This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE SECON 2005 proceedings.

Stimulating Cooperation and Defending Against Attacks in Self-Organized Mobile Ad Hoc Networks

Wei Yu and K. J. Ray Liu
Department of Electrical and Computer Engineering
and The Institute for Systems Research
University of Maryland, College Park, MD 20742
Email: weiyu, kjrliu@isr.umd.edu

***Abstract*— In self-organized mobile ad hoc networks, nodes belong to different authorities and pursue different goals, so they tend to be selfish, that is, they try to maximize the benefits they can get from the network, but are not willing to forward packets for the benefits of other nodes. Meanwhile, some nodes may be malicious whose objective is to degrade the network's performance and waste other nodes' valuable resource. In this paper, we propose an Attack-Resilient Cooperation Stimulation (ARCS) system for self-organized mobile ad hoc networks to stimulate cooperation among selfish nodes and to defend against attacks. In the ARCS system, the damage that can be caused by malicious nodes is bounded, and the cooperation among selfish nodes is enforced. Both theoretical analysis and simulation studies have confirmed the effectiveness of the ARCS system. Another key property of the ARCS system lies in that it is completely self-organizing and fully distributed, and does not require any tamper-proof hardware or central management points.**

I. Introduction

A *mobile ad hoc network* is a group of mobile nodes without requiring centralized administration or fixed network infrastructure, in which nodes can communicate with other nodes out of their direct transmission ranges through cooperatively forwarding packets for each other. In emergency or military situations, the nodes in an ad hoc network usually belong to the same authority and pursue the common goals. To maximize the overall performance, nodes usually work in a fully cooperative way. Recently, emerging applications of ad hoc networks have also been envisioned in civilian usage [1]–[8], where nodes typically do not belong to a single authority and may not pursue a common goal. Furthermore, such a network could be completely *self-organized*, where the network could be run solely by the operation of the end-users. Consequently, fully cooperative behaviors such as unconditionally forwarding packets for others cannot be taken for granted. On the contrary, in order to save limited resources, such as battery power, nodes tend to be "selfish". We refer to such networks as *self-organized ad hoc networks*.

Before ad hoc networks can be successfully deployed in a self-organized way, *cooperation stimulation* and *security* issues must be resolved first. To stimulate cooperation among selfish nodes, one possible way is to apply payment-based approaches, such as [2]–[4]. However, these schemes require either tamper-proof hardware or online central management points. Moreover, these schemes only considered nodes' selfish behaviors. Another possible way to stimulate cooperation is to employ reputation-based schemes [6]–[8]. However, these schemes also suffer from some problems. First, many attacks can cause a malicious behavior not being detected. Second, these schemes can only isolate misbehaving nodes, but cannot actually punish them, and the malicious nodes can still utilize the valuable network resources even after they have been suspected or detected. Third, extra energy needs to be consumed when a node keeps monitoring its neighbors' activities.

Previous experiences have also shown that before ad hoc networks can be successfully deployed, security concerns must be addressed [6], [9]–[13]. However, past experiences also show that security in ad hoc networks is particularly hard to achieve [11]. For self-organized ad hoc networks, things are even worse: there are no centralized monitoring or management points and nodes may tend to be selfish. In the literature many schemes have been proposed to address the security issues in ad hoc networks. However, most of the focus is on preventing attackers from entering the network through secure key distribution and secure neighbor discovery, such as [12]–[18], they cannot handle well the situation that the malicious nodes have entered the network, while in self-organized ad hoc networks the access control is usually loose, and malicious users can easily join the network.

In this paper we consider the scenarios that there exist both selfish and malicious nodes in self-organized ad hoc networks. The objective of selfish nodes is to maximize the benefits they can get from the network, while the objective of malicious node is to maximize the damage they can cause to the network. In this paper, we propose an Attack-Resilient Cooperation Stimulation (ARCS) system for self-organized ad hoc networks to stimulate cooperation among selfish nodes in adversarial environments. Besides being robust to various various, another key property of the ARCS system is that it does not require any tamper-proof hardware or central management points, which is very suitable for self-organized ad hoc networks. Both system analysis and simulation studies will confirm the effectiveness of the ARCS system.

The rest of the paper is organized as follows. Section II describes the system model and formulates the problem.

0-7803-9011-3/05/$20.00 ©2005 IEEE.

Section III describes the proposed ARCS system. Section IV presents the performance analysis of the system under various attacks. Simulation studies are presented in Section V. Finally Section VI concludes this paper.

II. Modelling and Formulation

A. System Model

In this paper we consider self-organized mobile ad hoc networks where nodes belong to different authorities and have different goals. We assume that each node is equipped with a battery with limited power supply, communicates with others through wireless connections, can move freely inside a certain area, and may act as a service provider: packets are scheduled to be generated and delivered to certain destinations with each packet having a specific delay constraint. If a packet cannot reach the destination within its delay constraint, it will become useless. If a packet can be successfully delivered to its destination within the specified delay constraint, the source of the packet will get some payoff, otherwise, it will be penalized.

According to their objectives, the nodes in the network can be classified into two types: selfish and malicious. The objective of selfish nodes is to maximize the payoff they can get using their limited resource, while the objective of malicious nodes is to maximize the damage they can cause to the network, where the damage is defined as the difference between the resource wasted by them due to their malicious behavior and the resource contributed by them in successfully forwarding packets for other nodes. Since energy is usually the most precious and valuable resource for battery-supplied nodes in ad hoc networks, we restrict the resource constraint to be energy. However, the proposed system is also applicable to other types of resource constraints.

For each node in the network, the energy consumption may come from many aspects, such as processing, transmitting and receiving packets. In this paper, we focus on the energy consumed in communication-related activities. One reason is that this portion of energy is necessary to support the basic networking functions, such as forwarding packets for each other. Another reason is that in wireless communications the energy consumed in communication-related activities usually plays a major role in overall energy consumption.

In this paper we mainly consider the situations that malicious nodes have entered the network, that is, all nodes in the network are legitimate, no matter selfish or malicious. To prevent illegitimate nodes from entering the network, the existing secure protocols can be used as the first defense line. For malicious nodes, in order to waste other nodes' energy, the following attacking models are considered:

- *Drop packet*: When a packet is dropped by an intermediate node on its route, all the energy that has been spent to transmit this packet from its source to this intermediate node is wasted.
- *Inject traffic*: Malicious nodes can inject an overwhelming amount of packets to overload the network and to consume other nodes's valuable energy. When other nodes have forwarded these packets but cannot get corresponding payback from attackers, the consumed energy is wasted. Besides injecting general data packets, the attackers can also inject control message packets.
- *Prevent good routes from being discovered*: Besides dropping packets and injecting traffic, attackers can also waste other nodes' energy by preventing good routes from being discovered. Two examples are wormhole [17] and rushing attacks [13].
- *Collusion*: Attackers can try to work together in order to improve their attacking capabilities.

TABLE I
Notations

E	The total amount of energy needed to encrypt/decrypt, encode/decode, and transmit/receive a data packet.
E_S^{max}	S's total energy when it first enters the network.
α_S	The payoff to S if a data delivery transaction with S being the source succeeds.
β_S	The penalty to S if a data delivery transaction with S being the source fails.
E_S	Energy spent by S until the current moment.
N_S^{succ}	# successful data packet deliveries until the current moment with S being the source.
N_S^{fail}	# unsuccessful data packet deliveries until the current moment with S being the source.
E_S^{waste}	Energy that has been wasted by S until the current moment due to its malicious behavior.
E_S^{contr}	Energy that S has spent until the current moment to successfully transmit packets for others.

We assume that each node has a public/private key pair, and assume that a node can know or authenticate other nodes' public keys, but no node will disclose its private key to others unless it has been compromised. To keep the confidentiality and integrity of the content, all packets should be encrypted and signed by their senders when necessary. We assume that the *acknowledgement* mechanism is supported in the link layer. That is, if node A has transmit a packet to node B and B has successfully received it, node B needs to immediately notify A of the reception through link level acknowledgement.

B. Problem Formulation

Before formulating the problem, we first introduce some notations to be used, as listed in Table I. Without loss of generality, we assume that all data packets have the same size, and the transmitting power is fixed for all nodes. We use "packet delivery transaction" to denote sending a packet from its source to its destination, and say that a transaction is "successful" if the packet has successfully reached its destination within its delay constraint, otherwise, the transaction is called as "unsuccessful".

For each selfish node S, its objective can be formulated as follows:

$$\max \left(\alpha_S N_S^{succ} - \beta_S N_S^{fail} \right) \quad \text{s.t.} \quad E_S \leq E_S^{max}. \qquad (1)$$

If S is malicious, then the total damage D_S that S has caused to other selfish nodes until the current moment can

TABLE II

RECORDS KEPT BY NODE S

$C_{redit}(A,S)$	Total energy spent by A until now on successfully transmitting packets for S.
$D_{ebit}(A,S)$	Total energy that S has spent until now on successfully transmitting packets for A.
$W_{by}(A,S)$	Total wasted energy that A has caused to S until the current moment.
$W_{to}(A,S)$	Total wasted energy that S has caused to A until the current moment.
$LB(A,S)$	Total wasted energy caused to S until now due to link breakages between A and S.
$B_{lacklist}(S)$	Set of nodes that S believes are malicious and S does not want to work together with.
$B_{lacklist}(A,S)$	The subset of A's blacklist known by S until the current moment.

be calculated as follows:

$$D_S = E_S^{waste} - E_S^{contr}. \tag{2}$$

Since in the current system model malicious nodes are allowed to collude, in this paper we only formulate the overall objective of malicious nodes, which is as follows:

$$\max \sum_{\text{S is malicious}} D_S \quad \text{s.t.} \quad E_S \le E_S^{max}. \tag{3}$$

III. DESCRIPTION OF ARCS SYSTEM

This section presents the proposed Attack-Resistant Cooperation Stimulation (ARCS) system for self-organized mobile ad hoc networks. In the ARCS system, each node S keeps a set of records indicating its interactions with other nodes, as listed in Table II. In a nutshell, the data delivery in the ARCS system works as follows: When a service provider has a packet scheduled to be sent, it first checks whether this packet should be sent and which route should be used. Once an intermediate node on the selected route receives a packet forwarding request, it checks whether it should forward the packet based on its relationship with the other nodes on the route. Once a node has successfully forwarded a packet on behalf of another node, it requests a receipt from its next node on the route and submits this receipt to the source of the packet to claim credits. After a packet delivery transaction finishes, no matter whether it is successful or not, all participating nodes will update their own records to reflect the changing relationships with other nodes and to detect possible malicious behavior.

A. Cooperation Degree

In [19], Dawkins illustrated that reciprocal altruism is beneficial for every ecological system when favors are granted simultaneously, and gave an example to explain the survival chances of birds grooming parasites off each other's head, which they cannot clean themselves. In that example, Dawkins divided the birds into three categories: *suckers*, which always help; *cheats*, which ask other birds groom parasites off their heads but never help others; and *grudgers*, which start out being helpful to every bird but refuse to help those birds that do not return the favor. The simulation study has shown that both cheats and suckers extinct finally, and only grudgers win over time.

Similar to those birds, in order to best utilize their limited resources, selfish nodes in self-organized ad hoc networks should also act like grudgers. In the ARCS system, each selfish node S keeps track of the *balance* $B(A,S)$ with any other node A known by S, which is defined as:

$$B(A,S) = (D_{ebit}(A,S) - W_{to}(A,S)) - (C_{redit}(A,S) - W_{by}(A,S)). \tag{4}$$

That is, $B(A,S)$ is simply the difference between what S has contributed to A and what A has contributed to S until the current moment in S's point of view. If $B(A,S)$ is a positive value, it can be regarded as the relative damage that A has caused to S, otherwise, it is the relative help that S has received from A.

Besides keeping track of the balance, each node S also sets a threshold $B_{max}(A,S)$ for each known node A in the network, which we called *cooperation degree*. A necessary condition for S to help A, e.g., forwarding packet for A, is

$$B(A,S) < B_{max}(A,S). \tag{5}$$

Setting $B_{max}(A,S)$ to be ∞ means S will always help A no matter what A has done, as the *suckers* act in the example. Setting $B_{max}(A,S)$ to be $-\infty$ means S will never help A, as the *cheats* act in the example. In the ARCS system, each selfish node will set $B_{max}(A,S)$ to be a relatively small positive value, which means that initially S is helpful to A, and will keep being helpful to A unless the relative damage that A has caused to S exceeds $B_{max}(A,S)$, as the *grudgers* act in the example where they set the balance to be 1 for any other bird. By specifying positive cooperation degrees, cooperation among selfish nodes can be enforced, while by letting the cooperation degrees to be relatively small, the possible damage caused by malicious nodes can be bounded.

B. Route Selection

Due to insufficient balance, malicious behavior and node mobility, not all packet delivery transactions can succeed. When a node has a packet scheduled to be sent, it needs to decide which route should be used. In the ARCS system, the following criterion is used when a node makes route selection decision.

In the ARCS system, each route is specified an expiring time indicating that after that time the route will become invalid, which is determined by the intermediate nodes during the route discovery procedure. Assume that S has a packet scheduled to be sent to D, route $R =$ "$R_0 R_1 \dots R_M$" is a valid route known by S with $R_0 = S$, $R_M = D$, and M being the number of hops. Let $P_{drop}(R_i,S)$ be the probability that node R_i will drop S's packet, and let $P_{delivery}(R,S)$ denote the probability that a packet can be successfully delivered from S to D through route R at the current moment. In the ARCS

system, S calculates $P_{delivery}(R,S)$ in the following way:

$$P_{delivery}(R,S)=\begin{cases}0 & (\exists R_i \in R)\ B(R_i,S) < -B_{max}(S,R_i)\\ 0 & (\exists R_i,R_j \in R)\ R_i \in B_{lacklist}(R_j,S)\\ \prod_{i=1}^{M-1}(1-P_{drop}(R_i,S)) & otherwise\end{cases} \quad (6)$$

That is, a packet delivery transaction has no chance to succeed unless S has enough balance to request help from all intermediate nodes on the route and no node has been marked as malicious by any other node on the route. Once a valid route R with non-zero $P_{delivery}(R,S)$ is used to send a packet by S, we define $E_{fail}(R,S)$ as

$$E_{fail}(R,S)=\sum_{n=1}^{M-1} nE\left(\prod_{k=1}^{n-1}(1-P_{drop}(R_k,S))\right)P_{drop}(R_n,S), \quad (7)$$

and define $E_{succ}(R,S)$ as

$$E_{succ}(R,S)=E\times M. \quad (8)$$

Then the expected energy consumption of using route R to send a packet from S to D becomes

$$E_{avg}(R,S)=E_{succ}(R,S)P_{delivery}(R,S)+E_{fail}(R,S), \quad (9)$$

and the expected profit of S is

$$P_{rofit}(R,S)=\alpha_S P_{delivery}(R,S)-\beta_S(1-P_{delivery}(R,S)). \quad (10)$$

Let $Q(R,S)$ be the expected *energy efficiency* of S using route R to send a packet to D, which is defined as

$$Q(R,S)=\frac{P_{rofit}(R,S)}{E_{avg}(R,S)}. \quad (11)$$

That is, $Q(R,S)$ is the expected profit per unit energy when S uses R to send a packet to D at the current moment. In the ARCS system, which route should be selected is decided as follows:

Route Selection Decision: *Among all valid routes $\mathcal{R}$ known by S which can reach D, route R^* will be selected if and only if $P_{delivery}(R^*,S) > 0$ and $Q(R^*,S) \geq Q(R,S)$ for any other route $R \in \mathcal{R}$.*

It is easy to see that the decision is optimal at the current moment in the sense that no other known routes can provide better expected energy efficiency than route R^*. Since the accurate value of $P_{drop}(R_i,S)$ is usually not known and may change over time, in the ARCS system, we can use the percentage of S's failed transactions caused by R_i in S's total transactions passing through R_i as the estimate of $P_{drop}(R_i,S)$.

C. Data Packet Delivery Protocol

Once a service provider has decided to send a packet to a certain destination using a certain route, a data packet delivery transaction should be started. In the ARCS system, the data packet delivery protocol consists of two stages: *forwarding data packet stage* and *submitting receipts stage*. In the forwarding data packet stage, the data packet is delivered from the source to the destination, while in the submitting receipts stage, each participating node on the route tries to submit a receipt to the source to claim credit. Some notations to be used in the protocols are listed below:

$sign_S(m)$	S creates a signature based on message m.
$verify_S(m,s)$	Verify whether s is the signature generated by node S based on message m.
$v \leftarrow m$	Assign the value of m to the variable v.
$MD()$	A message digest function.

Protocol 1 Forward a Data Packet

▷ A is the current node, S is the sender, D is the destination, and $(m,R,seq_S(S,D),s)$ is the received data packet from A's previous node if $A \neq S$, otherwise, $(m,R,seq_S(S,D),s)$ is the data packet generated by A.

if (A = S) **then**
  S forwards $(m,R,seq_S(S,D),s)$ to next node, increases $seq_S(S,D)$ by 1, and wait for receipts to be returned.
else if $((A = D)$ AND $(seq_S(S,D) > seq_A(S,D))$ AND $(verify_S((m,R,seq_S(S,D)),s) =$ TRUE$))$ **then**
  A assigns the value of $seq_S(S,D)$ to $seq_A(S,D)$, and returns a receipt to its previous node.
else
  if $((A \notin R)$ OR $(seq_S(S,D) \leq seq_A(S,D))$ OR $(verify_S((m,R,seq_S(S,D)),s) \neq$ TRUE)
    OR $(\exists R_i \in R, R_i \in B_{lacklist}(A))$) **then**
    A simply drops this packet.
  else if $((B(S,A) > B_{max}(S,A))$ OR (the link to A's next node is broken)) **then**
    A drops the packet, and returns a receipt to its previous node which also includes the dropping reason.
  else
    A assigns the value of $seq_S(S,D)$ to $seq_A(S,D)$, forwards $(m,R,seq(S,D),s)$ to its next node, and wait for a receipt to be returned by the next node.
  end if
end if

1) Forwarding Data Packet Stage: In the forwarding data packets stage, the protocol execution of each participating selfish node is illustrated in Protocol 1. Suppose that node S is to send a packet with payload *m* and sequence number $seq_S(S,D)$ to destination D through the route R. For the sender S, it first computes a signature $s = sign_S(MD(m),R,seq_S(S,D))$. Next, S transmits the packet $(m,R,seq_S(S,D),s)$ to the next node on the route, increases $seq_S(S,D)$ by 1, and waits for receipts to be returned by the following nodes on route R. Once a selfish node A has received the packet $(m,R,seq_S(S,D),s)$, A first checks whether itself is the destination of the packet. If it is the destination, after necessary verifications, A returns a receipt to its previous node on the route to confirm the successful delivery, otherwise, A checks whether the packet should be forwarded. A is willing to forward the packet only if all the following conditions are satisfied: 1) A is on the route R; 2) $seq_S(S,D) > seq_A(S,D)$, where $seq_A(S,D)$ is the sequence number of the last packet that A has forwarded with S being source and D being the destination; 3) the signature is valid; 4) $B(S,A) < B_{max}(S,A)$; 5) no node on route R has been marked as malicious by A.

Once A has successfully forwarded the a packet to the next node on the route, it will specify a time to wait for a receipt being returned by the next node before that time to confirm

the successful transmission, which A will use to claim credit from S. In the ARCS system, a selfish node sets its waiting time to be the value of T_{hop} multiplied by the number of hops following this node, where T_{hop} is a relatively small interval to account for the necessary time needed per hop. Since in general the waiting time is small enough, we can assume that if a node can return a receipt to its previous node in time, the two nodes will still keep connected.

2) Submitting Receipts Stage: In self-organized ad hoc networks, nodes may not be willing to forward packets on behalf of other nodes. So after a node (e.g., A) has forwarded a packet $(m, R, seq_S(S, D), s)$ for another node (e.g., S), A will try to claim corresponding credits from S which A can use later to request S to return the favor. To claim credits from S, A needs to submit necessary evidence to convince S that it has successfully forwarded packets for S. In the ARCS system, in order for A to show that it has successfully forwarded a packet for S, A only needs to request a receipt from its next node on the route (e.g., B) indicating that B has successfully received the packet. One possible format of such a receipt can be $\{MD(m), R, seq_S(S, D), B, sign_B(MD(m), R, seq_S(S, D), B)\}$. That is, it consists of $\{MD(m), R, seq_S(S, D), B\}$ and the signature generated by node B based on this message.

Actually, a receipt generated by any node following A on the route can be used as the evidence to convince S that A has successfully forwarded a packet for S. Sometimes a packet will be dropped due to link breakage or the requester running out of balance, and sometimes the next node may not return a receipt even if the transmission is successful, such as when the next node is malicious. For each selfish node, if it has dropped the packet or cannot get a receipt from its next node in time, or the received receipt is not valid, it will generate a receipt by itself and return it to its previous node, otherwise, it will simply send the received receipt back to its previous node on the route. The specific protocol execution of a selfish node submitting receipt to the requester is described in Protocol 2.

Protocol 2 Submit a receipt

▷ A is the current node, $(MD(m), R, seq_S(S, D), B, s)$ is the successfully received packet being processed.

if (($A = D$) OR (no valid receipts have been returned from the next node after waiting enough time)) **then**
  $s \leftarrow sign_A(MD(m), R, seq_S(S, D), A)$.
  Send the receipt $\{MD(m), R, seq_S(S, D), A, s\}$ to A's previous node on R.
else
  $receipt = \{MD(m), R, seq_S(S, D), B, s\}$, which is the returned receipt from the next node on the route.
  if ($verify_B((MD(m), R, seq_S(S, D), B), s)$ = TRUE)) **then**
    Send $receipt$ to A's previous node on R.
  else
    $s \leftarrow sign_A(MD(m), R, seq_S(S, D), A)$.
    Send the receipt $\{MD(m), R, seq_S(S, D), A, s\}$ to A's previous node on R.
  end if
end if

D. Update Records

In the ARCS system, after a packet delivery transaction has finished, no matter whether it is successful or not, each participating node will update its records to keep track of the changing relationships with other nodes and to detect possible malicious behavior. Next we use Fig. 1 to illustrate the records updating procedure, where S is the initiator of the transaction, D is the destination, and "$R = S \ldots AMB \ldots D$" is the associated route.

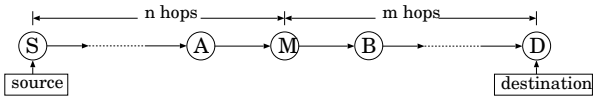


Fig. 1. Records updating

For the sender S, according to different situations, it updates its records as follows:

- *Case 1*: S has received a valid receipt signed by D which means that this transaction has succeeded. In this case, for each intermediate node X between S and D, S increases $C_{redit}(X, S)$ by E, where E is the amount of energy needed to transmit one packet.
- *Case 2*: S has successfully sent a packet to its next node, but cannot receive any receipt in time. In this case, let X be S's next node, S then increases $W_{by}(X, S)$ by E, and marks X as malicious. That is, refusing to return a receipt will be regarded as malicious behavior.
- *Case 3*: If S has received a valid receipt which is not signed by D, but signed by an intermediate node (e.g., M), which means either M has dropped the packet, or a returned receipt has been dropped by a certain node following M (including M) on the route in the submitting receipt stage. In this case, for each intermediate node X between S and M, S still increases $C_{redit}(X, S)$ by E. Since node M's transmission cannot be verified by S, S has enough evidence to suspect that the packet is dropped by M. To reflect this suspect, S increases $W_{by}(M, S)$ by nE to account for the amount of energy that has been wasted in this transaction with n being the number of hops between S and M.

When a packet delivery transaction fails, S also keeps a record of $(MD(m), R, seq_S(S, D), s)$ for this transaction as well as a copy of the returned receipt if there exists, where $MD(m)$ is the digest of the message m, $seq_S(S, D)$ is sequence number of this packet, and s is the signature generated by node S based on the message $\{MD(m), R, seq_S(S, D)\}$.

For each intermediate node (e.g., node M in Fig. 1) that has participated in the transaction, if it is selfish, it updates its records as follows:

- *Case 1*: M has successfully sent the packet to node B, and has got a receipt from B to confirm the transmission. Then M only needs to increases $D_{ebit}(S, M)$ by E.
- *Case 2*: M has successfully sent the packet to node B, but cannot get a valid receipt from B. In this case, M

increases $W_{to}(S, M)$ by nE, increases $W_{by}(B, M)$ by $(n+1)E$, and marks B as malicious.

- *Case 3*: M has dropped the packet due to link breakage between M and B. Although this packet dropping is not M's fault, since M cannot prove it to S, M will take the responsibility. However, since this link breakage may be caused by S who has selected a bad route, or caused by B who tries to emulate link breakage to attack M, M should also records this link breakage. In this case, M increases $W_{to}(S, M)$ by nE, increases $LB_{with}(B, M)$ and $LB_{with}(S, M)$ by nE. In the ARCS system, each selfish node (e.g., M) sets a threshold $LB_{th}(S, M)$ with any other node (e.g., S) to indicate the damage that M can tolerate which is caused due to link breakages between M and S. In this case, if $LB_{with}(B, M)$ exceeds $LB_{th}(B, M)$, B will be put in M's blacklist. Similarly, if $LB_{with}(S, M)$ exceeds $LB_{th}(S, M)$, S will be put in M's blacklist.
- *Case 4*: M has dropped the packet due to the reason that the condition in (5) is not satisfied or some nodes on R are in M's blacklist. In this case M does not need to update its records.

From the above update procedure we can see that a selfish node will always return a receipt to confirm a successful packet reception, since refusing to return receipt is regarded as malicious behavior and cannot provide any gain.

E. Secure Route Discovery

In the ARCS system, DSR [20] is used as the underlying routing protocol to perform route discovery. However, without security consideration, the routing protocol can easily become an attacking target. For example, malicious nodes can inject an overwhelming amount of route request packets to overload the network and consume other nodes' valuable resource. In the ARCS system, besides the necessary authentication, the following security enhancements have also been incorporated into the route discovery protocol:

1. When node S initiates a route discovery, it can also append the subset of its blacklist not known by others in the route request packet. After an intermediate node A has received the request packet, it will update its own record $B_{lacklist}(S, A)$ using the received blacklist.
2. When an intermediate node A receives a route request packet which originates from S and A is not this request's destination, A first checks the following conditions: 1) A has never seen this request before; 2) A is not in S's blacklist; 3) $B(S, A) < B_{max}(S, A)$; 4) no nodes that have been appended to the request packet are in A's blacklist; 5) A has not forwarded any request for S in the last $T_{min}(S, A)$ interval, where $T_{min}(S, A)$ is the minimum interval specified by A to indicate that A will forward at most one route request for S in each $T_{min}(S, A)$ interval. A will broadcast the request if and only if all of the above conditions can be satisfied, otherwise, A will discard the request.
3. During a discovered route is being returned to the requester S, each intermediate node A on the route appends the following information to the returned route: the subset of its blacklist that is not known by S, the value of $B_{max}(S, A)$ if not known by S, the value of $D_{ebit}(S, A)$, and node A's expected staying time in the current area. After S has received the route, for each node A on the discovered route, it updates the corresponding blacklist $B_{lacklist}(A, S)$, updates the value of $B_{max}(S, A)$, determines the expiring time of this route which is defined as the expected minimum staying time among all nodes on the route, and checks the consistency between $D_{ebit}(S, A)$ and $C_{redit}(A, S)$, where the procedure to handle inconsistent records is described below.

F. Resolve Inconsistent Records Update

In some situations, after a node (e.g., A) has successfully forwarded a packet for another node (e.g., S) and has sent a receipt back to S, the value of $C_{redit}(A, S)$ may not be increased immediately by S due to some intermediate node dropping the receipt returned by A. In this case, the value of $D_{ebit}(S, A)$ will be larger than the value of $C_{redit}(A, S)$, which we referred to as *inconsistent records update*. As a consequence, S may refuse to forward packets for A even the actual value of $B(A, S)$ is still less than $B_{max}(A, S)$, or S may continue requesting A to forward packets for it when the true value of $B(S, A)$ has exceeded $B_{max}(S, A)$. Next we describe how the inconsistent records update problem is resolved in the ARCS system.

In the route discovery stage, after route R has been returned to S, S will check whether there exists inconsistency. If S finds that a node A on route R has reported a larger value of $D_{ebit}(S, A)$ than the value of $C_{redit}(A, S)$, when calculating route quality, S should use the value of $D_{ebit}(S, A)$ to temporarily substitute the value of $C_{redit}(A, S)$. In the packet delivery stage, when route R is picked by S to send packets, for each intermediate node A on route R, the value of $C_{redit}(A, S)$ will also be appended to the payload of the data packet.

When A receives an appended value of $C_{redit}(A, S)$ from S, and finds $C_{redit}(A, S) < D_{ebit}(S, A)$, A will submit those receipts that target on S but have not been confirmed by S to claim corresponding credits, where we say a receipt received by A at time t_1 and targeting on S has been *confirmed* if there existed at least one moment $t_2 > t_1$ before now at which A and S have agreed that $C_{redit}(A, S) = D_{ebit}(S, A)$. Once S has received an unconfirmed receipt returned by A, S will check whether there is a failed transaction record associated to this receipt. If no such record exists, either the receipt is faked, or the corresponding credit has been issued to A. If there exists such a record, let B be the node who has signed the receipt associated to this transaction record, that is, all nodes between S and B have been credited by S. Let C be the node who has signed the receipt submitted by A. If B is in front of C on the route, S should use the new receipt singed by C to replace the previous receipt signed by B, and for each intermediate node

X between B and C on the route, S should update the values of $C_{redit}(X, S)$ and $W_{by}(C, S)$ according to Section III-D.

IV. Analysis of ARCS under Attacks

In this section we analyze the performance of the ARCS system under the following attack models: dropping packet, emulating link breakage, injecting traffic and collusion. The results show that the damage that can be caused by malicious nodes is bounded, and the system is collusion-resistant.

A. Dropping Packet Attacks

In the ARCS system, malicious nodes can waste other nodes' energy by dropping their packets, which can happen either in the forwarding data packet stage or in the submitting receipts stage. We use Fig. 2 as an example to analyze the possible dropping packet attacks that can be launched by malicious node M. Based on in which stage M drops packets and whether M will return receipts, there are four possible attacking scenarios:

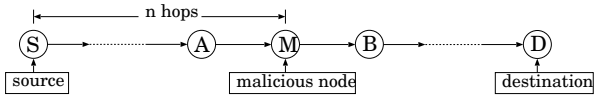


Fig. 2. Dropping Packets Attacks

- Scenario 1: M drops a packet in the forwarding data packet stage, but creates a receipt to send back to A to confirm successful receiving from A. In this scenario, after S gets the receipt, S will increase $W_{by}(M, S)$ by nE, which equals to the total amount of energy that has been wasted by M. That is, in this scenario, the damage caused by M has been recorded by S and needs to be compensated by M later if M still wants to get help from S.
- Scenario 2: M drops a packet in the forwarding data packet stage, and refuses to return a receipt to A. In this scenario, although A will be mistakenly charged by S who increases $W_{by}(A, S)$ by $(n-1)E$, A will mark M as malicious and will stop working with M further. That is, M can never get help from A and cause damage to A in the future.
- Scenario 3: M drops the receipt returned by B, but creates a receipt to send back to A. In this scenario, M will be charged nE by S, but the nodes after M who have successfully forwarded the packet will not be credited by S immediately. That is, by taking some charge (here nE), M can cause inconsistent records update. However, as described in Section III-F, this inconsistency can be easily resolved and will not cause further damage. That is, M can only cause temporary records inconsistency with the extra payment of $(n+1)E$.
- Scenario 4: M drops the receipt returned by B, and refuses to return a receipt to A. This scenario is similar to scenario 3 with the difference being that in this scenario A will be mistakenly charged by S, but M will be marked as malicious by A and cannot do any further damage to A in the future.

From the above analysis we can see that when a malicious node M launches dropping packet attacks, either it will be marked as malicious by some nodes, or the damage caused by it will be recorded by other nodes. Since for each node A, the maximum possible damage that can be caused by M is bounded by $B_{max}(M, A)$, the total damage that M can cause to the network is also bounded.

B. Emulating Link Breakage Attacks

Malicious nodes can also launch emulating link breakage attacks to waste other nodes' energy. For example, in Fig. 2, when node A has received a request from S to forward a packet to M, M can just keep silent to let A believe that the link between A and M is broken. By emulating link breakage, M can cause a transaction to fail and waste other nodes' energy.

In the ARCS system, each selfish node handles the possible emulating link breakage attacks as follows: For each known node M, S keeps a record $LB(M, S)$ to remember the damage that has been caused due to link breakage between M and S, and if $LB(M, S)$ exceeds the threshold $LB_{max}(M, S)$, S will mark M as malicious and will never work with M again. That is, the damage that can be caused to S by malicious node M who launched emulating link breakage attacks is bounded by $LB_{max}(M, S)$.

C. Injecting Traffic Attacks

Besides dropping packets, attackers can also inject an excessive amount of traffic to overload the network and consume other nodes' valuable energy. Two types of packets can be injected: general data packets and route request packets. In the ARCS system, according to the route discovery protocol, the number of route request packets that can be injected by each node is bounded by 1 in each time interval T_{min}. For general data packets, since an intermediate node A will stop forwarding packets for node M if $B(M, A) > B_{max}(M, A)$, the maximum damage that can be caused to node A by node M launching injecting general data traffic attacks is bounded by $B_{max}(M, A)$. In summary, by launching injecting traffic attacks, the maximum damage that can be caused by a malicious node M to node A is bounded.

D. Collusion Attacks

In order to increase their attacking capability, malicious nodes may choose to collude. Next we show that in the ARCS system colluding among malicious nodes cannot cause more damage to the network than working alone, that is, the ARCS system is collusion-resistant. First, it is easy to see that two nodes collude to launch injecting extra traffic attacks cannot increase the damage due to the existence of balance threshold (cooperation degree), and two nodes colluding to launch emulating link breakage attacks makes no sense, since each link breakage event has only two participants. Next we

consider two malicious nodes colluding to launch dropping packets attacks.

Given a packet delivery transaction, we first consider the case that the two colluding nodes are neighbor of each other. For example, as in Fig. 2, assume that M and B collude. When M drops the packet, M can still get (or generate by itself, since M may know B's private key) the receipt showing that M has successfully forwarded the packet. However, this cannot increase their total attacking capability, since B needs to take the charge for the damage caused by this packet dropping. That is, in this case M is released from the charge by sacrificing B.

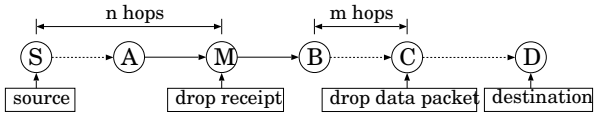


Fig. 3. Collusion Attacks

If two colluding nodes are not neighbor of each other, the only way that they can collude is that one node drops the data packet in the forwarding data packet stage, and the other node drops the receipt in the submitting receipt stage, as shown in Fig. 3, where node C drops the data packet and node M drops the receipt. By colluding in this way, if C has returned a receipt to its previous node, C will not be charged by S temporarily, and all the nodes between M and C cannot get credits from S immediately. For node M, if M will return a receipt to A, S will increase $W_{by}(M, S)$ by nE, and if M refuses to return a receipt to A, M will be marked as malicious by A. That is, in this case, temporary inconsistent records update can be caused, but the colluding nodes will be overcharged by nE. However, according to Section III-F, the inconsistency can be easily resolved.

Theorem 1 *Assume in the ARCS system there are L selfish nodes $\{S_1, \ldots, S_L\}$ and K malicious nodes $\{M_1, \ldots, M_K\}$. Let $B_{max}(M_k, S_l)$, $LB_{max}(M_k, S_l)$ and $T_{th}(M_k, S_l)$ be the cooperation degree, the link breakage threshold, and the minimum route request forwarding interval that node S_l has set for node M_k, respectively. Let T_{S_l} be node S_l's staying time in the system, and let $E_{request}$ (which is far less than E) be the consumed energy per route request forwarding, then the total damage that can be caused by all the malicious nodes is bounded by*

$$\sum_{k=1}^{K}\sum_{l=1}^{L}\left(B_{max}(M_k, S_l) + LB_{max}(M_k, S_l) + \frac{T_{S_l} * E_{request}}{T_{min}(M_k, S_l)}\right)$$

Proof: See the above analysis. ■

From the above theorem we can see that the damage that can be caused by malicious nodes is bounded, and is determined by those specified thresholds. Since the good values of $LB_{max}(M_k, S_l)$ and $T_{min}(M_k, S_l)$ can be obtained through training, and a relative small value of $B_{max}(M_k, S_l)$ can work well in most situations as shown in Section V, in general the above bound is small.

TABLE III
SIMULATION PARAMETERS

Total Number of Nodes	100
Number of Malicious Nodes	0-50
Maximum Velocity (v_{max})	10 m/s
Average Pause time	200 seconds
Dimensions of Space	1000m × 1000m
Maximum Transmission Range	250 m
Average Packet Inter-Arrival Time	2 seconds
Data Packet Size	1024 bytes
Link Bandwidth	1 Mbps

V. SIMULATION STUDIES

A. Simulation Configuration

We use an event-driven simulator to simulate self-organized ad hoc networks. There are 100 nodes in the network, which are randomly deployed inside a rectangular space. Each node moves randomly according to the *random waypoint* model [20]: a node starts at a random position, waits for a duration called the *pause time* that is modeled as a random variable with exponential distribution, then randomly chooses a new location and moves towards the new location with a velocity uniformly chosen between 0 and v_{max}. When it arrives at the new location, it waits for another random pause time and repeats the process. The physical layer assumes a fixed transmission range model, where two nodes can directly communicate with each other successfully only if they are in each other's transmission range. The MAC layer protocol simulates the IEEE 802.11 Distributed Coordination Function with a four-way handshaking mechanism [21]. Some simulation parameters are listed in Table III.

In the simulations, each selfish node acts as a service provider which randomly picks another selfish node as the receiver and packets are scheduled to be generated according to a Poisson process. Similarly, each malicious node also randomly picks another malicious node as the receiver to send packets. The total number of malicious nodes varies from 0 to 50. Among those malicious nodes, 1/3 launch dropping packets attacks which drop all packets passing through them whose sources are not malicious, 1/3 launch emulating link breakage attacks which emulate link breakage once receiving packet forwarding request from selfish nodes, and 1/3 launch injecting traffic attacks. For each selfish node or malicious node that does not launch injecting traffic attacks, the average packet inter-arrival time is 2 seconds, while for malicious nodes launching injecting traffic attacks, the average packet inter-arrival time is 0.1 second. In the simulations, all data packets have the same size.

Based on selfish nodes' forwarding decision, three types of systems have been implemented in the simulations: the proposed ARCS system, which we called "ARCS"; the ARCS system without balance constraint (i.e., cooperation degree is set to be infinity for all selfish nodes), which we called "ARCS-NBC"; and a fully-cooperative system, which we called "FULL-COOP". In "ARCS", all selfish nodes behave in the way as described in Section III. In "ARCS-NBC", the same

strategies as in "ARCS" have been used to detect launching dropping packets attacks and emulating link breakage attacks, but now (5) is not a necessary condition to forward packets for other nodes, and a selfish node will unconditionally forward packets for those nodes which have not been marked as malicious by it. In "FULL-COOP", all selfish nodes will unconditionally forward packets for other nodes, and no malicious nodes detection and punishment mechanisms have been used. In all three systems, the same route discovery procedure is used as described in Section III-E.

B. Performance metrics

We use $\{S_1, \ldots, S_L\}$ to denote the L selfish nodes and use $\{M_1, \ldots, M_K\}$ to denote the K malicious nodes in the network. In this paper, the following metrics are used to evaluate system performance:

- *Energy efficiency of selfish nodes*, which is defined as the total profits gained by all selfish nodes divided by the total energy spent by all selfish nodes until the current moment:

$$E_{fficiency} = \frac{\sum_{l=1}^{L}(\alpha_{S_l} N_{S_l}^{succ} - \beta_{S_l} N_{S_l}^{fail}) * E_{S_l}}{\sum_{l=1}^{L} E_{S_l}}. \quad (12)$$

- *Average damage received per selfish node*: which is defined as the total damage received by all selfish nodes divided by the total number of selfish nodes until the current moment:

$$D_{avg} = \frac{1}{L}\sum_{l=1}^{L} B(S_l). \quad (13)$$

  where $B(S_l)$ is node S_l's overall balance which is calculated as follows:

$$B(S_i) = \sum_{l=1}^{L} B(S_l, S_i) + \sum_{k=1}^{K} B(M_k, S_i). \quad (14)$$

- *Balance variation of selfish nodes*, which is defined as the standard deviation of selfish nodes' overall balance with the assumption that $\sum_{l=1}^{L} B(S_l) = 0$, that is,

$$V_{ariation} = \sqrt{\frac{1}{L}\sum_{l=1}^{L} B(S_l) * B(S_l)}. \quad (15)$$

  By assuming $\sum_{l=1}^{L} B(S_l) = 0$, this definition has incorporated the effects caused by malicious nodes, which will make $\sum_{l=1}^{L} B(S_l)$ deviate from 0. This definition also reflects the fairness for selfish nodes, where $V_{ariation} = 0$ implies absolute fairness, and the increase of $V_{ariation}$ implies the increase of possible unfairness for selfish nodes.

C. Simulation results

In our simulations, each configuration has been run 10 independent rounds using different random seeds, and the result are averaged over all the rounds. In the simulations, we set $\alpha_S = 1$, $\beta_S = 0.5$, and T_{min} to be 100 seconds for any selfish node S. The running time for each round is 5000 seconds. Fig. 4 shows the performance comparison among the three systems: ARCS, ARCS-NBC, and FULL-COOP, where in ARCS, B_{max} and LB_{max} are both set to be $60E$. From the selfish nodes' energy efficiency comparisons (Fig. 4(a)) we can see that ARCS has much higher efficiency than ARCS-NBC and FULL-COOP when there exist malicious nodes. When only selfish nodes exist, ARCS-NBC and FULL-COOP have the same efficiency, since they work in the same way, and both have slightly higher efficiency than ARCS with the payment of higher balance variation of selfish nodes, which is shown in Fig. 4(b). The balance variation comparison shows that ARCS has much lower balance variation than the other two systems, and almost keeps unchanged with the increase of malicious nodes number, while for the other two systems, the balance variation increases linearly and dramatically with the increase of malicious nodes number. This comparison also implies the lower unfairness for selfish nodes in the ARCS system. The average damage comparison (Fig. 4(c)) shows that in ARCS the damage that can be caused by malicious nodes is much lower than in other two systems, and increases very slowly with the increase of malicious nodes number.

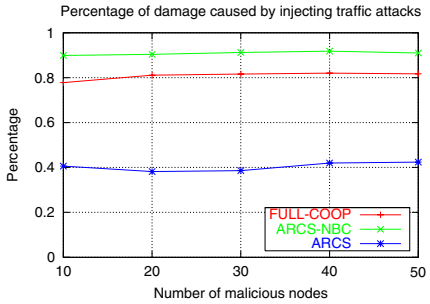


Fig. 5. Effects of injecting traffic attacks comparison among the three systems

From the results shown in Fig. 4(a), Fig. 4(b) and Fig. 4(c) we can also see that although ARCS-NBC has gained a lot of improvement over FULL-COOP by introducing mechanisms to detect dropping packet and emulating link breakage attacks, its performance is still much worse than ARCS. The reason is that ARCS-NBC cannot detect and punish those malicious nodes which launch injecting traffic attacks, so a large portion

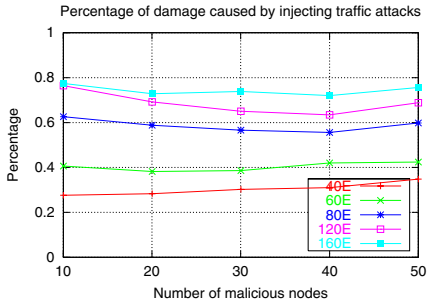
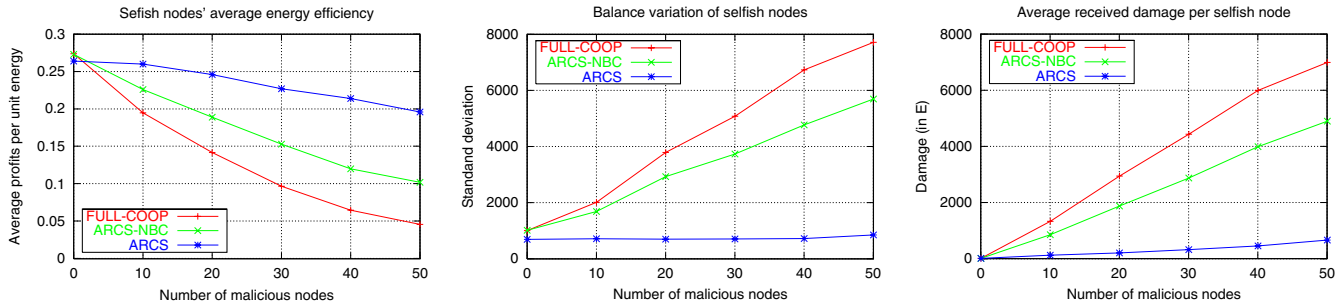
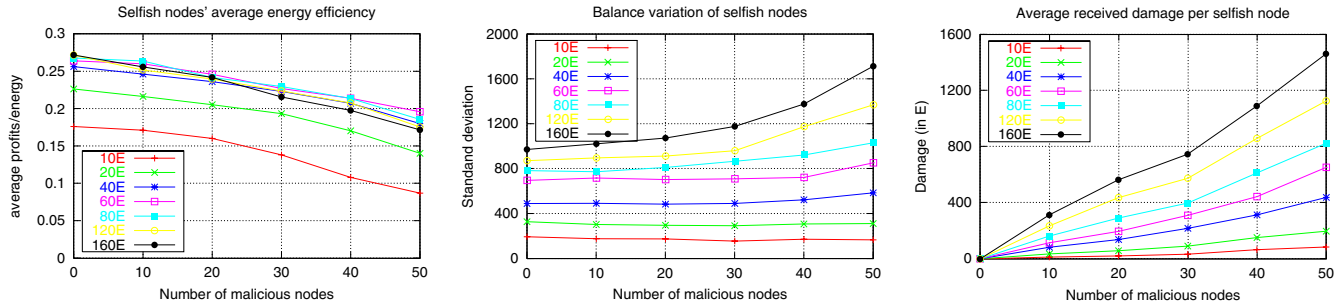


Fig. 7. Effects of injecting traffic attacks comparison of the ARCS system under different cooperation degrees



(a) Energy efficiency comparison (b) Balance variation comparison (c) Average damage comparison

Fig. 4. Performance comparison among the three systems



(a) Energy efficiency comparison (b) Balance variation comparison (c) Average damage comparison

Fig. 6. Performance comparison of the ARCS system under different cooperation degrees

of energy has been wasted to forward packets for those nodes. Fig. 5 illustrates the different effects of injecting traffic attacks in the three systems, where the y-axis shows the percentage of damage caused by injecting traffic attacks to the network. From these results we can see that in ARCS, only about 40% percentage of damage is caused by injecting traffic attacks, in FULL-COOP this percentage increases to around 80%, while in ARCS-NBC the percentage increases to more than 90%, although the overall damage caused by all malicious nodes to the selfish nodes in ARCS-NBC is less than that in FULL-COOP. These results explain why ARCS-NBC has much worse performance than ARCS, and clearly show that how necessary it is to introduce mechanisms to defend against such injecting traffic attacks.

Next we evaluate the ARCS system under different cooperation degree configurations, where all other parameters keep unchanged. Fig. 6 shows the performance of the ARCS system by varying cooperation degree from 10E to 160E. From Fig. 6(a) we can see that when the cooperation degree is 40E or more, the energy efficiency becomes almost identical. However, Fig. 6(b) and Fig. 6(c) show that with the increase of cooperation degree, both the balance variation of selfish nodes and the average received damage per selfish node increase. This can be explained using Fig. 7, which shows that with the increase of cooperation degree, the percentage of damage that is caused by injecting traffic attacks also increases. That is, the higher the cooperation degree, the more vulnerable to injecting traffic attacks. These results suggest that a relative small cooperation degree (for example 40E) is enough to achieve good performance for selfish nodes, such as high energy efficiency, low unfairness, and small damage.

VI. Conclusion

In this paper we have investigated the issues of cooperation stimulation and security in self-organized mobile ad hoc networks, and proposed an attack-resistant cooperation stimulation (ARCS) system to enforce cooperation among selfish nodes and to defend against various attacks, such as dropping packets, emulating link breakages, injecting traffic, and collusion. In the ARCS system, each node can adaptively adjust their own strategies according to the changing environments. The analysis has shown that in the ARCS system, the damage that can be caused by malicious nodes can be bounded, and the cooperation among selfish nodes can be enforced through introducing a positive cooperation degree. At the same time, the ARCS system maintains fairness among selfish nodes. The simulation studies have also agreed with the analysis. Another key property of the ARCS system is that it is fully distributive, and does not require any tamper-proof hardware or central management points.

VII. Acknowledgments

This work was supported in part by the Army Research Office under URI Award No. DAAD19-01-1-0494. The first

author would also like to thank Johannes Thorsteinsson for helpful discussion.

REFERENCES

[1] J.-P. Hubaux and T. Gross and J.-Y. Le Boudec and M. Vetterli, "Toward Self-Organized Mobile Ad Hoc Networks: The Terminodes Project," *IEEE Communications Magazine*, Jan. 2001.

[2] L. Buttyan and J. P. Hubaux, "Enforcing Service availability in mobile Ad-hoc Network," in *MobiHOC*, Aug. 2000.

[3] L. Buttyan and J.-P. Hubaux, "Stimulating Cooperation in Self-organizing Mobile Ad Hoc Networks," *Mobile Networks and Applications*, vol. 8, no. 5, pp. 579 – 592, Oct. 2003.

[4] S. Zhong, J. Chen, and Y. R. Yang, "Sprite: A Simple, Cheat-Proof, Credit-Based System for Mobile Ad-Hoc Networks," in *IEEE INFOCOM*, 2003.

[5] V. Srinivasan, P. Nuggehalli, C. F. Chiasserini, and R. R. Rao, "Cooperation in Wireless Ad Hoc Networks," in *IEEE INFOCOM*, 2003.

[6] S. Marti, T. J. Giuli, K. Lai, and M. Baker, "Mitigating Routing Misbehavior in Mobile Ad Hoc Networks," in *MobiCom*, Aug. 2000.

[7] P. Michiardi and R. Molva, "Core: a COllaborative REputation Mechanism to Enforce Node Cooperation in Mobile Ad Hoc Networks," in *IFIP - Communications and Multimedia Security Conference*, 2002.

[8] S. Buchegger and J.-Y. Le Boudec, "Performance Analysis of the CONFIDANT Protocol," in *MobiHOC*, June 2002.

[9] L. Zhou and Z. Haas, "Securing Ad Hoc Networks," *IEEE Network Magazine*, vol. 13, no. 6, pp. Nov./Dec., 1999.

[10] Y. Zhang and W. Lee, "Intrusion Detection in Wireless Ad-Hoc Networks," in *MobiCom*, Aug. 2000.

[11] J. P. Hubaux, L. Buttyan, and S. Capkun, "The Quest for Security in Mobile Ad Hoc Networks," in *MobiHOC*, May 2001.

[12] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Ariadne: A Secure On-Demand Routing Protocol for Ad Hoc Networks," in *MobiCom*, Sep. 2002.

[13] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Rushing Attacks and Defense in Wireless Ad Hoc Network Routing Protocols," in *WiSe*, San Diego, CA, USA, Sep. 2003.

[14] P. Papadimitratos and Z. Haas, "Secure Routing for Mobile Ad hoc Networks," in *SCS Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002)*, Jan 2002.

[15] K. Sanzgiri, B. Dahill, B. N. Levine, C. Shields, and E. M. Belding-Royer, "A Secure Routing Protocol for Ad Hoc Networks," in *Proceedings of the International Conference on Network Protocols (ICNP)*, Nov. 2002.

[16] M. G. Zapata and N. Asokan, "Securing Ad Hoc Routing Protocols," in *WiSe*, Sep. 2002.

[17] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Packet Leashes: A Defense against Wormhole Attacks in Wireless Networks," in *IEEE INFOCOM*, 2003.

[18] Y.-C. Hu, A. Perrig, and D. B. Johnson, "SEAD: Secure Efficient Distance Vector Routing for Mobile Wireless Ad Hoc Networks," *Ad Hoc Networks Journal*, vol. 1, pp. 175–192, 2003.

[19] R. Dawkins, *The Selfish Gene*, Oxford University Press, 2nd ed edition edition, 1990.

[20] D. B. Johnson and D. A. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks, Mobile Computing," In *Mobile Computing*, edited by Tomasz Imielinski and Hank Korth, chapter 5, pages 153-181, Kluwer Academic Publishers, 1996.

[21] IEEE Computer Society LAN MAN Standards Committee, "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE Std 802.11-1007," The Institue of Electrical and Electrics Engineers.