# Attack-Resistant Cooperation Stimulation in Autonomous Ad Hoc Networks

Wei Yu, *Student Member, IEEE,* and K. J. Ray Liu, *Fellow, IEEE*

***Abstract*—In autonomous ad hoc networks, nodes usually belong to different authorities and pursue different goals. In order to maximize their own performance, nodes in such networks tend to be selfish, and are not willing to forward packets for the benefits of other nodes. Meanwhile, some nodes might behave maliciously and try to disrupt the network and waste other nodes' resources. In this paper, we present an attack-resilient cooperation stimulation (ARCS) system for autonomous ad hoc networks to stimulate cooperation among selfish nodes and defend against malicious attacks. In the ARCS system, the damage that can be caused by malicious nodes can be bounded, the cooperation among selfish nodes can be enforced, and the fairness among nodes can also be achieved. Both theoretical analysis and simulation results 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.**

***Index Terms*—Autonomous ad hoc networks, cooperation stimulation, security.**

## I. Introduction and Background

An AD HOC NETWORK is a group of 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, nodes in an ad hoc network usually belong to the same authority and have a common goal. To maximize the overall system performance, nodes usually work in a fully cooperative way, and will unconditionally forward packets for each other. Recently, emerging applications of ad hoc networks are also envisioned in civilian usage [1]–[8]. In such applications, nodes typically do not belong to a single authority and may not pursue a common goal. Consequently, fully cooperative behaviors such as unconditionally forwarding packets for others cannot be directly assumed. On the contrary, in order to save limited resources, such as battery power, nodes may tend to be "selfish."

Before ad hoc networks can be successfully deployed in autonomous ways, the issues of *cooperation stimulation* and *security* must be resolved first. To stimulate cooperation among selfish nodes, one possible way is to use payment-based methods. In [2], a cooperation stimulation approach was proposed by using a virtual currency, called nuglets, as payments for packet forwarding, which was then improved in [3] using credit counters. However, tamper-proof hardware is required in each node to count the credits. In [4], Sprite was proposed to stimulate cooperation. It releases the requirement of tamper-proof hardware, but requires a centralized credit clearance service trusted by all nodes. Furthermore, these schemes consider only nodes' selfish behavior, while in many situations nodes can be malicious.

Another possible way to stimulate cooperation is to employ reputation-based schemes [6]–[8]. In [6], the first reputation-based system for ad hoc networks was proposed to mitigate nodes' misbehavior, where each node launches a "watchdog" to monitor its neighbors' packet forwarding activities and to make sure that these neighbors have forwarded the packets according to its requests. Following [6], CORE was proposed to enforce cooperation among selfish nodes [7], and CONFIDANT was proposed to detect and isolate misbehaving node and thus make it unattractive to deny cooperation [8]. However, these schemes suffer some problems. First, many attacks can cause a malicious behavior not being detected in these systems, and malicious nodes can easily propagate false information to frame up others. Second, these schemes can only isolate misbehaving nodes, but cannot actually punish them, and malicious nodes can still utilize the valuable network resources even after being suspected or detected.

Previous experiences have also shown that before ad hoc networks can be successfully deployed, security concerns must be addressed too [6], [9]–[13]. However, due to the sporadic nature of ad hoc networks, possible mobility and fragile wireless links, security in ad hoc networks is particularly hard to achieve [11]. For autonomous ad hoc networks, things are even worse: there is no centralized management point and nodes may tend to be selfish. Many schemes have been proposed in the literature 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]. These schemes cannot handle well the situation that the malicious nodes have entered the network, while in autonomous ad hoc networks the access control is usually loose, and malicious users can easily join the network.

In this paper, we consider the scenarios where there exist both selfish and malicious nodes in autonomous 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. Since no central management points are available, selfish nodes need to adaptively and autonomously adjust their strategies according to the environments. Accordingly, we propose an

Manuscript received October 1, 2004; revised April 1, 2005. This work was supported in part by the Army Research Office under URI Award DAAD19-01-1-0494.

The authors are with the Department of Electrical and Computer Engineering, Institute for Systems Research, University of Maryland, College Park, MD 20742 USA (e-mail: weiyu@isr.umd.edu; kjrliu@isr.umd.edu).

Digital Object Identifier 10.1109/JSAC.2005.857201

attack-resilient cooperation stimulation (ARCS) system for autonomous ad hoc networks which provides mechanisms to stimulate cooperation among selfish nodes in adversarial environments. Besides maintaining fairness among selfish nodes and being robust to various attacks, another key property of the ARCS system is that it does not require any tamper-proof hardware or central management point, which is very suitable for autonomous ad hoc networks. Both analysis and simulation confirm the effectiveness of the ARCS system.

The rest of this paper is organized as follows. Section II describes the system model and formulates the problem. 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. System Model and Problem Formulation

In this paper, we consider autonomous 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, 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 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 such networks 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 resources, and the objective of malicious nodes is to maximize the damage that they can cause to the network. Since energy is usually the most stringent and valuable resource for battery-supplied nodes in ad hoc networks, we restrict the resource constraint to be energy. However, the proposed schemes are 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. We focus on the situation that all nodes in the network are legitimate, no matter selfish or malicious. To prevent illegitimate nodes from entering the network, some existing schemes can be used as the first defense line, such as [9] and [12]–[14].

Next, we exploit the possible attacks that can be launched in such networks. We say a route $R = R_0R_1 \dots R_M$ is *valid at time* $t$ if for any $0 \leq i < M$, $R_i$ and $R_{i+1}$ are in each other's transmission range. We say a link $(R_i, R_{i+1})$ *is broken at time* $t$ if $R_i$ and $R_{i+1}$ are not in each other's transmission range. It is easy to see that at time $t$, a packet can be successfully delivered from its source $S$ to its destination $D$ through the route $R = R_0R_1 \dots R_M$ ($R_0 = S$ and $R_M = D$) within the delay constraint $\tau$ if and only if all of the following conditions are satisfied.

1) $R$ is a valid route at time $t$, and no links on route $R$ will break during the transmission.
2) No errors will be introduced to the packet during the transmission.
3) No nodes on route $R$ will drop the packet during the transmission.
4) The total transmission time is less than $\tau$.

In order to degrade the network performance, the attackers can either directly break the ongoing communications, or try to waste other nodes' valuable resources. In general, the possible attacks that can be used by attackers in ad hoc networks can be roughly categorized as follows.

A1) *Emulate link breakage*: When a node $R_i$ wants to transmit a packet to the next node $R_{i+1}$ on a certain route $R$, if $R_{i+1}$ is malicious, $R_{i+1}$ can simply keep silent to let $R_i$ believe that $R_{i+1}$ is out of $R_i$'s transmission range, which can dissatisfy the condition 1.

A2) *Drop/modify/delay packets*: Dropping a packet can dissatisfy the condition 3, modifying a packet can dissatisfy the condition 2, and delaying a packet can dissatisfy the condition 4.

A3) *Prevent good routes from being discovered*: Such attacks can either dissatisfy the condition 1, or increase attackers' chance of being on the discovered routes and then launching various attacks such as A1 and A2. Two examples are wormhole and rushing attacks [13], [14].

A4) *Inject traffic*: Malicious nodes can inject an overwhelming amount of packets to overload the network and consume other nodes's valuable energy. When other nodes forward these packets but cannot get payback from attackers, the consumed energy is wasted.

A5) *Collusion attack*: Attackers can work together in order to improve their attacking capability.

A6) *Slander attack*: Attackers can also try to say something bad about the others.

Before formulating the problem, we first introduce some notations to be used, as listed in Table I. We assume that all data packets have the same size, and the transmitting power is the same for all nodes. We use "packet delivery transaction" to denote sending a packet from its source to its destination. We say a transaction is "successful" if the packet has successfully reached its destination within its delay constraint; otherwise, the transaction is "unsuccessful."

For each node $S$, if it is selfish, its total profit $P_{rofit}(S)$ is defined as follows:

$$P_{rofit}(S) = \alpha_S N_{S,\text{succ}} - \beta_S N_{S,\text{fail}}. \tag{1}$$

Then, the objective of each selfish nodes $S$ can be formulated as follows:

$$\max P_{rofit}(S) \quad \text{s.t.} \quad E_S \leq E_{S,\max}. \tag{2}$$

If $S$ is malicious, then the total damage $D_S$ that $S$ has caused to other nodes until the current moment is calculated as

$$D_S = E_{S,\text{waste}} - E_{S,\text{contribute}}. \tag{3}$$

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:

$$\max \sum_{S \text{ is malicious}} D_S. \tag{4}$$

TABLE I
NOTATIONS USED IN THE PROBLEM FORMULATION

| | |
|---|---|
| $E$ | The amount of energy needed to transmit and receive a data packet and a receipt. |
| $E_{S,max}$ | $S$'s total available energy when it enters the network. |
| $\alpha_S$ | The payoff that $S$ can get for each successfully delivered data packet with $S$ being the source. |
| $\beta_S$ | The penalty that $S$ will receive for each unsuccessfully delivered data packet with $S$ being the source. |
| $E_S$ | The total energy that $S$ has spent until now. |
| $N_{S,succ}$ | The total number of successful data packet deliveries until now with $S$ being the source. |
| $N_{S,fail}$ | The total number of unsuccessful data packet deliveries until now with $S$ being the source. |
| $E_{S,waste}$ | The amount of selfish nodes' energy that has been wasted until now due to $S$'s its malicious behavior. |
| $E_{S,contribute}$ | The amount of $S$'s energy that it has spent until now on successfully transmitting packets for others. |

TABLE II
RECORDS KEPT BY NODE $S$

| | |
|---|---|
| $C_{redit}(A,S)$ | The energy that A has spent until now on successfully forwarding packets for S. |
| $D_{ebit}(A,S)$ | The energy that S has spent until now on successfully forwarding packets for A. |
| $W_{by}(A,S)$ | The wasted energy that A has caused to S until now. |
| $W_{to}(A,S)$ | The wasted energy that S has caused to A until now. |
| $LB_{with}(A,S)$ | The wasted energy caused to S until now due to the link breakages between A and S. |
| $B_{lacklist}(S)$ | The 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 now. |

We assume that each node has a public key and a private key. We also assume that a node can know or authenticate the other nodes' public keys, but no node will disclose its private key to the others unless it has been compromised. We do not assume that nodes trust each other. 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 *acknowledgment* mechanism is supported by the network link layer. That is, if node $A$ has transmitted a packet to node $B$ and $B$ has successfully received it, node $B$ needs to immediately notify $A$ of the reception through link level acknowledgment.

## III. DESCRIPTION OF ARCS SYSTEM

This section presents the proposed ARCS system for autonomous ad hoc networks. In the ARCS system, each node $S$ keeps a set of records indicating the interactions with other nodes, as listed in Table II. In a nutshell, when a node has a packet scheduled to be sent, it first checks whether this packet should be sent and which route should be used. When an intermediate node on the selected route receives a packet forwarding request, it will check whether it should forward the packet. Once a node has successfully forwarded a packet on behalf of another node, it will request a receipt from its next node on the route and submit this receipt to the source of the packet to claim credit. After a packet delivery transaction finishes, all participating nodes will update their own records to reflect the changed relationships with other nodes and to detect possible malicious behavior. For each selfish node $S$, all the records listed in Table II will be initiated to be 0 when $S$ first enters the network.

### A. Cooperation Degree

In [19], Dawkins illustrates that reciprocal altruism is beneficial for every ecological system when favors are granted simultaneously, and gives 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 divides 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 studies have shown that both cheats and suckers extinct finally, and only grudgers win over time. Such cooperation behaviors are also developed at length in [20] and [21].

In order to best utilize their limited resources, selfish nodes in autonomous ad hoc networks should also act like the 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

$$B(A,S) = (D_{ebit}(A,S) - W_{\text{to}}(A,S)) - (C_{redit}(A,S) - W_{\text{by}}(A,S)). \quad (5)$$

That is, $B(A,S)$ is the difference between what $S$ has contributed to $A$ and what $A$ has contributed to $S$, in $S$'s point of view. If $B(A,S)$ is a positive value, it can be viewed 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 $B_{\text{threshold}}(A,S)$ 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

$$B(A,S) < B_{\text{threshold}}(A,S). \quad (6)$$

Setting $B_{\text{threshold}}(A,S)$ to be $\infty$ means that $S$ will always help $A$ no matter what $A$ has done, as the *suckers* act in the example. Setting $B_{\text{threshold}}(A,S)$ to be $-\infty$ means that $S$ will never help $A$, as the *cheats* act in the example. In the ARCS system, each selfish node will set $B_{\text{threshold}}(A,S)$ 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 $S$ has exceeded $B_{\text{threshold}}(A,S)$, as the *grudgers* act in the example where they set the threshold 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 that can be caused by malicious nodes can be bounded.

### B. Route Selection

In the ARCS system, *source routing* is used, that is, when sending a packet, the source lists in packet header the complete sequence of nodes through which the packet is to traverse. Due to insufficient balance, malicious behavior and possible node mobility, not all packet delivery transactions can succeed. When a node has a packet scheduled to be sent, it needs to decide whether it should start the packet delivery transaction and which route should be used.

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 $R = R_0R_1,\ldots,R_M$ is a valid route known by $S$ with $R_0 = S$, $R_M = D$, and $M$ being the number of hops. Let $P_{\text{drop}}(R_i,S)$ denote the probability of node $R_i$ will dropping $S$'s packet, and let $P_{\text{delivery}}(R,S)$ denote the probability that a packet can be successfully delivered from $S$ to $D$ through route $R$ at the current moment. $S$ then calculates $P_{\text{delivery}}(R,S)$, as shown in (7) at the bottom of the page. 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 nonzero $P_{\text{delivery}}(R,S)$ is used to send a packet by $S$, the expected energy consumption can be calculated as:

$$E_{\text{avg}}(R,S) = EMP_{\text{delivery}}(R,S) + E_{\text{fail}}(R,S) \times \sum_{n=1}^{M-1} nE\left(\prod_{k=1}^{n-1}(1-P_{\text{drop}}(R_k,S))\right)P_{\text{drop}}(R_n,S) \quad (8)$$

and the expected profit of $S$ is

$$P_{rofit}(R,S)=\alpha_S P_{\text{delivery}}(R,S) - \beta_S\left(1-P_{\text{delivery}}(R,S)\right). \quad (9)$$

Let $Q(R,S)$ be the expected profit per unit energy when $S$ uses $R$ to send a packet to $D$ at the current moment, referred to as the expected *energy efficiency*. That is

$$Q(R,S) = \frac{P_{rofit}(R,S)}{E_{\text{avg}}(R,S)}. \quad (10)$$

Then, in the ARCS system, which route should be selected is decided as follows.

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

The above decision is optimal in the sense that no other known routes can provide better expected energy efficiency than route $R^*$. Since the accurate value of $P_{\text{drop}}(R_i,S)$ is usually not known, in the ARCS system, $P_{\text{drop}}(R_i,S)$ is estimated as the ratio between the number of $S$'s failed transactions caused by $R_i$ and $S$'s total transactions passing $R_i$.

After the route with the highest expected energy efficiency has been found by the sender $S$, suppose it is route $R^*$, in the next step $S$ should decide whether it should use $R^*$ to start a data packet delivery transaction. If the route quality is too low, simply dropping the packet without trying may be a better choice. Let $Q_{\text{avg}}(S)$ be $S$'s average energy efficiency over the past

$$Q_{\text{avg}}(S) = \frac{\alpha_S N_{S,\text{succ}} - \beta_S N_{S,\text{fail}}}{E_S}. \quad (11)$$

Then, in the ARCS system, the following decision rule is used.

**Packet delivery decision**: $S$ will use route $R^*$ to start a data packet delivery transaction if and only if the following condition holds:

$$P_{rofit}(R^*,S) \geq Q_{\text{avg}}(S)E_{\text{avg}}(R^*,S) - \beta_S. \quad (12)$$

The left-hand side of (12) is the expected profit when $S$ uses $R^*$ to start a packet delivery transaction, and the right-hand side of (12) is the predicted profit by simply dropping the packet without trying, where $\beta_S$ is the penalty due to dropping a packet and $Q_{\text{avg}}(S)E_{\text{avg}}(R^*,S)$ is the gain that $S$ predicts to get with energy $E_{\text{avg}}(R^*,S)$ based on its past performance. If $Q_{\text{avg}}(S)$ is stationary over time, the above decision is optimal in the sense that $S$'s total profit can be maximized under the energy constraint.

### C. Data Packet Delivery Protocol

In the ARCS system, a data packet delivery consists of two stages: *forwarding data packet stage* and *submitting receipts stage*. In the first stage, the data packet is delivered from its source to its destination, while in the second stage, each participating node on the route will submit a receipt to the source to claim credit. Table III lists some notations to be used.

*1) Forwarding Data Packet Stage:* Suppose that node $S$ is to send a packet with payload $M$ and sequence

$$P_{\text{delivery}}(R,S) = \begin{cases} 0, & (\exists R_i \in R)B(R_i,S) < -B_{\text{threshold}}(S,R_i) \\ 0, & (\exists R_i,R_j \in R)R_i \in B_{lacklist}(R_j,S) \\ \prod_{i=1}^{M-1}\left(1-P_{\text{drop}}(R_i,S)\right), & \text{otherwise} \end{cases} \quad (7)$$

TABLE III
NOTATIONS USED IN THE DATA PACKET DELIVERY PROTOCOLS

| | |
|---|---|
| $sign_S(m)$ | Node $S$ generates a signature based on the message $m$. |
| $verify_S(m, s)$ | Other nodes verify whether $s$ is the signature generated by node S based on the message $m$. |
| $v \leftarrow m$ | Assign the value of $m$ to the variable $v$. |
| $MD()$ | A message digest function, such as SHA-1 [24]. |
| $seq_S(S, D)$ | The sequence number of the current packet being processed with S being the source and D being the destination. |

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 it is the destination of the packet. If it is, after necessary verification, $A$ returns a receipt to its previous node on the route to confirm the successful delivery; otherwise, $A$ checks whether it should forward the packet. $A$ is willing to forward the packet if and only if all the following conditions are satisfied: 1) $A$ is on 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 the source and $D$ being the destination; 3) the signature is valid; 4) $B(S, A) < B_{\text{threshold}}(S, A)$; and 5) no node on route $R$ has been marked as malicious by $A$.

Once $A$ has successfully forwarded the packet $(m, R, seq_S(S, D), s)$ to the next node on route $R$, 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 $T_{\text{link}}$ multiplied by the number of hops following this node, where $T_{\text{link}}$ is a relatively small interval to account for the necessary processing and waiting time (e.g., time needed for channel contention) 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. The protocol execution of each participating selfish node in this stage is described in Protocol 1.

**Protocol 1** Forwarding data packet stage

▷ $A$ is the current node, $S$ is the sender, $D$ is the destination. $(m, R, seq_S(S, D), s)$ 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 waits for receipts to be returned.
**else if** $((A = D)$ and $(verify_S((m, R, seq_S(S, D)), s) = true)$ and $(seq_S(S, D) > seq_A(S, D)))$, **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 $(verify_S((m, R, seq_S(S, D)), s) \neq true)$ or $(seq_S(S, D) \leq seq_A(S, D))$ or $(\exists R_i \in R, R_i \in B_{lacklist}(A)))$ **then**
    $A$ simply drops this packet.
  **else if** $((B(S, A) > B_{\text{threshold}}(S, A))$ 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 $seq_A(S, D)$, forward $(m, R, seq(S, D), s)$ to its next node, and waits for a receipt to be returned by the next node.
  **end if**
**end if**

*2) Submitting Receipts Stage:* In autonomous 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 credit from $S$, which $A$ can use later to request $S$ to return the favor. To claim credit 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 submit a valid receipt generated by any node following $A$ on the route (e.g., $B$) indicating that $B$ has successfully received the packet. One possible format of such a receipt is

$$\{MD(m), R, seq_S(S, D), B, sign_B(MD(m), R, seq_S(S, D), B)\}.$$

That is, the receipt consists of the message $\{MD(m), R, seq_S(S, D), B\}$ and the signature generated by node $B$ based on this message. 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 invalid, 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 protocol execution of each participating selfish node in this stage is described in Protocol 2.

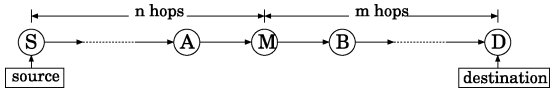



Fig. 1. Update records.

**Protocol 2** Submitting receipt stage
▷ $A$ is the current node, $(MD(m), R, seq_S(S,D), B, s)$ is the successfully received packet to be processed.
**if** ($(A = D)$ or (no valid receipts have been returned by 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 $R$'s previous node on $S$.
  **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 this transaction, $D$ is the destination, and $R = S \ldots AMB \ldots D$ is the associated route.

For the sender $S$ , according to the 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. Then, for each intermediate node $X$, $S$ updates $C_{redit}(X,S)$ as follows:

$$C_{redit}(X,S) = C_{redit}(X,S) + E. \tag{13}$$

- *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 updates its records as follows:

$$W_{\text{by}}(X,S) = W_{\text{by}}(X,S) + E \tag{14}$$

$$B_{lacklist}(S) = B_{lacklist}(S) \bigcup \{X\}. \tag{15}$$

  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 that 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 updates $C_{redit}(X,S)$ using (13). 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, $S$ updates $W_{\text{by}}(M,S)$ as follows:

$$W_{\text{by}}(M,S) = W_{\text{by}}(M,S) + nE. \tag{16}$$

  where $nE$ accounts for the amount of energy that has been wasted in this transaction with $n$ being the number of hops between $S$ and $M$.

If a 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.

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. In this case, $M$ only needs to update $D_{ebit}(S,M)$ as follows:

$$D_{ebit}(S,M) = D_{ebit}(S,M) + E. \tag{17}$$

- *Case 2*: $M$ has successfully sent the packet to node $B$, but cannot get a valid receipt from $B$. In this case, $M$ updates its records as follows:

$$\begin{aligned} W_{\text{to}}(S,M) &= W_{\text{to}}(S,M) + nE \\ W_{\text{by}}(B,M) &= W_{\text{by}}(B,M) + (n+1)E \\ B_{lacklist}(M) &= B_{lacklist}(M) \bigcup \{B\}. \end{aligned} \tag{18}$$

- *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 record this link breakage. In this case, $M$ updates its records as follows:

$$\begin{aligned} W_{\text{to}}(S,M) &= W_{\text{to}}(S,M) + nE \\ LB_{\text{with}}(B,M) &= LB_{\text{with}}(B,M) + nE \\ LB_{\text{with}}(S,M) &= LB_{\text{with}}(S,M) + nE. \end{aligned} \tag{19}$$

  In the ARCS system, each selfish node (e.g., $M$) will also set a threshold $LB_{\text{threshold}}(S,M)$ with any other node (e.g., $S$) to indicate the damage that $M$ can tolerate which is caused due to the link breakages happened between $M$ and $S$. In this case, if $LB_{\text{with}}(B,M)$ exceeds $LB_{\text{threshold}}(B,M)$, $B$ will be put into $M$'s blacklist. Similarly, if $LB_{\text{with}}(S,M)$ exceeds $LB_{\text{threshold}}(S,M)$, $S$ will be put into $M$'s blacklist.
- *Case 4*: $M$ has dropped the packet due to the reason that the condition in (6) is not satisfied or some nodes on $R$ are in $M$'s blacklist. In this case, $M$ does not need to update its records.

After finishing updating its records, $M$ will also keep a copy of the submitted receipt for possible future usage, such as resolving inconsistent records update problem, as will be described in Section III-F. From the above update procedure, we can see that a selfish node should 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, dynamic source routing (DSR) [22] is used as the underlying routing protocol to perform route discovery, which is an on-demand source routing protocol. However, without security consideration, the routing protocol itself can easily become an attacking target. For example, malicious nodes can inject an overwhelming amount of route request packets into the network. In the ARCS system, besides necessary identity authentication, the following security enhancements have also been incorporated into the route discovery protocol.

1) When node $S$ initiates a route discovery, it will also append its blacklist 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) $B(S, A) < B_{\text{threshold}}(S, A)$; 4) no nodes that have been appended to the request packet are in $A$'s blacklist; and 5) $A$ has not forwarded any request for $S$ in the last $T_{\text{interval}}(S, A)$ interval, where $T_{\text{interval}}(S, A)$ is the minimum interval specified by $A$ to indicate that $A$ will forward at most one route request for $S$ in each $T_{\text{interval}}(S, A)$ 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 $B_{\text{threshold}}(S, A)$ if not known by $S$, the value of $D_{ebit}(S, A)$, and node $A$'s expected staying time at the current position. 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 $B_{\text{threshold}}(S, A)$, determines the expiring time of this route which can be approximated as the expected minimum staying time among all nodes on the route, and checks the consistency between $D_{ebit}(S, A)$ and $C_{redit}(A, S)$.

### 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 $B_{\text{threshold}}(A, S)$, or $S$ may continue requesting $A$ to forward packets for it when the true value of $B(S, A)$ has exceeded $B_{\text{threshold}}(S, A)$. 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. We say a receipt received by $A$ at time $t_1$ and targeting on $S$ has been *confirmed* if there exists 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 $C_{redit}(X, S)$ using (13), also, if $C$ is not the destination of the associated packet, $S$ should update $W_{\text{by}}(C, S)$ using (16).

### G. Parameter Selection

In the ARCS system, for each selfish node $S$, it needs to specify three types of thresholds regarding to any other node $A$ in the network: the cooperation degree $B_{\text{threshold}}(A, S)$, the maximum tolerable damage due to link breakage $LB_{\text{threshold}}(A, S)$ and the minimum route request forwarding interval $T_{\text{interval}}(A, S)$, which are determined in the following way.

For each known node $A$, $S$ initially sets $T_{\text{interval}}(A, S)$ to be a moderate value, such as a value equal to its own average pause time. During staying in the network, $S$ will keep estimating a good route discovery frequency for itself, and will set $T_{\text{interval}}(A, S)$ to be the inverse of its own route discovery frequency. Similarly, $S$ initially sets all link breakage thresholds using a (relatively small) constant value $LB_{\text{init}}$, and keeps estimating its own average link breakage ratio over time, assuming $P_{S,LB}$. For each node $A$, let $N_{\text{trans}}(A, S)$ be the total number of transactions that simultaneously revolve $S$ and $A$ with $A$ either

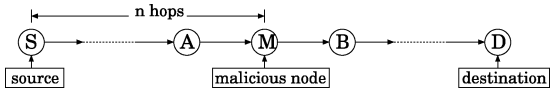



Fig. 2. Dropping packets attacks.

being $S$'s next node or being the initiator of the transactions, then $S$ may set

$$LB_{\text{threshold}}(A,S) = L_{\text{hop}} P_{S,LB} N_{\text{trans}}(A,S) E + LB_{\text{init}} \tag{20}$$

where $L_{\text{hop}}$ is the average number of hops per route.

For $B_{\text{threshold}}(A,S)$, if favors can be granted simultaneously, a small value (for example 1, as grudgers do in the ecological example) can work perfectly. However, in many situations favors cannot be granted immediately. For example, after $S$ has helped $A$ several times, $S$ may not get similar amount of help from $A$ due to that $S$ does not need help from $A$ currently or $A$ has moved. Many factors can affect the selection of $B_{\text{threshold}}(A,S)$, among them some are unknown to $S$, such as other nodes' traffic patterns and behaviors, and some are unpredictable, such as mobility, which make selecting an optimal value for $B_{\text{threshold}}(A,S)$ hard or impossible. However, our simulation studies in Section V have shown that in most situations a relatively small constant value can achieve good tradeoff between energy efficiency and robustness to attacks.

## IV. Analysis of the ARCS System Under Attack

In this section, we analyze the performance of the ARCS system under the following types of attacks: dropping packet, emulating link breakage, injecting traffic, collusion, and slander. Since the attacks of preventing good routes from being discovered are mainly used to increase attackers' chance of being on the discovered routes, they can be regarded as part of dropping packets or emulating link breakage attacks, and will not be analyzed separately. Similarly, modifying or delaying packets attacks can also be regarded as specific types of dropping packets attacks, and will not be analyzed separately. The results show that the damage that can be caused by malicious nodes is bounded, and the system is collusion-resistant.

*1) 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 study 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.

- 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 $W_{\text{by}}(M,S)$ 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 $S$ which 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 only 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 $B_{\text{threshold}}(M,A)$, the total damage that $M$ can cause is also bounded.

*2) 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 $LB_{\text{with}}(M,S)$ to remember the damage that has been caused due to link breakage between $M$ and $S$, and if $LB_{\text{with}}(M,S)$ exceeds the threshold $LB_{\text{threshold}}(M,S)$, $S$ 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 $LB_{\text{threshold}}(M,S)$.

*3) Injecting Traffic Attacks:* Besides dropping packets, attackers can also inject an excessive amount of traffic to overload the network and to 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 $T_{\text{interval}}$. For general data packets, since an intermediate node $A$ will stop forwarding packets for node $M$ if $B(M,A) > B_{\text{threshold}}(M,A)$, the maximum damage that can be caused to node $A$ by node $M$ launching injecting general data traffic attacks is bounded by $B_{\text{threshold}}(M,A)$. In

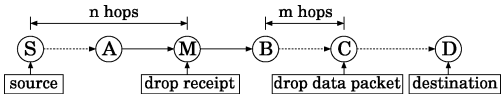



Fig. 3. Collusion attacks.

summary, by launching injecting traffic attacks, the maximum damage that can be caused by a malicious node $M$ to node $A$ is bounded.

*4) 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 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 $B$.

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 $W_{\text{by}}(M, S)$ 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, temporarily 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.

*5) Slander Attacks:* In ARCS, each node can propagate its blacklist to the network, which may give attackers chances to slander the others. Next, we show that instead of causing damage, such attacks can even benefit selfish nodes in some situations. Suppose that an attacker $M$ tells the others that node $X$ is malicious. For any selfish node $S$, this information will only be used when $S$ wants to calculate a route's successful packet delivery probability and both $X$ and $M$ are on this route. In this situation, the successful packet delivery probability of this route will be calculated as 0 according to (7), and this route will not be used by $S$, which is just one goal of secure route discovery: preventing attackers from being on the discovered route. In all other situations, such information will not affect $S$'s decision.

TABLE IV
SIMULATION PARAMETERS

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

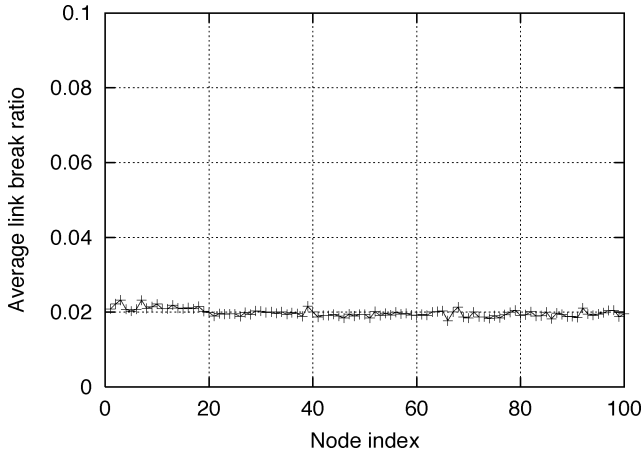



Fig. 4. Self-estimated link breakage ratio.

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

$$D_{amage} \leq \sum_{k=1}^{K} \sum_{l=1}^{L} \Big( B_{\text{threshold}}(M_k, S_l) + LB_{\text{threshold}}(M_k, S_l) + \frac{T_{S_l} * E_{\text{request}}}{T_{\text{interval}}(M_k, S_l)} \Big). \quad (21)$$

*Proof:* See the above analysis. ■

From Theorem 1, we can see that the damage that can be caused by malicious nodes is bounded, which is determined by the thresholds specified by each selfish node.

## V. SIMULATION STUDIES

### A. Simulation Configuration

In our simulations, nodes are randomly deployed inside a rectangular space. Each node moves randomly according to the *random waypoint* model [22]: a node starts at a random position, waits for a duration called the *pause time*, then randomly chooses a new location and moves toward the new location with

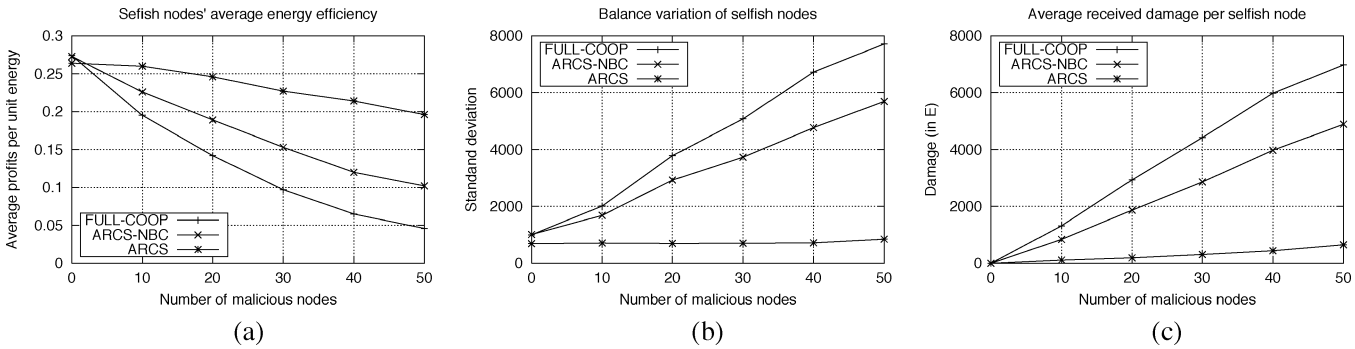



Fig. 5. Performance comparison between the three systems. (a) Energy efficiency comparison. (b) Balance variation comparison. (c) Average damage comparison.

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 medium access control (MAC) layer protocol simulates the IEEE 802.11 distributed coordination function (DCF) with a four-way handshaking mechanism [23]. Some simulation parameters are listed in Table IV.

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 or malicious node that does not launch injecting traffic attacks, the average packet interarrival time is 2 seconds, while for malicious nodes launching injecting traffic attacks, the average packet interarrival time is 0.1 second. In the simulations, all data packets have the same size.

Based on selfish nodes' forwarding decision, three 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 (6) 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.

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 section, the following performance metrics are used.

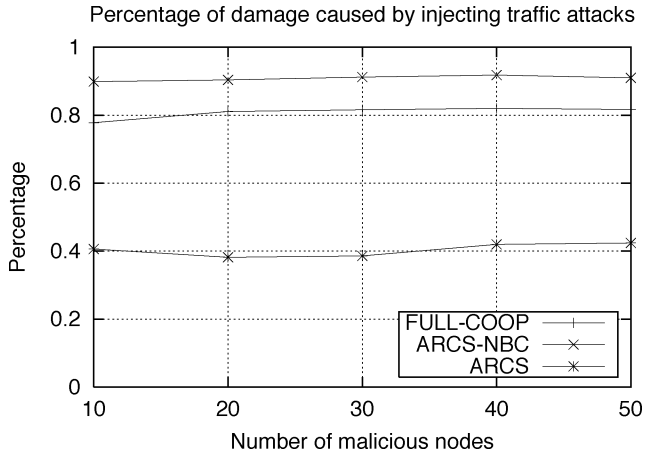



Fig. 6. Effects of injecting traffic attacks in the three systems.

- *Energy efficiency of selfish nodes*, which is the total profit gained by all selfish nodes divided by the total energy spent by all selfish nodes.
- *Average damage received per selfish node*, which is the total damage received by all selfish nodes divided by the total number of selfish nodes, that is

$$D_{\text{avg}} = \frac{1}{L}\sum_{i=1}^{L}\left(\sum_{l=1}^{L} B(S_l, S_i) + \sum_{k=1}^{K} B(M_k, S_i)\right). \quad (22)$$

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

$$V_{ariation} = \sqrt{\frac{1}{L}\sum_{l=1}^{L} B(S_l) * B(S_l)}. \quad (23)$$

By assuming $\sum_{l=1}^{L} B(S_l) = 0$, the effects caused by malicious nodes have incorporated, which will make $\sum_{l=1}^{L} B(S_l)$ deviate from 0. This metric 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.

### B. Simulation Results

In our simulations, each configuration has been run ten 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 $T_{\text{interval}}$ to be 100 seconds for each selfish node

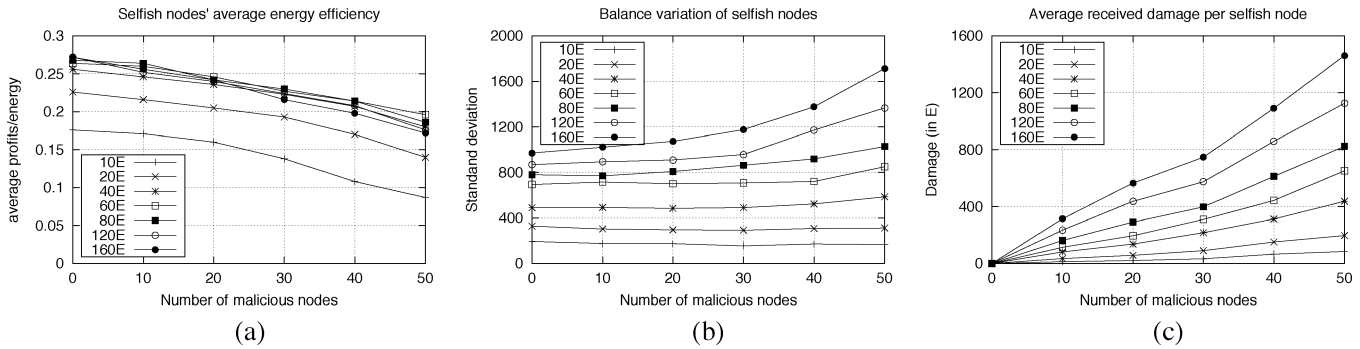



Fig. 7. Performance comparison of the ARCS system under different cooperation degrees. (a) Energy efficiency comparison. (b) Balance variation comparison. (c) Average damage comparison.

$S$, which is equal to the average pause time. The running time for each round is 5000 seconds. For each selfish node, the link breakage ratio is estimated through its own experience, which is the ratio between the total number of link breakages it has experienced with itself being the transmitter and the total number of transmissions it has tried. Fig. 4 shows the estimated values of link breakage ratio by each node, which shows that all nodes have almost the same link breakage ratio (here 2%).

Fig. 5 shows the performance comparison among the three systems: ARCS, ARCS-NBC, and FULL-COOP, where in ARCS, $B_{\text{threshold}}$ is set to be $60E$, and the value of $LB_{\text{threshold}}$ is set according to (20) with $LB_{\text{init}} = 20E$. The experiments based on other values of $B_{\text{threshold}}$ have also been conducted, which shows that $60E$ can achieve good tradeoff between performance and possible damage (demonstrated in Fig. 7). From the comparisons of selfish nodes' energy efficiency [Fig. 5(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. 5(b). The balance variation comparison shows that ARCS has much lower balance variation than the other two systems, and keeps almost unchanged with the increase of malicious nodes, while for the other two systems, the balance variation increases linearly and dramatically with the increase of malicious nodes. This comparison also implies the lower unfairness for selfish nodes in the ARCS system. The average damage comparison [Fig. 5(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.

From the results shown in Fig. 5, 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 of energy has been wasted to forward packets for those nodes. Fig. 6 illustrates the different effects of injecting traffic attacks in the three systems, where the vertical 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

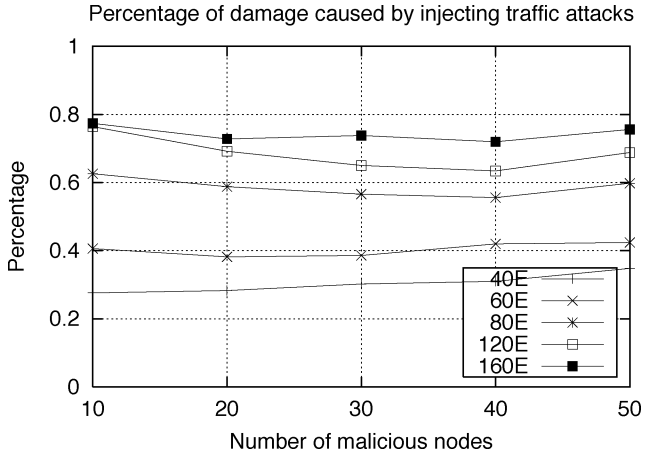



Fig. 8. Effects of injecting traffic attacks under different cooperation degrees in the ARCS system.

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. In other words, in Fig. 5(c), the gap between the results corresponding to "ARCS" and the results corresponding to "ARCS-NBC" is caused by injecting traffic attacks, while the gap between the results corresponding to "ARCS-NBC" and the results corresponding to "FULL-COOP" is caused by dropping packets/emulating link breakage attacks. 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 injecting traffic attacks.

Next, we evaluate the ARCS system under different cooperation degrees, where all other parameters keep unchanged. Fig. 7 shows the performance of the ARCS system by varying cooperation degree from 10E to 160E. From Fig. 7(a), we can see that when the cooperation degree is 40E or more, the energy efficiency becomes almost identical. However, Fig. 7(b) and (c) shows 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. 8, 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 autonomous ad hoc networks, and proposed an ARCS system to stimulate cooperation among selfish nodes and defend against various attacks launched by malicious nodes. In the ARCS system, each node can adaptively and autonomously 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 good fairness among selfish nodes. The simulation results have also agreed with the analysis. Another key property of the ARCS system is that it is fully distributive, completely self-organizing, and does not require any tamper-proof hardware or central management points.

## Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable comments and feedback. Meanwhile, W. Yu would like to thank J. Thorsteinsson for the helpful discussions.

## References

[1] J. P. Hubaux, T. Gross, J. Y. Le Boudec, and M. Vetterli, "Toward self-organized mobile ad hoc networks: The terminodes project," *IEEE Commun. Mag.*, vol. 39, no. 1, pp. 118–124, Jan. 2001.

[2] L. Buttyan and J. P. Hubaux, "Enforcing service availability in mobile ad-hoc network," in *Proc. 1st Annu. Workshop Mobile Ad Hoc Netw. Comput.*, Boston, MA, Aug. 2000, pp. 87–96.

[3] ——, "Simulating cooperation in self-organized mobile ad hoc WANs," *Mobile Netw. Applicat.*, 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 *Proc. IEEE INFOCOM*, San Francisco, CA, Mar. 2003, pp. 1987–1997.

[5] V. Srinivasan, P. Nuggehalli, C. F. Chiasserini, and R. R. Rao, "Cooperation in wireless ad hoc networks," in *Proc. IEEE INFOCOM*, San Francisco, CA, Mar. 2003, pp. 808–817.

[6] S. Marti, T. J. Giuli, K. Lai, and M. Baker, "Mitigating routing misbehavior in mobile ad hoc networks," in *Proc. 6th Annu. Int. Conf. Mobile Comput. Netw.*, Boston, MA, Aug. 2000, pp. 255–265.

[7] P. Michiardi and R. Molva, "Core: A collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks," in *Proc. IFIP—Commun. Multimedia Security Conf.*, Portoroz, Slovenia, Sep. 2002, pp. 107–121.

[8] S. Buchegger and J.-Y. Le Boudec, "Performance analysis of the CONFIDANT protocol: Cooperation of nodes fairness in dynamic ad-hoc networks," in *Proc. 3rd ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, Lausanne, Switzerland, Jun. 2002, pp. 226–236.

[9] L. Zhou and Z. Haas, "Securing ad hoc networks," *IEEE Netw. Mag.*, vol. 13, no. 6, pp. 24–30, Nov./Dec. 1999.

[10] Y. Zhang and W. Lee, "Intrusion detection in wireless ad-hoc networks," in *Proc. 6th Annu. Int. Conf. Mobile Computi. Netw.*, Boston, MA, Aug. 2000, pp. 275–283.

[11] J. P. Hubaux, L. Buttyan, and S. Capkun, "The quest for security in mobile ad hoc networks," in *Proc. ACM Symp. Mobile Ad Hoc Netw. Comput.*, Long Beach, CA, Oct. 2001, pp. 146–155.

[12] Y. C. Hu, A. Perrig, and D. B. Johnson, "Ariadne: A secure on-demand routing protocol for ad hoc networks," in *Proc. 8th Annu. Int. Conf. Mobile Comput. Netw.*, Atlanta, GA, Sep. 2002, pp. 12–23.

[13] ——, "Rushing attacks and defense in wireless ad hoc network routing protocols," in *Proc. ACM Workshop Wireless Security*, San Diego, CA, Sep. 2003, pp. 30–40.

[14] ——, "Packet leashes: A defense against wormhole attacks in wireless networks," in *Proc. IEEE INFOCOM*, San Francisco, CA, Mar. 2003, pp. 1976–1986.

[15] ——, "SEAD: Secure efficient distance vector routing for mobile wireless ad hoc networks," *Ad Hoc Netw. J.*, vol. 1, pp. 175–192, 2003.

[16] P. Papadimitratos and Z. Haas, "Secure routing for mobile ad hoc networks," in *Proc. SCS Commun. Netw. Distrib. Syst. Modeling Simulation Conf.*, San Antonio, TX, January 2002.

[17] K. Sanzgiri, B. Dahill, B. N. Levine, C. Shields, and E. M. Belding-Royer, "A secure routing protocol for ad hoc networks," in *Proc. 10th IEEE Int. Conf. Network Protocols*, Paris, France, Nov. 2002, pp. 78–87.

[18] M. G. Zapata and N. Asokan, "Securing ad hoc routing protocols," in *Proc. ACM Workshop Wireless Security (WiSe)*, Atlanta, GA, Sep. 2002, pp. 514–519.

[19] R. Dawkins, *The Selfish Gene*, 2nd ed. London, U.K.: Oxford Univ. Press, 1990.

[20] R. Axelrod, *The Evolution of Cooperation*. New York: Basic Books, 1984.

[21] ——, *The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration*. Princeton, NJ: Princeton Univ. Press, 1997.

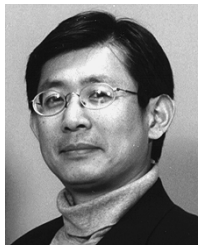
[22] D. B. Johnson and D. A. Maltz, "Dynamic source routing in ad hoc wireless networks, mobile computing," in *Mobile Computing*, T. Imielinski and H. Korth, Eds. Norwell, MA: Kluwer, 1996, ch. 5, pp. 153–181.

[23] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Standard 802.11-1007.

[24] *Federal Information Processing Standards Publication 180-1*, Secure Hash Standard, Apr. 1995.

**Wei Yu** (S'04) received the B.S. degree in computer science from the University of Science and Technology of China (USTC), Anhui, in 2000, and the M.S. degree in computer science from Washington University, St. Louis, MO, in 2002. Currently, he is working towards the Ph.D. degree in electrical and computer engineering at the University of Maryland, College Park.

From 2000 to 2002, he was a Graduate Research Assistant at the Washington University. From 2002 to 2005, he was a Graduate Research Assistant with the Communications and Signal Processing Laboratory and the Institute for Systems Research, University of Maryland. His research interests include security, wireless communications and networking, game theory, and wireless multimedia.



**K. J. Ray Liu** (F'03) received the B.S. degree in electrical engineering from the National Taiwan University, Taipei, Taiwan, R.O.C., in 1983, and the Ph.D. degree in electrical engineering from the University of California, Los Angeles (UCLA), in 1990.

He is a Professor and Director of Communications and Signal Processing Laboratories, Electrical and Computer Engineering Department, Institute for Systems Research, University of Maryland, College Park. His research contributions encompass broad aspects of wireless communications and networking, information forensics and security, multimedia communications and signal processing, signal processing algorithms and architectures, and bioinformatics, in which he has published over 350 refereed papers.

Dr. Liu is the recipient of numerous honors and awards including the IEEE Signal Processing Society 2004 Distinguished Lecturer, the 1994 National Science Foundation Young Investigator Award, the IEEE Signal Processing Society's 1993 Senior Award (Best Paper Award), the IEEE 50th Vehicular Technology Conference Best Paper Award, Amsterdam, 1999, and the EURASIP 2004 Meritorious Service Award. He received the 2005 Poole and Kent Company Senior Faculty Teaching Award from A. James Clark School of Engineering, University of Maryland, as well as the George Corcoran Award in 1994 for outstanding contributions to electrical engineering education, and the Outstanding Systems Engineering Faculty Award in 1996 in recognition of outstanding contributions in interdisciplinary research from the Institute for Systems Research. He is the Editor-in-Chief of the *IEEE Signal Processing Magazine*, the prime proposer and architect of the new IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, and was the founding Editor-in-Chief of the *EURASIP Journal on Applied Signal Processing*. He is on the Board of Governors of the IEEE Signal Processing Society.