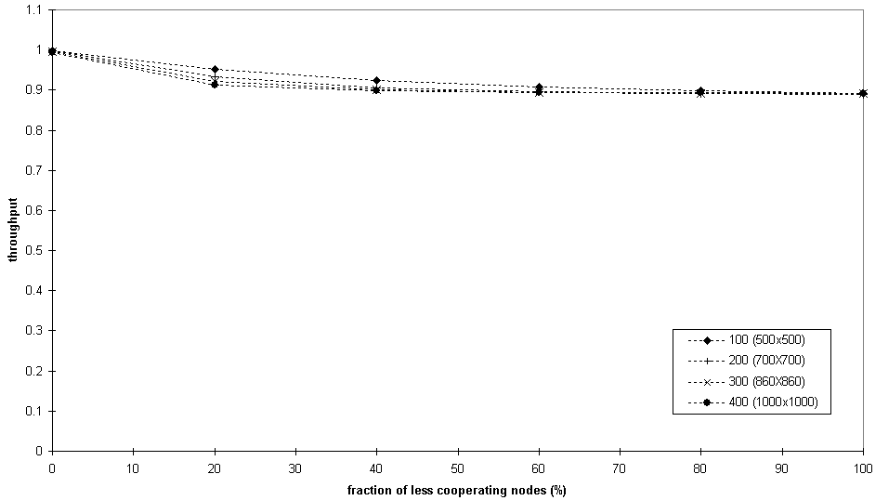

Figure 10: Effect of less cooperative nodes on the throughput of the network

sent by the node by observing the credit incomes and expenses. Thus, it can reset the credit counter after a given number of packets has been sent. This would eliminate the problem of ever increasing or ever decreasing average credit level in the network. However, it is not yet clear to us, what the consequences of this resetting mechanism are on the performance of the different forwarding rules. In particular, it seems, that in this case, the node's goal is not only maximizing z_o, but at the same time, it may want to minimize its credit loss due to the resetting mechanism. It is an open question which forwarding rule would be the best with respect to this new goal.

5 Related work

To the best of our knowledge, there are only two papers addressing the problem of non-cooperating nodes in mobile ad hoc networks: [11] and our previous paper [6]. The authors of [11] consider the case in which some malicious nodes agree to forward packets but fail to do so. In order to cope with this problem, they propose two mechanisms: a *watchdog*, in charge of identifying the misbehaving nodes, and a *pathrater*, in charge of defining the best route avoiding these nodes.

The paper shows that these two mechanisms make it possible to maintain the total throughput of the network at an acceptable level, even in the presence of a high amount of misbehaving nodes. However, the problem is that the selfishness of the nodes does not seem to be castigated; on the contrary, by the combination of the watchdog and the pathrater, the misbehaving nodes will not be bothered by the transit traffic while still enjoying the possibility to send and to receive packets. The proposed mechanisms could be enriched in such a way that a misbehaving node would be locked out by its neighbors. However, this possibility could be exploited to mount denial of service attacks.

In [6], we addressed the same problem as in this paper, and proposed a stimulation mechanism that is based on a virtual currency, called *nuglets*. Nuglets are used to pay for

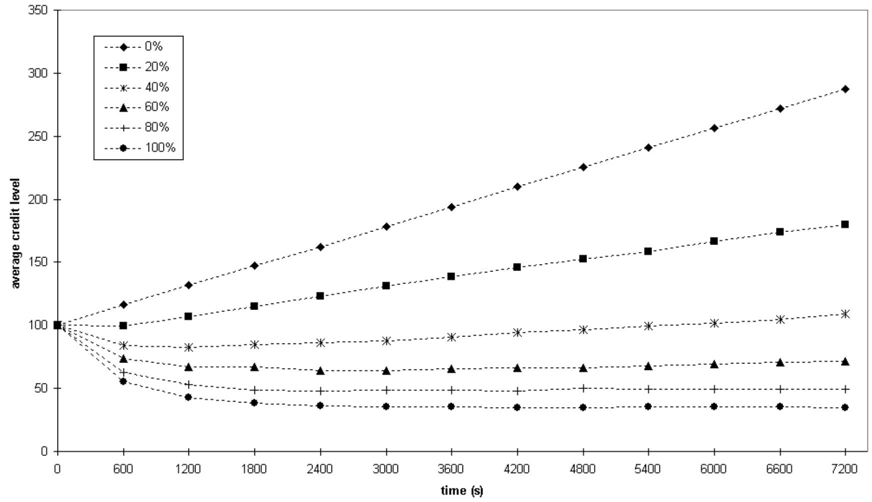

Figure 11: The effect of less cooperative nodes on the average credit level in the network

packet forwarding. We proposed two payment models for this purpose. In the Packet Purse Model, the source of the packet pays by loading some nuglets in the packet before sending it. Intermediate nodes acquire some nuglets from the packet when they forward it. If the packet runs out of nuglets, then it is dropped. In the Packet Trade Model, the packet does not carry nuglets, but it is traded for nuglets by intermediate nodes: Each intermediate node "buys" it from the previous one for some nuglets, and "sells" it to the next one (or to the destination) for more nuglets. In this way, each forwarding node earns some nuglets, and the total cost of forwarding the packet is covered by the destination.

A serious disadvantage of the Packet Trade Model is that it allows overloading of the network, since the sources do not have to pay. For this reason, mainly the Packet Purse Model has been studied. However, the Packet Purse Model has a problem too: it seems to be difficult to estimate the number of nuglets that the source should put in the packet initially. If the source under-estimates this number, then the packet will be discarded with a high probability, and the source loses its investment. The source may over-estimate the number, but this leads to a rapid decrease of the total number of nuglets in the system due to the dropping of packets (for networking reasons) with many nuglets inside.

The mechanism proposed in this paper overcomes this estimation problem, because the packets do not need to carry credits. At the same time, the property of refraining users from overloading the network is retained. Otherwise, the two mechanisms have a very similar flavor, just like their protection schemes.

6 Conclusion and future work

In this paper, we addressed the problem of stimulating cooperation in self-organizing, mobile ad hoc networks for civilian applications, where the nodes are assumed to be "selfish", meaning that they try to maximize the benefits that they get from the network, while minimizing their contribution to it. We focused on a particular instance of this problem, namely, stimulating

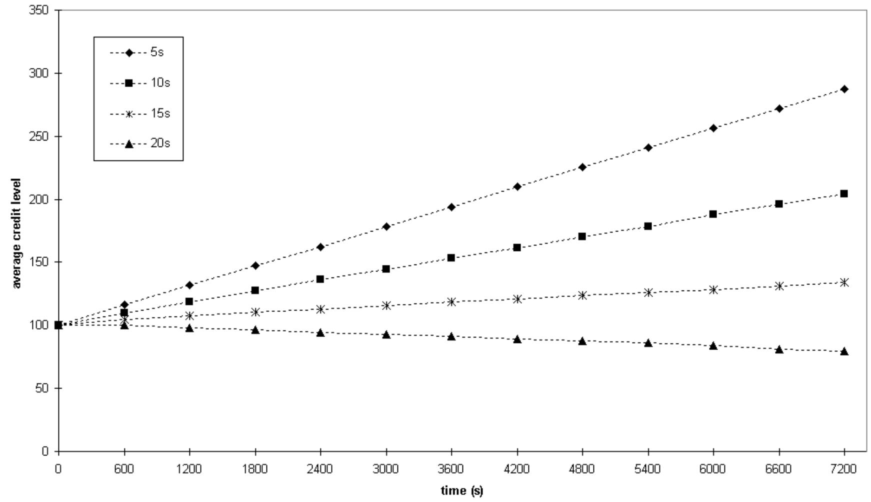

Figure 12: The effect of the credit synchronization interval on the average credit level in the network

packet forwarding. Our approach is based on a counter, called credit counter, in each node. Besides stimulating packet forwarding, the proposed mechanism encourages the users to keep their nodes turned on and to refrain from sending a large amount of packets to distant destinations.

In order to protect the proposed mechanism against misuse, we presented a scheme based on a trusted and tamper resistant hardware module, called security module, in each node, which generates cryptographically protected security headers for packets and maintains the credit counters of the nodes.

It is important to understand that the proposed stimulation mechanism and the proposed protection scheme are not intended to make misbehavior of the nodes impossible. For instance, nodes can still deny packet forwarding, or they may bypass the security module, and send a packet without a valid security header. What our design tries to ensure is that *misbehavior is not beneficial* for the nodes, and therefore, it should happen only rarely. For instance, if the node denies packet forwarding, then it runs out of credits, and it cannot send its own packets. Or, if the node sends a packet without a valid security header, then intermediate nodes will be reluctant to forward it. This is because an intermediate node can earn credits with packet forwarding only if it passes the forwarding packet to its security module. However, in the absence of a valid security header, the security module will reject the packet.

We studied the behavior of the proposed mechanism analytically and by means of simulations. We showed convincing arguments that it indeed stimulates the nodes for packet forwarding assuming that

- each node of the network generates packets continuously;
- generated packets cannot be buffered, which means that if they cannot be sent, then they must be dropped; and

- selfishness of the nodes is represented by the goal of dropping as few own packets as possible.

In our future work, we intend to study the behavior of the proposed mechanism, when these assumptions are weakened. For instance, we could allow buffering of packets, but limit the size of the buffer and the time that the packets can spend in it. Or, we could allow the nodes to generate packets in bursts instead of continuously. Finally, we would like to study the effect of regularly resetting the credit counters to a reference value (see discussion in Subsection 4.2.3), in which case the assumed goal of the nodes needs to be extended.

The work presented in this paper is focused on packet forwarding, as this is probably the most fundamental function of an ad hoc network. However, we are well aware of the fact that many other functions are required, including at the networking layer; an important example thereof is route discovery and route repair in on-demand protocols. In our future work, we intend to explore the way to generalize the proposed mechanism to these functions as well.

We also intend to address application-level aspects. In peer-to-peer computing, there is a growing concern that some users might parasitically take advantage of resources provided by others (see e.g., [1]). Some researchers have made early attempts to introduce a virtual currency to encourage "good citizenship" (e.g., www.mojonation.net). A further, more general ambition of our research is to explore how mechanisms like the one proposed in this paper could be used to application-level issues. An example thereof could be the mutual provision of information services in an ad hoc network. We believe that such aspects must be properly addressed in order to fully take advantage of the self-organization property of ad hoc networks.

Acknowledgement

We are grateful to Bharath Anathasubramaniam for implementing parts of the simulator and performing parts of the simulations.

References

[1] E. Adar and B. Huberman. Free Riding on Gnutella. *First Monday*, 5(10), October 2000.

[2] R. Anderson and M. Kuhn. Tamper Resistance – a Cautionary Note. In *Proceedings of the Second Usenix Workshop on Electronic Commerce*, Oakland, California, November 1996.

[3] L. Blažević, S. Giordano, and J.-Y. Le Boudec. Self-Organizing Wide-Area Routing. In *Proceedings of SCI 2000/ISAS 2000*, Orlando, July 2000.

[4] L. Blažević, L. Buttyán, S. Čapkun, S. Giordano, J.-P. Hubaux, and J.-Y. Le Boudec. Self-Organization in Mobile Ad Hoc Networks: The Approach of Terminodes. *IEEE Communications Magazine*, June 2001.

[5] J. Broch, D. Maltz, D. Johnson, Y. C. Hu, and J. Jetcheva. A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom)*, Dallas, 1998.

[6] L. Buttyán and J.-P. Hubaux. Enforcing Service Availability in Mobile Ad-Hoc WANs. In *Proceedings of the 1st IEEE/ACM Workshop on Mobile Ad Hoc Networking and Computing (MobiHOC)*, Boston, August 2000.

[7] S. Corson, J. Freebersyser, and A. Sastry (eds.). *Mobile Networks and Applications (MONET).* Special Issue on Mobile Ad Hoc Networking, October 1999.

[8] P. Gupta and P. R. Kumar. The Capacity of Wireless Networks. *IEEE Transactions on Information Theory*, March 2000.

[9] J.-P. Hubaux, Th. Gross, J.-Y. Le Boudec, and M. Vetterli. Towards Self-Organized Mobile Ad Hoc Networks: The Terminodes Project. *IEEE Communications Magazine*, January 2001.

[10] J.-P. Hubaux, L. Buttyán, and S. Čapkun. The Quest for Security in Mobile Ad Hoc Networks. In *Proceedings of the 2nd IEEE/ACM Workshop on Mobile Ad Hoc Networking and Computing (MobiHOC)*, Long Beach, CA, October 2001.

[11] S. Marti, T. J. Giuli, K. Lai, and M. Baker. Mitigating Routing Misbehavior in Mobile Ad Hoc Networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom)*, Boston, August 2000.

[12] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.

[13] C. Perkins (ed). *Ad Hoc Networking.* Addison-Wesley, 2001.

[14] A. Pfitzmann, B. Pfitzmann, and M. Waidner. Trusting Mobile User Devices and Security Modules. *IEEE Computer*, February 1997.

[15] G. J. Pottie and W. J. Kaiser. Wireless Integrated Sensor Networks. *Communications of the ACM*, May 2000.