2008

# Enhancing Node Cooperation in Mobile Wireless Ad Hoc Networks with Selfish Nodes

Yongwei Wang
*University of Kentucky*

ABSTRACT OF DISSERTATION

Yongwei Wang

The Graduate School
University of Kentucky
2008

Enhancing Node Cooperation in Mobile Wireless Ad Hoc Networks with Selfish Nodes

---

ABSTRACT OF DISSERTATION

---

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in the
College of Engineering
at the University of Kentucky

By

Yongwei Wang

Lexington, Kentucky

Director: Dr. Mukesh Singhal, Professor of Computer Science

Lexington, Kentucky

2008

ABSTRACT OF DISSERTATION

Enhancing Node Cooperation in Mobile Wireless Ad Hoc Networks with Selfish Nodes

In Mobile Ad Hoc Networks (MANETs), nodes depend on each other for routing and forwarding packets. However, to save power and other resources, nodes belonging to independent authorities may behave selfishly, and may not be willing to help other nodes. Such selfish behavior poses a real threat to the proper functioning of MANETs.

One way to foster node cooperation is to introduce punishment for selfish nodes. Based on neighbor-monitoring techniques, a fully distributed solution to detect, punish, and re-admit selfish nodes, is proposed here. This solution provides nodes the same opportunity to serve/and be served by others. A light-weight solution regarding battery status is also proposed here. This solution requires neighbor monitoring only when necessary, thereby saving nodes battery power.

Another effective way to solve the selfish-node problem is to reward nodes for their service according to their cost. To force nodes to show their true cost, truthful protocols are needed. A low overhead truthful routing protocol to find optimal routes is proposed in this thesis. The most prominent feature of this protocol is the reduction of overhead from existing solutions $O(n^3)$ to $O(n^2)$. A light-weight scalable truthful routing protocol (LSTOP) is further proposed, which finds near-least-cost paths in dense networks. LSTOP reduces overhead to $O(n)$ on average, and $O(n^2)$ in worst case scenarios.

Multiple path routing protocols are an effective alternative to single path routing protocols. A generic mechanism that can turn any table-driven multipath routing protocol

into a truthful one, is outlined here. A truthful multipath routing protocol (TMRP), based on well-known AOMDV protocol, is presented as an example. TMRP incurs an only $2n$ message overhead for a route discovery, and can also achieve load balancing without compromising truthfulness.

To cope with the selfish-node problem in the area of position-based routing, a truthful geographic forwarding (TGF) algorithm is presented. TGF utilizes three auction-based forwarding schemes to stimulate node cooperation. The truthfulness of these schemes is proven, and their performance is evaluated through statistical analysis and simulation studies.

KEYWORDS: Ad Hoc Networks, Selfish Nodes, Routing, Neighbor monitoring, Truthfulness.

Yongwei Wang

March 23, 2008

Enhancing Node Cooperation in Mobile Wireless Ad Hoc Networks with Selfish Nodes

By

Yongwei Wang

Dr. Mukesh Singhal

---

Director of Dissertation

Dr. Raphael A Finkel

---

Director of Graduate Studies

March 25, 2008

---

RULES FOR THE USE OF DISSERTATIONS

Unpublished dissertations submitted for the Doctor's degree and deposited in the University of Kentucky Library are as a rule open for inspection, but are to be used only with due regard to the rights of the authors. Bibliographical references may be noted, but quotations or summaries of parts may be published only with the permission of the author, and with the usual scholarly acknowledgements.

Extensive copying or publication of the dissertation in whole or in part also requires the consent of the Dean of the Graduate School of the University of Kentucky.

A library which borrows this dissertation for use by its patrons is expected to secure the signature of each user.

| Name | Date |
| --- | --- |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

DISSERTATION

Yongwei Wang

The Graduate School
University of Kentucky
2008

Enhancing Node Cooperation in Mobile Wireless Ad Hoc Networks with Selfish Nodes

---

DISSERTATION

---

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in the
College of Engineering
at the University of Kentucky

By

Yongwei Wang

Lexington, Kentucky

Director: Dr. Mukesh Singhal, Professor of Computer Science

Lexington, Kentucky

2008

Dedicated to my son and my wife

ACKNOWLEDGEMENTS

First of all, I would like to sincerely thank my advisor, Dr. Mukesh Singhal, for his indispensable guidance and assistance during the whole process of my research and dissertation writing.

I am also grateful to my other committee members, Dr. D. Manivannan, Dr. Jerzy W Jaromczyk, Dr. Caicheng Lu and Dr. Kevin D. Donohue for their invaluable suggestions and support.

I would also like to thank all my colleagues in the Hardyman Building for their kind help and discussions.

Finally, I thank my wonderful wife for her full support.

Table of Contents

List of Tables

List of Figures

**List of Files**

1. Dissertation_YW.pdf 978 kilobytes

# Chapter 1

# Introduction

This dissertation addresses the node cooperation problem in wireless ad hoc networks with selfish nodes. A fair distributed solution to detect and punish selfish nodes, together with a light-weight solution considering battery status is first presented. Then, two routing protocols, a low overhead truthful routing protocol and a light-weight scalable truthful routing protocol to improve efficiency of truthful routing, are presented. Next, the paper discusses a generic mechanism to turn any table-driven multipath routing protocol into a truthful one. Finally, a truthful mechanism for position-based routing and forwarding is presented in this thesis.

## 1.1 Ad Hoc Networks

With the rapid development of electronics and mobile computing devices, wireless data communication is becoming more and more pervasive. Wireless networks are developed by connecting these wireless devices, and a wireless network can range in size from as small as a BAN (Body Area Network), which covers the transmission range of the human body, to as large as wireless WANs (Wide Area Networks)[1]. In addition to providing the platform for mobile computing, a wireless network provides connection to various resources such as Internet services. The emergence of wireless networks satisfies the need for ubiquitous computing by accessing information at any time and any place. For example, at Starbucks, we can always see customers surfing the Internet while enjoying coffee. Among the various wireless networks, some have the support of infrastructures. For example, cellular networks are equipped with base stations, and wireless LANs are equipped with access points. Other wireless networks are not facilitated by an infrastructure support system. These networks are established for specific purposes, or in an 'ad hoc' manner, and thus are called *ad hoc networks*.

A mobile ad hoc network is an autonomous system, whereby a set of nodes (entities), which could be laptops or PDAs, are connected through wireless links. The system

allows the seamless interconnection of entities in areas without preexisting infrastructures. Without an infrastructure, ad hoc networks can be deployed rapidly, and were initially used for scenarios where establishing an infrastructure was not feasible or economical, such as disaster relief operations, emergency operations, and in the battlefield. It has now been extended to civilian and commercial applications. A prototypical example is mobile conferencing, where group of mobile users can establish an ad hoc network for collaborative computing, using their laptops. Ad hoc networks can also be used as edge networks to extend the coverage of cellular networks [2]. The potential applications of ad hoc networks include home networking, Personal Area Networks, embedded computing applications, sensor networks, and automotive interaction [1].

In general, mobile ad hoc networks share the following properties:

- Lack of infrastructure. In cellular networks, there are base stations and other centralized servers such as mobile switch centers. However, in ad hoc networks, there are no centralized entities in charge of network activities. Therefore, networking functions have to be executed by nodes. For example, each node has to work as a router for routing and data forwarding.

- Topology changes. In mobile ad hoc networks, mobility is inevitable. Nodes may move together as a group in the same direction, or to the same destination, as soldiers move on the battlefield. In most scenarios, nodes are more likely to move independently. Nodes may also dynamically join, or leave, the network at any time. Due to the mobility of nodes and their dynamic presence, the topology of the network link may change rapidly and unpredictably.

- Scarcity of resource. Nodes in mobile ad hoc networks usually have limited resources, such as battery power and bandwidth. While nodes in wired networks never take power into consideration, battery power is a primary concern in mobile ad hoc networks. Usually, the batteries in laptops last for only a couple of hours under normal working conditions. Moreover, packet transmission is very energy consumptive, and accelerates the depletion of battery power.

- Poor channel quality. Communication over wireless channels suffers from propagation path loss, fading, and interference from other ongoing communications. Different terrain contours and environments (city or rural, open space or inside office buildings), have varying impact on path loss. Even the weather, i.e., sunny or rainy, has an effect. The distance between two nodes is an important factor in channel quality, as received

signal strength deteriorates exponentially with increased distance. Nodes within radio range may interfere with each other. A more subtle problem is hidden nodes [3, 4]. In one scenario, a node is sending packets to a second node. However, a third node, out of radio range of the sending node but within the transmission range of the receiving node, may simultaneously send out packets, thus destroying ongoing communications. Mobile nodes also suffer from frequent link breaks.

- Multi-hop forwarding. Nodes have limited radio ranges to minimize required transmission power. Usually the expanse of an ad hoc network is much larger than a node's transmission coverage. Therefore, multi-hop packet forwarding is necessary for communication over the entire network range. In cellular networks or wireless LANs, the base station and the access point can cover the entire network.

- Device heterogeneity. Mobile devices in ad hoc networks can exist in many forms, such as laptops, PDAs, and cellular phones. These devices differ greatly in features such as size, memory, power consumption, and processing ability [5].

### 1.2 Motivations

Routing and data forwarding are two basic functions of networks. In wired networks, routers are in charge of routing and data forwarding, while in cellular networks base stations have this responsibility. However, in mobile ad hoc networks, in the absence of dedicated routers or base stations, nodes have to work as both routers and end systems. On one hand, these nodes usually have limited radio power and can cover a very limited area. On the other hand, the range of an ad hoc network can be quite large. Data communication therefore usually involves multi-hops between nodes. In such a system, without the forwarding help of nodes, communication beyond a node's radio range is impossible. Most routing protocols assume cooperative network settings, whereby nodes are willing to help each other by forwarding packets without bias. The primary challenge of routing in such an environment is to cope with node mobility, which, as has been noted, leads to frequently changing network topology. Many routing protocols [6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22] have been proposed for ad hoc networks. The key idea behind all of these protocols is to establish and maintain loop-free paths to respective destinations, while achieving a good balance with important performance metrics.

However, in some ad hoc networks, (and particularly civilian ones), such cooperation cannot be assumed. In these ad hoc networks, nodes belong to different authorities and have

their own priorities. In ad hoc networks, nodes have very limited resources, particularly battery power. On the other hand, the transmission of packets is very power-consuming. To expend this scarce resource on helping other nodes is not in a node's best interests. Thus a *selfish node* is not willing to forward data to other nodes freely, although it may expect service from other nodes. A selfish node is distinct from a malicious node, which intends to destroy the network or harm other nodes. When a minority of nodes engages in selfish behavior, network performance will be degraded, but when the majority of nodes exhibit selfish behavior, the collapse of the network may result. Thus selfish nodes pose a real threat to the correct functioning of ad hoc networks.

### 1.2.1 Punishment Based Solutions

One way to solve the selfish nodes problem is to introduce a punishment mechanism to force node cooperation. Nodes have the responsibility to help each other, i.e., they have to forward packets upon request. Uncooperative nodes can be punished by being excluded from the network. The key issue for such a mechanism is the design of an effective monitoring and evaluation system to evaluate node forwarding performance, and to detect selfish behaviors.

A good punishment based solution should have the following features:

- In the absence of centralized servers in ad hoc networks, the solution (both detection and evaluation tasks) should be distributed among nodes. Due to the properties of wireless communication (such as packet collision and interference), a node can not make accurate observations regarding other nodes. Thus, a single node's opinion/decision regarding conviction of another node cannot be relied upon.

- A selfish node should be identified and punished. Simply finding another route that bypasses a suspicious selfish node is not enough to force cooperation, since it encourages the selfish node not to forward packets.

- False conviction should be avoided, i.e., a cooperative node should not be convicted of being selfish, and be punished. Avoiding false punishment is more important than identifying all selfish nodes.

- A resource-depleting solution is not feasible. If excessive battery power and bandwidth consumption are needed to identify selfish nodes, it may be enough to just identify suspicious nodes. it is important to remember that a selfish node is not a *malicious node*, as it just wants to save its resource, not destroy or attack others.

A node's selfishness is evaluated based on its forwarding behavior. However, nodes in different positions in the network (particularly when the network is in a two dimensional area) have different opportunities to serve, or be served by the network. Some nodes in certain positions have obvious privilege over other nodes. This unfair utility of network is known as the *location privilege* problem. Generally, routing algorithms select short routes (ideally the shortest routes), and therefore routes are very likely to pass through the middle of the network. Nodes located in the middle of the network have more opportunities to be included in routes than those in the periphery. Nodes in the center area have to forward more packets and therefore spend more resources than those in the periphery. In other words, nodes in the periphery benefit from saving their resources. The evaluation of selfish behavior should consider this problem.

### 1.2.2 Motivation Based Solutions

The selfish nodes problem can be countered by inducing these nodes to cooperate by motivating them with incentives. This is diametrically opposed to the punishment approach. All nodes are assumed to be selfish in nature and they have the choice to forward packets, or not. But, why consume valuable power resources when nothing is received in return? By being reimbursed for their forwarding cost, and even gaining bonuses, selfish nodes can benefit from packet forwarding, and will be more willing to cooperate with other nodes. Reimbursement is usually in the form of virtual money, which can be converted into real money.

A natural question is how much a node should be paid. A simple way is to reimburse each node equally for packet forwarding. However, expenditures vary among nodes for forwarding the same packet, since they may use different emitting power and have different costs for emitting power units. Also the cost of a node may vary over time due to its battery status. For example, a node with low battery status is more likely to save the battery for its own use than one with full battery status. Thus, it is desirable to reimburse nodes according to their particular situation at any point in time. Meanwhile nodes getting forwarding services can pay for these services with a reasonably low reimbursement. However, to maximize their utility, selfish nodes may not divulge their true cost. This poses the requirement for truthful protocols (strategies) to prevent cheating.

A protocol is *truthful* (or strategy-proof) if it maximizes the utility of nodes only when they reveal their true cost. Truthful protocols are very useful in practice. For example, if ad hoc networks are used as edge networks to extend coverage of access points, or cellular

networks, truthful protocols provide the means for optimal payment to the forwarding nodes.

In a truthful protocol, selfish nodes have no incentive to lie about their cost. Note that the cost here refers to that of data packet transmission only. However, any routing protocol produces control messages[1] for route discovery and route maintenance. Since the cost of overhead in terms of control messages incurred by such truthful protocols must also be reimbursed, it is not desirable to reduce cost for data transmissions at the expense of high overhead, in terms of control messages. If the overall overhead incurred exceeds the savings for data packet transmission, there is no advantage to the truthful protocol. The following example illustrates this point. There are 5 intermediate nodes between a given source and a destination. Each intermediate node asks 5 cents for one packet forwarding although its true cost (to forward one packet) is only 1 cent. If the source has 1000 packets to send, it should pay 250 dollars for the service. With a truthful protocol, every node will show its true cost, which is 1 cent. However, the protocol incurs an overhead of 500 dollars. Of course, as a rational node, the source just wants to pay what an intermediate node asks. Thus controlling the overhead is as important as guaranteeing the truthfulness for any practical protocol. Moreover, a lower overhead usually leads to higher network performance.

Finding a truthful routing protocol for geographic routing is also a challenge. Topology-based routing algorithms, such as AODV or DSR, usually invoke global route discoveries to find routes from sources to destinations. Data packets will travel along the found paths. However, typical geographic routing algorithms[23, 24, 25, 26, 27, 28, 29] do not have such a global route discovery. Instead, the routing decision is based on local topology information, particularly 1-hop or 2-hop neighbor information. Without global information, some truthful methods used in topology-based routing can not be applied directly to geographic routing.

### 1.3 Contributions of the Dissertation

In this dissertation, we address the selfish nodes problem. We explore various mechanisms to solve this problem and have developed two detection-based solutions, two low-overhead truthful routing protocols, one multiple path truthful routing mechanism, and one

[1] In this dissertation, we use packet and message interchangeably, meaning formatted blocks of data transmitted over the network.

truthful greedy forwarding algorithm for position-based routing. Specifically, the following contributions are made:

- A fair distributed solution [30] to force node cooperation is presented. It judges, punishes, and readmits selfish nodes. Unlike previous solutions, the focus is on fairness, i.e., to provide the same chance for all nodes to gain services from the network and to offer services to the network. Also considered is *location privilege* which has not been emphasized in any of the previous solutions. Simulation results show that the proposed scheme improves data forwarding capability substantially in the presence of selfish nodes.

- Also presented here is a light-weight solution considering battery status [31]. The solution is light-weight in that neighbor monitoring is on-demand, and nodes work in promiscuous mode only part-time to save power. In addition, the solution is fair, in that battery status is considered. Compared with the first solution, this solution can more accurately judge selfish nodes as well as deliver more packets.

- A Low Overhead Truthful routing protocol (LOTTO) [32] reduces the overhead from $O(n^3)$ [33] to $O(n^2)$.

  - An algorithm is developed that collects topology information using a much lower overhead (from $O(n^3)$ to $O(n^2)$) compared to that of ad hoc-VCG [33]. Thus LOTTO provides much higher network performance than ad hoc-VCG while achieving truthfulness and least-cost routing.
  - Due to the reduced overhead of the protocol, the net cost for truthful routing is substantially reduced.

- A Light-weight Scalable Truthful routing Protocol (LSTOP) [34, 35] is also presented, which further reduces overhead to $O(n)$ on average, and $O(n^2)$ in the worst case.

  - A scalable method is shown to find near-optimal routes (in simulation) with a low overhead of $O(n)$ on average.
  - Due to the low overhead, the protocol is scalable and provides high network performance. Simulation results show that LSTOP generates far lower (30 to 60 times less) overhead, incurs far lower (2 magnitude of order) end-to-end delay, delivers significantly more packets, and has an even better overpayment ratio. Moreover, the net cost for truthful routing is significantly reduced.

  - An easy and accurate way to calculate payment is shown for nodes involved in route discoveries.

- GTMR [36], a generic mechanism to transform any table-driven multipath routing protocol into a truthful one, is presented, and shown for guarantee of truthfulness. In addition, there is TMRP - a truthful multipath routing protocol based on an AOMDV protocol as an instance of GTMR. A prominent feature of TMRP is that it incurs only $2n$ control packets for a route discovery and doesn't require new types of control messages over AOMDV. To the best of our knowledge, this is the lowest overhead incurred for any truthful routing protocol. TMRP can also achieve load balancing without compromising truthfulness.

- TGF [37], a truthful geographic forwarding algorithm for position-based routing is also discussed. Three auction schemes are introduced for forwarding packets to the next node. We show the truthfulness and effectiveness of TGF. To the best of our knowledge, TGF is the first truthful geographic forwarding algorithm for ad-hoc networks.

- Extensive simulation studies have also been conducted to evaluate the network performance of all solutions and protocols. Specifically, LOTTO and LSTOP are compared with the ad hoc-VCG. To the best of our knowledge, we are the first to conduct an extensive simulation study to evaluate important network metrics, such as packet delivery ratio, overhead, and end-to-end delay for protocols, using mechanism design for selfish nodes problems. Simulation results show that the protocols greatly outperform the ad hoc-VCG in all metrics.

### 1.4 Organization of the Dissertation

The reminder of the dissertation is arranged as follows. Chapter 2 reviews existing studies in this field. Chapter 3 presents two distributed detection-based solutions: a distributed solution which provides fairness and mitigates location privilege, and a light-weight solution which takes battery status into account. Chapter 4 presents two truthful routing protocols LOTTO and LSTOP, which incur $O(n^2)$ and $O(n)$ overhead on average, respectively. Chapter 5 presents a generic mechanism to turn any table-driven multiple routing protocol into a truthful one, and gives an instance of such a mechanism. Chapter 6 focuses on position-based routing and presents a truthful greedy-forwarding algorithm. Chapter 7 concludes the dissertation and discuss possible future studies.

# Chapter 2

# Related Work

The problem of nodes cooperation in MANETs has received a lot of research interest recently. Numerous solutions [33, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 30] have appeared in the literature, and each solution has a niche of applicability. This chapter gives a brief overview of these works.

Based on different philosophies of treating selfishness, solutions can be broadly classified as *detection-based* approaches and *motivation-based* approach. Detection-based approaches assume that nodes have the responsibility to forward packets for others, i.e., selfishness is not allowed. Nodes are monitored for their performance and necessary actions are taken once selfish behavior is detected. Motivation-based approaches, on the other hand, assume that nodes are selfish in nature and try to motivate them to cooperate by using some form of incentive such as virtual money (given to nodes upon serving the network). In such protocols, packet forwarding services are not free. Nodes having incentives/credit can use it to gain services from the network. When there are no accumulated incentives/credit, the nodes cannot take any services from the network. However, they can earn credit by serving other nodes.

## 2.1 Detection-based Approach

Marti et al. [50] were the first to introduce the selfish-node problem, as a type of misbehaving nodes. While not focusing on this problem alone, they provided a generic solution for handling misbehaving nodes. They introduced two tools, watchdog and pathrater, into DSR extensions to mitigate routing misbehavior (including selfish nodes) in ad hoc networks. Watchdog is based on neighbor monitoring and is used to identify malicious and selfish nodes. Pathrater evaluates the overall reputation of nodes on a path. It is used to select routes which bypass misbehaving nodes, including malicious nodes and selfish nodes. In their solution, selfish nodes are not penalized. Instead, they can still use the network while not forwarding packets.

Buchegger et al. [48, 49] proposed the *CONFIDANT* protocol to monitor the behavior of nodes, evaluate the reputation of corresponding nodes, and punish selfish nodes. CONFIDANT consists of four parts: a monitor, a reputation system, a trust manager and a path manager. The Monitor records the behavior of neighboring nodes. The reputation system evaluates the reputation of nodes based on direct observation and friends' observation. The trust manager collects indirect warning messages from friends, and the path manager is used to manage routing with selfish nodes. In CONFIDANT, each node monitors its next-hop neighbor's behavior. Detected misbehavior is reported to the reputation system. If the misbehavior is significant and intolerable, the information is relayed to the path manager, which will delete the related node from its path. Also, a warning message will be sent from the trust manager to friends. Upon receiving the warning message, the trust manager of a receiver evaluates the trustworthiness of the message and passes it on to its reputation system if necessary. CONFIDANT suffers from an inconsistent evaluation problem, i.e., different nodes may have different evaluations for the same node and therefore, some may regard a node as selfish whereas others do not. It also suffers from a location privilege problem since it punishes nodes if they do not forward packets regardless of how they contributed to the network before. To avoid punishment, nodes situated in the center of network have to keep forwarding packets and thus have to spend much more battery power than those on the periphery of the network.

Michiardi et al. [46] suggested using reputation to measure a node's contribution to a network. Reputation is actually a combination of three kinds of reputations: subjective, indirect and functional. Subjective reputation is calculated based on a node's direct observation. Indirect reputation is calculated based on the information provided by others. Functional reputation refers to subjective and indirect reputation with respect to different functions. Only routing function and packet forwarding function are under consideration. All three reputations will be aggregated into a collaborative reputation. Michiardi et al. proposed the *CORE* [47] protocol to evaluate nodes according to the above collaborative reputation. In CORE, each node maintains a set of Reputation Tables (RT) and a watchdog mechanism (WD). WD is used to verify whether a required function is correctly executed by the requested node by comparing the observed execution of function with the expected result. RT is used to maintain the reputation value of other nodes. Reputation is created and updated along time, based on direct observation by the node itself, or indirect information provided by others. With reputation, a node can judge the selfishness of a service requester and thus decide to refuse the request or provide the service.

Miranda et al. [51] proposed that a node periodically broadcast messages stating its view of its neighboring nodes. Also, nodes are allowed to publicly declare their refusal to forward messages to certain nodes. This mechanism causes heavy communication overhead. Paul and Westhoff [52] introduced security extensions to the DSR protocol to detect attacks to the routing process. The scheme relies on neighbors monitoring the routing message's context in order to find the attacker.

## 2.2 Motivation-based Approach

### 2.2.1 Traditional Methods

Traditionally, all nodes participating in data forwarding in traffic get the same payment. The cost discrepancy (for packet forwarding) between nodes is not taken into consideration.

The use of virtual money (termed a nuglet or credit) to simulate nodes' cooperation has been suggested [42, 53]. A node earns money by providing a forwarding service to others, and has to pay to get service from other nodes. Two payment models, the Packet Purse Model and Packet Trade Model, have been proposed. In the Packet Purse Model, the sender attaches nuglets to each packet. Each intermediate node gets some nuglets from the packet upon forwarding it. In the Packet Trade Model, each intermediate node buys a packet from its previous node and sells it to its next node. The destination node finally pays the cost. Security modules independent of nodes are used to protect the nuglets or credit value from modification and other attacks. Such security modules include some tamper-resistant hardware which stores certificates of its public key and others from some/all manufacturers.

Fratkin et al. [45] proposed a software solution to avoid the tamper resistant module in their APE (Ad hoc Participation Economy) system. A trusted third party, termed the banker node, is used to assure payment consolidation and integrity.

Ben Salem et al. [44] proposed a charging and rewarding scheme in multi-hop cellular networks. With the help of trustable base stations, the scheme combines symmetric cryptography with nuglets so that it can prevent some attacks such as refusal to pay and dishonest rewards. Crowcroft et al.[54] also proposed a pricing model where nodes update their cost based on bandwidth and power usage.

### 2.2.2 Game Theory Methods

From the point of view of game theory [55, 56, 57], the node-cooperation problem in wireless ad hoc networks falls into the framework of the typical non-cooperative game. Here, game players are all selfish nodes. They are rational in that they make decisions or adopt strategies

which can maximize their own benefit. Several protocols [33, 58, 59, 60, 41, 39, 38, 40, 61] have addressed the selfish-node problem from a game theory perspective.

#### 2.2.2.1 General Methods

Zhong et al. [58] proposed Sprite to motivate nodes to report their actions honestly. A central server, the Credit Clearance Service is placed in each network. The introduction of CCS relieves the necessity of tamper-proof hardware. In Sprite, every node reports a receipt, the digest of received or forwarded packets, to CCS whenever it has a connection to CCS. CCS then determines the charge and credit to each node involved in the forwarding of the packets. In the basic strategy to motivate nodes to forward packets, CCS asks the sender to pay the last node on the path receiving a packet $\alpha$ and all predecessors $\beta$. However, an intermediate node may report a (false) receipt even though it does not receive a packet. To prevent cheating, if the destination does not report a receipt, payment for every node will be reduced by multiplying a factor of $\gamma$ to the values in the basic scheme, where $\gamma \leq 1$.

Srinivasan et al. [59] proposed a distributed and scalable acceptance algorithm, named Generous TIT-FOR-TAT, based on which nodes decide whether to accept a relay request or not. Specifically, a node $i$ maintains a record of its past experience, including the acceptance ratio of its own relay request for type $j$ sessions and the acceptance ratio by node $i$ for type $j$ sessions of other nodes. If node $i$ has relayed more traffic sessions than it should, or for type $j$ session, it has already relayed more traffic than its own to be relayed by other nodes, it rejects the relay request. Otherwise, node $i$ accepts the request. It has been showed that this algorithm leads to a Nash equilibrium. Game theory has also been used to evaluate the CORE algorithm [60].

### 2.2.3 Mechanism Design Methods

Since nodes incur different costs to forward packets, it is desirable that they get reimbursed accordingly so that enough incentive is provided and the total payment charged is the least possible. This poses the requirement that nodes declare their cost honestly, or $turthfully$. Basically, this problem is related to research in mechanism design [62].

Anderegg and Eidenbenz [33] were the first to introduce mechanism design into ad hoc networks. They proposed the ad hoc-VCG for ad hoc networks with selfish agents. This routing protocol using the VCG mechanism is truthful and cost-efficient. In ad hoc-VCG, a route request message is flooded over the network and includes information (power and cost) of all links it has traversed. Upon receiving such a message and finding that this message

includes any new unknown link, a node appends its own cost and power information and broadcasts it. Thus, when a destination receives all the broadcast request messages, it can construct the whole topology and find a least cost path. The VCG mechanism is applied to the payment calculation for data messages, and it prevents nodes from cheating over their cost. However, this protocol has some shortcomings and leaves several questions unanswered. First, it needs $O(n^3)$ control messages for a route discovery. It is prohibitive for the source to pay this cost for its traffic. Second, with an $O(n^3)$ overhead, this protocol cannot provide good performance on metrics such as packet delivery ratio and end-to-end delay. Third, mobility is not taken into consideration. Fourth, the calculation of payment for the route discovery is complicated. All of these problems make this protocol difficult to use in practice.

Zhong et al. [41] proposed CORSAC, a truthful routing protocol which combines the VCG mechanism and cryptography. For each available power level, a node sends a signed test signal message. A route message is also signed. This type of cryptography is used to prevent possible under-declaration of a receiver. Based on these signed messages, which are flooded over the network by a route discovery similar to that of the ad hoc-VCG, the destination can determine the topology of the network and apply the VCG mechanism. However, this protocol has same shortcomings as the ad hoc-VCG protocol. The message overhead of this protocol is $O(\rho \cdot E \cdot n)$, where $\rho$ is the number of power levels, $E$ the number of links and $n$ is the number of nodes. More details of CORSAC are discussed in Chapter 4.3.3.

Chen and Nahrstedt [39] proposed iPass, an auction system where nodes get forwarding service by bidding in the intermediate nodes. A node sets an auction mechanism in itself and gives its power and bandwidth to the winner of the bid at the cost of the highest loser's price. To set the auction, iPass requires that any intermediate node needs at least two flows to pass through it concurrently. In addition, it pays intermediate nodes the same amount of payment, not taking into consideration their cost diversity. Also it does not specify how to select the next hop and thus find a path from the source to the destination.

Cai and Pooch [38] proposed another method, $TEAM$, in an attempt to provide a truthful and low-cost method. A node that can overhear a sender and a receiver becomes a redirector of data forwarding, i.e., the sender sends data to the redirector and then the redirector forwards the data to the receiver. The payment to the redirector is the power saving of this redirection. However, the payment received by intermediate nodes may not cover their cost, and thus nodes may have no incentive to forward packets. On the other

hand, the sender saves nothing and gets worse service due to the greater number of hops used.

Wang and Li [40] also discussed truthful routing based on the VCG mechanism, with the assumption that a node has a fixed cost of sending a packet to any of its outgoing neighbors. In particular, they proposed a time optimal algorithm to calculate payment to nodes in both centralized and distributed manners. The scheme is not suitable in cases where the forwarding cost to different neighbors varies.

Eidenbenz et al. [61] proposed another VCG mechanism-based protocol, COMMIT, which allows a source to set a reserve price for its data transmission to a destination. By utilizing underlying topology control protocols, COMMIT incurs an overhead of $O(n^2 \log n)$. Different from the model of the ad hoc-VCG and CORSAC, COMMIT assumes that a node incurs the same cost to send packets to different neighbors. On the other hand, selfish nodes may behave selfishly in routing, and they may do so in the topology control. Therefore, how to achieve topology control with selfish nodes itself is challenging.

# Chapter 3

# Detecting Selfish Nodes Behavior

## 3.1 Introduction

Punishment is usually an effective way to enforce obligation and cooperation. It has been shown to be even more effective in fostering cooperation than award-giving in animal societies [63]. However, such a method faces many challenges in ad hoc networks. Without trusted authorities, each node has to share the responsibility for detecting selfish behavior. Besides, the characteristics of wireless communication make accurate neighbor monitoring impossible. Different nodes have distinct observations for a given node due to their different sensing conditions/positions. Moreover, nodal mobility makes it difficult to trace node behavior. A good solution should be distributed, catch selfish nodes accurately, and introduce low overhead. It should also provide fairness such that nodes have the same chance to serve others and to be served by others.

We introduce two distributed detection-based solutions to solve the selfish nodes problem. Nodes in the network monitor, judge, punish and re-admit selfish node distributively. Both solutions use a polling mechanism to ensure fairness and cope with false accusations, and take into account the scarcity of resources such as battery life. They provide each node a fair chance to serve and be served.

The reminder of the chapter is arranged as follows: section 3.2 presents a fair distributed solution [30]. Section 3.3 presents a light-weight solution considering battery status [31]. Section 3.4 summarizes this chapter.

## 3.2 A Fair Distributed Solution

### 3.2.1 Overview

To evaluate the selfishness of nodes, we need a criterion to quantify the selfishness of a node. We define the information used to evaluate selfishness as *credit*, and use the packet forwarding ratio as *criterion*. The packet forwarding ratio is the ratio of the packets

forwarded to the total packets meant to be forwarded. Packet forwarding ratio reflects a node's contribution to the network. Each node must satisfy a preset minimum value for the packet forwarding ratio. The minimum forward ratio is based on the battery status of a node. A node may send (its own packet) far more packets than the number it has forwarded, as long as this node always satisfies the required forwarding ratio. This is not possible in motivation based schemes, where a node can only send as many packets as it forwards. The rationale is that every node provides a different contribution to the network. Because of location privilege, some nodes can contribute more and others less. However, to be *fair*, as long as they contribute, they should not be regarded as selfish nodes, and the network should provide service to them.

A node's contribution is evaluated by its neighbors based on their direct observations. These observations are compared with the node's self-evaluation, to justify behavior. In our scheme, nodes broadcast their own credit information, and their neighbors verify the credibility based on what they observe. To achieve this, every node calculates its credit and broadcasts it periodically to its 2-hop neighbors. Neighbors monitor the node's behavior and compare the declared value of credit with what they observe. If the deviation between the broadcast self-evaluation and the observed value exceeds a certain threshold, the monitoring node sends out a *warning* message about the monitored node. If more than $k$ nodes accuse the same node, the accused node is considered as a *selfish node* by all nodes in the network. The selfish node is punished by other nodes by dropping packets intended for, or originated from, such a node. Routes to be established bypass the selfish node. After a predetermined amount of time, a selfish node is re-admitted to the network. Thus, a selfish node knows that selfishness will be harmful, and will be forced to be cooperative.

### 3.2.2 Evaluating Selfish Behavior

Each node maintains *records* to monitor and evaluate the selfishness of 1-hop and 2-hop neighbor nodes. A record has the following fields: *NodeID,* $Pkt_{drop}$*,* $Pkt_{fwd}$*,* $Per_{fwd}$*, Batt_stat,* $Credit_{acc}$*, Seq_num. NodeID* denotes the identity of the monitored node. $Pkt_{drop}$ and $Pkt_{fwd}$ are the number of packets that the monitored node dropped and forwarded, respectively as observed by a monitoring (in the promiscuous mode) node. $Per_{fwd}$ is defined as $\frac{Pkt_{fwd}}{(Pkt_{drop}+Pkt_{fwd})} * 100\%$. *Batt_stat* represents the battery status of the monitored node. There are four levels, *Normal, Low1, Low2 and Recharged. Normal* indicates the node has enough battery. *Low1* indicates that the node does not have enough battery and thus, will not participate in new route discoveries or flooding (if any). *Low2* indicates that the node

has very low battery and implies that it will not participate in packet forwarding. *Recharged* indicates the node has been recharged and has enough battery. Different battery statuses corresponds to different criteria of selfishness (See below for further discussion). $Credit_{acc}$ is used to record accumulated net forwarded packets, i.e., $Credit_{acc} = \sum Pkt_{fwd} - \sum Pkt_{drop}$ in each battery life cycle (a cycle is the period during which the battery status changes from *Normal* to *Recharged*). *Seq_num* is the latest version of the credit message known to the monitoring node. A node also keeps its own credit information.

Initially, each node sets up an entry for each of its neighbors. For every entry, the initial value of all fields is set to zero, except $Per_{fwd}$, which is set to 100% (that is, initially all nodes are considered unselfish). Every node maintains its own credit by adding $Pkt_{drop}$ and $Pkt_{fwd}$, respectively, taking retransmissions into consideration. Periodically, it sends out a *Credit* message to its neighbors consisting of *NodeID,* $Pkt_{drop}$*,* $Pkt_{fwd}$*, Batt_stat*, and *Seq_num*. Each node increases *Seq_num* by one for every credit update message it broadcasts. The credit message is updated upon the expiration of the *credit_update_timer* or when the *Batt_stat* is changed due to energy dissipation.

### 3.2.3 Monitoring Selfishness

Nodes monitor neighbors' activity by listening (in the promiscuous mode) to all packets within the radio range. If a neighbor forwards a packet, then it increases $Pkt_{fwd}$ for that neighbor, taking the retransmissions into consideration. Similarly, if the neighbor drops a packet, it increases the corresponding $Pkt_{drop}$. When a node receives its neighbor's *Credit* message, it compares $Pkt_{drop}$ and $Pkt_{fwd}$ data in the message with what it has monitored. If the data differs from the observed value beyond a threshold, then the monitoring node suspects that the neighboring node is lying. In practice, there are some cases where it is hard to judge whether a packet has been forwarded or not [50]. Therefore, the data declared and the data monitored may not be equal. However, if the originator of the credit message is honest, then the deviation between these data will not be large. If a node finds the deviation is under threshold $\tau$, it adapts the declared value and overwrites its own observed value, and continues monitoring based on this new value. Otherwise, if the deviation exceeds $\tau$, the node sends out a *warning* message to accuse the monitored node with the following fields, *AccusedID, AccuserID, Seq_num*, and*Mac*. *AccusedID* and *AccuserID* denote the monitored node and the originator of this message, respectively. *Seq_num* is the same as in the latest *Credit* message. *MAC* (Message Authentication Code) is a signed digest over all previous fields with a private key. It is used to provide integrity and non-repudiation.

The conviction is based on voting, i.e., if more than $k$ neighbors accuse the same node, the accused is convicted of being selfish. A selfish node may accuse a normal node of being selfish. A larger $k$ can prevent more false accusations and avoid unnecessarily false punishment of a good node. However, a larger $k$ requires more time to converge. On the other hand, as the nodes on the periphery of the network have fewer neighbors, a smaller value works well in their situation. Generally, a node in an ad hoc network may have 8-10 direct neighbors [14]. So $k$ can be set as 5 or 6, or at least 60% of the node's neighbors.

### 3.2.4 Confirming Selfishness

To confirm that a node is selfish, nodes use $Ratio_{fwd}$ (packet forwarding ratio) and *Batt_stat*. We use three different thresholds, *th1*, *th2* and *th3* for $Ratio_{fwd}$, corresponding to battery status. When a node is in *normal* status, its $Ratio_{fwd}$ should be above *th1*. Otherwise, the monitored node is convicted of selfishness. Similarly, *th2* and *th3* correspond to *Low1* and *Low2* battery status. It is also important to note that $th1 > th2 > th3$. Also, *th1* should be much higher than *th3* (*th3* is at least 50%) because if the node has enough battery, it should forward packets.

A node may pretend to have lower battery status to forward few packets and save its energy. To prevent this, we introduce two strategies. The first strategy is to restrict the benefits a lying node can get. If a node declares its status in *Low1*, then it is free from participating in routing. However, it cannot initiate a new route discovery for itself, and it has to participate in packet forwarding. A node in *Low2* status is free from forwarding packets. However, it is allowed to send out its own packets, as many as $Pkt_{fwd}$ - $Pkt_{drop}$, i.e., its net contribution is recorded as credit. After a node is in *Low2* status, it is given a grace period of $\Delta T$ seconds to get recharged. During $\Delta T$, the node may not send or forward packets. After getting recharged, it issues a new *Credit* message to update its neighbors. Upon receiving this announcement, the neighbors set $Credit_{acc}$ to $Credit_{acc}$ + $Pkt_{fwd}$ - $Pkt_{drop}$ and resets $Pkt_{fwd}$ and $Pkt_{drop}$ for the originator to zero and $Ratio_{fwd}$ to 100%. The data used for $Ratio_{fwd}$ is based on the new $Pkt_{fwd}$ and $Pkt_{drop}$ values. This is because as the node has recharged and has enough battery, and therefore should behave normally and be evaluated normally. On the other hand, its previous contribution should not be ignored. So, $Credit_{acc}$ is used to record its previous contribution and the node can use it when in *Low2* status.

Another strategy is to have every node estimate the battery consumption of its neighbors. A node's battery consumption consists of four parts: power used for routine task,

power used for working in the promiscuous mode, power used for forwarding and sending packets, and power used for receiving packets, denoted by $e_R,\ e_p,\ e_s,\ e_r$, respectively. Thus, total power consumption $P = e_R + e_p + e_s + e_r$ . $E_R$ and $e_p$ are time dependent quantities and are assumed to be same for all the nodes. Depending on how long a node has been working during a battery cycle, nodes can estimate $e_R$ and $e_p$. $E_s$ and $e_r$ are proportional to the number of packets sent or received. Supposing that forwarding a packet requires $e$ units of power and every node consumes the same energy for packet forwarding, $e_s$ can be estimated as $e \star Pkt_{fwd}$(the difference in packet length is ignored); similarly we can estimate $e_r$. Since every node can estimate its neighbor's battery consumption, it can verify the credibility of *Batt_stat* in the *Credit* message broadcast by the neighbor. Using the above criteria, every node can independently decide if a node is selfish.

### 3.2.5 Punishment and Re-admission

Once a selfish node has been identified, it is punished. The neighbors of a convicted node refuse to forward any packets originating from this selfish node. Thus, a selfish node will be excluded from the network. However, the selfish node is excluded only temporally (for some predetermined *temp_bypass* time). The exclusion is not a perfect solution; instead the goal is to force the nodes to cooperate and thus benefit each other. After punishment, a selfish node is likely to be more cooperative. As a selfish node is not a malicious node, it's fair to give it a chance to provide service and use the network. However, some nodes may continue to behave selfishly even after being initially punished. In such a case, the *temp_bypass* time can be increased exponentially as $2^i \star temp\_bypass$ where $i$ represents the number of times the node has behaved selfishly. Thus, nodes continuing to behave selfishly are punished severely.

### 3.2.6 Combining with Routing Protocols

This protocol can be easily combined with routing protocols such as DSR [13] and AODV [19]. In AODV, whenever a source node needs a route to a destination node, it broadcasts the route request *RREQ*. Any intermediate node which knows a fresh route to the destination may reply to this request, otherwise it propagates the request. Upon receiving the request, a destination replies with an *RREP* towards the source along the reverse route. During routing maintenance, upon detecting an inaccessible downstream node, a node sends *RERR* along the active route(s) toward the source(s). Combined with our scheme, a node needs to check the selfish node list whenever it gets routing packets. It

can drop any packet sent from a selfish node. Thus during route discovery, any known selfish node will be excluded. During routing maintenance, upon convicting its downstream node as a selfish node, a node sends an *RERR* to the source. The source may initiate a new route request to find a good route. The combination is even simpler for DSR. An intermediate node can exclude selfish nodes as in the case of AODV. The source itself can check the node list in the *RREP* and drop routes containing any known selfish node. Similarly, during data forwarding, when a source receives a message declaring a newly found selfish node, it deletes all routes containing that selfish node.

### 3.2.7 Performance Evaluation

#### 3.2.7.1 Simulation Setup

We conducted simulations to evaluate the impact of our scheme on a network with selfish nodes. Specifically, our simulation focused on the following metrics:

- Packet delivery ratio is defined as $\frac{\sum Packets_{received}}{\sum Packets_{sent}}$, i.e., the ratio of the total number of packets received by the intended receivers to total packets originated by all nodes. To be realistic, we considered the packet delivery ratio as the comprehensive result of all factors which could affect the packet delivery rate, from the application layer to the physical layer, such as collisions in the MAC layer, rerouting in the network layer, and so on.

- Overhead is defined as $\sum_{i=1}^{n} Packets_i^{control}$, i.e., the overall protocol control packets exchanged by nodes in the network. It includes *Hello* messages and routing packets such as *RREQ*, *RREP* and *RERR*. In our study, three additional types of packets (or messages) are introduced: *Credit* messages, *Accuse* messages and *Black* messages. A *Credit* message is periodically broadcast by a sender to provide credit information. An *Accuse* message is sent whenever a node suspects that its direct neighbor is behaving selfishly based on its observations. Finally, a *Black* message is flooded all over the network whenever a node has been convicted as a selfish node.

- A false conviction is the number of nodes that convicted as selfish that are actually cooperative.

- A missed conviction is defined as $\frac{Nodes_{selfish}^{undetected}}{Nodes_{selfish}} * 100\%$, i.e., the percentage of nodes that not detected as selfish nodes that behave selfishly.

We use AODV as the underlying routing protocol and have two implementations for it. The original AODV, named *base*, is used for comparison with our enhancement, termed *enhance*.

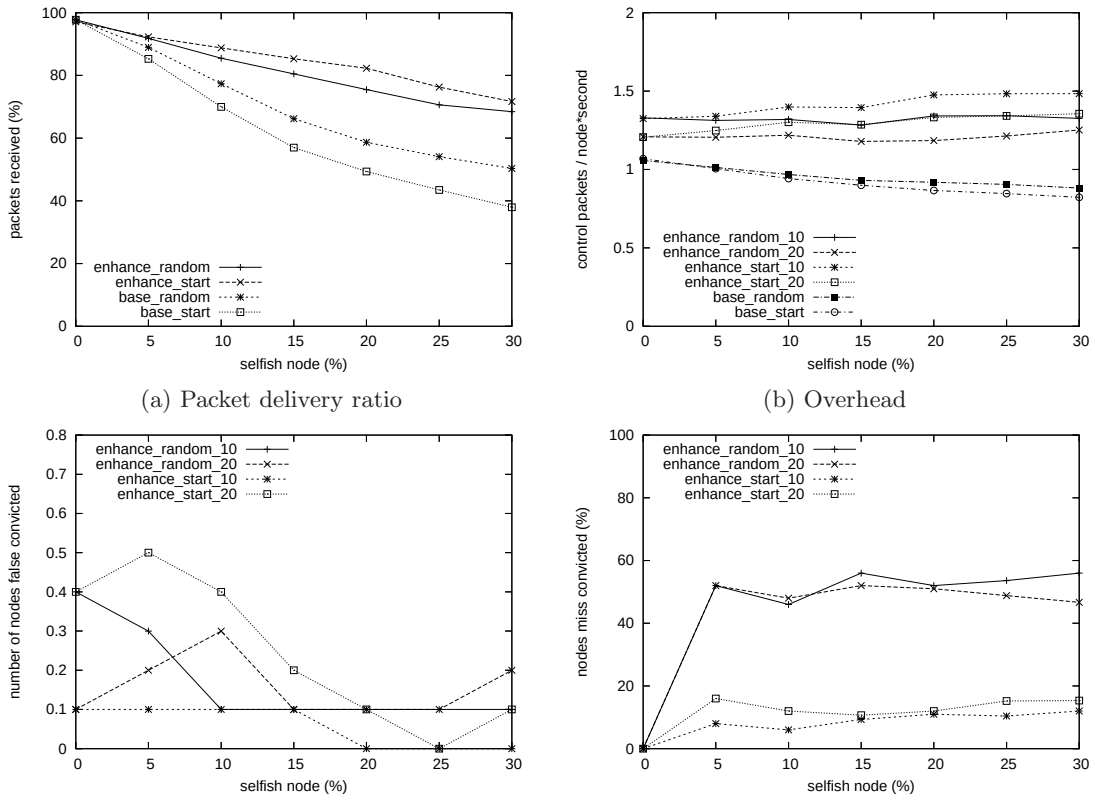
We modeled selfish behavior in two ways. In the first model, a selfish node starts behaving selfishly from the beginning. We denote this version as *enhance_start* and *base_start*, corresponding to *enhance* and *base* version, respectively. In the second model, the selfish node begins to behave selfishly at a random time during the simulation. Thus a node may drop packets at the beginning of simulation while another node may drop packets 400 seconds into simulation time. Nodes tend to be selfish after they dissipate more energy, thus we believe this model to be more realistic than the first model. We denote the version corresponding to this model as *enhance_random* and *base_random*.

Also, we implemented two versions to detect selfish behavior. One is directed at data packet forwarding only. The other aims to detect selfish routing behavior.

We used Glomosim, a scalable network simulator [64], for simulation. Unless specified otherwise, the following parameters were used in our simulations. 50 nodes were placed uniformly in an area of 1000m by 1000m, with a radio range of 250m. 802.11 protocol with DCF was used as the MAC protocol. All nodes followed the Random Waypoint mobility model [65] with a speed up to 10 m/s and a pause time of 30 seconds. Each simulation lasted for 900 seconds of simulated time. 10 CBR flows were simulated, each sending 4 512-byte data packets per second. Each flow started at 120 seconds and ended at 880 seconds. The first 120 second period of the simulation was used to provide sufficient randomization to nodal positions. The last 20 seconds of the simulation were used to prevent the data transmissions from sudden stopping because of the end of the simulation. Data points represented in graphs were averaged over 10 simulation runs, each with a different seed.

#### 3.2.7.2 Simulation Results

**3.2.7.2.1 Packet Delivery Ratio** Fig. 3.1(a) shows the packet delivery ratio with respect to percentage of selfish nodes. Both *enhance* versions have a higher packet delivery ratio than the corresponding *base* version: *enhance_start* yields as much as a 90% improvement over *base_start* and *enhance_random* shows up to a 36% improvement over *base_random*. The packet delivery ratio decreases as the percentage of selfish nodes increases because more nodes drop packets. An important feature is that *enhance_start* performs better than *enhance-random* while *base_start* performs better than *base-random*. This is because both *base* and *enhance* systems suffer from selfish nodes dropping packets.




(a) Packet delivery ratio

(b) Overhead

(c) False conviction

(d) Missed conviction ratio

Figure 3.1: Performance under different selfish node percentages

However, in the *base* case, no action is taken to bypass these selfish nodes. In *base_start*, all selfish nodes behave selfishly from the beginning and thus they drop all packets passing through them. In *base_random*, selfish nodes only behave selfishly from a random moment. They may behave cooperatively for some time and forward packets. Some selfish nodes even behave well for most of the simulation time. Thus more packets are forwarded in the *base_random* case. However, under an enhanced scheme, selfish behavior will be detected and those selfish nodes will be bypassed. The earlier the behavior is detected, the fewer the dropped packets and thus the higher the packet delivery ratio. If selfish nodes behave selfishly from the beginning, they are much easier to detect. However, if they behave well initially for a period of time and selfishly thereafter, the deviation of credit between the observed and the self-declared may not exceed the threshold or the deviation may not be sharp enough and thus they are hard to be detected. As shown in Fig. 3.1(d), there are more undetected selfish nodes in *enhance_random* than in *enhance_start*. Finally, in the presence of 30% selfish nodes, the packet delivery ratio is still as high as 37% in *base_start*. This is because a portion of traffic occurs between direct neighbors. In addition, some traffic has

short paths, with only one or two intermediate nodes behaving well.

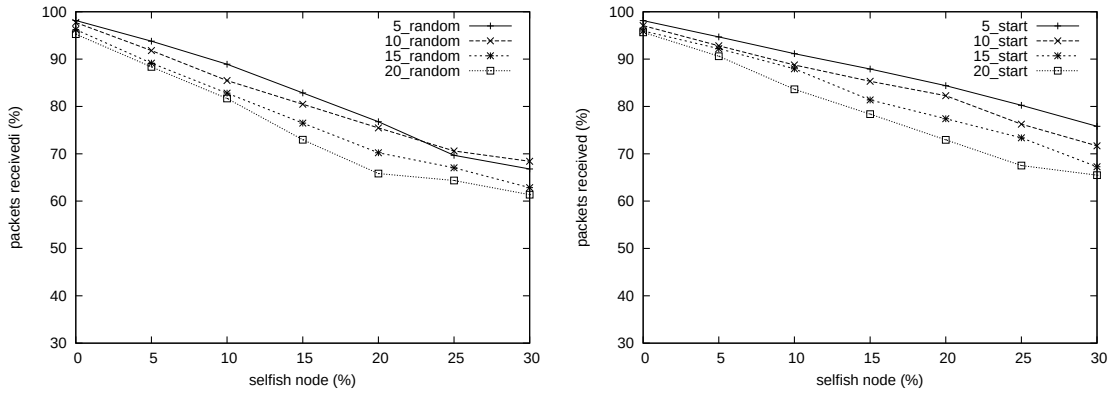
**3.2.7.2.2 Overhead** Fig. 3.1(b) shows the overall protocol control overhead introduced by our enhancement over different evaluating intervals with respect to the percentage of selfish nodes. It is normalized as packets per node per second. *Enhance_random_10* and *enhance_random_20* correspond to *enhance_random* with an evaluating interval of 10 and 20 seconds, respectively. This is the same for *enhance_start_10* and *enhance_start_20*. The overhead is mainly due to the *Hello* messages, which are broadcast every two seconds by every node, and *RREQ* messages. The periodic *Credit* messages also contribute some overhead. As the evaluating interval increases, the *Credit* messages decrease and thus the overhead decreases. As the percentage of selfish nodes increases, the overhead increases slowly. This is because more selfish nodes will be detected and thus more new *RREQ*s will be initiated. This also explains why overhead caused by *enhance_start* is higher than that caused by *enhance_random* although they produce the same number of credit messages. In *enhance_start*, selfish behavior is easier to detect and thus produces more *RREQ* packets.

**3.2.7.2.3 False Conviction** Fig. 3.1(c) shows the number of false convictions with respect to percentage of selfish nodes. In any case, no more than 0.5 nodes, averaging over 10 runs, are falsely convicted. So the rate of false conviction is very low. A node will file an accusation as long as the deviation between the observed and the declared exceeds the threshold. Due to radio interference, a node cannot always monitor its neighbors accurately and thus convicting a node based on a single accusation is not feasible. A voting system is used to avoid false convictions. However, false convictions still happen in special cases. For example, if the majority of the observed node's neighbors are between the observed node and some other traffic, due to interference from these traffic, neighboring nodes may misjudge the node as a selfish node and accuse it. The accused node will be wrongly convicted. We can raise the threshold in the voting system to lower false convictions. However, this will raise the missed convictions and thus lower the performance (i.e., packet delivery ratio). Therefore, there is a tradeoff between false convictions and missed convictions. A false conviction is not sensitive to the evaluating interval since it occurs in some special cases.

**3.2.7.2.4 Missed Conviction** Fig.3.1(d) shows the missed conviction ratio with respect to the percentage of selfish nodes. *Enhance_random* has a high missed conviction ratio, averaging about 50% while *enhance_start* has a much lower ratio, averaging less than 14% (about 9.7% for the evaluating interval of 10s). As discussed earlier, detection of selfish

behavior is difficult in the former case and easy in the latter one. We also observed that the evaluating interval affected the missed conviction ratio. A larger evaluating interval reduces the missed conviction ratio in *enhance_random* while increasing the ratio in *enhance_start*. This can be explained as follows. In *enhance_random*, a selfish node may behave well in the beginning and selfishly later, so we need a credit deviation between the observed and the declared sharp enough to detect this behavior. A larger interval helps this deviation. In *enhance_start*, such a requirement is not necessary and a smaller interval is more aggressive in detecting the misbehavior. Of course, too small an interval will blur the credit deviation and thus raise the missed conviction ratio.

**3.2.7.2.5 Impact of Mobility** Fig. 3.2(a) and (b) show packet delivery ratio with respect to maximum node speed, corresponding to selfish behavior from a random moment and from the beginning, respectively. Four speeds were used in the mobility model - 5m/s, 10m/s,15m/s and 20m/s, keeping a maximum pause time of 30 seconds. In both scenarios, as speed increases, the packet delivery ratio drops. There are two reasons for this. One reason is that higher mobility causes more link breakages. Another reason is that missed convictions increase with higher mobility because it is harder to trace a highly mobile selfish node. Again, the *start* cases performed better than the *random* cases.




(a) From random time

(b) From start

Figure 3.2: Packet delivery ratio vs. selfish node percentage, with changing speed

### 3.2.7.3 Impact of Routing Packets

Fig. 3.3 shows false conviction based on the detection of selfish routing behavior, where a node does not participate in routing, with an evaluating interval of 20 seconds. As in the data forwarding case, we name the cases where nodes behave selfishly from the be-

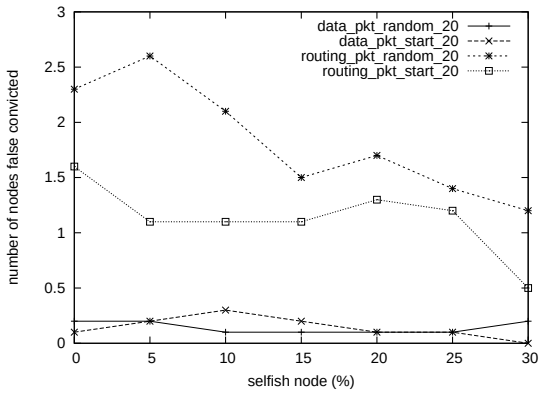



Figure 3.3: False conviction for detecting routing behavior

ginning and from random moment for routing as *routing_pkt_random* and *routing_pkt_start*, correspondingly. Compared to the result based on data packets, termed *data_pkt_random* and *data_pkt_start*, false conviction in *routing_pkt_random* and *routing_pkt_start* is much higher. In fact, the credit deviation threshold set for routing is less strict than that for data forwarding. If using same threshold, the false conviction will be even higher and cannot be used for detecting misbehavior at all. This is ascribed to the broadcast property of the routing message. Unlike a unicast message, broadcast does not use RTS/CTS. A broadcast packet is sent out as long as the channel is free and will not be retransmitted even if collision occurs. Routing message (RREQ) is flooded over the network and causes collision and interference here and there. So the credit counting for a routing packet forwarded is much less accurate than that in data forwarding and thus results in a high rate of false convictions. This simulation reveals the very important fact that a purely detection-based method is not suitable for detecting routing selfish behavior.

### 3.3 A Light-weight Solution Considering Battery Status

In the previous section, we presented a fair distributed solution for the selfish-node problem, focusing on *location privilege* problems. Selfish nodes should be detected. However, a good solution for selfish nodes should also be light-weight. If the cost for identifying selfish nodes in terms of battery and bandwidth is very high, it may be enough to just identify suspicious nodes. A selfish node is not a *malicious node* as it just wants to save its resource, not to destroy or attack others.

In this section, we present a new solution for the selfish-node problem, considering all the above issues. The solution is based on neighbor monitoring. However, this monitoring is on demand. We consider battery status such that nodes work in promiscuous mode only

when necessary. They can ask for exemption from data forwarding based on their battery status.

### 3.3.1 Overview

To enable neighbor monitoring, nodes work in the promiscuous mode. However, working in the promiscuous mode has a high cost. Feeney [66] showed that receiving traffic in the promiscuous mode is much more expensive than discarding traffic and even more expensive than working in idle mode. Thus, to save energy, nodes should work in the promiscuous mode as little as possible.

Generally, the majority of traffic will pass through the center of the network. So, nodes in the center will forward more packets (and thus consume more energy) than those in the periphery. This results in a location privilege problem. To mitigate this problem and to provide fairness to all nodes, we evaluate a node's behavior based on its battery status. A node can declare low battery status to exempt itself from forwarding packets. Measures can be taken to prevent a node from cheating on battery status. Also, a cooperative node should be able to get service from others.

Initially, all nodes need not work in the promiscuous mode. After sending a data packet, a transmitter $S$ turns onto promiscuous mode to monitor the downstream node $A$'s behavior. Upon detecting the dropping of a certain number of consecutive packets, the transmitter $S$ sends a broadcast message to ask for the help of $A$'s neighbors. These neighbor nodes turn on the promiscuous mode to monitor $A$'s behavior for some time and turn off promiscuous mode thereafter. Node $A$ defends itself by declaring the number of packets it has forwarded. Based on their own observation, neighbors make their judgment about node $A$. If the majority of neighbors accuse $A$, $A$ is convicted as a selfish node. A selfish node will be punished by other nodes by dropping packets intended for and originated from such a selfish node. Routes will be re-established to bypass the selfish node. After a predetermined amount of time, a selfish node can be re-admitted to the network. Thus, a selfish node knows that the selfish behavior will be harmful and it will be forced to be more cooperative.

### 3.3.2 Monitoring Selfish Behavior

After a node $S$ sends (or forwards) a packet to its downstream node $A$, it turns onto the promiscuous mode to observe if $A$ forwards the packet. No other node monitors $A$'s forwarding behavior at this time. If $S$ does not overhear that $A$ forwarded that packet within a specific period, $S$ suspects that $A$ dropped that packet. If $S$ finds $A$ has not

forwarded packet for a consecutive number of times (e.g., 20 packets), $S$ has good reason to believe that $A$ is selfish. However, it is possible that $S$ did not overhear packet forwarding due to collisions or interference. So $S$ asks the help of other nodes by broadcasting an *Alert* message to other adjacent nodes. An *Alert* message includes the following fields, $Target, Requestor, Period, Seq\_num$, where $Target$ is the ID of the node that needs to be monitored, *Requestor* is the ID of the node originating this request (in this example, it is $S$), *Period* indicates how long the observation should last. *Period* can be decided based on the rate of the traffic forwarded by the requester. The requester can ask to monitor a predefined number of packets. Thus, the *Period* is inversely proportional to the packet sending rate of a traffic, i.e., the higher the traffic rate, the shorter the period. Node $A$ can also get an *Alert*.

Upon receiving the *Alert* message, every neighbor of $A$ establishes a record for node $A$. They turn onto promiscuous mode and overhear the forwarding packets at node $A$ and increase the packets dropped, sending counts accordingly. This observation lasts for the *observation* period in the *Alert* message. On the other hand, node $A$ also gets the *Alert* message. By the end of this *observation* period, node $A$ has to broadcast a self-declared $Defence$ message to state how many packets it forwarded during the *observation* period, including $< NodeID, Pkt_{fwd}, Pkt_{drop}, Seq\_num >$, where $NodeID$ is the originator of this message, $Pkt_{fwd}$ and $Pkt_{drop}$ indicates the number of packets forwarded and dropped during the *observation* period, $Seq\_num$ is used to prevent replay of an old $Defence$.

### 3.3.3 Conviction of a Selfish Node

Upon receiving a $Defence$ message, $A$'s neighbors will compare their observation with this self-declared value. If the deviation exceeds a threshold $\tau$, an *Accuse* message against $A$ is sent out. An *Accuse* message includes the following fields: *AccusedID, AccuserID, Seq_num, Mac*. *AccusedID* and *AccuserID* denote the monitored node and the originator of this message, respectively. $Seq\_num$ is the same as in the last $Defence$ message. *MAC* (Message Authentication Code) is a signed digest over all the previous fields with the private key of the sender of the *Accuse* message. It is used to provide integrity and non-repudiation. It is possible that $A$ does not broadcast $Defence$ or its neighbor does not receive this message. Then the node that does not receive the $Defence$ message will send an *Accuse* against $A$. After sending their opinions, $A$'s neighbors can turn off their promiscuous mode for the purpose of monitoring $A$ unless requested to do so by other nodes.

The conviction is based on voting, i.e., if more than the majority of neighbors or over a predefined number of neighbors accuse a node $A$, $A$ will be convicted as a selfish node.

***A Simplification:*** The above algorithm can prevent some attacks from *malicious* nodes. If we assume no malicious nodes, we can further simplify the algorithm. Some nodes may have an accurate observation for the observed node. Upon hearing an *Accuse* message, if a node finds that its observed value is almost the same as the self-declared value in the $Defence$ message, it can send an $Innocence$ message to suppress further *Accuse* messages. The originator of *Alert* can verify if the $Defence$ coincides with what the originator forwarded. This should reduce the overhead caused by *Accuse* messages. Moreover, it may reduce $false\ convictions$.

***Punishment*** As discussed in Section 3.2, a convicted selfish node is temporarily excluded from the network for a predetermined *temp_bypass* time and then will be re-admitted. If it still behaves selfishly thereafter, the *temp_bypass* time is increased exponentially as $2^i \star temp\_bypass$ where $i$ represents the number of times the node has behaved selfishly.

***Enhancing Routing Protocols*** As discussed in Section 3.2, this scheme can be combined with routing protocols such as DSR [13] and AODV [19]. During route discovery, nodes drop route request sent (forwarded) from the known selfish nodes. During data packet forwarding, if a node $A$ convicts its downstream node $B$ as a selfish node, $A$ sends *RERR* to the source. The source may initiate a new route request to find a good route. In DSR, the source can also check the node list in the Route Reply and drop a route containing any known selfish node. Also, upon receiving a notification that declares a new convicted selfish node in DSR, the source can delete routes including that selfish node.

### 3.3.4 Battery Caring

A node may lose battery over time and thus become unsuitable for forwarding packets. Such a node can declare that it has low battery and is free from forwarding packets. The traffic going through this node should change routes to bypass this node. The node in low battery status will not be regarded as a selfish node and can still get service from others. Under this scheme, nodes in the center of the network, usually consuming more battery power, can have a rest and get no punishment, and therefore the location privilege problem is mitigated.

When a node $C$ has low battery, it broadcasts a $Battery$ message to announce its low battery status, including $NodeID, Batt\_stat, Seq\_num$, where $NodeId$ is the ID of the

originator, *Batt_stat* indicates battery status of that node (defined as *Normal* or *Low*), *Seq_num* indicates the freshness of the message. After receiving $C$'s *Battery* message, its upstream node on the route should inform the source to change its route.

A node in low battery status is given a grace period $\Delta T$ seconds to get recharged. After recharging, the node issues a new *Battery* message to inform its neighbors. Thereafter it is expected to forward packets normally.

A node may pretend to be in low battery status and thus it may forward few packets. As discussed in Section 3.2.4, two possible strategies can be used to prevent such cheating. The first strategy is to restrict the benefits a lying node can get. A node in low battery status is free from forwarding packets. However, it is allowed to send out only $Pkt_{allowed}$ number of its own packets. If a node is cooperative, it will contribute to the network. This contribution should be reimbursed. However, due to the limited remaining battery power, even though a node may have a great net contribution, it can only send a limited number $Pkt_{allowed}$ of its own packets while in low battery status. Nodes with less net contribution may thus get additional benefits. However, this benefit is limited due to the above packet limitation $Pkt_{allowed}$.

Another strategy is to have every node estimate the battery consumption of its neighbors. A node's battery consumption consists of four parts: power used for routine tasks, power used for working in the promiscuous mode, power used for forwarding and sending packets, and power used for receiving packets, denoted by $e_R$, $e_p$, $e_s$, $e_r$, respectively. $E_R$ is time dependent quantity, decided by how long a node has worked during a battery cycle, and is assumed to be same for all the nodes. $E_p$ consists of two parts, one to monitor its downstream node's behavior, which is proportional to the number of packets sent, and the other to monitor a neighbor's behavior under the request of some neighbor(s). This can be assumed to be same for all the nodes. $E_s$ and $e_r$ are proportional to the number of packets sent or received. Assuming that forwarding a packet requires the same unit of energy for all nodes, $e_s$ and $e_r$ can be estimated based on the number of packets sent or received. Based on this estimation of battery consumption, a node can verify the credibility of the *Battery* message broadcast by the neighbor.

### 3.3.5 Performance Evaluation

#### 3.3.5.1 Simulation Setup

We conducted an extensive simulation study to evaluate the impact of our scheme on a network with selfish nodes. In additional to the metrics (packet delivery ratio, overhead,

false conviction and missed conviction) measured in Section 3.2, we evaluated another metric, that is, *Averaged promiscuous time*. This indicates how much time, on the average, a node works in the promiscuous mode. As Glomosim does not provide enough information for energy consumption in the promiscuous mode, we use this metric to evaluate the effect of power saving.

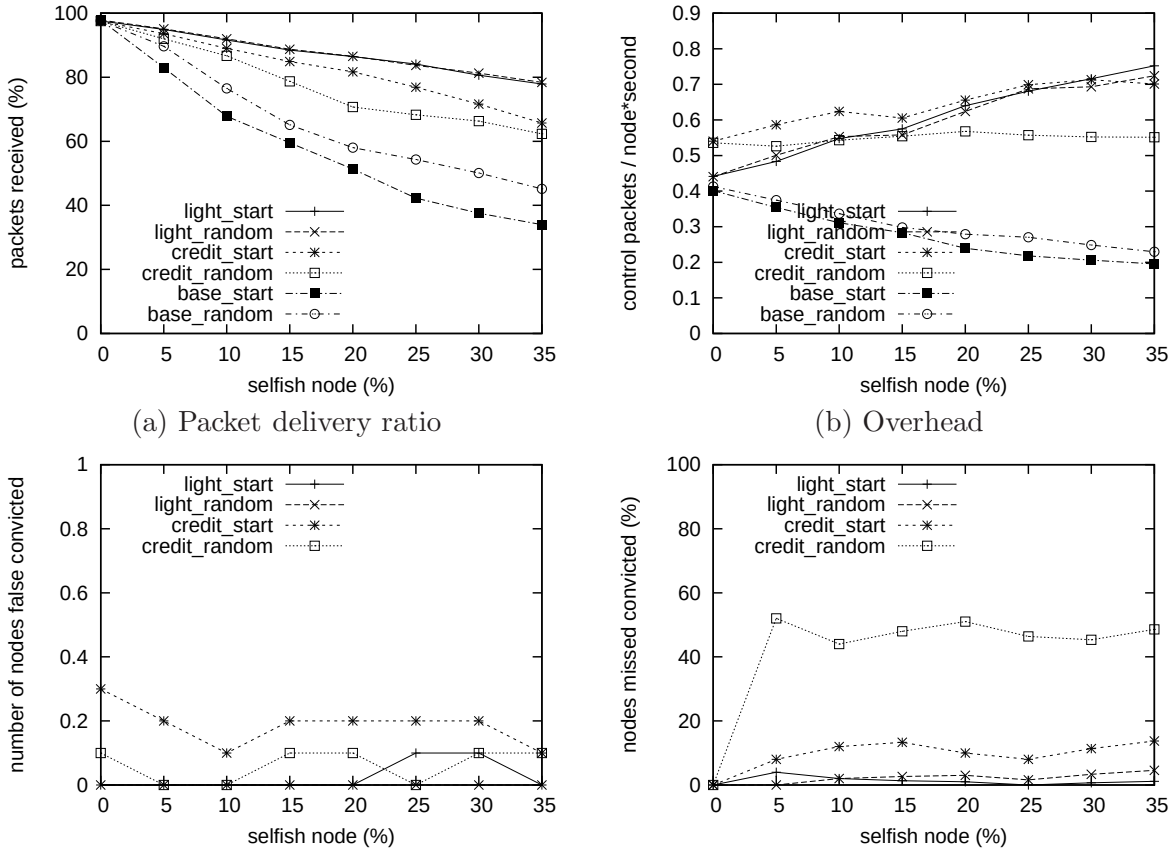
We compared the proposed protocol with work carried out in Section 3.2. The curves for the algorithm in Section 3.2 are denoted as *credit*. The curves for the system without any enhancement, which is used as the baseline for comparison, are denoted as *base*. The curves for the light-weight algorithm are denoted as *light*.

As in Section 3.2.7.2, we modeled selfish behavior in two ways. First, a selfish node starts behaving selfishly from the beginning. We denoted this version as *light_start*, *credit_start* and *base_start*, corresponding to *light*, *credit* and *base* versions, respectively. In the second model, a selfish node begins to behave selfishly at a random time during the simulation. We denoted the version corresponding to it as *light_random*, *credit_random* and *base_random* corresponding to *light*, *credit* and *base* versions, respectively.

We used Glomosim for simulation. 50 nodes were placed uniformly in an area of 1000m by 1000m. The radio range of nodes was 250m. We used AODV as the underlying routing protocol and 802.11 protocol with DCF as the MAC protocol. All nodes followed the Random Waypoint mobility model with a speed range of 0 m/s to 10 m/s and a pause time of 30 seconds. Each simulation lasted for 900 seconds. 10 CBR flows were simulated. Each flow sent four 512-byte data packets per second, started at 120 seconds and ended at 880 seconds. Data points represented in the graph were averaged over 10 simulation runs, each with different seed.

#### 3.3.5.2 Simulation Results

**3.3.5.2.1 Packet Delivery Ratio** Fig. 3.4(a) shows the packet delivery ratio with respect to the percentage of selfish nodes, with a maximum speed of 10m/s. As the percentage of selfish nodes increases, the packet delivery ratio decreases as more nodes drop packets. In the figure, when 35% nodes are selfish, *light_start* improves the delivery ratio by 130% over *base_start*, and *light_random* improves the delivery ratio by 74% over *base_random*. Compared with the fair solution in Section 3.2, *light_start* improves the delivery ratio by 18% over *credit_start*; *light_random* improves the delivery ratio by 25% over *credit_random*. Also we can see that *light_start* performs the same as *light-random* whereas *credit_start* performs better than *credit_random* and *base_random* performs better




(a) Packet delivery ratio

(b) Overhead

(c) False conviction

(d) Missed conviction percentage

Figure 3.4: New performance under different selfish node percentages.

than *base_start*. The performance of *base* and *credit* have been discussed in Section 3.2.7.2. In the *base* case, no action was taken to bypass these selfish nodes. In the *credit* system, selfish nodes behaving selfishly from a random moment are more difficult to find than those behaving selfishly from the beginning. Under the *light* scheme, once a node is suspected to behave selfishly, it will be traced and only its behavior during the traced period will be considered. Thus, a node can be caught regardless of when it behaves selfishly. Compared with *credit*, *light* can catch more selfish nodes, as shown in Fig. 3.4(d), and thus has a higher packet delivery ratio.

**3.3.5.2.2 Overhead** Fig. 3.4(b) shows the overall overhead introduced by our enhancement with respect to percentage of selfish nodes, with a maximum speed of 10m/s.. It is normalized as packets per node per second. The overhead is mainly due to *RREQ* messages. In *light* scheme, total control packets not including *RREQ* is only 6% - 17% of the overall overhead, and the total overhead in *light_start* is almost the same as in *light_random*. As the

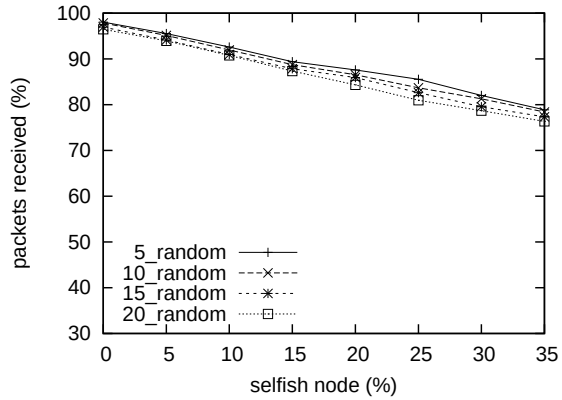
percentage of selfish nodes increases, the overhead increases. This is because more selfish nodes will be detected and thus more new *RREQ* will be initiated. Compared with the *credit* scheme, *light_start* has a lower overhead than *credit_start* whereas *light_random* has a higher overhead than *credit_random*. Nevertheless, the overhead is still low, with less than 0.75 packet per node per second.

**3.3.5.2.3 False Conviction** Fig. 3.4(c) shows false convictions with respect to the percentage of selfish nodes, with a maximum speed of 10m/s. *light_random* causes no false convictions. Under the *light_start*, false conviction was only 0.1 node (less than 0.8%) and happens only in 25% and 30% selfish nodes case. Compared with *credit*, *light* catches the selfish node more accurately. We use a voting system to avoid false convictions. Only two false convictions occurred in some extreme cases. In these cases, the majority of an observed node's neighbors are between the observed node and other traffics. These observing nodes suffer from interference from these traffics and hence have inaccurate observations for the observed node. Thus, such nodes accuse the observed node.

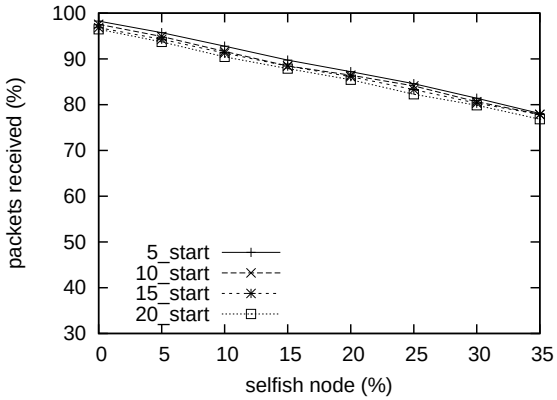
**3.3.5.2.4 Missed Conviction** Fig. 3.4(d) shows the missed conviction ratio with respect to the percentage of selfish nodes, with a maximum speed of 10m/s. Both *light_random* and *light_start* catch almost all the selfish nodes, with maximum missed conviction ratio of 4.5%. In most cases of *light_start*, the missed conviction ratio is under 1%. It catches more selfish nodes than *light_random* because nodes behaving selfishly from the very beginning give the system more time and chance to detect them. On the contrary, missed conviction in *credit* is pretty high, particularly for *credit_random*, for the same reason discussed in the *packet delivery ratio* section.

**3.3.5.2.5 Impact of Mobility** Fig. 3.5(a) and (b) show the packet delivery ratio with respect to maximum speed, corresponding to selfish behavior from a random moment and from the beginning, respectively. We used four speeds in the mobility model as 5m/s, 10m/s, 15m/s and 20m/s, keeping the maximum pause time at 30 seconds. In both figures, an increase in speed has little effect on packet delivery ratio, mainly due to higher mobility causing more link breakages. It shows our scheme is insensitive to mobility.

**3.3.5.2.6 Time in the Promiscuous Mode** Fig. 3.6 shows the time a node spends in the promiscuous mode. *light_start* and *light_random* spend almost the same time, which is less than 35% of the total data traffic time. Unexpectedly, as the percentage of selfish




(a) Behaving selfishly from a random moment




(b) Behaving selfishly from the start

Figure 3.5: New packet delivery ratio vs. selfish node percentage, with changing speed

node increases, the time spent in the promiscuous mode decreases. On the one hand, as selfish nodes increase, more *Alert* messages will be sent and more nodes will turn on the promiscuous mode. Thus more nodes may increase their promiscuous time. On the other hand, more selfish nodes will drop more packets. If a packet is dropped, the successors in the path do not need to turn on the promiscuous mode to monitor their downstream nodes and thus, decrease the time in the promiscuous mode. In some cases, the source cannot find a path within a period. During that period, no node needs to work in the promiscuous mode for that traffic. The decrease factor outweighs the increase factor, so the overall time for the promiscuous mode decreases with increase in the number of selfish nodes.

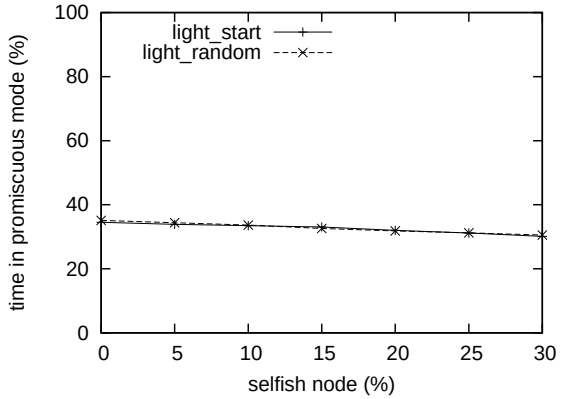



Figure 3.6: Time spent in the promiscuous mode

### 3.4 Chapter summary

In this chapter, we have presented two distributed detection-based solutions for the selfish-node problem. In both solutions, we used a voting system to confirm a node's selfishness to avoid the necessity for a centralized service. It can also solve the inconsistent evaluation problem. We emphasized fairness to provide same chance for each node to serve and be served by others. The first solution uses credit as a metric for evaluating a node's contribution to the network. Such a metric is different from the virtual money and reputation used previously. It can greatly mitigate the *location privilege* problem not addressed in other proposals. Our second solution is a light-weight solution, requiring neighbor monitoring only when necessary. It can identify almost all selfish nodes with very few mistakes and reduces the time spent in the promiscuous mode to save battery. Simulation results show our solutions are effective. They raise the packet delivery rate efficiently, cause few false accusations and have low overhead.

## Chapter 4

## Improving the Efficiency of Truthful Routing

### 4.1 Introduction

An effective way to motivate nodes' cooperation is to pay them for forwarding packets. Obviously, selfish nodes want to get paid as much as possible, while the nodes utilizing packet forwarding want to pay as little as possible. To achieve an equilibrium, it is desirable to reimburse nodes according to their cost. However, nodes incurs different cost for forwarding a packet since they may use different level of power and have different costs per unit of power. On the other hand, to maximize their utility, selfish nodes may not expose their true cost. Thus, truthful protocols (strategies) are needed to prevent cheating.

Controlling message overhead in the context of truthful routing is critical. First, the cost of overhead in terms of control messages incurred by such truthful protocols must also be reimbursed. A rational sender may have no reason to send packets at a very high cost, and thus truthful routing may lose its significance. Second, high overhead often results in low network performance. Usually, the less the overhead, the better the network performance. Anderegg and Eidenbenz [33] recently proposed a truthful routing protocol, named ad hoc-VCG, for mobile ad hoc networks with selfish nodes. The complexity of the routing overhead of their protocol is $O(n^3)$, where $n$ is the number of nodes in the network. Such an overhead may be prohibitively large as the network size grows.

We introduced two low overhead truthful routing protocols. The first protocol, *LOTTO* [32] - a low overhead truthful routing protocol can find the least cost path for data transmissions with an overhead of $O(n^2)$, a significant improvement over ad hoc-VCG. The second protocol, *LSTOP* [34, 35] – a light-weight scalable truthful routing protocol incurs an even lower overhead of $O(n)$ on average, and $O(n^2)$ in the worst case.

The remainder of this chapter is organized as follows: Section 4.2 provides background on mechanism design and system models. Section 4.3 presents details of the LOTTO protocol, including protocol description and truthfulness analysis, and a simulation study. Section 4.4 presents details of LSTOP and simulation study. Section 4.5 summarizes this chapter.

## 4.2 System Model and Preliminaries

### 4.2.1 The VCG Mechanism

Mechanism design [67], a subfield of game theory, studies how to design a system so that its behavior results in the desired system goal, under the assumption that agents are selfish and rational. It is traditionally used in market settings and now is being introduced into networking for resource and task allocations.

One of the important results in mechanism design is the VCG mechanism [62], named after Vickrey [68], Clarke [69] and Grove [70]. The most salient feature of the VCG mechanism is that it can maximize the total welfare of the system and achieve truthfulness [62]. Nisan and Renon [62] illustrated the application of the VCG mechanism in networking as follows. They describe a network as a biconnected graph. Each edge $e$ of the graph is an agent and has a cost $T_i$ of sending a single message along this edge. The mechanism design goal is to find a shortest path $sp$ between two given nodes $s$ and $d$. Then the following mechanism is truthful: $e$ gets no payment if it is not on the shortest path. Otherwise, it gets the payment $P_e = T_{sp}^{-e} - T_{sp}^{e=0}$ [62]. Where $C_{sp}^{-e}$ is the cost (length) of the shortest path not containing $e$, and $T_{sp}^{e=0}$ is the cost of the shortest path assuming zero cost of $e$. Feigenbaum et al. [71] showed that this claim holds true with nodes as agents.

### 4.2.2 The System Model

We consider a mobile ad hoc network as a directed weighted graph $G = (V, E, W)$. $V$ is the set of nodes, or agents in the mechanism design. $E$ is the set of links between nodes. $W$ is the set of weights for each link, indicating the cost to forward a packet along that link. Each link may have a different weight. A node (agent) has a different cost to forward a packet to different neighbors. The network is biconnected, i.e., the graph is still connected on removing any node and its incident links.

A node $v_j$ within radio range of node $v_i$ is represented as the link $(v_i, v_j)$. Power consumption is used as the cost metric. The weight $w_{ij}$ of the link $(v_i, v_j)$ is decided as the product of $v_i$'s emitting power $P_i^{emit}$ and its cost of unit power $c_i$, i.e., $w_{ij} = P_i^{emit} \cdot c_i$. Control messages are sent with maximum emitting power to reach more nodes. However, for data transmissions, and to save power, the sender sends a packet using the least power with which the receiver can receive the packet. Technically, a sender $i$ can choose its emitting power $P_i^{emit}$ and this power determines the radio range. From the wireless propagation model, the signal strength received by the receiver $j$ is $P_{i,j}^{rec} = \frac{K \cdot P_i^{emit}}{d^{\alpha}}$, where $K$ is a constant and $\alpha \in [1,6]$ is the distance-power gradient depending on the environment condition ($\alpha$ is 2

for free space and 4 for a two-way ground reflection model). If $P_{i,j}^{rec}$ exceeds a threshold $P_{min}^{rec}$, then $j$ can receive the data correctly. With the assumption of omnidirectional antenna, data will be overheard by all the nodes within the radio range of the sender. Therefore, if the transmitting power $P_i^{emit}$ is known, a receiver $j$ can estimate the minimum emitting power needed for the sender $i$ to forward data to it (node $j$) as $P_{i,j}^{min} = \frac{P_i^{emit} \cdot P_{min}^{rec}}{P_{i,j}^{rec}}$.

Virtual money is used to stimulate cooperation between nodes. A node can earn money by forwarding packets for others. A source is charged for the forwarding service. The payment is on a per packet basis. It should cover the cost for data transmissions and control messages, and includes some bonuses. The cost for data transmissions is the sum of links cost along the least cost paths from sources to destinations. It is important to note that the cost for route discoveries is high and cannot be ignored due to the large number of control messages and the high emitting power used. Nodes are assumed to be selfish and rational. They pursue maximization of their utility and thus may falsely declare their cost. However, they do not intend to destroy the network or attack other nodes.

In this chapter, we focus on effective truthful routing under the assumption of no collusion between nodes. Other issues such as the bootstrap of the virtual money, and securely crediting and transferring money are out of the scope of this chapter. We assume a payment mechanism [58] that takes care of accounting and transferring of payment between nodes. Tamper-proof hardware [42] can also be used to protect virtual money from modification or other attacks.

### 4.3 LOTTO - A Low Overhead Truthful Routing Protocol

LOTTO is a low-overhead truthful routing protocol which can find a truthful least cost path for data transmissions by applying the VCG mechanism. To date, the VCG mechanism is the only truthful mechanism for the least cost path problem [71]. LOTTO is a reactive routing protocol and is invoked when a new route is needed. It consists of two parts. The first part is route discovery and is discussed in Section 4.3.1. It intends to determine the topology of the network, which is essential for applying the VCG mechanism. Our route discovery greatly reduces the message overhead complexity from $O(n^3)$ [33] to $O(n^2)$. The second part consists of payment calculation and is discussed in Section 4.3.2. By using the VCG mechanism, the payments to nodes are set such that rational nodes have no incentive to cheat over their cost. Thus truthfulness is guaranteed. Sections 4.3.3 and 4.3.4 presents the truthfulness analysis and message complexity analysis, respectively.

### 4.3.1 Route Discovery

The VCG mechanism can guarantee that a rational node tells the truth about its cost. However, to apply the VCG mechanism, the source needs to find not only the least cost path $LCP$ as $< s, M^1, M^2, \cdots, M^i, d >$ to the destination, but also the least cost paths to the destination excluding node $M^1$, the least cost paths to the destination excluding node $M^2$, $\cdots$, the least cost paths to the destination excluding node $M^i$, respectively. Thus route discovery should provide the underlying directed weighted graph of the network to the source. LOTTO provides a method to collect this topological information effectively. It reduces the message overhead complexity from $O(n^3)$ [33] to $O(n^2)$.

Whenever a source $s$ wants to communicate with a destination $d$, it sends a $RREQ$ message. The $RREQ$ message is of the form $< s, d, seqNo, c_s, P_s^{emit} >$, where $s$ and $d$ are the node IDs of the source and destination, respectively. $SeqNo$ is the sequence number of this request. $C_s$ is the sender's cost per unit of power and $P_s^{emit}$ is the emitting power of the sender.

Upon receiving a $RREQ$ message from node $i$, a node $j$ takes the following actions:

1. Determine the power $P_{i,j}^{rec}$ at which $j$ received the packet.
2. Estimate the minimum power from node $i$ to node $j$ as $P_{i,j}^{min} = \frac{P_i^{emit} \cdot P_{min}^{rec}}{P_{i,j}^{rec}}$.
3. Append node $i$, $c_i$ and $P_{i,j}^{min}$ to its neighbor list. A neighbor list is a list of structures consisting of node ID, the minimum emitting power from the neighbors of node $j$ to node $j$, and cost per unit of power of these neighbors.
4. Check the freshness of the $RREQ$ message from its sequence number. If it's a new message, node $j$ appends its ID $j$, its cost per unit of power $c_j$ and its emitting power $P_j^{emit}$ to the $RREQ$ message and rebroadcasts it.[1]
5. Set a report timer ($RT$) for reporting a neighbor list after receiving the first $RREQ$ message for the session.

When $RT$ times out, nodes in the network send their neighbor lists to the source using a $Neighbor$ message. A $Neighbor$ message is of the form $< s, d, seqNo, i, (j, c_j, P_{j,i}^{min}), \cdots, (k, c_k, P_{k,i}^{min}) >$, where $s$ and $d$ are the IDs of the source and destination; $seqNo$ is the sequence number obtained from the $RREQ$; $i$ is the ID of this message sender; $j, \cdots, k$ are the IDs

[1] An alternative is that the $RREQ$ message makes no use of source routing, i.e., rather than appending its information in the $RREQ$ message, node $j$ replaces node $i$'s information to the $RREQ$ message with its own information (ID $j$, its cost per unit of power $c_j$ and its emitting power $P_j^{emit}$).

of $i$'s neighboring nodes; $c_j, \cdots, c_k$ are the cost per unit of power of neighbors $j, \cdots, k$, respectively; and $P_{j,i}^{min}, \cdots, P_{k,i}^{min}$ are the minimum power needed from neighbor $j, \cdots, k$ to node $i$, respectively.

The $Neighbor$ message can be sent through source routing, which is established as the reverse route to the source when a node receives a $RREQ$ message. However, a node which is not on the least cost path may selectively drop $Neighbor$ messages so that the source node gets incomplete topology information, thus increasing the probability that this node will lie on the selected least cost path. To prevent such an occurrence, each $Neighbor$ message is flooded over the network. Thus, with the assumption that the network is biconnected, a selective $Neighobr$ message dropped by a node will not thwart the $Neighbor$ messages from reaching the source through other routes. Nodes within the radio range of the source can unicast their $Neighbor$ messages directly to the source instead of flooding.

We assume that forwarding nodes will not modify $Neighbor$ messages since such modification action is a *malicious* behavior rather than a *selfish* behavior. However, if nodes exhibit malicious behavior, cryptography can be applied to detect such modifications. The neighbor list information in the $Neighbor$ messages can be encrypted using a symmetric key shared only between the sender and the source node. A symmetric key can be established a priori or with the help of a public cryptography system.

Upon receiving all the $Neighbor$ messages, the source knows all nodes' incident links and thus can construct the underlying directed weighted graph of the network. The source finds the least cost path using a Dijkstra or Bellmen-Ford algorithm [72] and calculates the payment to intermediate nodes, as discussed below in Section 4.4.3.2.

***Data Forwarding:*** After finding the least cost path, the source can send data packets along this path. The header of data packets includes the node list $< s, v_1, v_2, \cdots, v_k, d >$, the minimum power $P_i^{min}, i \in 1, k$ needed for every intermediate node $v_i$ to forward data to its downstream node, and payment $P_i$ to every intermediate node $v_i$. The payment to each intermediate node can also be accounted and reimbursed accordingly by a central server [58].

***Route Rediscovery:*** Link breakage occurs during data transmission due to node mobility or the shutting down of a node. Also a node's cost to forward packets may increase over time and thus the assigned payment is not feasible for that node. In such cases, the corresponding node sends a route error ($RERR$) message to the source. Upon receiving the $RERR$ message, the source initiates a new route discovery to find an alternate path to the destination.

### 4.3.2 Payment Calculation

All nodes are rewarded for their packets forwarding. To pay nodes accordingly, we need to know their cost. As stated in Section 4.2.1, the VCG mechanism can guarantee that nodes show their true cost. The following passage describes how the payment is calculated.

#### 4.3.2.1 Payment for Route Discovery

To encourage nodes' participation in a route discovery, a source reimburses all nodes involved in the route discovery. A unit amount is paid to each node to forward a routing message. This unit payment can cover the maximum cost of a node for a routing message. Credits to each node can be counted in a way similar to ad hoc-VCG [33].

#### 4.3.2.2 Payment for Data Forwarding

Data is forwarded along the least cost path. The payment to the nodes on the least cost path is calculated according to the VCG mechanism [62]. Nodes not on the least cost path get no payment for data transmissions. Specifically, we can suppose that the least cost path is $< s, v_1, v_2, \cdots, v_k, d >$, denoted as $LCP$, and the cost of $LCP$ is $C_{lcp}$. We denote the declared cost of a node $v_i$, $i \in \{1, \cdots, k\}$ as $T_i$. We denote the least cost path from $s$ to $d$ excluding node $v_i$ as $LCP^{-v_i}$ and the cost of this path as $C_{lcp}^{-v_i}$. Then, $P_{v_i}$, the payment to a node $v_i$, $i \in \{1, \cdots, k\}$ [62, 71], is calculated as

$$P_{v_i} = C_{lcp}^{-v_i} - C_{lcp} + T_i \tag{4.1}$$

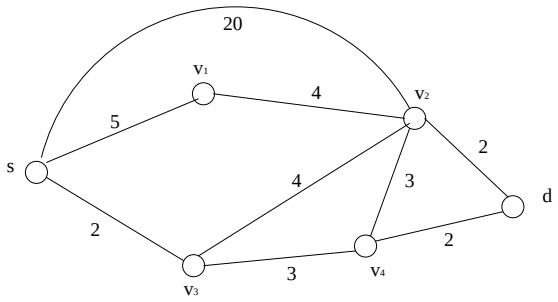



Figure 4.1: A weighted graph

Figure 4.1 shows an example of a weighted graph. In this graph the least cost path from $s$ to $d$ is $LCP =< s, v_3, v_4, d >$, and $C_{lcp} = 2 + 3 + 2 = 7$. The least cost path without $v_3$ is $LCP^{-v_3} =< s, v_1, v_2, d >$, and its cost is $C_{lcp}^{-v_3} = 5 + 4 + 2 = 11$, then the payment to $v_3$ is $P_{v_3} = C_{lcp}^{-v_3} - C_{lcp} + T_{v_3} = 11 - 7 + 3 = 7$. Similarly, the least cost path without $v_4$ is

$LCP^{-v_4} =< s, v_3, v_2, d >$, and its cost is $C_{lcp}^{-v_4} = 2 + 4 + 2 = 8$. Thus the payment to $v_4$ is $P_{v_4} = C_{lcp}^{-v_4} - C_{lcp} + T_{v_4} = 8 - 7 + 2 = 3$. Note that the total payment to the nodes on the least cost path is greater than the cost of that path since the intermediate nodes are awarded more than their own cost. These marginal utilities are what those nodes pursue.

An alternative is to distribute the overpayment evenly among all intermediate nodes along a path [33]. In this case, every node is reimbursed equal to its cost. Bonus (payment subtracted by cost) of all intermediate nodes on a path will be summed up and then be distributed evenly among the nodes.

### 4.3.3 Truthfulness Analysis

As described in Section 4.3.1, a link $l_{ij}$'s weight is determined by the sender $v_i$ and the receiver $v_j$. We now show that telling the truth about the cost (emitting power, cost per unit of power and minimum estimated power) is the dominant strategy for every rational node [33].

First we show that the sender $v_i$ has no incentive to cheat over cost.

***Case of Under-declaration*** A sender $v_i$ may under-declare its cost $T_i$ by under-declaring its emitting power $P_i^{emit}$ or its cost per unit of power $c_i$ in an attempt to get extra utility (margin). There are two possibilities. In the first case, node $v_i$ is not on the least cost path initially. After under-declaration of its cost, if $v_i$ is still not on the least cost path, then it gets no payment for data forwarding. On the other hand, $v_i$ may be selected to be on the least cost path. We denote the actual least cost path before $v_i$'s under-declaration of its cost as $LCP_t$ and its cost as $C_{lcp}^t$. We denote the least cost path after under-declaration as $LCP$ and its cost as $C_{lcp}$. It is important to note that $LCP_t$ will be the least cost path after excluding $v_i$ (on $LCP$) from the network, i.e., $C_{lcp}^{-v_i} = C_{lcp}^t$. We denote the actual cost of $v_i$ as $T_i^t$ (its declared cost is $T_i$). Then $\sum_{j \in LCP \wedge j \neq i} T_j + T_i^t > C_{lcp}^t = C_{lcp}^{-v_i}$ and thus $C_{lcp}^{-v_i} - \sum_{j \in LCP \wedge j \neq i} T_j < T_i^t$. From Equation 4.1, we have

$$\begin{aligned} P_{v_i} &= C_{lcp}^{-v_i} - C_{lcp} + T_i = C_{lcp}^{-v_i} - \sum_{j \in LCP} T_j + T_i \\ &= C_{lcp}^{-v_i} - \sum_{j \in LCP \wedge j \neq i} T_j. \end{aligned} \tag{4.2}$$

Thus $P_{v_i} < T_i^t$, i.e., it implies that $v_i$'s payment cannot cover its cost and thus $v_i$ gets negative utility. In the second case, $v_i$ is already on the least cost path initially. In this case items in Equation 4.2 do not change even if $v_i$ under-declares its cost. Thus $v_i$ gets the

same payment as when it tells the truth. Therefore, we conclude that $v_i$ has no incentive to under-declare its cost.

***Case of Over-declaration*** Alternatively, $v_i$ may over-declare its cost by over-declaring its emitting power $P_i^{emit}$ or cost per unit of power $c_i$ with the hope of getting a higher reimbursement. The over-declaration of cost makes the path through $v_i$ more expensive. On one hand, this may cause the least cost path not go through $v_i$ and thus $v_i$ losses the chance to get the utility. On the other hand, if $v_i$ is still on the least cost path, then items in Equation 4.2 do not change due to $v_i$'s over-declaration and thus $v_i$'s payment will be still the same as the payment when it tells the truth. Therefore, $v_i$ has no incentive to over-declare its cost. In summary, $v_i$ has no incentive to cheat over its emitting power or cost per unit of power.

Similarly, a receiver $v_j$'s best strategy is to declare the true estimated minimum emitting power $P_{i,j}^{min}$. If $v_j$ under-declares $P_{i,j}^{min}$, it may let the least cost path pass through it. However, the sender $v_i$ will use a lower power to send packets and thus $v_j$ will be out of the radio range of $v_i$. $v_j$ can forward nothing and gets no payment. If $v_j$ over-declares $P_{i,j}^{min}$ and thus making the path through $v_j$ more expensive, the least cost path may not go through link $l_{ij}$. Even if link $l_{ij}$ is still on the least cost path (after the over-declaration), $v_j$'s over-declaration of cost increases the cost of the path $C_{lcp}$. From Equation 4.1, since $C_{lcp}^{-v_j}$ and $T_j$ do not change while $C_{lcp}$ is increased, $v_j$ will get less payment than it gets when it tells truth about $P_{i,j}^{min}$. Thus $v_j$ has no incentive to tell lies about its estimated minimum emitting power $P_{i,j}^{min}$.

Zhong et al. [73] discuss $v_j$'s strategy in case of $v_i$'s over-declaring $P_i^{emit}$. If $v_i$ over-declares $P_i^{emit}$ by $\alpha$ times and $v_j$ declares cost truthfully, then the estimated minimum power to cover link $l_{ij}$ will be $\alpha P_{i,j}^{min}$. Thus $l_{ij}$ may not be on the least cost path. If $v_j$ under-declares the cost by $\beta$ times ($1 < \beta \leq \alpha$), the estimated minimum power will be $\alpha P_{i,j}^{min}/\beta < \alpha P_{i,j}^{min}$, and thus $l_{ij}$ has more chance to be on the least cost path. Since $\alpha P_{i,j}^{min}/\beta > P_{i,j}^{min}$, $v_i$ can still reach $v_j$. Thus $v_j$ can get a better payoff than it could get by declaring the true minimum power. However, $v_i$ will not use this estimated minimum power $\alpha P_{i,j}^{min}/\beta$ to send packets to $v_j$. Instead, it will factor out its over-declared ratio $\alpha$ and use $P_{i,j}^{min}/\beta < P_{i,j}^{min}$ to send the packet. Without this, $v_i$'s over-declaration does not have a point. Thus $v_i$ can not reach $v_j$. $V_j$ gets no payoff, even if $l_{ij}$ is still on the least cost path after the over-declaration of $P_i^{emit}$. Thus $v_j$'s best strategy is still to declare the true estimated minimum power $P_{i,j}^{min}$.

Base on their argument, Zhong et al. provided a protocol to prevent a node from under-

declaring $P_{i,j}^{min}$ by combining the VCG mechanism with cryptography. Their protocol results in the $O(\rho \cdot E \cdot n)$ message overhead, where $\rho$ is the number of power levels, $E$ the number of links and $n$ the number of nodes. LOTTO's route discovery can be used in their protocol and it reduces their message overhead from $O(\rho \cdot E \cdot n)$ to $O(n^2)$.

### 4.3.4 Message Complexity Analysis

In LOTTO, a route discovery consists of two stages. In the first stage, the *RREQ* message from the source is flooded over the network. Each node forwards the *RREQ* message once, resulting in a total of $n$ messages, where $n$ is the number of nodes. In the second stage, each node issues a *Neighbor* message. If a node is within the radio range of the source, it can directly send its *Neighbor* message to the source. Otherwise, it floods the *Neighbor* message over the network to prevent being selectively dropped by certain nodes. Thus, in the worst case, a node's *Neighbor* message will be forwarded once by each node, resulting in an $O(n)$ message overhead. In the second stage, $n$ nodes' *Neighbor* messages result in $O(n^2)$ message overhead. To sum up these two stages, a single route discovery incurs $n + O(n^2) = O(n^2)$ message overhead.

In ad hoc-VCG, a route discovery also consists of two stages. In the first stage, the *RREQ* message is flooded over the network. However, unlike LOTTO, where a node forwards only one *RREQ* for a route discovery, in ad hoc-VCG, a node forwards multiple *RREQ* messages for a route discovery. A *RREQ* message is initiated by the source and includes information ( weight) about all links it has traversed. Upon receiving a *RREQ* message from its neighbor node $B$, a node $A$ checks if this *RREQ* message contains *any* link that it ($A$) has not known about. If so, node $A$ forwards (broadcasts) this *RREQ* message by appending its identification, emitting power and cost of unit power. A node $A$ may even forward (broadcasts) multiple *RREQ* messages from the same neighbor $B$ as long as they contain new links unknown to node $A$. In a network, the possible number of links is $O(n^2)$, where $n$ is the number of nodes. Thus, each node in ad hoc-VCG (except the source and the destination) may need to forward (broadcast) $O(n^2)$ *RREQ* messages containing at least one new link weight, resulting in a total $O(n^3)$ message overhead for $n$ nodes. In the second stage, the destination sends an *RREP* message to the source after it collects topology information and calculates a least cost path. This *RREP* message will traverse at most $n$ hops, resulting in at most $n$ message overhead. To sum up, a route discovery in ad hoc-VCG incurs an $O(n^3)$ message overhead.

### 4.3.5 Performance Evaluation

We conducted an extensive simulation study to evaluate the performance of our protocol on a network with selfish nodes. We compared the proposed protocol with the ad hoc-VCG protocol [33], since both the protocols give incentives to nodes to forward packets under the assumption that nodes are selfish and rational, and both achieve the design objective of truthfulness of the selected routes. We did not compare our protocol with generic routing protocols such as DSR and AODV, because our protocol works under a different model of node behavior. The generic routing protocols are feasible in networks where all nodes are obedient (i.e. they follow the protocol), and are willing to cooperate. Our protocol, on the other hand, is aimed at networks where nodes are selfish and need some incentives to forward packets.

In our simulation study, we consider the following performance metrics:

• Packet delivery ratio, as defined in Section 3.2.7.1. The packet delivery ratio is a comprehensive result of all factors which could affect the packet delivery ratio, such as collisions in the MAC layer, rerouting in the network layer, and so on.

• Overhead is the sum total of the protocol control messages exchanged by nodes in the network, including routing messages as *RREQ*, *RREP*, *Neighbor* and *RERR*.

• Delay is defined as $\frac{\sum_{i=1}^{n} packet_{delay}^{i}}{n}$, i.e., the average end-to-end delay of data packets from senders to receivers.

• Overpayment ratio is defined as $\frac{\sum_{i=1}^{n} Pay^{i} + Cost^{s}}{\sum_{i=1}^{n} Cost^{i}}$, i.e., the ratio of total payment to the intermediate nodes paid by all the source nodes plus the cost of the sources to total cost incurred by all nodes for data transmission.

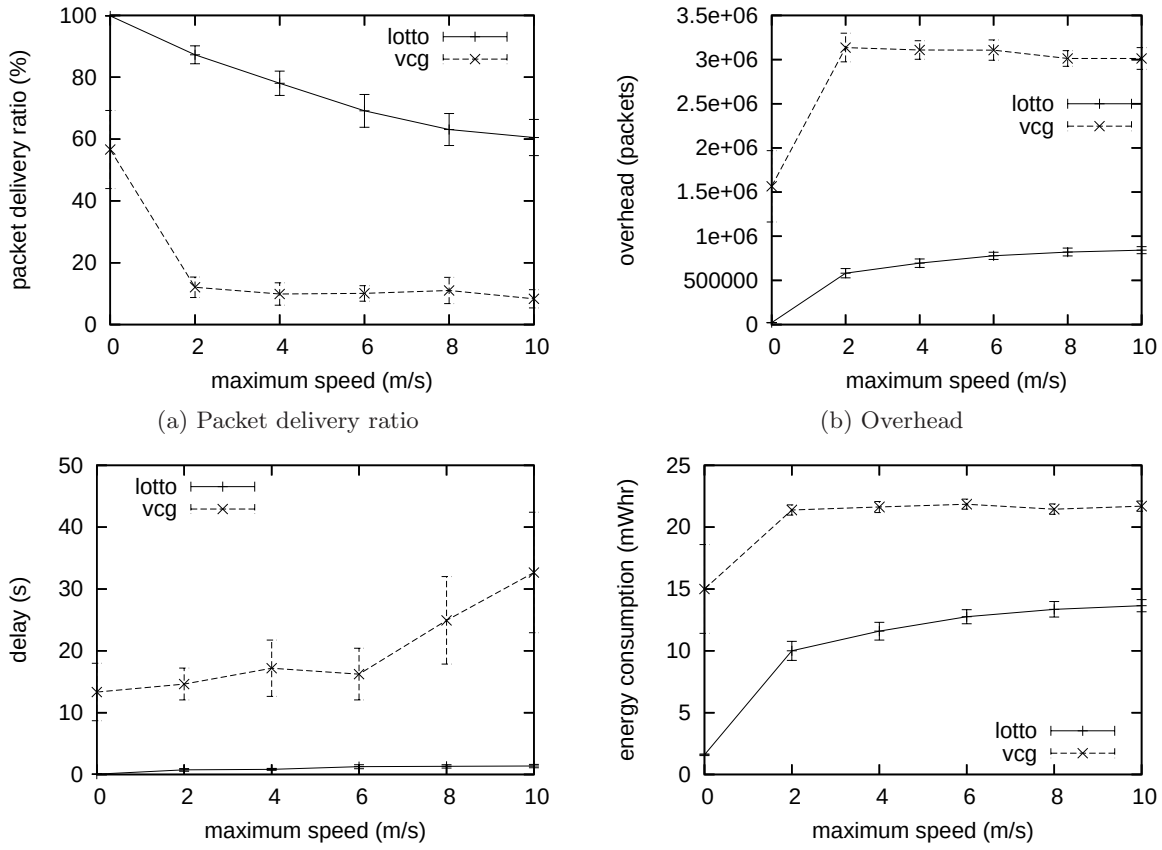
• Energy consumption. This is the total energy (power) consumed to send all packets, including route discoveries and data transmissions.

In the following discussion, *vcg* stands for the ad hoc-VCG protocol [33] and *lotto* stands for the proposed protocol LOTTO.

We used Glomosim [64] for simulations. Unless specified otherwise, the following parameters were used in simulations. Sixty nodes were placed uniformly in a square area of 600 meters by 600 meters. Since a wireless card usually has only a few discrete power level settings instead of a continuum power level, we simulated four levels of power emission for nodes, i.e., 1, 3, 5, 7 dBm, corresponding to the radio ranges of 125, 158, 198 and 250 meters, respectively. The routing messages were always sent with the highest power level. For simplicity, we assumed that nodes have the same cost of unit power. We used an 802.11 protocol with DCF as the MAC protocol and modified it to send packets at different power

levels. Default values are used for MAC layer parameters. All nodes followed the Random Waypoint mobility model [65] with a speed range of 0 m/s to 10 m/s and a pause time of 30 seconds. Each simulation lasted for 900 seconds of simulated time. 10 CBR flows were simulated, starting at 100 seconds and ending at 880 seconds. Each CBR flow sent four 256-byte data packets per second. Data points represented in the graphs were averaged over 10 simulation runs, each with a different seed. Confidence intervals (95%) are shown with vertical bars in the graphs.

#### 4.3.5.1 Impact of Mobility




(a) Packet delivery ratio

(b) Overhead

(c) End-to-end delay

(d) Energy consumption

Figure 4.2: Performance of different speeds for 60 nodes

**4.3.5.1.1 Packet Delivery Ratio** Fig. 4.2(a) shows the packet delivery ratio with respect to maximum speed. We observe that *lotto* provides a much higher packet delivery ratio over *vcg*. This can be ascribed to three reasons, all resulting from *vcg*'s $O(n^3)$ message overhead.

First, a large number of *RREQ*s cause heavy message propagation congestion in the MAC layer and result in a high delay of message propagation. Results of our simulation study show that, in a static network, a destination node needs several seconds to collect information about *most* of the links, and even more time to collect information about *all* links. In a dynamic network, it is impossible to collect all link information as the topology keeps changing during the *RREQ* flooding period. In order to guarantee that the selected path is of least cost and truthful, comprehensive topological information is needed. After collecting link information, the destination sends an *RREP* message to the source, including the least cost path and related information. However, the propagation of this *RREP* message is time consuming due to heavy congestion in the MAC layer, and in our simulations we found that it may take a several seconds to reach the source node. Therefore, the whole route discovery can take long time. In a dynamic network, after the source node receives the *RREP*, the selected route (the least cost path) may be outdated. The higher the mobility of nodes, the higher the probability that the network topology changes, and the higher the probability that the selected route is outdated. Thus data transmissions along the selected route may encounter link breakages, resulting in increased route discoveries, which generate even more *RREQ* messages and make the situation worse.

The second reason is that *vcg*'s large number of *RREQ* messages cause severe overflows of the network output queues and thus a lot of packets are dropped, including *RREP* packets. The dropping of *RREP* messages results in increased route discoveries, which generates even more *RREQ* and increase the problem. Table 4.1 shows the packets dropped due to overflow of output queues. The number of packets dropped due to queue overflow in *vcg* is about two magnitudes of order higher than that in *lotto*.

| speed | 0 | 2 | 4 | 6 | 8 | 10 |
|---|---|---|---|---|---|---|
| lotto | 0 | 2810 | 4786 | 6682 | 8078 | 8474 |
| vcg | 467431 | 1476108 | 1423999 | 1408228 | 1346671 | 1328511 |

Table 4.1: The number of packets dropped due to output queue overflow

The third reason is that *vcg*'s large number of *RREQ* messages cause severe radio interference [74], resulting in data packet being dropped. Nodes outside the radio range but within the interference range of a communicating node pair may interfere with that communication and result in packets drops. The interference range is larger than the radio range, and can be twice that of the radio range. The data packet drops result in increased route discoveries, which generate even more *RREQ* messages and worsen the situation.

Even in a static network, many packets are dropped and multiple route discoveries are invoked during a single data session.

As mobility increases, packet delivery ratio decreases for both protocols. Higher mobility causes more link breakages and thus results in more packets dropped. Link breakages result in increased route discoveries, which generate even more *RREQ* messages and thus make the situation worse.

**4.3.5.1.2 Overhead** Fig. 4.2(b) shows the overhead with respect to maximum speed. We observe that *lotto*'s overhead is much lower than that of *vcg*. As discussed in Section 4.3.4, *vcg* results in a total $O(n^3)$ overhead for a route discovery while *lotto* results in a total overhead of $O(n^2)$.

**4.3.5.1.3 End-to-end Delay** Fig. 4.2(c) shows the end-to-end delay with respect to maximum speed. We observe that *lotto*'s end-to-end delay is significantly lower than that of *vcg*. This is due to the reasons described in Section 4.3.5.1.1. First, *vcg* takes a long time to discover a route. Second, in *vcg*, route discoveries may fail due to *RREP* messages dropped resulting from the output queue overflows. Route discoveries may also fail due to the late arrival of *RREP* messages, i.e., when *RREP* messages reach the source nodes, the waiting timer for the route discoveries might have already timed out, and the source nodes might have already invoked the route discoveries again. Thus, source nodes need more time to find paths to the destination nodes. Third, in *vcg*, data packets have to wait in the output queue for a long time before being transmitted over the channel. As mobility increases, the end-to-end delay increases for both protocols. Higher mobility causes more link breakages and thus results in even more *RERR*s, which slows down route discoveries and causes even longer output queues in the MAC layer.

**4.3.5.1.4 Energy Consumption** Fig. 4.2(d) shows the energy consumed in sending packets with respect to maximum speed. We observe that *lotto* consumes much less energy than *vcg* because it sends much less control packets than *vcg*. As mobility increases, the energy consumption increases for both the protocols because an increase in mobility results in more route discoveries, which causes transmission of more control packets and thus consumes more energy.

As shown above, increased mobility degrades the network performance, mainly due to the increased routing overhead. Thus in case of higher mobility, a proactive routing protocol

may incur less overhead, and thus may perform better than a reactive routing protocol. This would be an interesting subject for future research.

#### 4.3.5.2 Impact of the Network Size

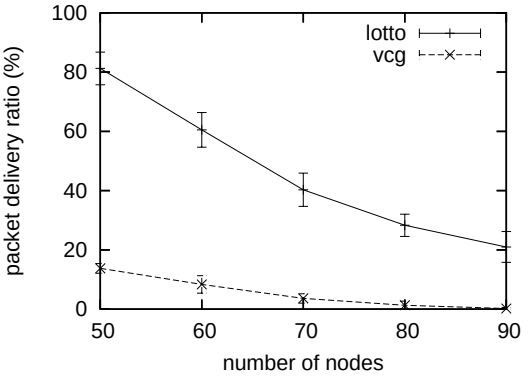
To study the impact of network size, we changed the number of nodes. Two cases were considered. In the first case, we fixed the node density at 6000 $m^2$/node and changed the number of nodes from 50 to 90. In the second case, we fixed the network area at 600m*600m and changed the number of nodes from 50 to 80. We set the maximum speed at 10 m/s in both the cases.

**4.3.5.2.1 The case of Fixed Node Density** Fig. 4.3(a) shows the packet delivery ratio with respect to the number of nodes, while keeping the density fixed. We observe that the packet delivery ratio of *lotto* is much higher than that of *vcg*. This is due to the reasons discussed in Section 4.3.5.1.1. As the number of nodes increases, the packet delivery ratio decreases for both protocols. With a fixed node density, an increase in the number of nodes results in an increase in network area and an increase in the average hop count of a path. Thus node mobility is more likely to cause link breakages, resulting in more packet drops and increased *RREQ* messages, particularly for *vcg*. These control packets weigh down the network traffic, cause severe output queue overflow, and increase interference for data transmission, both resulting in more packets being dropped. Also, the number of links increases with the number of nodes, and thus route discoveries in *vcg* need more time to collect link information. Thus the selected paths are more likely to be outdated, causing more link breakages, which results in increased route discoveries and a worsening situation.

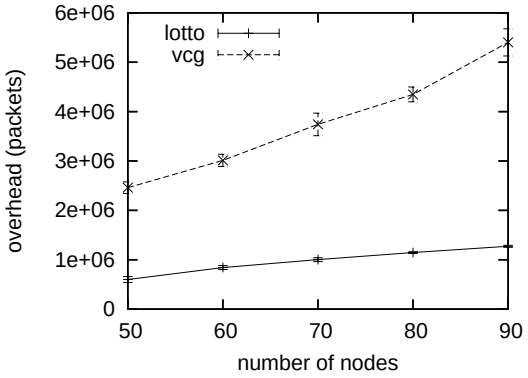
Fig. 4.3(b) shows the overhead with respect to the number of nodes keeping the density fixed. We observe that the overhead of *lotto* is much lower than that of *vcg*. This is because *vcg* generates $O(n^3)$ overhead while *lotto* generates an overhead of $O(n^2)$. As the number of nodes increases, the overhead increases for both the protocols for two reasons. First, an increase in the number of nodes results in an increase in the number of *RREQ* messages (and *Neighbor* messages for *lotto*). Second, an increase in the number of nodes results in more link breakages, as discussed above, and thus results in more route discoveries.

| nodes | 50 | 60 | 70 | 80 | 90 |
|---|---|---|---|---|---|
| lotto (pkts.) | 1070 | 1661 | 2332 | 3163 | 4038 |
| vcg (pkts.) | 11884 | 13328 | 19185 | 18932 | 21697 |

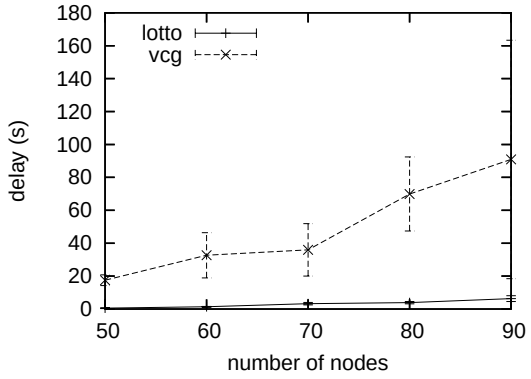
Table 4.2: Average overhead per route discovery with fixed node density



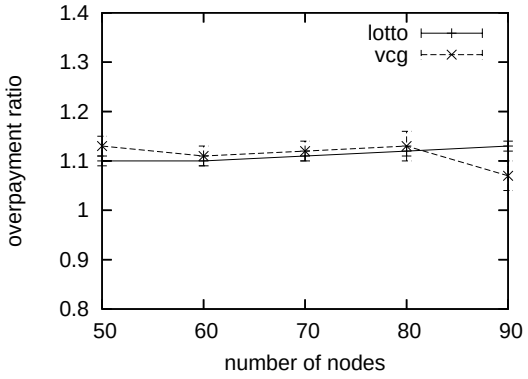

(a) Packet delivery ratio, fixed node density



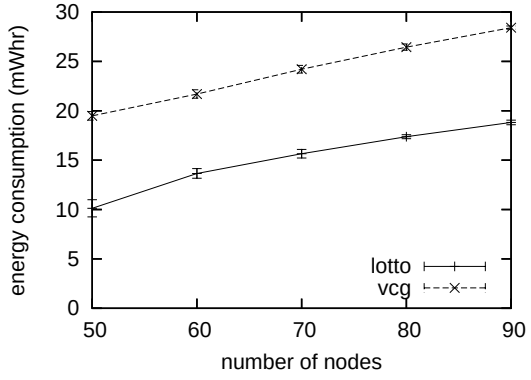

(b) Overhead, fixed node density



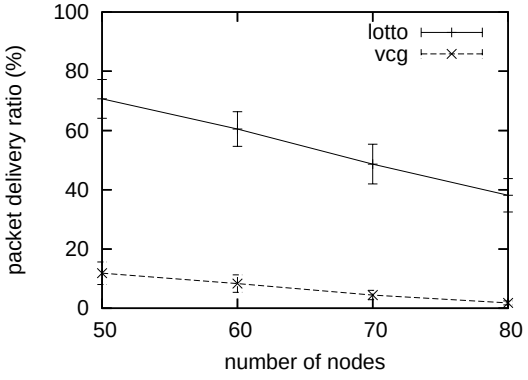

(c) Delay, fixed node density



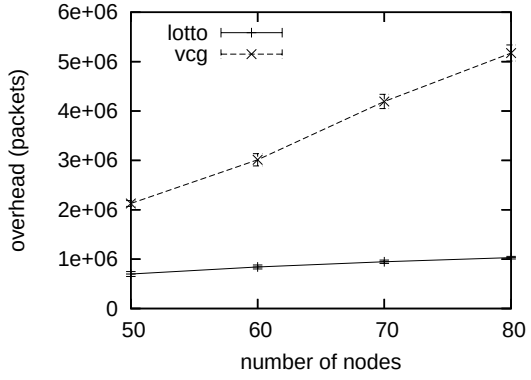

(d) Overpayment ratio, fixed node density



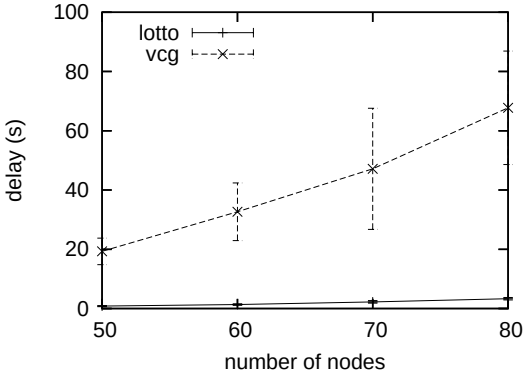

(e) Energy consumption, fixed node density




(f) Packet delivery ratio, fixed area




(g) Overhead, fixed area




(h) Delay, fixed area

Figure 4.3: Performance for different numbers of nodes

To further analyze the overhead, we normalize the overhead as control packets per route discovery. Table 4.2 shows the overhead per route discovery with different numbers of nodes. The overhead of *lotto* is almost 1 magnitude of order lower than that of *vcg*.

Fig. 4.3(c) shows the end-to-end delay with respect to the number of nodes keeping the density fixed. We observe that *lotto* incurs a much lower delay than *vcg*. This is due to the reasons discussed in Section 4.3.5.1.3. As the number of nodes increases, the delay increases for both the protocols. An increase in the number of nodes results in an increase in the overhead and thus results in more packets in the output queues which aggravate queue overflow. Longer output queues increase the waiting time of data packets. Queue overflow causes the failure of a route discovery. In addition, route discovery is slower as it needs more time to collect link information. Both result in data packets staying in the source node' buffers for a longer period of time.

Fig. 4.3(d) shows the overpayment ratio with respect to the number of nodes keeping the density fixed. The ratio is no more than 1.13 for *lotto*. *lotto*'s overpayment ratio is a little lower than that of *vcg*. It implies that *lotto* can find better paths than *vcg*. The main reason is that in *vcg*, with node mobility, it is difficult to collect complete link information to get the most effective routes and thus the most effective overpayment ratio. The ratio changes a little for both the protocols as the number of nodes changes, except at the point of 90 nodes, where *vcg* has a lower overpayment ratio than *lotto*. This is because in *vcg* a large portion of data packets delivered to the destinations are sent from the sources to the destinations directly, meaning that the sources and the destinations are neighbors.

Fig. 4.3(e) shows the energy consumed for sending packets with varying number of nodes keeping the density fixed. We observe that *lotto* consumes much less energy than *vcg* even though it delivers many more data packets because it generates much fewer control packets. As the number of nodes increases, more nodes are involved in sending packets, particularly for route discoveries, and the total energy consumption increases for both protocols.

**4.3.5.2.2 The Fixed Area Case** Fig. 4.3(f) shows the packet delivery radio with respect to the number of nodes keeping the network area fixed. We observe that *lotto* provides a much higher delivery ratio over *vcg*. This is due to the reasons discussed in Section 4.3.5.1.1. As the number of nodes increases, i.e., the node density increases, the packet delivery ratio decreases for both protocols. For *vcg*, as the number of nodes increases, the number of links increases, resulting in longer route discoveries. The probability that selected paths are outdated increases and thus link breakages increase, causing more packets

to be dropped. Furthermore, an increase in overhead due to route discoveries results in severe output queue overflow and radio interference with data transmission, causing more packet drops. *Lotto* has less output queue overflow and radio interference than *vcg*.

| nodes | 50 | 60 | 70 | 80 |
|---|---|---|---|---|
| lotto (pkts.) | 1176 | 1661 | 2219 | 2860 |
| vcg (pkts.) | 11317 | 13328 | 21287 | 27239 |

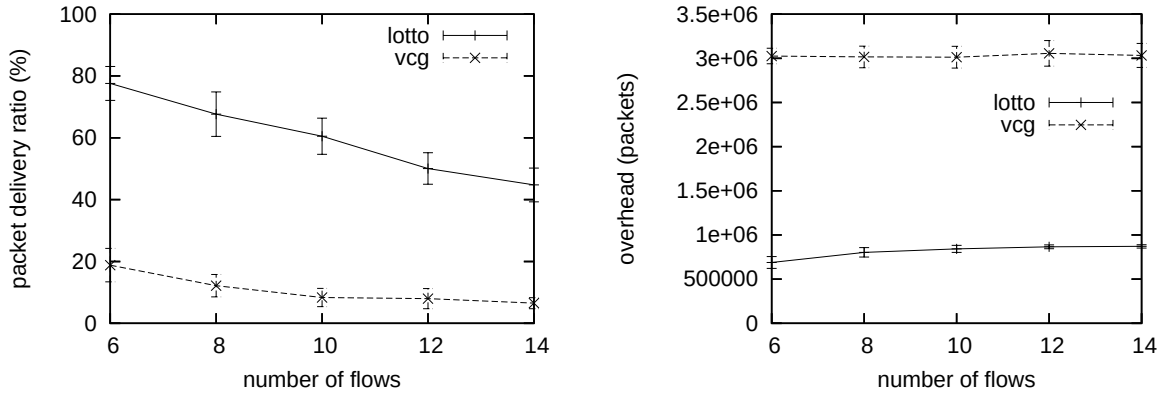
Table 4.3: Average overhead per route discovery with a fixed area

Fig. 4.3(g) shows the overhead with respect to the number of nodes keeping the area fixed. We observe that *lotto*'s overhead is much lower than that of *vcg* because *lotto* generates $O(n^2)$ $RREQ$ messages while *vcg* incurs $O(n^3)$ $RREQ$ messages. As the number of nodes increases, more nodes are involved in the route discoveries, thus increasing the overhead for both protocols. Table 4.3 shows the overhead per route discovery. *Vcg* incurs much higher overhead than *lotto*. Comparing Table 4.2 with Table 4.3, we can see that, for *lotto* the overhead per route discovery in the case of fixed density is almost the same as that in the case of fixed area. In *lotto*, every node sends only one $RREQ$ message and one $Neighbor$ message, which are flooded over the network, for a route discovery. However, for *vcg*, the overhead per route discovery in the case of fixed area increases faster than that in the case of fixed density. This is due to the fact that an increase in the number of links in a small area is faster than that in a larger area as the number of nodes increases.

Fig. 4.3(h) shows the end-to-end delay with respect to the number of nodes keeping the area fixed. We observe that *lotto* incurs a much lower delay than *vcg*. This is because of the reasons discussed in Section 4.3.5.1.3. As the number of nodes increases, the delay increases for both protocols. This is due to the same reason discussed in the case of fixed node density.

#### 4.3.5.3 Impact of Load

**4.3.5.3.1 Packet Delivery Ratio** Fig. 4.4(a) shows the packet delivery ratio with respect to the number of flows for 60 nodes and a maximum speed of 10 m/s. We observe that *lotto* provides a much higher packet delivery ratio than *vcg*. This is due to the reasons discussed in Section 4.3.5.1.1. As the load increases, the packet delivery ratio decreases for both protocols. This is because the output queue overflow, together with packets collisions and interference, increases when the number of flows increases, causing more packet drops.




(a) Packet delivery ratio

(b) Overhead

Figure 4.4: Performance for different loads with 60 nodes

**4.3.5.3.2 Overhead** Fig. 4.4(b) shows the overhead with respect to the number of flows for 60 nodes and a maximum speed of 10 m/s. We observe that *lotto* causes significantly lower overhead than *vcg* since *vcg* generates $O(n^3)$ *RREQ* messages for route discoveries while *lotto* incurs an only $O(n^2)$ overhead. In *lotto*, an increase in the number of flows causes an increase in route discoveries, resulting in an increase in the overhead. The overhead of *vcg* changes little with respect to the number of flows. This occurs for two reasons. First, with more flows, each route discovery collects fewer *RREQ* messages due to collisions and interference of these *RREQ* messages. Second, due to bounded simulation time, the long time of route discovery restricts the number of new route discoveries.

## 4.4 A Light-weight Scalable Truthful Routing Protocol

### 4.4.1 Overview

It is desirable to find a $truthful$ and optimal (*least-cost*) path between a source and a destination. By far, the VCG mechanism is the only solution[71]. The VCG mechanism requires multiple optimal routes and thus a complete underlying weighted graph should be determined. However, the least cost path established by the VCG mechanism is meant for data forwarding. The cost of network topology discovery is beyond the consideration of the VCG mechanism, and finding network topology is not trivial. It costs ad hoc-VCG [33] an $O(n^3)$ overhead. Moreover, finding the complete topology of a network is not practical. A network topology can be acquired by some routing protocols, in practice, which resort to a broadcast approach. Due to collision and radio interference, a broadcast message is not reliable. There is no guarantee that nodes within the radio range of a broadcaster

will receive the broadcast message. Thus some links in the network topology may remain undetected.

A near-optimal (near least-cost) path for data traffic with very low overhead may reduce total cost and may be more scalable. This is the stimulus for our LSTOP protocol. Compared to the ad hoc-VCG, LSTOP provides near-optimal paths for data traffic with significantly low overhead. It reduces the message overhead to $O(n^2)$ in the worst case and to $O(n)$ on the average. The basic idea of how LSTOP functions can be summarized as follows: the source sends a route request on demand. Every node forwards the route request once. Based on route requests, nodes construct neighbor lists. One route is selected by the destination as the base to construct a subgraph of $G^p$. Only the nodes around this selected route report their neighbor lists to the source. Using these neighbor lists, the source constructs a subgraph $G^p$ and finds the least cost path in $G^p$ using the Dijkstra algorithm. Then, the VCG mechanism is applied to calculate the payment to nodes so that a rational node has no incentive to lie about its cost.

### 4.4.2 Route Discovery

Route discovery intends to construct a subgraph $G^p$ of the whole network. The subgraph $G^p$ includes the source, destination and some intermediate nodes. To apply the VCG mechanism, this subgraph should be biconnected.

Whenever a source $s$ wants to communicate with a destination $d$, it initiates an $RREQ$ message. This message includes $< s, d, seqNo, c_s, P_s^{emit} >$, where $s$ and $d$ are the node IDs of the source and the destination, respectively; $seqNo$ is the sequence number of this request; $c_s$ is the sender's cost per unit of power and $P_s^{emit}$ is the emitting power of the sender.

Upon receiving a $RREQ$ message from a node $i$, a node $j$ takes the following actions:

1. Determine the power $P_j^{rec}$ at which $j$ received the message.
2. Estimate the minimum power from node $i$ to node $j$ as $P_{i,j}^{min} = \frac{P_i^{emit} \cdot P_{min}^{rec}}{P_{i,j}^{rec}}$.
3. Append node $i$, $c_i$ and $P_{i,j}^{min}$ to its neighbor list. A neighbor list is a list of structures consisting of node ID and the minimum emitting power from the neighbors of node $j$ to node $j$ and cost per unit of power of these neighbors.
4. Check the freshness of the $RREQ$ message by its sequence number. If it's a new message, node $j$ appends its ID $j$, its cost per unit of power $c_j$ and its emitting power $P_j^{emit}$ to the $RREQ$ message and rebroadcasts it.

After receiving the first $RREQ$, the destination waits for a specific period $\tau$ and then sends an $RREP$ to the source. We denote the path along which the first $RREQ$ arrived as $SP^{hop}$. The $RREP$ is sent along $SP^{hop}$ in the reverse direction and includes the node list of that path. The waiting period $\tau$ is used to let $RREQ$ messages reach as many nodes as possible so that nodes can get as complete neighbor lists as possible.

After forwarding $RREP$ to the source, nodes on the path $SP^{hop}$ report their neighbor lists to the source. Nodes work in the promiscuous mode and thus nodes not on the path $SP^{hop}$ may overhear the $RREP$ message. We denote the set of nodes on the path $SP^{hop}$ as $V_{sp}^{on}$ and the set of nodes not on path $SP^{hop}$ but within radio range of $SP^{hop}$ (i.e., within radio range of any node in $V_{sp}^{on}$) as $V_{sp}^{near}$. When nodes in $V_{sp}^{near}$ overhear $RREP$ messages, they send their neighbor lists to the source using $Neighbor$ messages. A $Neighbor$ message includes such fields as $< s, d, seqNo, i, (Neighbor_j, c_j, P_{j,i}^{min}), \cdots, (Neighbor_k, c_k, P_{k,i}^{min}) >$, where $s$ and $d$ are the IDs of the source and destination, respectively; $seqNo$ is the sequence number acquired from the $RREQ$; $i$ is the ID of this message sender; $Neighbor_j, \cdots, Neighbor_k$ are the IDs of $i$'s neighboring nodes; $c_j, \cdots, c_k$ are the cost per unit of power of neighbor $j, \cdots, k$, respectively; and $P_{j,i}^{min}, \cdots, P_{k,i}^{min}$ are the minimum power needed from neighbor $j, \cdots, k$ to node $i$, respectively. These $Neighbor$ messages use source routing. The routes to the source are established as the reverse routes of the routes along which nodes received the $RREQ$ messages. Nodes not overhearing $RREP$ messages don't send $Neighbor$ messages.

Upon receiving all these neighbor lists from nodes in $V_{sp}^{near}$ and $V_{sp}^{on}$, the source constructs a subgraph $G^p$ which includes the source, destination and all nodes within the radio range of nodes on path $SP^{hop}$. In a network with sufficient high density, this subgraph $G^p$ is biconnected. In cases where the acquired subgraph is not biconnected, the source node invokes a new route discovery. By using the Dijkstra or Bellmen-Ford algorithm [72], the source can find a least cost path in $G^p$, denoted as $SP^p$.

Figure 4.5 illustrates this route discovery. $s$ and $d$ denote the source and the destination, respectively; the thin solid line denotes the $SP^{hop}$; the thick dash_dot line denotes the $SP^p$; the area enclosed by the dotted line is the found subgraph $G^p$. Only nodes within this enclosed area send $Neighbor$ messages to the source node. With high probability, the least cost path is present in the area enclosed by the dotted line. Thus $SP^p$ may be a good approximation of the least-cost path for the whole network.

***Data Forwarding:*** After finding the least cost path, the source can send data packets along this path. The header of data packets includes the node list $< s, v_1, v_2, \cdots, v_k, d >$,

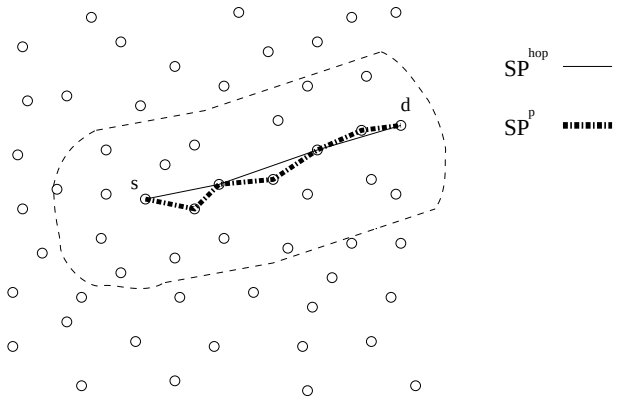



Figure 4.5: Illustration of LSTOP

the minimum power $P_i^{min}, i \in \{1, 2, \cdots, k\}$ needed for every intermediate node $v_i$ to forward data to its downstream node, and payment $P_i$ to every intermediate node $v_i$. The payment to each intermediate node can also be accounted for and reimbursed accordingly by a central server [58]. Tamper-proof hardware [42] can be used to protect virtual money or payment from modification and other attacks.

Upon link breakage due to mobility or cost change, the corresponding node sends a route error ($RERR$) message to the source. The source then invokes a new route discovery.

***Message Complexity:*** In LSTOP, route discovery consists of two stages. In the first stage, $RREQ$ floods over the network. Every node forwards one $RREQ$ message. Thus a total of $n$ $RREQ$ messages are sent. In the second stage, nodes in $V_{sp}^{on}$ and $V_{sp}^{near}$ send one $Neighbor$ message each to the source using source routing. The total number of $Neighbor$ messages forwarded is $\sum_{i \in V_{sp}^{on}} hop_s^i + \sum_{i \in V_{sp}^{near}} hop_s^i$, where $hop_s^i$ denotes the hop counts from node $v_i$ to the source $s$. Typically, the diameter of a network is $\sqrt{n}$, i.e., the number of nodes on a path is $\sqrt{n}$. The number of neighbors of a node, denoted as $\mu$, is determined by the node density. Thus the neighboring nodes along the path $SP^p$ is $\mu\sqrt{n}$, where $\mu$ is typically $6 \sim 8$. As some nodes are common neighbors of two consecutive nodes on $SP^p$, i.e, there is a high probability that the neighbor sets of two nodes will intersect, the actual value of $\mu$ is smaller. Under the assumption that a typical network diameter is $\sqrt{n}$, a total of $O(\mu\sqrt{n} \cdot \sqrt{n}) = O(n)$ $Neighbor$ messages are forwarded. In the worst case, every node in the network is within the radio range of at least one of the nodes on the path $SP^p$. In this case all nodes report their neighbor lists and thus the overhead could be $O(n^2)$ in the worst case and $O(n^{3/2})$ on average. However, in such a case, a least cost path is guaranteed to be found as the information on the whole graph is available to the source node. To summarize

the two stages, a single discovery incurs a message overhead of $O(n^2)$ in the worst case and $O(n)$ on average.

### 4.4.3 Payment Calculation

#### 4.4.3.1 Payment to Route Discovery

We pay a unit amount for each node to forward a routing message. This unit payment can cover the maximum cost of a node for routing a message. Payment for a route discovery consists of two parts. The first part pays nodes involved in flooding $RREQ$. Every node broadcasts $RREQ$ exactly once. The second part pays nodes involved in the $Neighbor$ message reporting. Every node that overhears the $RREP$ message along $SP^{hop}$ sends its neighbor list to the source using a $Neighbor$ message. The message may be relayed by several intermediate nodes. Since a $Neighbor$ message uses source routing, the source node can determine all the intermediate nodes that relayed the message by checking the header of the $Neighbor$ message. Upon receiving a $Neighbor$ message, the source increases the corresponding credits for the nodes forwarding the packet, and for the initiator of a $Neighbor$ message.

#### 4.4.3.2 Payment for Data Forwarding

Payment calculation for data forwarding is based on subgraph $G^p$ only. Nodes not in $G^p$ get no payment for data forwarding. Nodes not on the least cost path get no payment for data transmissions. Payment to the nodes on the least cost path is calculated according to the VCG mechanism, as discussed in Section 4.3.2.

### 4.4.4 Truthfulness Analysis

We show that telling the truth about its cost is a node's dominant strategy. Path selection is determined in two stages. In the first stage, $SP^{hop}$ is chosen. $SP^{hop}$ is the path along which the first $RREQ$ arrives at the destination. It is not dependent on the declared emitting power or cost per unit of power of any node. Thus a node's under-declaration or over-declaration of its cost makes no sense at this stage. In the second stage, $SP^p$ is found. After $SP^{hop}$ is found, the subgraph $G^p$ is constructed based on neighbor list information of nodes along and around $SP^{hop}$. The payment to nodes in the subgraph $G^p$ is determined by applying the VCG mechanism. Only links in the subgraph are used. A link $l_{ij}$'s cost is determined by the sender $i$ and the receiver $j$. As shown in Section 4.3.3, nodes in the subgraph have no incentive to lie about their cost.

### 4.4.5 Performance Evaluation

We conducted an extensive simulation study to evaluate the performance of our protocol on a network with selfish nodes. We compared the proposed protocol with the ad hoc-VCG protocol [33]. In the following discussion, *vcg* stands for the ad hoc-VCG protocol [33] while *lstop* stands for the proposed protocol LSTOP. The same parameters used for simulating LOTTO were used in the simulations.

In our simulation study, we evaluate the same metrics that we evaluated for *LOTTO*, i.e., packet delivery ratio, overhead, end-to-end delay, overpayment, and energy consumption. In addition, we evaluate a new metric, approximation ratio, defined as follows,

• The approximation ratio of cost is the ratio of total cost of nodes based on *lstop* to that based on the whole graph (which gives the optimal solution, i.e., least cost paths). Similarly, the approximation ratio of payment is the ratio of total payment plus cost of source nodes based on *lstop* to that based on the whole graph. Approximation ratio shows how well *lstop* approaches the optimal solution.

#### 4.4.5.1 Approximation Ratio

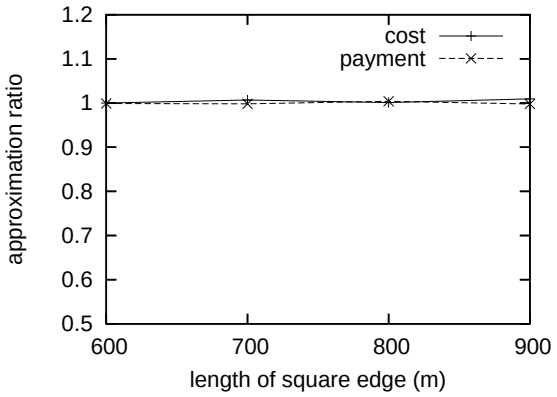
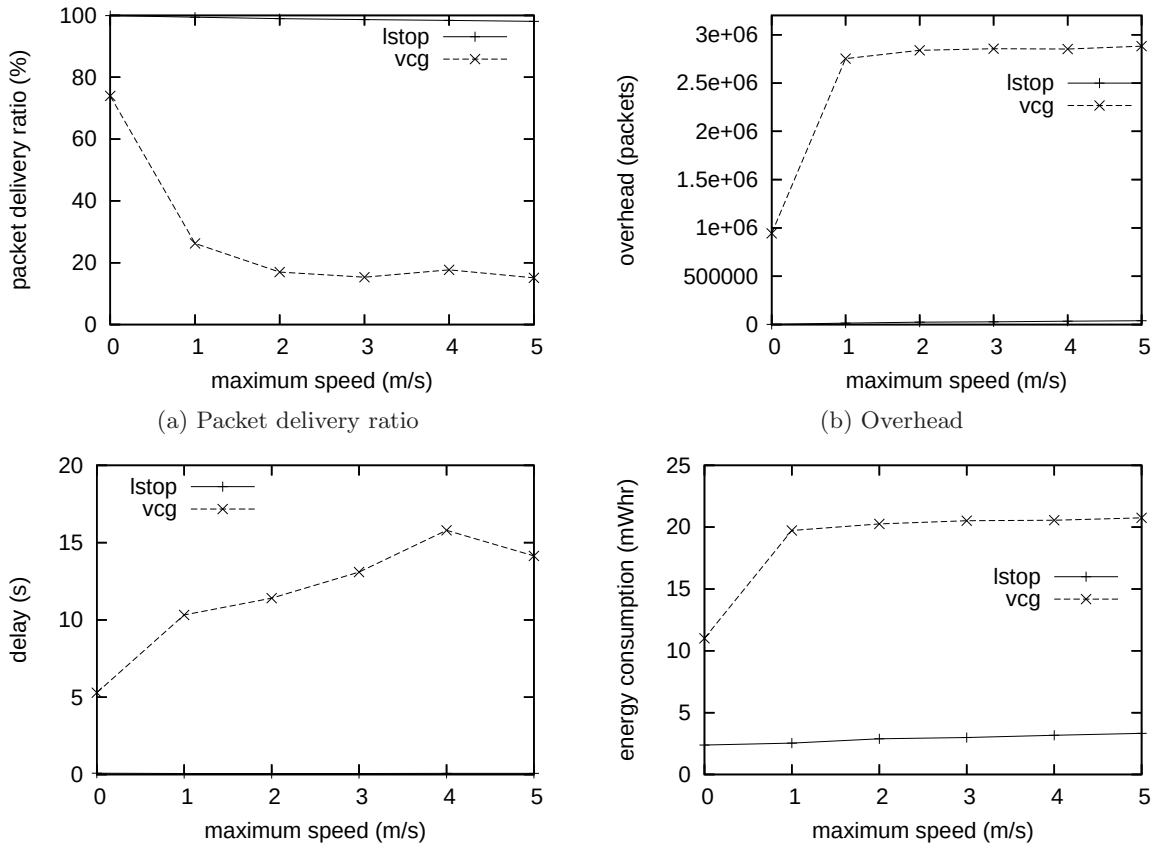



Figure 4.6: Approximation ratio

To study the approximation ratio of *lstop*, we collected the entire topology. Based on this, we applied the VCG mechanism to calculate the total cost on the least cost paths and total payment to all intermediate nodes for data traffic. As the paths found based on the whole graph and those based on the sub graph may be different, mobility may cause different link breakages and thus results in different route discoveries. To allow comparison, we set the network to be static.

Figure 4.6 shows the approximation ratio of cost and payment with respect to node density. Both the approximation ratio of cost and the approximation ratio of payment are almost equal to 1. As the network area increases, i.e., node density decreases, the approximation ratio changes very little. The highest approximation ratio for either cost or payment is less than 1.007. It shows that selecting a path using *lstop* approaches the optimal solution of selecting a path based on the whole topology of a network with sufficient high density.

#### 4.4.5.2 Impact of Mobility




(a) Packet delivery ratio

(b) Overhead

(c) End-to-end delay

(d) Energy consumption

Figure 4.7: Performance at different speeds for 60 nodes

**4.4.5.2.1 Packet Delivery Ratio** Fig. 4.7(a) shows the packet delivery ratio with respect to maximum speed. We observe that *lstop* provides much higher packet delivery ratio than *vcg*. As mobility increases, the packet delivery ratio of *lstop* drops slightly. The mobility increases link breakages and thus results in more packet drops. When a link breaks,

*lstop* can find new paths to the destinations quickly with low overhead (See subsection 4.4.2). On the contrary, *vcg*'s packet delivery ratio drops dramatically with an increase in mobility. This can be ascribed to three reasons, as discussed in Section 4.3.5.1. First, *vcg*'s large number of *RREQ*s cause much delay in message propagation and the source is very likely to collect outdated route information. Second, large numbers of *RREQ* messages cause severe overflow of the network output queues and thus a lot of packets are dropped. Third, a large number of *RREQ* messages causes severe radio interference [75], resulting in data packets drops.

**4.4.5.2.2 Overhead** Fig. 4.7(b) shows the overhead with respect to maximum speed. We observe that *lstop*'s overhead is significantly (by 2 magnitudes of order) lower than that of *vcg*. In *vcg*, each node forwards (broadcasts) every *RREQ* message carrying link information unknown to it and thus a node forwards $O(n^2)$ *RREQ* messages, resulting in a total of $O(n^3)$ overhead for a route discovery. On the contrary, in *lstop*, every node sends only one *RREQ* message and one *Neighbor* message, which may be forwarded for several hops, for a route discovery, resulting in a total overhead of $O(n^2)$ in the worst case and $O(n)$ on average. As mobility increases, the overhead increases in both protocols because higher mobility results in more link breakages, and thus results in more *RERR*s and new route discoveries.

**4.4.5.2.3 End-to-end Delay** Fig. 4.7(c) shows end-to-end delay with respect to maximum speed. *lstop*'s end-to-end delay is between 0.06∼0.07 second and is on the horizontal axis. We observe that *lstop*'s end-to-end delay is significantly (by 2 magnitudes of order) lower than that of *vcg*. This is due to the reasons discussed in Section 4.3.5.1. First, *vcg* takes a long time to discover a route. Second, multiple route discoveries for a single route request increase the delay. Third, congestion and long queues in the MAC layer increase the delay.

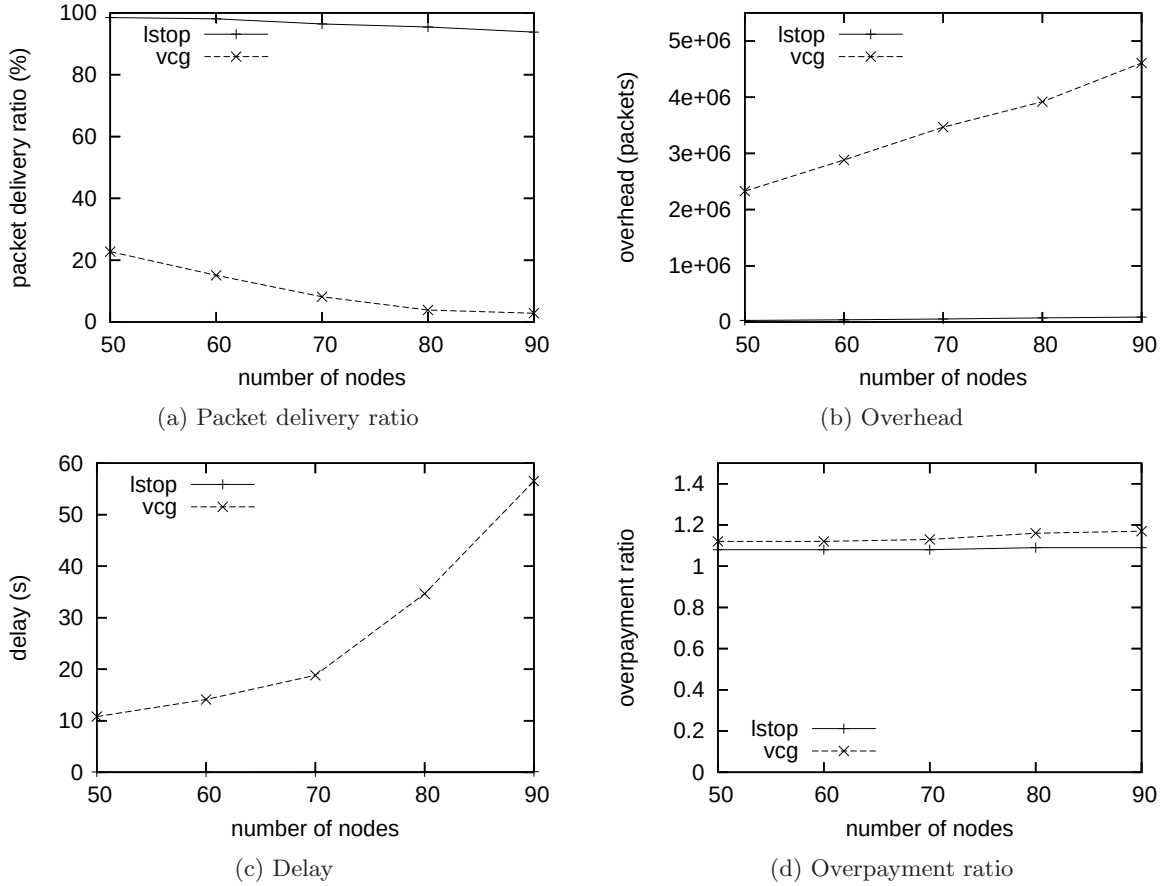
As the mobility increases, the end-to-end delay increases for *vcg*. Higher mobility causes more link breakages and thus results in new route discoveries, producing more *RREQ* messages. This in turn slows down route discoveries and causes even longer output queues in MAC layer.

**4.4.5.2.4 Energy Consumption** Fig. 4.7(d) shows the energy consumed in sending packets with respect to maximum speed. We observe that *lstop* consumes much less energy than *vcg* because it sends far less control packets than *vcg*. As mobility increases, the energy

consumption increases for both protocols because an increase in mobility results in more route discoveries, which causes transmission of more control packets and thus consumes more energy.

#### 4.4.5.3 Impact of the Network Size

These simulations were conducted with a density of one node per 6000 $m^2$ and a maximum speed of 5 m/s, and the number of nodes was changed from 50 to 90.




(a) Packet delivery ratio

(b) Overhead

(c) Delay

(d) Overpayment ratio

Figure 4.8: Performance for different numbers of nodes

**4.4.5.3.1 Packet Delivery Ratio** Fig. 4.8(a) shows the packet delivery ratio with respect to the number of nodes. We observed that the packet delivery ratio of *lstop* was significantly higher than that of *vcg*. This is due to the reasons discussed in Section 4.3.5.1(a). As the number of nodes increases, the network area increases and the average hop count of a path increases, and thus node mobility is more likely to cause link breakages. In addition, an increase in the number of nodes results in an increase in

the number of links, and therefore route discoveries in *vcg* need more time to collect link information. The selected paths are more likely to be outdated, causing more link breakages. An increase in link breakages increases the number of packets drops and route discoveries, which generates more control messages particularly for *vcg*, thus making the situation worse.

**4.4.5.3.2 Overhead** Fig. 4.8(b) shows the overhead with respect to the number of nodes. We observe that the overhead of *lstop* is far lower (by 2 magnitudes of order) than that of *vcg*. This is because *vcg* generates an $O(n^3)$ overhead while *lstop* generates an overhead of $O(n)$ on average. As the number of nodes increases, the overhead increases for both protocols for two reasons. First, an increase in the number of nodes results in an increase in the control messages of a route discovery. Second, an increase in the number of nodes results in more link breakages, as discussed above, and thus results in more route discoveries.

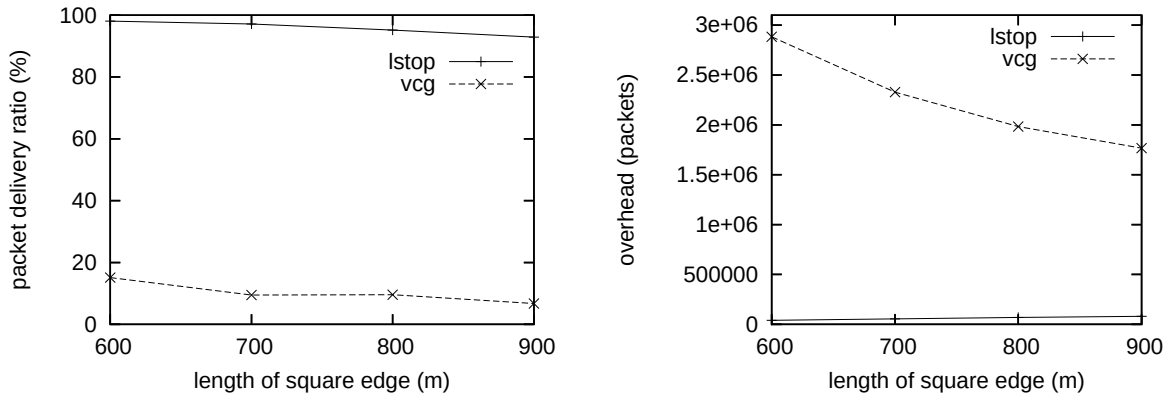
To further analyze the overhead, we normalized the overhead as control packets per route discovery, including $RREQ$ messages, $Neighbor$ messages and $RREP$ messages. Table 4.4 shows the overhead per route discovery with different numbers of nodes. The overhead of *lstop* is almost 2 orders of magnitude lower than that of *vcg*.

| nodes | 50 | 60 | 70 | 80 | 90 |
|---|---|---|---|---|---|
| lstop | 114 | 136 | 160 | 183 | 206 |
| vcg | 15743 | 18479 | 21676 | 23187 | 24513 |

Table 4.4: Average overhead per route discovery with fixed node density

**4.4.5.3.3 End-to-end Delay** Fig. 4.8(c) shows the end-to-end delay with respect to the number of nodes. The delay of *lstop* is around 0.1 second and is on the horizontal axis. We observe that *lstop* incurs a far shorter delay (by 2 order of magnitude) than *vcg*. This is further discussed in Section 4.3.5.1(c). As the number of nodes increases, the delay increases for *vcg*. An increase in the number of nodes increases the number of control messages, and thus results in longer output queues and increases the queue overflow and congestion in the MAC layer. Longer output queues and congestion increase the data packets' waiting time. Queue overflow causes the failure of route discovery. Route discoveries need more time to collect link information. Both result in data packets' waiting at the source nodes' buffers for a longer period of time.

**4.4.5.3.4 Overpayment Ratio** Fig. 4.8(d) shows the overpayment ratio with respect to the number of nodes. The ratio changes a little for both protocols as the number of nodes changes. *vcg*'s overpayment ratio is a little higher than that of *lstop*. It implies that *lstop* can find better paths than *vcg*. The main reason is that in *vcg* with node mobility, it is difficult to collect complete link information to get the most effective routes. Therefore the most effective overpayment ratio is not achieved.




(a) Packet delivery ratio (b) Overhead

Figure 4.9: (a) Performance for different densities with 60 nodes

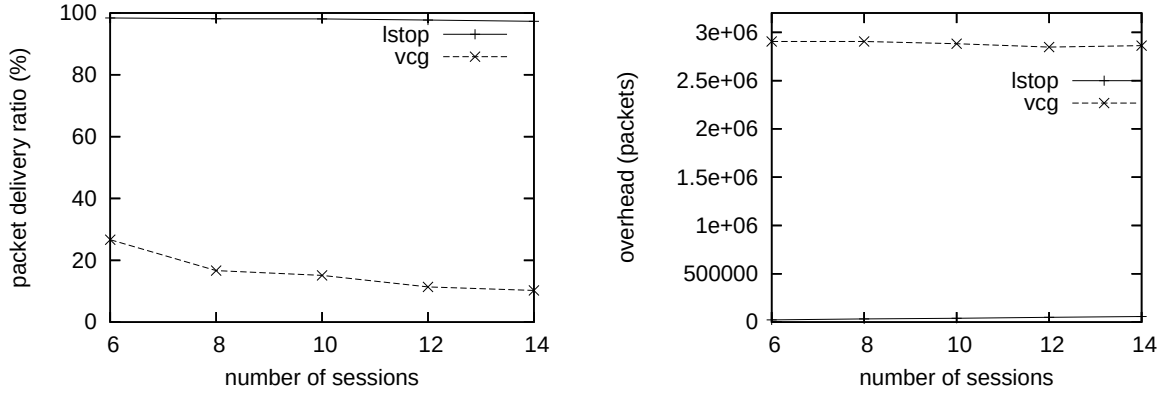
### 4.4.5.4 Impact of Node Density

These simulations were conducted with 60 nodes and a maximum node speed of 5 m/s, and the network area was changed from 600*600 $m^2$ to 900*900$m^2$.

**4.4.5.4.1 Packet Delivery Ratio** Fig. 4.9(a) shows the packet delivery ratio with respect to the size of network area. We observe that *lstop* provides a much higher delivery ratio over *vcg*. This is discussed further in Section 6.4.1(a). As the area increases (i.e., the density decreases), the packet delivery ratio decreases for both protocols. This is because an increase in the area increases the probability of nodes' moving out of each other's radio range, and increases the hop counts of data transmission paths, causing more link breakages and more packet drops.

**4.4.5.4.2 Overhead** Fig. 4.9(b) shows the overhead with respect to size of the network area. We observe that *lstop* incurs a far lower overhead than *vcg* as *vcg* generates $O(n^3)$ *RREQ* messages for route discoveries. As the network area increases (i.e., the density decreases), the overhead of *lstop* increases. This is because an increase in the area results

in more link breakages, resulting in increased route discoveries, which incurs more control messages. *Vcg*'s overhead decreases as node density decreases. This is because, as the network area increases, with a fixed number of nodes, the number of links in the network decrease and thus the number of *RREQ* messages (which include links information) per route discovery decrease. Although the number of route discoveries increases due to an increase in link breakages, the decrease in per route discovery overhead outweighs the increase in the number of route discoveries.




(a) Packet delivery ratio

(b) Overhead

Figure 4.10: Performance of different loads with 60 nodes

#### 4.4.5.5 Impact of Load

**4.4.5.5.1 Packet Delivery Ratio** These simulations were conducted with 60 nodes and a maximum speed of 5 m/s, and the number of flows was changed from 6 to 14.

Fig. 4.10(a) shows the packet delivery ratio with respect to the number of flows. We observe that *lstop* provides a much greater packet delivery ratio than *vcg* (See Section 6.4.1(a)). As load increases, the packet delivery ratio of *lstop* changes little, whereas that of *vcg* decreases. This is because an increase in the number of flows increases the output queue overflow, packet collisions and interference, causing more packet drops.

**4.4.5.5.2 Overhead** Fig. 4.10(b) shows the overhead with respect to the number of flows. We observe that *lstop* causes far lower (about 2 order of magnitude) overhead than *vcg* since *vcg* generates $O(n^3)$ *RREQ* messages for route discoveries. In *lstop*, an increase in the number of flows causes an increase in route discoveries, resulting in an increase in overhead. *Vcg*'s overhead changes a little with respect to the number of flows. This is because, with more flows, each route discovery collects significantly fewer *RREQ* messages

due to collisions and interference of these *RREQ* messages. Also, due to bounded simulation time, the long route discovery restricts the number of new route discoveries.

### 4.5 Chapter Summary

In this chapter we have presented two low overhead truthful routing protocols, LOTTO and LSTOP. We introduced a simple and effective way to collect the topology information of the network. Each node needs to broadcast one *RREQ* message instead of $O(n^2)$ [33]. By applying the VCG mechanism, both LOTTO and LSTOP guarantee that nodes get enough payment and have no incentive to cheat over their cost. LOTTO achieves cost efficiency by finding the least cost path from the source node to the destination node. The most prominent feature of LOTTO is that it reduces overhead from $O(n^3)$ [33] to $O(n^2)$ and greatly mitigates message congestion and queue overflows in the MAC layer. LSTOP, on the other hand, approaches an optimal routing solution in a dense network and incurs extremely low overhead of $O(n)$ on average and $O(n^2)$ in the worst case, thus providing far better performance and scalability.

We conducted an extensive simulation study to evaluate our protocols and compare them with the ad hoc-VCG protocol [33]. To the best of our knowledge, we are the first to conduct an extensive simulation study for mechanism design methods to evaluate important network metrics such as packet delivery ratio, overhead and end-to-end delay.

Simulation results show that LOTTO achieves a much higher packet delivery ratio, generates much lower overhead and has much lower end-to-end delay than ad hoc-VCG, while LSTOP achieves an even better performance, by generating 2 orders of magnitude lower overhead, and end-to-end delay in comparison to ad hoc-VCG.

## Chapter 5

## Achieving Multipath Truthful Routing

### 5.1 Introduction

On-demand routing protocols [13, 19] establish paths to respective destinations *only* when necessary, and maintain only the active routes between sources and destinations, thereby reducing the control overhead. Similarly, on-demand multipath routing protocols [76, 16, 77] establish multiple paths to a given destination in a single route discovery phase. Multiple paths to a destination provide better fault-tolerance to path-breaks. In case of on-demand multipath protocols, a new route discovery (which typically requires a network-wide flooding of a route request message) is necessary only when *all* paths to a given destination break. Thus, they provide an (overhead) efficient means to recover from routing failure when compared to single path on-demand routing protocols.

One might think that truthfulness in conjunction with loop-free routing implies increased complexity of protocol design, and consequently the related control overhead. On the contrary, we present here a generic mechanism [36] that makes *any table-driven multipath* routing protocol a truthful one, *without introducing additional control messages* and prove its truthfulness. As an instance of implementation, we present a Truthful Multipath Routing Protocol (*TMRP*) which is based on AOMDV [77]. TMRP establishes multiple loop-free paths to a given destination and incurs only $2n$ control overhead, asymptotically the same as AOMDV.

The rest of the chapter is organized as follows: Section 5.2 provides the system model and preliminaries. Section 5.3 presents the generic truthful multipath routing mechanism GTMR. Section 5.4 presents TMRP, an example of GTMR. Section 5.5 presents a performance evaluation of TMRP through simulations. Section 5.6 summarizes the chapter.

## 5.2 System Model and Preliminaries

We use the well known *Unit Disk Graph* (UDG) model for ad-hoc networks. In a UDG, nodes are distributed in a two dimensional Euclidean plane. All nodes use a constant radio range assumed to be normalized to 1. The nodes $u$ and $v$ in a UDG are connected iff the Euclidean distance between them $\overline{uv} \leq 1$. The network is assumed to be dense enough such that there is more than one path between any two nodes in the network, unless the two nodes are direct neighbors.

Nodes in the network are selfish and rational, but not malicious and do not collude. Each node uses constant power to send packets and it incurs the same cost to send a packet to different neighbors. However, the value of a node's forwarding cost may vary over time [1]. Since different nodes may incur different costs for packet forwarding, it is desirable to pay nodes according to their cost. As with other truthful routing protocols, nodes assume a payment mechanism [78] that takes care of accounting and transferring payments between nodes. Tamper-proof hardware [42] can also be used to protect virtual money from modification or other attacks.

### 5.2.1 A Hello Protocol

Nodes use *Hello* messages to exchange bid (cost) information. Each node establishes a neighbor table to record its neighbors' information. The neighbor table consists of multiple entries, each corresponding to a neighbor. A neighbor table entry includes the fields $<ID_i, Cost_i, TS_i>$, where $ID_i$ is the neighbor node's identity, $Cost_i$ is its cost of sending one packet and $TS_i$ is the time of establishing this entry, respectively. Fig. 5.1 shows an example of a node's neighbor table.

Further, *Hello* messages are also used to refresh routing tables. A node has multiple valid paths to a destination. Whenever a path is used to forward packets, the corresponding route entry in the routing table is refreshed. If a node tends to select the same next hop to forward packets, only limited paths will be refreshed during a given period. All other valid path entries will be timed out gradually. These paths will not be "fresh" enough and they will become invalid paths even if they are in existence. This may result in unnecessary route discoveries since our auction mechanism requires multiple paths. With the help of *Hello* messages, nodes refresh valid paths periodically. When a node X receives a *Hello* message from a neighbor, and if the neighbor is on a valid path, X increases the link life time for the neighbor by a predetermined number of time units.

[1] For example, a node's cost increases when its battery power becomes low.

Each node maintains two timers: a hello timer ($HT$) and a neighbor table flush timer ($NTFT$). Upon the expiration of $HT$, each node broadcasts a *Hello* message. A *Hello* message includes the fields $<ID, Cost>$, corresponding to its identity and cost of sending a packet, respectively. Upon receiving a *Hello* message, neighbors add an entry corresponding to the issuer of the *Hello* message into neighbor tables if it is not already present, or refresh the entry. The nodes then reset the $NTFT$ for this entry. A node also refreshes its neighbor entry when it overhears the corresponding neighbor's sending packet. Upon the expiration of $NTFT$, a node checks the timestamp of the entry. If the entry is older than a predetermined period (typically 2 *Hello* intervals), the node deletes it from its neighbor table.

### 5.2.2 An Introduction of an Auction

In GTMR, nodes use an auction-based approach to select the next hop for packet forwarding. To clarify, we briefly describe the auction schemes below.

An auction is a gathering of persons to bid for an item/good according to certain rules declared a priori. It has been used since prehistory and is still a very common mechanism in today's economics. According to the rules and information known to the bidders, an auction can be classified into different types [57]. For example, the bidders can make their bids simultaneously where each bidder puts his bid into a sealed envelop, or sequentially, where the auctioneer gives successive bids and the bidders vie for those bids. An auction can also be classified based on how the winner is selected and payed. In the *first-price* scheme, the winner is the one who bids the highest (lowest) price and pays (receives) the same. In the *second-price* scheme the winner is the one who bids the highest (lowest) price but pays (receives) the second-best price quoted by the bidders. The second-price sealed bid auction is well studied in economics theory. This auction is also known as a *Vickrey auction*, named after Richard Vickrey, a Nobel Prize winner in economics. One of the salient features of this auction is that making bids equal to their true valuations is the dominant strategy for the bidders to win the auction. By doing so, bidders will always get non-negative utilities (payoffs).

## 5.3 A Generic Protocol

In this section we present a generic truthful multipath routing protocol GTMR that transforms any table-driven multipath routing protocol MRP into a *truthful* protocol, if the MRP satisfies the following two general requirements:

**R1** The MRP must establish and maintain multiple *loop-free* paths to each destination, such that each intermediate node has at least two next hop neighbors.

**R2** The MRP must be table-driven, i.e., each intermediate node should maintain only next hop(s) for different destinations in a routing table.

Loop-freedom is a basic requirement for any routing protocol, while multiple-path and table-driven routing are necessary for establishing auctions. Furthermore, in addition to *Hello* messages, which are commonly used in ad hoc networks for neighbor discovery, GTMR uses only the control messages necessary for the proper functioning of MRP.

### 5.3.1 Description of GTMR

GTMR guarantees the *truthfulness* of the MRP by establishing a coordination between the neighbor table and the routing table, which is maintained by the MRP. Fig. 5.1 shows an example of a node $F$'s neighbor table and routing table. $A$, $B$, $C$ and $E$ are $F$'s direct neighbors and $F$ has three paths to $D$ through $A$, $B$ and $C$. The routing table consists of destination IDs, next hops and other fields such as sequence number or timestamp, distance to the destination or height, etc., depending on the multipath routing protocol being used. Generally, to forward packets, an MRP selects a single next hop towards the destination, among available multiple next hops, using selection policies like round-robin (in the order of path creation), least hop, or random. However, in GTMR, the next hop selection is based on the bid value (for packet forwarding). Precisely, for each packet, nodes select a next hop by using the second-bid auction scheme. In a later subsection we show that such a next-hop selection guarantees truthfulness.

| ID | cost | timestamp |
|---|---|---|
| A | 25 | 2500 |
| B | 22 | 2650 |
| C | 35 | 2700 |
| E | 37 | 2770 |
| ··· | ··· | ··· |

Neighbor Table

| dest | nexthop | others |
|---|---|---|
| A | A | ··· |
| ··· | ··· | ··· |
| *D* | *A* | ··· |
| | *B* | ··· |
| | *C* | ··· |
| E | E | ··· |
| ··· | ··· | ··· |

Routing Table

Figure 5.1: Illustration of node $F$'s neighbor table and routing table

Nodes establish neighbor tables using the hello protocol described in Section 5.2.1, in which nodes exchange periodic *Hello* messages containing their bid values. The bid value specified by a node in the *Hello* message represents the bid (cost) for which it is willing

to forward a packet from its neighbors. It is valid for a hello period (typically one or two seconds). By exchanging bid values proactively via *Hello* messages, nodes declare *a priori* the bids (cost) per packet-forwarding without any bias towards neighboring nodes.

Upon receiving a packet destined for a certain node, a node $F$ selects a next hop using the second-price sealed bid auction scheme. All neighboring nodes of $F$ that are next hops towards the destination in $F$'s routing table are qualified bidders. From this point view of the node that is selecting a next hop, i.e., $F$, is an *auctioneer*. Note that nodes do not bid for each packet. Instead they advertise their bids (on a per packet basis) periodically by attaching them in their periodic *Hello* messages. Since the *Hello* interval is very short, it is reasonable to assume that a node's cost will not change during that period.

Let the set of qualified bidders be $\mathcal{N}$ obtained from the routing-table of the node $F$. $F$ retrieves the bids of qualified bidders from its neighbor table and compares these bids. The winner of the bid is a node $N \in \mathcal{N}$ that seeks the least cost for forwarding packets. The payment to $N$ will be the second least cost (bid) among nodes in $\mathcal{N}$. If more than one neighboring node seeks the least bid value, then $F$ randomly selects one of them. However, the payment to this node is the same as its bid since the second lowest bid equals the lowest bid. Algorithm 1 presents the pseudo-code for selecting the next hop.

In cases where the destination is a direct neighbor, no auction is established since the destination will not forward the packet further and should not get payment. $F$ sends the packet directly to the destination.

**Algorithm 1**: A Packet forwarding algorithm

**Input**: node $F$, destination $D$
**Output**: $N$ // the next hop of $F$ toward $D$ ;
$Pay^N$ // payment to next hop $N$

```
1  /*Check if multiple paths are available*/;
2  if (!IsMultiplePathAvailable(F, D)) then
3      send route error message;
4  else
5      /*Nexthop_F^D is the set of neighbor nodes of F in valid paths toward D*/;
6      foreach node j ∈ {Nexthop_F^D} do
7          Retrieve Cost_j from the neighbor table;
8          Record the lowest cost min and the second lowest cost min_next;
9          if Cost_j is lowest then
10             N = j;
11             Pay^N = min_next;
12         end
13     end
14     return (N,Pay^N);
15 end
```

After selecting the next hop node $N$, $F$ adds the node ID $N$ and its payment $Pay^N$ to the packet header, and forwards the packet to node $N$. Under the assumption that nodes are selfish but not malicious, a forwarding node $N$ is not supposed to modify its payment $Pay^N$. Otherwise, tamper-proof hardware [42] can be used to protect this payment information from modification or other attacks.

In addition, two methods can be applied to avoid such modifications. The first method resorts to cryptography. Besides items $N$ and $Pay^N$, $F$ calculates a $MAC$ (Message Authentication Code) over these two items, digitally signs it with its private key, and appends this $MAC$ to the packet. When the destination receives the packet, it can verify the amount of payments. The second method resorts to neighbor monitoring. Neighbor monitoring is a key mechanism in detection-based methods for the selfish-node problem. Each node works in the promiscuous mode and can overhear packets transmitted by nodes within the radio range. To prevent the modification of payment amount $Pay^N$, each neighboring node of $F$ saves the overheard packet $P$ sent to $N$ in its buffer. Upon overhearing packet $P$ sent out from node $N$, they compare the corresponding header area of two instances of packet $P$ and check if $Pay^N$ is modified.

**Lemma 5.3.1.** *Given an MRP that establishes and maintains loop-free multiple paths to each destination, the packet forwarding algorithm of GTMR routes a packet along a loop-free path to the destination.*

*Proof.* The proof follows from the loop-freedom insured by MRP. Let $\mathcal{H}(D)_i$ represent a set of next hops for the destination $D$, maintained by node $i$ in its routing table. If $j \in \mathcal{H}(D)_i$, then the hop-distance of $j$ towards the destination is strictly shorter than the hop-distance of $i$ towards the destination (along $j$). Since the packet forwarding algorithm of GTMR selects a next hop from $\mathcal{H}(D)_i$ at each intermediate node $i$, the path traced by packet using the forwarding algorithm of GTMR contains only those nodes which strictly decrease the hop-distance to the destination, which implies loop-freedom.

### 5.3.2 Truthfulness of GTMR

In this section, we show that truthful bidding is the dominant strategy for each node for GTMR. It is important to note that a source node pays other nodes but not itself, and conducts an auction for other nodes only. Neither destination node gets payment.

**Theorem 5.3.2.** *Given an MRP that establishes and maintains a routing table containing multiple routes to each destination, bidding true cost is the dominant strategy for every*

*qualified bidder, when the next hop (a bidder) is selected using the packet forwarding algorithm of GTMR.*

*Proof.* To prove the theorem, we need to show that any bidder $i$, will benefit only when it bids its true cost [57] (sent in a Hello message).

Suppose a bidder $i$ with true cost $v_i$ declares a bid $b_i$. Let $i$'s utility be $u_i$. Let $m^{-i}$ denote the minimum bid besides $b_i$. According to the auction rule, the winner of the auction receives the second-best (second-least) bid quoted by the bidders and the other bidders get nothing. Thus the utility $u_i$ of a bidder $i$ is

$$u_i = \begin{cases} m^{-i} - v_i, & b_i < m^{-i} \\ 0, & b_i > m^{-i} \end{cases}$$

There are two possibilities: Bidder $i$ may over-bid or under-bid its cost $v_i$ [57].

**Case of over-bidding:** If the bid is higher than the cost, i.e., $b_i > v_i$, then there are three possibilities.

1. If $b_i < m^{-i}$, then $i$ wins the auction and $u_i = m^{-i} - v_i$. However, $i$ can get the same payoff by bidding its cost $v_i$.

2. If $v_i < m^{-i} < b_i$, then $i$ loses the auction and gets zero payoff. However, by bidding $v_i$, it can get a positive payoff.

3. If $m^{-i} < v_i$, then $i$ gets zero payoff, the same as it if it had bid $v_i$.

Thus, $i$ has no incentive for over-bidding.

**Case of under-bidding:** If the bid is lower than the cost, i.e., $b_i < v_i$, then there are three possibilities.

1. If $v_i < m^{-i}$, then $i$ gets payoff $m^{-i} - v_i$, the same as if it had bid $v_i$.

2. If $b_i < m^{-i} < v_i$, then $i$ gets a negative payoff as $m^{-i} - v_i < 0$. However, by bidding $v_i$, $i$ gets zero payment, which is better than a negative payoff.

3. If $b_i > m^{-i}$, then $i$ gets zero payoff, the same as if it had bid $v_i$.

Thus, $i$ has no incentive for under-bidding.

In both the cases $i$'s dominant strategy is to bid its cost $v_i$, and hence the theorem.

## 5.4 A Truthful Multipath Routing protocol

In this section, we present an instance of GTMR based on the Ad-hoc On-demand Multipath Distance Vector routing protocol (AOMDV) [77]. AOMDV satisfies the general requirements specified in the previous section. We call this instance of GTMR with AOMDV a Truthful Multipath Routing Protocol (TMRP). To clarify, we first explain the AOMDV protocol. Then, we present a detailed description of TMRP.

### 5.4.1 AOMDV

AOMDV [77] is an extension to AODV [19] that aims to find multiple paths, especially link-disjoint paths. Like AODV, AOMDV is based on a distance vector concept and uses hop-by-hop routing. Moreover, it ensures loop-freedom.

In AOMDV, when a source has packets to send to a destination and finds no routes in its routing table, it invokes route discovery by broadcasting *RREQ* packets. Route discovery in AOMDV is similar to AODV. An *RREQ* packet in AOMDV includes all fields that occur in AODV. It includes an additional field called the last hop[2], i.e., the neighboring node of the source. This information and the next hop information, i.e., the node from which to receive the *RREQ*, are used to achieve link disjointness for reverse paths to the source [77]. A node may receive multiple duplicate *RREQ* packets. For each packet received, it examines if an alternate reverse path to the source can be formed such that loop-freedom and link-disjointness are preserved.

Upon establishing a reverse path to the source, an intermediate node checks if it has any valid paths to the destination. If so, it generates an *RREP* packet, including a forwarding path not used in any previous *RREP*s for this *RREQ*, and sends the *RREP* back to the source through the reverse path. Otherwise, it checks if it has forwarded this request before and forwards it it has not.

Upon receiving a *RREQ* packet, the destination tries to form a reverse path to the source as above. It then generates a *RREP* packet for each *RREQ* copy that arrives through a loop-free path to the source. Multiple *RREP*s intend to increase the probability of finding multiple disjoint paths.

Upon receiving an *RREP* packet, an intermediate node checks if it can form a loop-free and disjoint path to the destination using the same rule as when it receives a *RREQ* packet. If not, it drops the *RREP*. Otherwise, it checks if there are any reverse paths to the source

[2]The last hop of a path to a destination is the node just proceeding the destination on that path.

that has not been used to forward an *RREP* for this route discovery. If so, it chooses one of the unused reverse paths to forward the *RREP*. Otherwise, it drops the packet.

Loop-freedom is guaranteed by satisfying the following sufficient conditions: 1) The sequence rule. For each destination, multiple paths maintained by a node should have the same sequence number, i.e., the highest known destination sequence. 2) For the same destination sequence number, a node never advertises a route shorter than one already advertised, and never accepts a route longer than one already advertised. More details and a proof are available in [77].

Route maintenance is also very similar to AODV. Upon link breakage of the last path to the destination, a node generates or forwards a *RERR* packet.

### 5.4.2 An Implementation of Multipath Routing

AOMDV [77] is a reactive hop-by-hop routing protocol which finds node/link disjoint multiple paths. However, it cannot guarantee finding multiple paths from a node. In AOMDV, a node forwards packets as long as there is one path to the destination. On the other hand, TMRP requires multiple paths from a node to establish an auction. To achieve this, upon receiving a packet, an intermediate node checks if it has at least two valid next hop nodes to the destination. An exception is that an intermediate node is the direct neighbor of the destination. In that case the intermediate node forwards the packet directly to the destination. If no multiple paths are available, the node drops the packet and sends an *RERR* packet. While keeping most implementation of AOMDV, especially the necessary condition for loop-freedom, and thus keeping the feature of loop-freedom, we modify AOMDV implementation to satisfy general requirements for auctions.

We modified AOMDV to maximize the number of multiple paths. First, we relax the requirement for node/link disjoint paths. Node/link disjointedness is a good feature for routing robustness. However, it limits the number of multiple paths. Second, the *RREP* packet is handled in the same manner as *RREQ* packets. *RREP* is also flooded over the whole network. Whenever a node receives an *RREP* packet, it examines if it can form an alternate reverse path to the destination. If so, it checks if it has forwarded the first copy of *RREP* for this route discovery and forwards it if it has not. Otherwise, it drops the *RREP*. Third, while guaranteeing that each node maintains routes only to the highest known destination sequence number, we changed the rule for sequence number updating in case of link breakages and link timeout so that the sequence number will not be changed during the propagation of *RREP*s. A node may change the destination sequence number

included in an *RREP* if it has a higher one. Upon receiving a copy of *RREP* with a higher destination sequence number, a node creates a new route to replace valid routes created by *RREP* copies with lower destination sequence numbers and thus reduces the number of multiple paths.

We do not allow an intermediate node to generate an *RREP* even if it has multiple valid forwarding paths to the destination. Only the destination can generate *RREP*s for a route discovery. By doing so, all nodes have chances to refresh their routing entries to this destination and get more accurate routing information. Also periodic *Hello* messages instead of adaptive *Hello* messages are used to help maintain routing tables, as discussed in Section 5.2.1.

Upon expiration of the timer for a route discovery, the source node checks if it has established multiple paths to the destination. If not, it retries the route discovery. Upon receiving a packet, an intermediate node also checks if there are multiple paths to the destination. If not, it drops the packet and sends (broadcasts) an *RERR* packet.

Route maintenance is also modified from AOMDV. Upon link breakage, a node on an active path checks if it still has valid multiple paths to the destination. If not, it generates an *RERR* to inform other nodes that it can not reach the destination for auction-based packet forwarding (it can still reach the destination). A receiver of *RERR* invalidates the routing entry corresponding to the *RERR* initiator and repeats the above actions.

### 5.4.3 TMRP

Nodes establish and maintain multiple routes using a modified version of AOMDV (as described in Section 5.4.2). Nodes maintain a neighbor table using the hello protocol described in Section 5.2.1.

We simulated two variations of TMRP for nodal cost. In the first variation, nodes don't change their packet forwarding cost over time. In the second variation, a node's packet forwarding cost may change over time. A node increases its cost proportionally to the number of packets sent (or forwarded). However, when a node's cost reaches an upper boundary, it goes down to the low boundary. This is to simulate the scenario where a node can have its battery recharged upon battery depletion.

A good feature of the latter implementation is that it can achieve good load balancing. Our auction mechanism always selects nodes with the lowest cost as forwarding nodes. Without changing their cost, nodes with a low cost have a high probability of being forwarding nodes. They are likely to forward more traffic than other nodes and thus they

become hot spots or are in hot areas, resulting in packet collisions and radio interferences. However, if these nodes' cost increases as the packets are forwarded, their cost will gradually increase more than other nodes. Thus they are less likely to be selected as forwarding nodes. Other nodes will get more chances to forward packets. Packets are therefore forwarded along different paths by different nodes over time and load balance is achieved. Note that such load balancing does not compromise truthfulness due to the use of the auction mechanism.

To forward packets, nodes use the next hops from the routing table entries, and execute the packet forwarding algorithm (Algorithm 1) of GTMR with inputs from the neighbor table. It follows from Lemma 1, that such a forwarding scheme guarantees loop-freedom, given that the underlying AOMDV is loop-free.

### 5.4.4 Message complexity

Note that the *Hello* messages, which are very commonly used by nodes for neighbor discovery in most ad-hoc networks, are the *only* additional messages for TMRP, and they are broadcast only locally.

Compared with AOMDV, TMRP incurs a little higher overhead for route discovery. In both the protocols, each node except the destination broadcasts an $RREQ$ packet, and thus a total of $n-1$ packets are transfered. In TMRP, each node except the source forwards one $RREP$ packet. Thus $n-1$ $RREP$ packets are transfered. On the other hand, in AOMDV, several instances of $RREP$s are unicast back to the source, resulting in $k\sqrt{n}$ packets on average and $n-1$ packets in the worst case, where $k$ is the number of instances. In summary, a route discovery in TMRP incurs $2n-2$ packets while that in AOMDV incurs $n+k\sqrt{n}$ on average. Thus both protocols incur an $O(n)$ overhead. To the best of our knowledge, this is the lowest overhead incurred by a truthful routing protocol.

The other overhead incurred is due to $RERR$ packets. Upon link breakage, a node sends an $RERR$ to a destination. The receiver of this packet forwards the $RERR$ if there are no multiple paths to the destination, excluding the path containing the new link breakage indicated by the $RERR$. In the best case, only one node, which is a neighbor of the source, sends an $RERR$. In the worst case, which is very unlikely, each node except the source and the destination sends an $RERR$ and thus at most $n-2$ packets are transfered. Our simulation study shows that only a very small fraction of nodes send $RERR$s.

## 5.5 Performance Evaluation

We conducted an extensive simulation study to evaluate the performance of TMRP in ad-hoc networks with selfish nodes. To the best of our knowledge, TMRP is the only truthful multipath routing protocol in the literature. For this reason, we did not have another protocol for comparison. Generic routing protocols such as AODV [19] or AOMDV [77] are not comparable as those protocols work under the *different assumption* that nodes follow the protocol accurately and are willing to cooperate.

Our simulations focused on the following five metrics: packet delivery ratio, routing overhead, average end-to-end delay, overpayment ratio and average hop count, which is the average number of hops needed from a source to a destination.

We conducted two implementations of TMRP and compared simulation results between them. In one version, viz. *tmrp-c*, nodes don't change their packet forwarding cost during the simulation. In the other version, viz. *tmrp-v*, nodes' cost change over time. A node's packet forwarding cost increases proportionally to the number of data and control packets sent or forwarded. However, when a node's cost reaches the upper boundary, i.e., 40, it goes down to the lower boundary, i.e., 20. This is to simulate battery recharging.

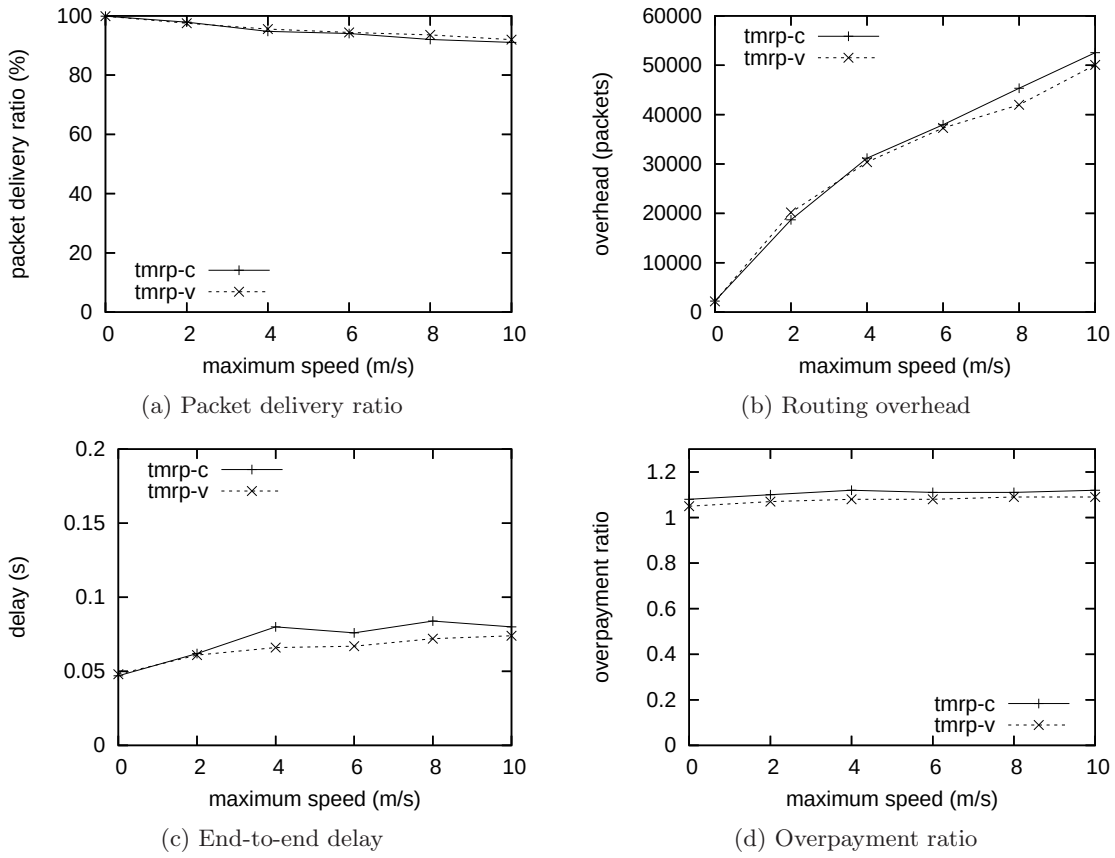
We used GloMoSim [64] for our simulations. Unless specified otherwise, the following parameters were used in the simulations. 100 nodes were initially placed uniformly in an area of 600m by 1500m. Nodes used a radio range of 250 meters, and picked a packet forwarding cost randomly between 20 and 40. All nodes followed the Random Way-point mobility model [65] with a maximum speed of 1 m/s to 10 m/s and a pause time of 30 seconds. Each simulation lasted for 900 seconds. To generate traffic we simulated 10 CBR flows. Each flow sent four 512-byte data packets per second, starting at 120 seconds and ending 880 seconds. Nodes used the 802.11 protocol with DCF as the MAC protocol. Data points represented in a graph were averaged over 20 simulation runs, each with a different seed.

### 5.5.1 Impact of Node Mobility

These simulations were conducted with a maximum node speed ranging from 1 to 10 m/s.

#### 5.5.1.1 Packet delivery ratio

Fig. 5.2(a) shows the packet delivery ratio with respect to the maximum node speed. In both implementations, more than 91% of the packets are delivered. As mobility increases, the packet delivery ratio drops as mobility increases link breakages, resulting in more packet




(a) Packet delivery ratio

(b) Routing overhead

(c) End-to-end delay

(d) Overpayment ratio

Figure 5.2: Performance with respect to mobility.

drops. We didn't implement local packet salvage, or the packet delivery ratio could have been higher. We observe that *tmrp-v* performs better than *tmrp-c*. This is in *tmrp-c*, node cost does not change over time. Nodes with lower cost have a higher probability of being selected as next hops than those with higher cost. Such nodes are more likely to become hot nodes or be in a hot area and thus incur more packet collision or radio interference, resulting in packet drops. On the other hand, in *tmrp-v*, nodes increase their cost as they send packets. The more packets sent, the higher the cost. After forwarding packets, hot nodes have a high cost and are less likely to be continued as next hops. Thus the problem of hot nodes or hot areas is mitigated. So *tmrp-v* achieves some good load balancing.

#### 5.5.1.2 Routing Overhead

Fig. 5.2(b) shows the routing overhead with respect to the maximum node speed. TMRP incurs a $2n$ overhead per route discovery. As mobility increases, link breakage increases, resulting in more $RERR$s and more route discoveries. Thus the overall route overhead

increases. *tmrp-v* incurs less overhead than *tmrp-c* since it mitigates the hot node/area problem, as discussed above, and thus results in fewer link breakages.

#### 5.5.1.3 End-to-end Delay

Fig. 5.2(c) shows the end-to-end delay with respect to the maximum node speed. The delay is very low for both versions, and less than 0.09 second for each point. Mobility increases link breakages, resulting in more route discoveries and thus packets need more time to reach destinations. *tmrp-v* incurs a lower delay than *tmrp-c* since it results in fewer route discoveries and shorter output queues by avoiding hot nodes/areas.

#### 5.5.1.4 Overpayment Ratio

Fig. 5.2(d) shows the overpayment ratio with respect to maximum node speed. The overpayment ratio for both versions is lower than 1.12 and varies little over mobility. TMRP therefore overpays nodes very little over their cost. *tmrp-v* incurs a lower overpayment ratio than *tmrp-c*, as cost change and load balancing may result in less difference between bids (nodes' cost).

### 5.5.2 Impact of Network Size

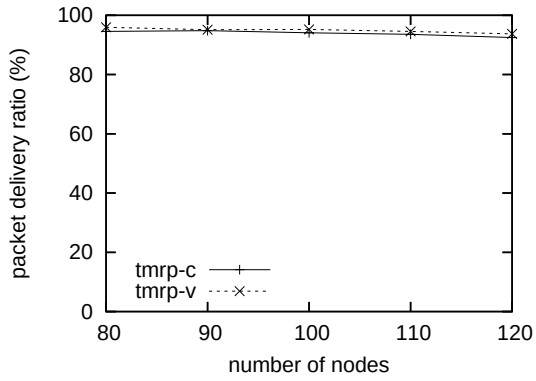
These simulations were conducted with a density of node/9000m*m and a maximum node speed of 5 m/s, and the number of nodes was changed from 80 to 120.

#### 5.5.2.1 Packet Delivery Ratio

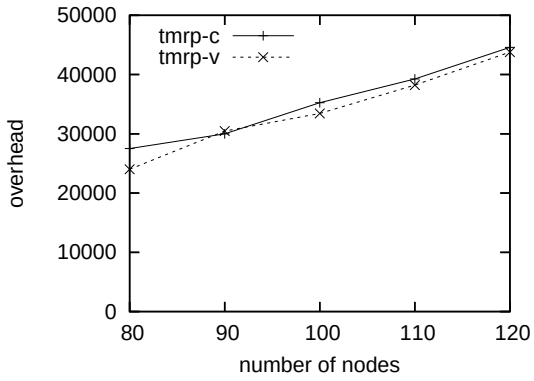
Fig. 5.3(a) shows packet delivery ratio with respect to the number of nodes. Both versions deliver more than 92% of the packets. As the number of nodes increases, the packet delivery ratio decreases for both versions. An increase in the number of nodes results in an increase in the network area and an increase in the average hop count of a path, as shown in Fig. 5.3(e). Thus node mobility is more likely to cause link breakages, resulting in more packet drops. *tmrp-v* performs better than *tmrp-c* due to its load balancing, as discussed in Section 5.5.1.1.

#### 5.5.2.2 Routing Overhead

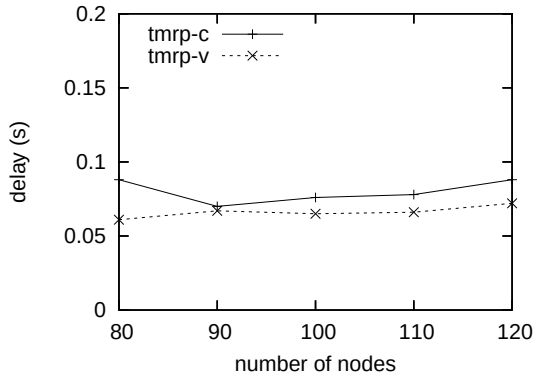
Fig. 5.3(b) shows the routing overhead with respect to the number of nodes. As the number of nodes increases, the overhead increases for both versions. First, more nodes are involved in routing and thus transfer more control packets. Second, an increase in link breakages due to an increase in the area results in more *RERR*s and route discoveries, which incurs



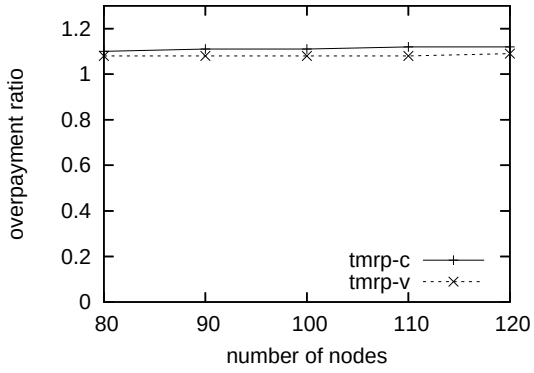

(a) Packet delivery ratio



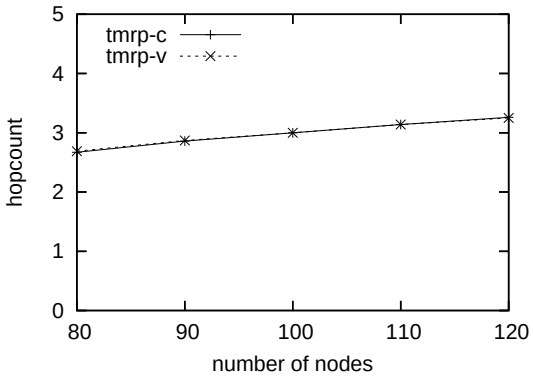

(b) Routing overhead




(c) End-to-end delay




(d) Overpayment ratio




(e) Average hop count

Figure 5.3: Performance with respect to number of nodes.

more overhead. Due to load balancing, *tmrp-v* incurs few route discoveries and thus results in less overhead than *tmrp-c*.

#### 5.5.2.3 Average End-to-end Delay

Fig. 5.3(c) shows end-to-end delay with respect to the number of nodes. The delay is very low for both versions and less than 0.09 second for each point. As the number of nodes increases, the network area increases and the average hop count for a path increases, thereby increasing end-to-end delay. Also, an increase in nodes results in more contention for the radio channel. Data packets are likely to need more time to get the channel. Again, *tmrp-v* performs better than *tmrp-c* due to its load balancing.

#### 5.5.2.4 Overpayment Ratio

Fig. 5.3(d) shows overpayment ratio with respect to the number of nodes. The ratio is less than 1.12 for both versions and does not change over the number of nodes. *tmrp-v* still performs better than *tmrp-c* due to its load balancing.

#### 5.5.2.5 Average Hop Count

Fig. 5.3(e) shows the average hop count of a path with respect to the number of nodes. As the number of nodes increases, with a fixed node density, the network area increases, thus the average hop count for a path increases. *tmrp-c* and *tmrp-v* generate identical hop counts. This is because multiple paths from a node to a destination are formed by the multiple-path routing protocol such that they are no longer than an advertised hop count, which is set to the length of the longest available path at the time of first advertisement for a sequence number. These different paths almost have the same length.

### 5.5.3 Impact of Node Density

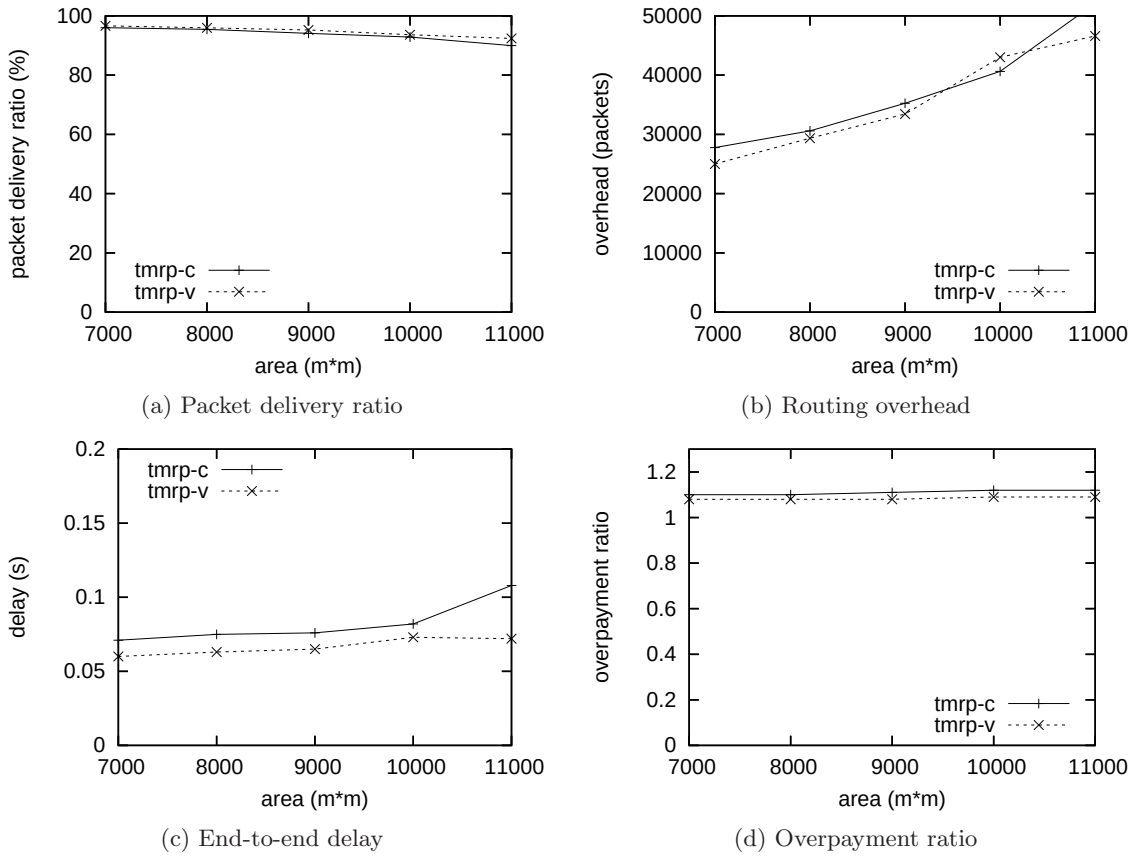
These simulations were conducted with a maximum node speed of 5 m/s and the node density was changed from node/$7000m^2$ to node/$11000m^2$.

#### 5.5.3.1 Packet Delivery Ratio

Fig. 5.4(a) shows the packet delivery ratio with respect to node density. As density decreases, the network area increases and thus mobility is more likely to cause link breakages, resulting in more packet drops. *Tmrp-v* performs better than *tmrp-c* due to its capacity of load balancing.

#### 5.5.3.2 Overhead

Fig. 5.4(a) shows the routing overhead with respect to node density. As node density decreases, the overhead increases. First, an increase in the network area due to a decrease




(a) Packet delivery ratio

(b) Routing overhead

(c) End-to-end delay

(d) Overpayment ratio

Figure 5.4: Performance with respect to node density.

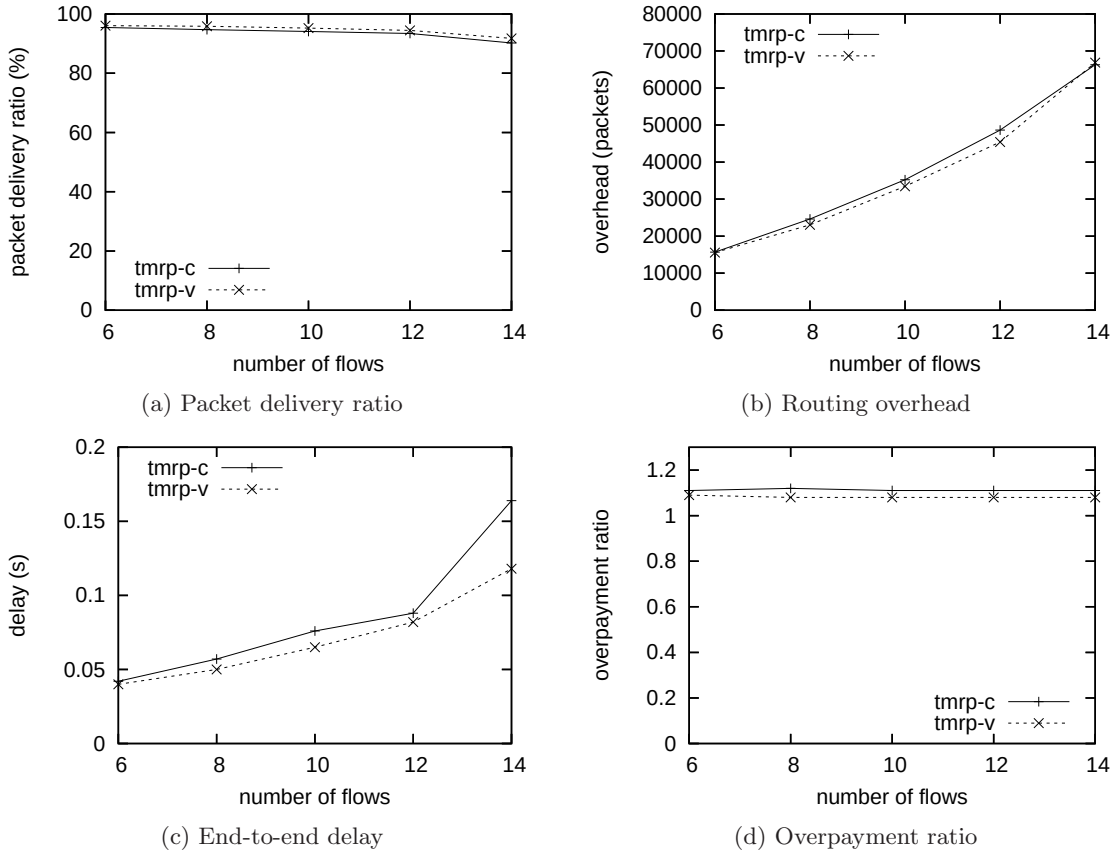
in node density results in more $RERR$s and more route discoveries. Second, as node density decreases, a node has fewer neighbors and can form fewer multiple paths. Thus, a route discovery is more likely to occur.

#### 5.5.3.3 Average End-to-end Delay

Fig. 5.4(c) shows end-to-end delay with respect to node density. As node density decreases, end-to-end delay increases for both versions. A decrease in node density results in an increase in route discoveries, and thus packets need a longer time to reach destinations. A decrease in node density also results in an increase in network area, and thus packets may travel over more hops. Nevertheless, all these end-to-end delays are very low, and the highest recorded was only 0.11 second.

#### 5.5.3.4 Overpayment Ratio

Fig. 5.4(d) shows overpayment ratio with respect to node density. The ratio is less than 1.12 for both the versions and does not change over density changes. *tmrp-v* still performs better than *tmrp-c* due to its capacity of load balancing.




(a) Packet delivery ratio

(b) Routing overhead

(c) End-to-end delay

(d) Overpayment ratio

Figure 5.5: Performance with respect to load.

### 5.5.4 Impact of Load

These simulations were conducted with a maximum node speed of 5 m/s and the number of flows was changed from 6 to 14.

#### 5.5.4.1 Packet Delivery Ratio

Fig. 5.5(a) shows packet delivery ratio with respect to the number of flows. As load increases, the packet delivery ratio decreases for both the versions. This is because an increase in the number of flows increases the number of data and control packets, resulting in more packet

collisions and more severe radio interference, which cause more packet drops. *Tmrp-v* delivers more packets than *tmrp-c* due to load balancing.

#### 5.5.4.2 Overhead

Fig. 5.5(b) shows routing overhead with respect to the number of flows. As the load increases, the overhead increases, as an increase in flow results in more route discoveries. In addition, a heavier load results in more packet drops, as discussed above, producing more *RERR*s and causing even more route discoveries. *tmrp-v* incurs less overhead than *tmrp-c* as it invokes less route discoveries.

#### 5.5.4.3 Average End-to-end Delay

Fig. 5.5(c) shows end-to-end delay with respect to the number of flows. As flows increase, the delay increases for both versions. A heavier load increases the number of packets and thus increases the contention for a radio channel. Nodes need more time to get the radio channel to transfer packets. This results in a longer route discovery and slower data packet forwarding. *Tmrp-v* incurs shorter delay than *tmrp-c* since it invokes fewer route discoveries and has a shorter output queue by avoiding hot nodes/areas.

#### 5.5.4.4 Overpayment Ratio

Fig. 5.5(d) shows overpayment ratio with respect to the number of flows. The ratio is less than 1.12 for both versions and does not change over the number of flows. *Tmrp-v* performs better than *tmrp-c* due to its capacity of load balancing.

### 5.6 Chapter Summary

We presented a generic method, GTMR, to transform a table-driven multipath routing protocol into a truthful routing protocol. By applying an auction mechanism for packet forwarding, GTMR achieves truthfulness and stimulates nodes to declare their true cost. As an example of implementation, we presented TMRP, a Truthful Multipath Routing Protocol which is based on the AOMDV protocol. A prominent feature of TMRP is that it incurs only $2n$ control packets for a route discovery and needs no new types of control messages over AOMDV. To the best of our knowledge, this is the lowest overhead incurred for truthful routing protocols. TMRP can also achieve load balancing without compromising truthfulness. We have proved that GTMR guarantees truthfulness and thus TMRP. We also conducted an extensive simulation study to evaluate the performance of two variations of

TMRP. Simulation results showed that TMRP provided high packet delivery ratio and had low overhead and low end-to-end delay without compromising the overpayment of nodes.

# Chapter 6

# Truthful Greedy Forwarding

## 6.1 Introduction

Geographic routing protocols [79, 80, 23, 24, 25, 26, 27, 28, 81, 82, 83, 84], also known as position-based routing protocols for mobile ad-hoc networks use the position information of nodes in the network for routing and location service. Unlike topology-based routing protocols [19, 13], nodes in geographic routing protocols do not establish or maintain routes in the network. A node forwards packets towards a destination solely based on the position of the destination, its own position, and the position of neighboring nodes. By using only local topological information, geographic routing protocols cope with node mobility, and exhibit better scalability than topology-based routing protocols [79].

Recent research [33, 73, 39, 38] has focused on designing truthful protocols for the selfish-node problem in the context of topology-based routing protocols. Such protocols rely on the discovery and maintenance of routes in the network, which require substantial overhead (in the order of $O(n^3)$ [33, 73] control messages, where $n$ is the number of nodes in the network). On the other hand, geographic forwarding incurs a localized overhead of control messages [79]. It is therefore desirable that a geographic forwarding algorithm designed to cope with selfish nodes should also be localized in nature.

In this chapter, we present such an algorithm, viz., the *T*ruthful *G*eographic *F*orwarding algorithm (*TGF*) [37] for data forwarding in ad-hoc networks. TGF introduces three auction-based packet forwarding schemes that *guarantee truthfulness* while maintaining the localized nature of geographic forwarding. We prove that TGF is truthful, statistically analyze the average progress made per hop for proposed auction-based packet forwarding schemes, and present results from our extensive simulation study. To the best of our knowledge, TGF is the first algorithm to address truthfulness in the context of geographic forwarding.

We have used the same system model as in Chapter 5. Additionally, each node knows its own position by means of a positioning system such as GPS, and reveals its true position.

Furthermore, the source node is assumed to know the position of the destination [85, 86, 87, 82, 88]. The network is assumed to be dense enough such that there is more than one path between any two nodes in the network, and avoids dead-ends due to geographic forwarding.

The rest of the chapter is organized as follows: Section 6.2 presents the truthful geographic forwarding protocol, Section 6.3 presents an analysis of TGF, Section 6.4 presents the performance evaluation through simulations and Section 6.5 summarizes the chapter.

## 6.2 Truthful Geographic Forwarding

### 6.2.1 The Basic Idea

The truthful geographic forwarding algorithm (TGF) is a combination of greedy forwarding with an auction scheme. Unlike pure greedy forwarding where the selection of the next hop is based on the progress (Euclidean distance) made towards the destination, in TGF the selection of the next hop is based on a combination of bid value (for packet forwarding) along with progress made by the node towards the destination. To achieve this, nodes exchange periodic *Hello* messages containing their positions and bid values, and establish neighbor tables. The bid value specified by a node in the *Hello* message represents its bid (cost) for which it is willing to forward a packet from its neighbors. This bid value in a *Hello* message is valid for a hello period (typically one or two seconds). By exchanging bid values proactively via *Hello* messages, nodes declare *a priori* the bids (cost) per packet-forwarding without any bias towards neighboring nodes.

When a node has a packet to forward to the destination, it uses an auction scheme to select a next hop for the destination from its neighbors. As the bid values and the positions of the neighbors are known a priori (because of *Hello* message exchange) selecting a next hop requires a neighbor table look-up and selecting a neighbor according to the auction scheme. The auction scheme(s) guarantee that TGF is truthful. It is important to note that the control overhead incurred by TGF is only due to the *Hello* messages, which are one hop broadcast messages. Thus, the control overhead of TGF is $O(1)$ per node every $t$ seconds, where $t$ is the hello interval.

### 6.2.2 Packet Forwarding

Generally, geographic routing protocols [79] use greedy forwarding in which a node forwards a packet to a node that is geographically closest to its destination among neighboring nodes. However, this approach is not suitable in the context of selfish nodes, where nodes are more interested in cost and payment. To address this issue TGF takes into account the cost

of packet forwarding of nodes, and forces nodes to show their true cost. TGF uses three forwarding schemes, the basic scheme, the restricting bidders scheme, and the unit price bid scheme.

#### 6.2.2.1 Basic Forwarding Scheme (BaFS)

Upon receiving a packet destined for a certain node, node $F$ selects the next hop using the second-price sealed bid auction scheme. All neighboring nodes of $F$ that make progress towards the destination are qualified bidders, and $F$ is an *auctioneer*. Nodes do not bid for each packet. Instead they bid on a per packet basis periodically by attaching bids to their periodic *Hello* messages.

Let $\mathcal{D}(a, b)$ be the Euclidean distance between node $a$ and node $b$, and $\mathcal{NB}^a$ be the set of neighboring nodes of $a$. Then, the qualified bidders $K \in \mathcal{NB}^{\mathcal{F}}$ are nodes that satisfy the condition $\mathcal{D}(K, D) < \mathcal{D}(F, D)$, where $D$ is the destination ID. We denote the set of qualified bidders as $\mathcal{N}$. The winner of the bid is node $N \in \mathcal{N}$ that seeks the lowest cost for forwarding packets. The payment to $N$ will be the second least bid among nodes in $\mathcal{N}$. If there is more than one neighboring node seeking the lowest bid value, the node making the maximum progress towards the destination (i.e., having $min(\mathcal{D}(K, D))$) will be the winner. However, the payment to this node is the same as its bid, since the second lowest bid equals the lowest bid. After selecting the next hop node $N$, $F$ adds the node identity $N$ and its payment $Pay^N$ to the packet header, and forwards the packet to node $N$. The same method as discussed in Section 5.3.1 can be used to secure payment.

#### 6.2.2.2 Average Plus Forwarding Scheme ($A^+FS$)

In the basic scheme, each node making progress towards the destination is a potential next hop. However, it is possible that the lowest bidder selected using the basic scheme may make the least progress (among the potential next hops) towards the destination. Thus, the basic forwarding scheme can increase the number of hops, resulting in a higher total payment. Hence, it is desirable to reduce total cost as long as truthfulness is guaranteed.

If an auctioneer selects only those nodes making *enough* progress towards the destination as qualified bidders, then the average hop count to the destination will be reduced. To achieve this, an auctioneer calculates the average distance ($AvgDst$) over all neighboring nodes that are closer to the destination than the auctioneer. It selects only nodes that make greater progress more than $AvgDst$ towards the destination as qualified bidders ($QB'$). The bidder with the lowest bid in $QB'$ will win the auction. Its payment will be the second lowest

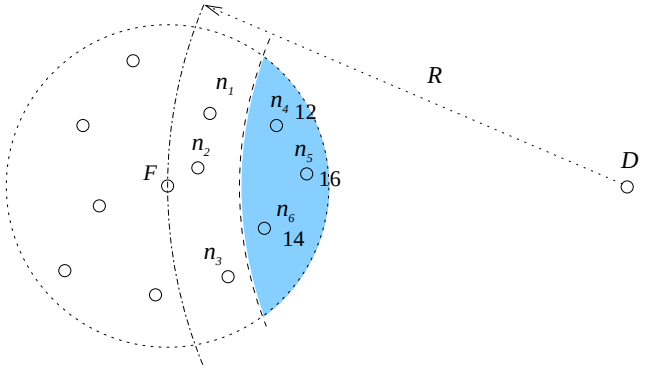



Figure 6.1: Illustration of the Average Plus Forwarding Scheme (A$^+$FS).

bid in $QB'$. If there are less than two qualified nodes in $QB'$, then the auction fails. In such a case, the auctioneer returns to the BaFS, i.e., sets all nodes making progress towards the destination as the qualified bidder, and selects the lowest bidder as the winner.

Fig. 6.1 illustrates an example of A$^+$FS. In the figure, $F$ is the auctioneer and the dotted circle denotes the radio range of $F$. $R$ denotes the distance from $F$ to destination $D$. Nodes $n_k, k \in 1, \cdots, 6$ are the neighbors of $F$ that are closer to the destination than $F$. The dashed line denotes the $AvgDst$ of these six nodes. Thus only nodes in the shadowed area, $n_4$, $n_5$ and $n_6$, are the qualified bidders (i.e., belong to $QB'$). Among these nodes, $n_4$ has the lowest bid (12) and $n_6$ has the second lowest bid (14). Thus $n_4$ will be the winner of the auction and gets a payment of 14.

#### 6.2.2.3 Unit Price Bid

In both BaFS and A$^+$FS, bidders quote their bids for sending a packet and the lowest bidder wins the auction. An alternative way to bid in both schemes is to use the price of unit progress as a criterion. In this case, the bidder asking the least price per unit progress will be the winner. For example, in BaFS the auctioneer $F$ selects all neighboring nodes that make progress towards destination $D$ as qualified bidders ($QB$). Based on the bid $b_i$ value and progress ($\mathcal{D}(n_i, D) - \mathcal{D}(F, D)$) of each node $n_i \in QB$, $F$ calculates the unit price of each node $n_i$ as:

$$\mu_i = \frac{b_i}{(\mathcal{D}(n_i, D) - \mathcal{D}(F, D))}.$$

The auctioneer $F$ selects the node with the least $\mu_i$, denoted as $n_s$, as the next hop. The payment to $n_s$ will be made according to the unit price instead of the bid for the packet. The next hop node, $n_s$, gets the unit payment as the second least unit price among $QB$, denoted as $\mu'$. Thus, the payment to the node $n_s$ per packet will be $\mu' * (\mathcal{D}(n_s, D) - \mathcal{D}(F, D))$.

### 6.3 Analysis of TGF

In this section, we first show that truthful bidding is the dominant strategy for each node for BaFS [57], A$^+$FS and the unit price bid schemes. Second, we present an analysis of progress made per hop for both BaFS and A$^+$FS schemes.

#### 6.3.1 Truthfulness of TGF

**Theorem 6.3.1.** *In BaFS, bidding true cost is the dominant strategy for every qualified bidder.*

*Proof.* This proof is similar to that in Section 5.3.2. Suppose a bidder $i$ with true cost $v_i$ declares a bid $b_i$. Let $i$'s utility be $u_i$. We denote the minimum bid besides $b_i$ as $m^{-i}$, and the distance between $i$ and the destination $D$ as $\mathcal{D}_i$. According to the auction rule, the utility $u_i$ of a bidder is

$$u_i = \begin{cases} m^{-i} - v_i, & b_i < m^{-i} \\ m^{-i} - v_i, & b_i = m^{-i} \wedge \mathcal{D}_i = \min\limits_{\forall j \in \text{bidders}} \{\mathcal{D}_j\} \\ 0, & b_i > m^{-i} \end{cases}$$

There are two possibilities: Bidder $i$ may over-bid or under-bid its cost $v_i$ [57].

**Case of over-bidding:** If the bid is higher than the cost, i.e., $b_i > v_i$, then there are five possibilities:

1. If $b_i < m^{-i}$, then $i$ wins the auction and $u_i = m^{-i} - v_i$. However, $i$ can get the same payoff by bidding its cost $v_i$.
2. If $v_i < m^{-i} < b_i$, then $i$ loses the auction and gets zero payoff. However, by bidding $v_i$, it can get a positive payoff.
3. If $m^{-i} < v_i$, then $i$ gets zero payoff, the same as it could have received by bidding $v_i$.
4. If $m^{-i} = b_i$ and $\mathcal{D}_i = \min\limits_{\forall j \in \text{bidders}} \{\mathcal{D}_j\}$, then $i$ gets the same payoff as by bidding $v_i$.
5. If $m^{-i} = b_i$ and $\mathcal{D}_i \neq \min\limits_{\forall j \in \text{bidders}} \{\mathcal{D}_j\}$, then $i$ loses the auction and gets zero payoff. However, by bidding $v_i$ it can get a positive payoff.

Thus, $i$ has no incentive for over-bidding.

**Case of under-bidding**: If the bid is lower than the cost, i.e., $b_i < v_i$, then there are five possibilities:

1. If $v_i < m^{-i}$, then $i$ gets payoff $m^{-i} - v_i$, the same as it could have received by bidding $v_i$.

2. If $b_i < m^{-i} < v_i$, then $i$ gets a negative payoff as $m^{-i} - v_i < 0$. However, by bidding $v_i$, $i$ gets zero payment, which is better than a negative payoff.

3. If $b_i > m^{-i}$, then $i$ gets zero payoff, the same as it could have received by bidding $v_i$.

4. If $b_i = m^{-i}$ and $\mathcal{D}_i = \min_{\forall j \in \text{bidders}} \{\mathcal{D}_j\}$, then $i$ gets $m^{-i} - v_i < 0$. However, by bidding $v_i$, $i$ gets zero payment, which is better than a negative payoff.

5. If $b_i = m^{-i}$ and $\mathcal{D}_i \neq \min_{\forall j \in \text{bidders}} \{\mathcal{D}_j\}$, then $i$ gets zero payoff, the same as it could have received by bidding $v_i$.

Thus, $i$ has no incentive for under-bidding.

In both the cases $i$'s dominant strategy is to bid its cost $v_i$, and hence, the theorem.

**Corollary 6.3.1.** *In A$^+$FS, bidding the true cost is the dominant strategy for each node.*

*Proof.* There are only two possibilities.

1. There is more than one node in $QB'$ (refer to the A$^+$FS algorithm). Only nodes in $QB'$ are qualified bidders. They may over-bid or under-bid their cost. However, from Theorem 6.3.1, their dominant strategy is to bid their true cost.

2. There is at most one node in $QB'$. In this case, the auction for $QB'$ cannot be set up. Thus the auctioneer sets up the auction as in BaFS. From Theorem 6.3.1, the dominant strategy is to bid their true cost.

**Theorem 6.3.2.** *In the unit price bid scheme along with BaFS, bidding the true cost is the dominant strategy for each bidder.*

*Proof.* Suppose a bidder $i$ with true cost $v_i$ bids with a value $b_i$, and $i$'s utility is $u_i$. We denote $\mathcal{D}(F, D) - \mathcal{D}(i, D)$ as $\delta_i$, where $F$ is the position of the auctioneer, and the minimum unit bid besides $i$ as $t^{-i}$. The utility of a node (qualified bidder) $i$ is:

$$u_i = \begin{cases} t^{-i}\delta_i - v_i, & \frac{v_i}{\delta_i} < t^{-i} \\ t^{-i}\delta_i - v_i, & \frac{v_i}{\delta_i} < t^{-i} \wedge \mathcal{D}_i = \min_{\forall j \in \text{bidders}} \{\mathcal{D}_j\} \\ 0, & \frac{v_i}{\delta_i} > t^{-i} \end{cases}$$

Bidder $i$ may over-bid or under bid its cost $v_i$ [57].

**Case of over-bidding**: The bid is higher than the cost, i.e., $b_i > v_i$, and therefore there are five possibilities:

1. If $b_i/\delta_i < t^{-i}$, then $i$ wins the auction and gets the payoff as $t^{-i}\delta_i - v_i$. However, $i$ can get the same payoff by bidding its cost $v_i$.

2. If $v_i/\delta_i < t^{-i} < b_i/\delta_i$, then $i$ loses the auction it should win and gets zero payoff. By bidding $v_i$, it can get a positive payoff.

3. If $t^{-i} < v_i/\delta_i$, then $i$ gets zero payoff, the same as it would have received by bidding its cost $v_i$.

4. If $t^{-i} = b_i/\delta_i$ and $\mathcal{D}_i = \min\limits_{\forall j \in \text{bidders}} \{\mathcal{D}_j\}$, then $i$ gets the same payoff as it would have received by bidding $v_i$.

5. If $t^{-i} = b_i/\delta_i$ and $\mathcal{D}_i \neq \min\limits_{\forall j \in \text{bidders}} \{\mathcal{D}_j\}$, then $i$ loses the auction it should win and gets zero payoff. By bidding $v_i$, it can get a positive payoff.

Thus, $i$ has no incentive for over-bidding.

**Case of under-bidding**: The bid is lower than the cost, i.e., $b_i < v_i$, and there are therefore five possibilities:

1. If $v_i/\delta_i < t^{-i}$, then $i$ gets payoff $t^{-i}\delta_i - v_i$, the same as it would have received by bidding $v_i$.

2. If $b_i/\delta_i < t^{-i} < v_i/\delta_i$, then $i$ gets a negative payoff as $t^{-i}\delta_i - v_i < 0$. By bidding $v_i$, $i$ gets zero payoff, which is better than a negative payoff.

3. If $b_i/\delta_i > t^{-i}$, then $i$ gets zero payoff, the same as it would have received by bidding $v_i$.

4. If $b_i/\delta_i = t^{-i}$ and $\mathcal{D}_i = \min\limits_{\forall j \in \text{bidder}} \{\mathcal{D}_j\}$, then $i$ gets $t^{-i}\delta_i - v_i < 0$. By bidding $v_i$, $i$ gets zero payoff, which is better than a negative payoff.

5. If $b_i/\delta_i = t^{-i}$ and $\mathcal{D}_i \neq \min\limits_{\forall j \in \text{bidder}} \{\mathcal{D}_j\}$, then $i$ gets zero payoff, the same as it could have received by bidding $v_i$.

Thus, $i$ has no incentive for under-bidding.

In both the cases $i$'s dominant strategy is to bid its cost $v_i$, and hence the theorem.

**Corollary 6.3.2.** *In the unit price bid scheme along with $A^+FS$, bidding true cost is the dominant strategy for each bidder.*

*Proof.* This follows from Theorem 6.3.2 and Corollary 6.3.1 above.

### 6.3.2 Average Progress Made Per Hop in BaFS

Intuitively, selecting the next hop towards a destination based on Euclidean distance makes more sense than basing the choice on BaFS. However, greedy distance based forwarding *does not guarantee truthfulness*. On the other hand, progress made per hop (towards the destination) is another important design issue for forwarding algorithms. Thus, we statistically analyze the average progress made per hop in BaFS.

In the basic scheme, a node $F$ selects a neighbor with the lowest bid as the next hop. Let $\mathcal{N}$ be the set of neighbors that are closer to destination $D$ than $F$, such that $m = |\mathcal{N}|$. Let $\xi$ be the random variable representing the next hop location. Then the expected next hop location is:

$$E_\xi = \sum_{i=1}^{m} p_i l_i \ .$$

where $p_i$ is the probability that node $i$ is selected as the next hop, and $l_i$ is $i$'s location. Since $i$'s cost is independent of its location, every node $i \in \mathcal{N}$ has the same probability of being the lowest bidder. Thus $p_i$ is same for all $i \in \mathcal{N}$. Thus,

$$E_\xi = \frac{1}{m} \sum_{i=1}^{m} l_i \ .$$

If $m$ is large enough, under the assumption of uniform distribution, discrete random variable $\xi$ can be considered as a continuous variable $\xi'$, then:

$$E_{\xi'} = \int l\rho(l)dl \ .$$

where $\rho(l)$ is the probability density function of location. One way to interpret this change in variable is as follows: even though $m$ is small, if such a next hop selection runs many times, from the statistical point of view, it is reasonable to use the expected continuous variable $\xi'$ to estimate the expectation of the discrete variable $\xi$.

The next question is how to calculate $\xi'$. Under the assumption of uniform distribution, $\xi'$ is the center of geometry of the shaded area in Figure 6.2. In the figure, the line connecting auctioneer $F$ and destination $D$ is set as the x-axis. Without loss of generality, let $F$ be at the origin (0,0), and $D$ be at ($R$,0). Due to the symmetry of the upper and lower parts, $\xi'$ must be located on the x-axis, i.e., its y-coordinate equals zero. We denote the radio range

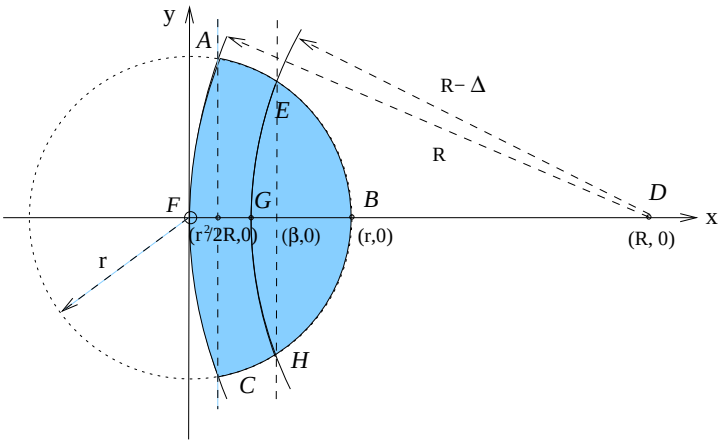



Figure 6.2: The expected position of a next hop node.

of node $F$ as $r$ and the distance from $F$ to $D$ as $R$. $\xi'$'s x-coordinate is calculated as follows:

$$\begin{aligned} E_{\xi'_x} &= \frac{\iint_{FABCF} x dxdy}{\iint_{FABCF} dxdy} \\ &= \frac{\int_0^{\frac{r^2}{2R}} x\sqrt{R^2-(x-R)^2}dx + \int_{\frac{r^2}{2R}}^{r} x\sqrt{r^2-x^2}dx}{\int_0^{\frac{r^2}{2R}} \sqrt{R^2-(x-R)^2}dx + \int_{\frac{r^2}{2R}}^{r} \sqrt{r^2-x^2}dx}. \end{aligned} \tag{6.1}$$

As both $F$ and $\xi'$ are on the x-axis, the average progress in BaFS is $|E_{\xi'_x}|$.

### 6.3.3 Average Progress Per Hop in A$^+$FS

In A$^+$FS only nodes making more progress than $AvgDst$ will be qualified as bidders. As shown in Fig. 6.2 above, only nodes in the enclosed area $EBHGE$ can be potential candidates for selection as the next hop. Statistically, the expectation of the next hop's position is the geometric center of the shaded area of $EBHGE$ in Fig. 6.2, which can be calculated in a similar way to the geometrical center of $FABCF$ if we have $G$'s x-coordinate (for convenience denoted as $\Delta$). With uniform distribution, $G$ is the geometry center of area $FABCF$. Thus $\Delta$ can be calculated from Equation 6.1. Using the value of $\Delta$, the expectation of the next hop's x-coordinate can be calculated, in a similar way:

$$\begin{aligned} E_{\xi'_x} &= \frac{\iint_{EBHGE} x dxdy}{\iint_{EBHGE} dxdy} \\ &= \frac{\int_{\Delta}^{\beta} x\sqrt{(R-\Delta)^2-(x-R)^2}dx + \int_{\beta}^{r} x\sqrt{r^2-x^2}dx}{\int_{\Delta}^{\beta} \sqrt{(R-\Delta)^2-(x-R)^2}dx + \int_{\beta}^{r} \sqrt{r^2-x^2}dx} \\ where\ \beta &= \frac{2\Delta R - \Delta^2 + r^2}{2R}. \end{aligned} \tag{6.2}$$

Thus the average progress made by A$^+$FS is $|E_{\xi_x'}|$.

Table 6.1 shows the expected progress made using the BaFS and the A$^+$FS with respect to ratio of $r/R$.

| r/R | BaFS | A$^+$FS |
|---|---|---|
| 2 | 0.466r | 0.721r |
| 3 | 0.453r | 0.702r |
| 4 | 0.446r | 0.693r |
| 5 | 0.442r | 0.687r |
| 6 | 0.439r | 0.682r |

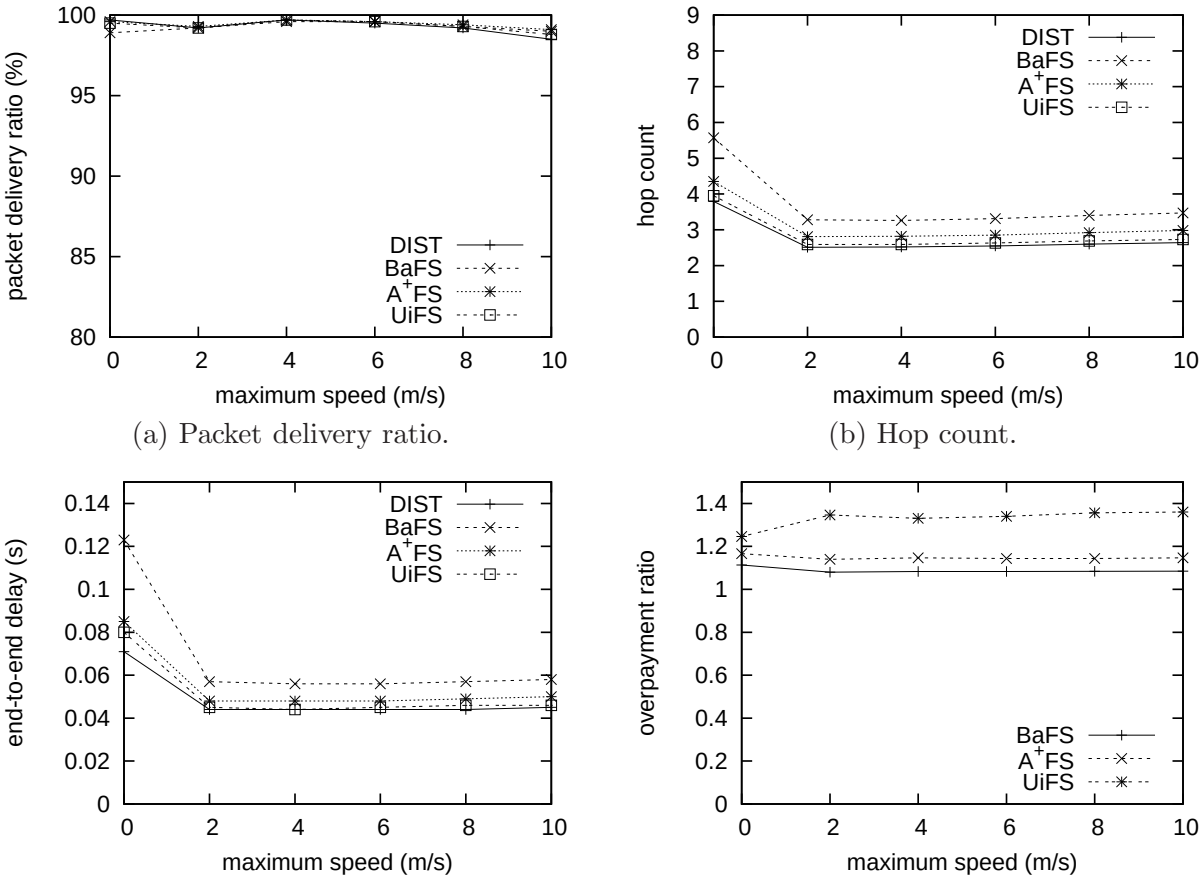
Table 6.1: The expected progress made towards a destination

### 6.4 Performance Evaluation

To evaluate the performance of proposed schemes, we conducted extensive simulations using GloMoSim [64]. Unless specified otherwise, the following parameters were used in the simulations. 100 nodes were initially placed uniformly in an area of 600m by 1500m. Nodes used a radio range of 250 meters, and picked a cost randomly between 10 and 30. All nodes followed the Random Waypoint mobility model [65] with a maximum speed of 0 m/s to 10 m/s and a pause time of 30 seconds. Each simulation lasted for 900 seconds. To generate traffic we simulated 10 CBR flows. Each flow sent four 512-byte data packets per second, starting at 120 seconds and ending at 880 seconds. Nodes used the 802.11 protocol with DCF as the MAC protocol. Data points represented on a graph were averaged over 10 simulation runs, each with a different seed. Simulations focus on five metrics, including packet delivery ratio, average hop count, average end-to-end delay, overpayment ratio and total payment, which is the total amount paid to all intermediate nodes.

TGF *does not* introduce any additional control packets in comparison to the pure geographic forwarding algorithm. In both algorithms, the overhead is only due to the *Hello* messages. TGF needs only one additional field (4 bytes) in each *Hello* message. The overhead incurred by TGF is .66 *Hello* packet per node per second.

Since TGF is the first algorithm to address truthfulness in the context of geographic forwarding, we compared results among the different schemes proposed. In the following discussion and in the graphs, UiFS represents the unit price bid scheme along with BaFS, and DIST represents distance-based greedy forwarding. Our simulation emphasizes the effect of node mobility and the number of nodes on the proposed schemes.




(a) Packet delivery ratio.

(b) Hop count.

(c) End-to-end delay.

(d) Overpayment ratio.

Figure 6.3: Performance of different schemes with respect to mobility.

### 6.4.1 Impact of Mobility

#### 6.4.1.1 Packet Delivery Ratio

Fig. 6.3(a) shows packet delivery ratio with respect to maximum node speed. All four schemes show almost identical behavior, and deliver more than 99% of packets. AT the given node density, nodes in various schemes can find feasible next hops and forward packets to destinations. As the mobility increases, the packet delivery ratio drops slightly. This is because with the fixed *Hello* message interval, neighbor information is more likely to be outdated. Thus, a supposed neighbor may have already moved out of radio range, resulting in more packets being dropped.

#### 6.4.1.2 Average Hop Count

Fig. 6.3(b) shows the average hop count for a packet to reach a destination with respect to maximum node speed. We observe that different schemes result in different hop counts, with

BaFS, A$^+$FS, UiFS and DIST in descending order (highest to lowest hop counts). DIST needs the lowest average hop count as it always selects the node closest to the destination as the next hop. BaFS needs more hops than other schemes as it does not take into account the progress (distance) when selecting the next hop. A$^+$FS needs less hops than BaFS because it selects a node that makes enough progress towards a destination as the next hop. In UiFS, the closer a node is to the destination, the lesser its unit price, and the higher the probability of the node being selected as the next hop. In other words, nodes making more progress towards the destination are more likely (depending upon their bid value) to be selected as the next hop. In all schemes, more hop counts are needed in a static network than in a mobile network. This is because mobility randomizes the position of source-destination pairs as well as the intermediate nodes, and increases network connectivity.

#### 6.4.1.3 End-to-end Delay

Fig. 6.3(c) shows end-to-end delay with respect to maximum node speed. We observe that the performance of all schemes is similar to that in Fig. 6.3(b), and the curves are identical to their corresponding parts: DIST has the shortest end-to-end delay, followed by UiFS, then A$^+$FS, with BaFS incurring the largest delay. This is because the more hops a packet travels, the more delay it incurs. Nevertheless, the delay difference between various schemes are negligible.

#### 6.4.1.4 Overpayment Ratio

Fig. 6.3(d) shows overpayment ratio with respect to maximum node speed. BaFS has the lowest overpayment ratio since it always finds the lowest bidder and pays it with the second lowest bid. A$^+$FS has a slightly higher overpayment ratio than BaFS as the bidders in this scheme constitute only a fraction of those in BaFS. UiFS incurs the highest overpayment ratio. In UiFS, the second lowest unit price multiplying the distance and progress made towards a destination, instead of the second lowest bid, will be paid to the winner. Thus, the scheme incurs a higher rate of overpayment than the other schemes.

#### 6.4.1.5 Total Payment

Fig. 6.4 shows total payment incurred with respect to maximum node speed. A$^+$FS requires the least total payment. Although its overpayment ratio is a little higher than that of BaFS, it needs less hops. The decrease in average hop count outweighs the increase in overpayment ratio, and thus the overall payment is less than the other two schemes. UiFS needs fewer

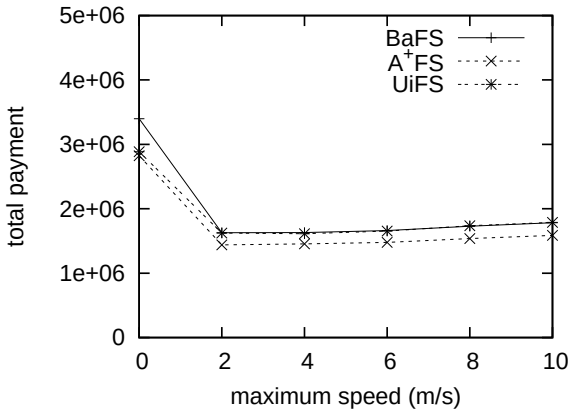



Figure 6.4: The payment of different schemes with respect to mobility

hops but incurs a higher overpayment ratio, whereas BaFS incurs the lowest overpayment ratio but needs the highest number of hops. Overall, UiFS and BaFS incur similar payments.

The above result shows that A$^+$FS has a low end-to-end delay and incurs the least total payment, and thus performs better than BaFS and UiFS from the point of view of source nodes.

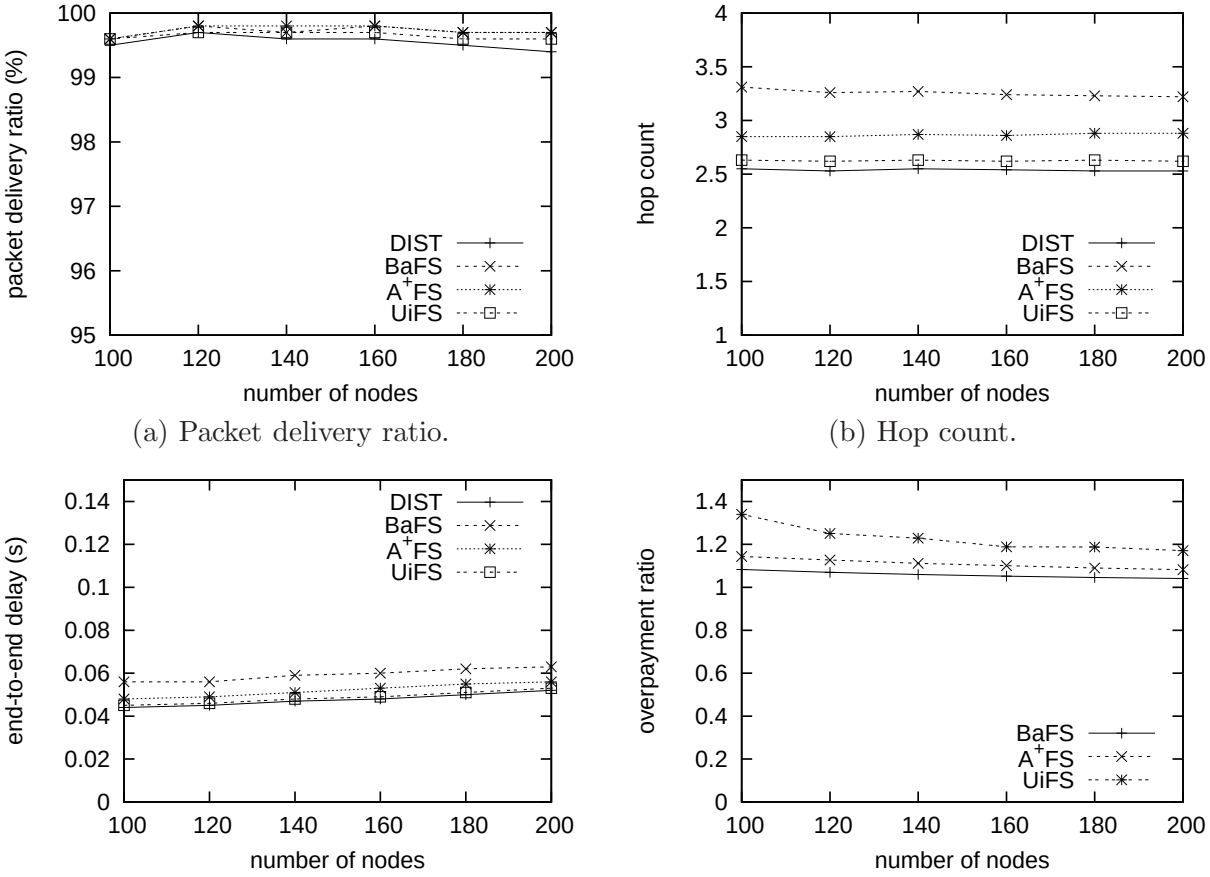
### 6.4.2 Impact of Node Density

#### 6.4.2.1 Packet Delivery Ratio

Fig. 6.5(a) shows the packet delivery ratio with respect to node density, with a maximum node speed of 6 m/s. All four schemes show almost identical behavior, and deliver more than 99.4% of packets to their destination. DIST delivers a slightly fewer packets than the other three schemes. This is because DIST always selects the node closest to the destination, and is more likely to be far away from the sender than those selected by other schemes. Under mobile scenarios, such nodes are more likely to move out of the sender's radio range, resulting in more packets being dropped.

#### 6.4.2.2 Average Hop Count

Fig. 6.5(b) shows its average hop count for a packet to reach the destination with respect to node density, with a maximum node speed of 6 m/s. We observe that different schemes result in different average hop counts, for reasons discussed in Section 6.4.1.2. As the node density increases, the average hop count incurred by DIST, A$^+$FS and UiFS change little whereas that incurred by BaFS decreases by a small amount.




(a) Packet delivery ratio.

(b) Hop count.

(c) End-to-end delay.

(d) Overpayment ratio.

Figure 6.5: Performance of different schemes with respect to node density.

#### 6.4.2.3 Average End-to-end Delay

Fig. 6.5(c) shows end-to-end delay with respect to the node density, with a maximum node speed of 6 m/s. As node density increases, the end-to-end delay increases a little for all four schemes. With a fixed *Hello* message interval, an increase in the number of nodes incurs more *Hello* messages, resulting in more contention for the radio channel. Thus, data messages are likely to need more time to attain a channel, resulting in a longer delay. Nevertheless, all these end-to-end delays are very low, as the longest one is only 0.063 seconds.

#### 6.4.2.4 Overpayment Ratio

Fig. 6.5(d) shows overpayment ratio with respect to node density, with a maximum node speed of 6 m/s. As node density increases, the overpayment ratio decreases for all three auction-based schemes, and in particular for BaFS. With a fixed cost range, an increase in

nodes is more likely to decrease the cost difference between the best bid and the second best bid, i.e., the second best bid is more likely to be close to the best bid.

#### 6.4.2.5 Total Payment

Fig. 6.6 shows the total payment incurred with respect to node density, with a maximum node speed of 6 m/s. A$^+$FS incurs the least payment, UiFS incurs a higher payment, and BaFS incurs the highest payment. For an analysis of the reason for this, see Section 6.4.1.5. As node density increases, the total payment decreases for all the three schemes. Within a fixed cost range, the greater the number of nodes (bidders), the higher the probability that the second best bid is close to the best bid, i.e., the second best bid is more likely to decrease as node density increases. As payment is determined by the second best bid, total payment will decrease.

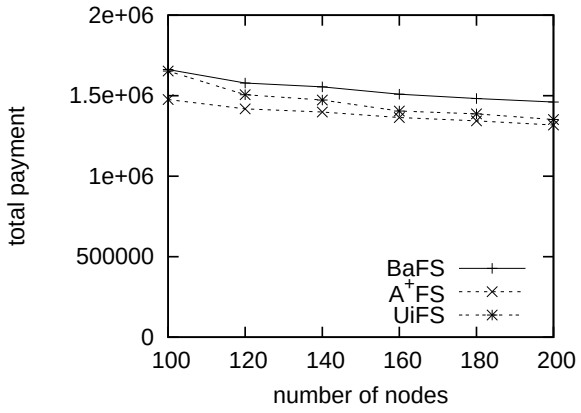



Figure 6.6: Payment of different schemes with respect to node density

## 6.5 Chapter Summary

In this chapter, we have presented TGF, a truthful protocol for geographic routing in mobile ad hoc networks with selfish nodes. TGF uses the auction-based forwarding schemes, BaFS, A$^+$FS and unit price bid scheme that selects a next hop based on the winner of the auction. To stimulate nodes ooperation, all BaFS, A$^+$FS and unit price bid schemes provide incentive to nodes to forward packets correctly, and to prevent nodes from cheating over their cost. We have theoretically proved that all three schemes guarantee truthfulness, i.e., nodes maximize their utilities only when they reveal their true cost. We also statistically analyze the average progress made per hop for BaFS and A$^+$FS. Our simulation results show that

these schemes provide a high packet delivery ratio and have a low average hop count and low end-to-end delay without compromising overpayment to the nodes.

# Chapter 7

# Conclusion

## 7.1 Conclusions

In this dissertation, node-cooperation problem in mobile ad hoc networks with selfish nodes were addressed. We explored several methods to solve this problem.

First we applied the methodology of punishing uncooperative nodes to enforce cooperation. By resorting to neighbor monitoring technology, a fair and distributed solution to detect, punish and readmit selfish nodes was presented. It emphasized providing nodes with equal opportunities to serve, and be served by others, and introduced credit as a metric for evaluating node contribution to the network. The solution greatly mitigated the location privilege problem. We further presented a light-weight detection-based solution considering battery status. This solution requires neighbor monitoring only when necessary, and nodes working in the promiscuous mode only part-time, thereby reducing the overhead and saving battery life. A voting system was introduced in both solutions to avoid the necessity of a centralized server. This system can also solve the inconsistent evaluation problem. Simulation results showed that both solutions could catch selfish node very effectively, incur low overhead and improve network performance significantly.

We then explored the methodology of rewarding nodes, specifically mechanism design methods, to motivate selfish but rational nodes to forward packets to other nodes. A low overhead truthful routing protocol (LOTTO) was first presented. It rewards nodes according to their individual cost, which may be different even for the same node according to its different neighbors. In LOTTO, we introduced a simple and effective way to collect the topological information of a network. Based on this, LOTTO can find least cost paths from source nodes to destination nodes. By applying the VCG mechanism, this protocol guarantees that nodes get enough payments and have no incentive to cheat over their cost. The most prominent feature of LOTTO is that it reduces the message overhead from $O(n^3)$ [33] to $O(n^2)$ and greatly mitigates message congestion and queue overflows in the MAC layer. Thus it has a much better performance than ad hoc-VCG.

We conducted extensive simulation studies for LOTTO, as well as for our other solutions and algorithms. To the best of our knowledge, we are the first to conduct an extensive simulation study for mechanism design methods to evaluate important network metrics such as packet delivery ratio, overhead and end-to-end delay. Our simulation results showed that LOTTO greatly outperforms ad hoc-VCG.

We further presented a light-weight scalable truthful routing protocol (LSTOP). It requires partial topological information of a network and incurs a very low overhead of $O(n)$ on average and $O(n^2)$ in the worst case. LSTOP provides near-least-cost paths and even least-cost paths with a high probability in dense networks. Simulation results showed that LSTOP achieves far a better network performance when compared to ad-hoc VCG as it generates 30-50 times less overhead, results in 2 orders of magnitude lower end-to-end delay and delivers far more packets. Moreover, it greatly reduces overall cost, and thus makes mechanism design method significant.

We also presented a generic mechanism, GTMR, that can turn any table-driven multiple routing protocol into a truthful one. By applying an auction mechanism to packet forwarding, GTMR stimulates nodes to show their true cost. Furthermore, a truthful multipath routing protocol (TMRP), as an example of GTMR, was presented. TMRP is derived from a well-known AOMDV protocol and incurs only $2n$ overhead in route discovery, without introducing new types of control messages. By far, this is the least overhead incurred in any truthful routing protocol. TMRP can also achieve load balancing without compromising truthfulness.

The selfish nodes problem, as related to location-based routing protocols is addressed here. A truthful geographic forwarding protocol (TGF) was presented. TGF utilizes three auction-based forwarding schemes to stimulate node cooperation. In all schemes the next hop node is typically the winner of the auction. We theoretically proved the truthfulness of all three schemes, and showed their performance through simulation studies. Also shown is a statistical analysis of the average progress made per hop for the two schemes. TGF is the first algorithm to address truthfulness in the context of geographic forwarding.

### 7.2 Future Work

The selfish-node problem is still an active research area and there are still many unanswered questions. In the following section, directions and areas for further research are outlined.

### 7.2.1 Detecting Byzantine Selfish Behavior

In the fair distributed and light-weight solutions, as well as other detection-based solutions, it is assumed that selfish behavior is straightforward, i.e., a node may behave selfishly from the very beginning or at a random moment. Once behaving selfishly, it continues until being detected. However, a selfish node may behave more subtly. It may behave selfishly for some time and then behave cooperatively, then selfishly again. Such behavior is termed *byzantine* selfish behavior. We propose the following scheme as a tentative solution to detect byzantine selfish behavior.

Each node is assigned *agent*s. The agents' IDs can be acquired by applying a pre-defined hash function to the node's ID. So every node can know its own agents and those of other nodes. A node's agents are used to record the lifetime of packet forwarding information of the client node. Suppose node $A$ is selected as an intermediate node for a data flow. Node $P$ and node $N$ are node $A$'s upstream node and down stream node, respectively. $P$ and $N$ should report the number of packets forwarded to $A$, $Pkt_{fwd}$, and received from $A$, $Pkt_{rcvd}$, to $A$'s agents, respectively. Reporting will be invoked in two cases. In the first case, the data flow through $A$ completes normally. Suppose a node can determine whether the flow is complete or not. At the end of the flow, $P$ and $N$ report the corresponding packet number to $A$'s agents. $A$ should also report its forwarding statistics, $Pkt_{self}^{rcvd}$ and $Pkt_{self}^{fwd}$, to its agents. In the second case, upon finding the link to $N$ broken, $A$ sends an $RERR$ message to the source. Upon receiving the $RERR$, the upstream node $P$ reports the number of packets forwarded from $A$ to $A$'s agents. Also, $N$ will report the number of packets received from $A$ upon timeout of a certain timer. Meanwhile, $A$ reports its forwarding statistics to its agents.

Upon receiving reports from the upstream node $P$ and downstream node $N$ of a client node $A$, an agent $R$ adds $Pkt_{fwd}$ and $Pkt_{rcvd}$ to $Pkt_{fwd}^{acc}$ and $Pkt_{rcvd}^{acc}$, respectively, where $Pkt_{fwd}^{acc}$ is the total number of packets sent to node $A$ and $Pkt_{rcvd}^{acc}$ is the total number of packets received by $A$'s downstream node(s). Also, $R$ calculates the accumulated self-declared number of packets from $A$, $Pkt_{self}^{acc}$. The agent checks if the deviation between $Pkt_{fwd}^{acc}$, $Pkt_{rcvd}^{acc}$ and $Pkt_{self}^{acc}$ is beyond a certain threshold. If it is, it convicts the client node as a selfish node.

A node will be equipped with multiple agents. Using multiple agents can mitigate the dropping of reports containing the forwarding statistics of the client node. It can also enhance reliability and robustness. The distribution scheme of agents should be well designed and takes the following issues into consideration. The first issue is the location

of agents. An agent close to a client is more likely to receive all reports. The second issue is node mobility, which may change initial layout substantially . The third issue is the number of agents. More agents provide more robustness against report dropping (intentionally by selfish nodes, or unintentionally due to collision). However, having more agents incurs a higher overhead for reporting. A possible solution to this is to distribute agents hierarchically, as in GLS [89] and DLM [90]. In such a scheme, a node will have $\lg N$ agents, where $N$ is the number of nodes in the network.

### 7.2.2 Detection-based Cost-efficient Method

Detection-based methods are combined with conventional routing algorithms such as DSR or AODV, where cost-efficiency is not a consideration. Due to limited battery life, which is the main reason for selfish behavior, it is necessary to use battery efficiently. Researchers have focused on this issue and proposed quite a few algorithms to reduce overall power consumption or to extend the life of the network [91, 92, 93, 94, 95, 96, 97]. These solutions use some metrics related to emitting power or power consumption information. However, it is almost impossible to accurately check if a node lies about such information, unless a truthful protocol is used. Thus some questions are raised. Is it possible to find a detection-based cost efficient method for selfish-node problem? Although they are based on incompatible philosophies, can we combine the detection-based method and the motivation-based method?

### 7.2.3 Truthful Mechanism Prevention of Node Collusion

The VCG mechanism is truthful. However, it cannot prevent the collusion of nodes, i.e., it is not *group-strategyproof*. A mechanism is group-strategyproof if not lying about its cost is its dominant strategy even if agents collude. Informally, a group-strategyproof mechanism requires that "if any agent in the group benefits from the group's collusion and lying to the mechanism, then at least one agent will lose benefit"[98]. Finding a group-strategyproof and cost-efficient mechanism is a challenge.

### 7.2.4 Truthful Mechanism Based on Cryptography

Mitchell et al. [99] observed that if we apply cryptography to every message received and sent in an Autonomous System on the Internet, then all forms of cheating can be detected. Feigenbaum and Shenker [98] provided some open questions using such a cryptography mechanism in a BGP protocol. Such an observation and open questions should be applicable to MANET. One question is "Can a protocol detect all forms of cheating without using

public key infrastructure?" [98]. Salem et al. [44] proposed a charging and rewarding scheme in multi-hop cellular networks. The scheme uses symmetrical cryptography to assist in rewarding honesty. However, it is inherent in the scheme, with the help of base stations, that all forwarding nodes get the same reimbursement. With different reimbursements to different nodes, the question remains whether cryptography alone can help to prevent cheating.

### 7.2.5 Payment Management

A complete motivation-based solution should include a payment scheme and a money management mechanism in charge of (virtual) money crediting and transferring. These two components are decoupled so that they are tractable. Most researchers in the literature focus on how to design an effective payment scheme according to different criteria, assuming an existing payment management mechanism. Only two schemes have been proposed for payment management, and both have limitations. The first solution [58, 45] assumes there is a centralized service. Nodes report their credits to the central server (or Banker node) when they are connected to it. The banker node manages the accounting of all nodes. Such a centralized service can be provided in an ad hoc network that works as an edge network of a cellular network or by access points. However, typically centralized service is not available in an ad hoc network. Thus a distributed payment management mechanism is required. Another solution [42] assumes there is a secure module which is independent of a node (user) and is tamper-proof. This hardware module can determine payment correctly and fairly. For economic reasons, it is difficult to attach such a hardware module to each node in the network.

To deploy a motivation method for the selfish-node problem in practice, a feasible payment management mechanism is a prerequisite. On designing such a mechanism, the following issues should be considered.

- Who assumes the responsibility of accounting and transferring credit? In edge networks, access points or base stations are natural choices. However, for other ad hoc networks, no single node can work as an accounting server. A possible solution is to distribute the responsibility to a subset of nodes, as in the threshold cryptography key management scheme [100]. As discussed in [101], such a task sharing scheme is not suitable for civilian networks, since nodes in these networks are selfish, and have no interest in taking on added responsibilities. However, with additional awards,

these nodes may have the incentives to do such a job. One issue for such a distributed solution is that it should not incur a high overhead.

- How is the credit to be accounted for? This does not pose a problem in a single server scheme. However, in a distributed scheme, accounting coordination is an important issue due to possible loss of payment-related control messages.

- What should the money format be? How can it be turned into real currency? Intuitively, virtual currency should be introduced for virtual money circulation. Such virtual currency could be official tokens signed by certain authorities.

- When is money paid? The money can be distributed immediately at the end of the forwarding service, or upon request.

**Bibliography**


[1] C. Silva Ram Murthy and C. Siva Ram. *Ad Hoc wireless networks : architectures and protocols*. Prentice Hall PTR, 2004.

[2] H. Luo, R. Ramjee, P. Sinha, L. Li, and S. Lu. Ucan: A unified cellular and ad-hoc network architecture. In *Proceedings of MobiCom'03*, 2003.

[3] F. A. Tobagi and L. Kleinrock. Packet switching in radio channels: Part ii - the hidden terminal problem in carrier sense multiple-access modes and the busy-tone solution. *IEEE Transaction on Communication*, 23(12):1417–1433, 1975.

[4] Chane L. Fullmer and J. J. Garcia-Luna-Aceves. Solutions to hidden terminal problems in wireless networks. *SIGCOMM Comput. Commun. Rev.*, 27(4):39–49, 1997.

[5] C. K Toh. *Ad Hoc Mobile Wireless Networks: Protocols and System*. Prentice Hall PTR, 2002.

[6] Rendong Bai and M. Singhal. Doa: Dsr over aodv routing for mobile ad-hoc networks. *IEEE Transactions on Mobile Computing*, 5(10):1403–1416, 2006.

[7] Athanasios Bamis, Azzedine Boukerche, Ioannis Chatzigiannakis, and Sotiris Nikoletseas. A mobility sensitive approach for efficient routing in ad hoc mobile networks. In *MSWiM '06: Proceedings of the 9th ACM International Symposium on Modeling Analysis and Simulation of Wireless and Mobile Systems*, pages 357–364, New York, NY, USA, 2006. ACM Press.

[8] Kwan-Wu Chin, John Judge, Aidan Williams, and Roger Kermode. Implementation experience with manet routing protocols. *SIGCOMM Comput. Commun. Rev.*, 32(5):49–59, 2002.

[9] Douglas S. J. De Couto, Daniel Aguayo, John Bicket, and Robert Morris. A high-throughput path metric for multi-hop wireless routing. In *MobiCom '03*, pages 134–146, New York, NY, USA, 2003. ACM Press.

[10] Venkata C. Giruka, Mukesh Singhal, and Siva Prasad Yarravarapu. A path compression technique for on-demand ad-hoc routing protocols. In *MASS '04: The 1st IEEE International Conference on Mobile Ad-hoc and Sensor Systems*, Oct 2004.

[11] Tom Goff, Nael Abu-Ghazaleh, Dhananjay Phatak, and Ridvan Kahvecioglu. Preemptive routing in ad hoc networks. *Journal of Parallel and Distributed Computing*, 63:123–140, 2003.

[12] Zygmunt J. Haas and Marc R. Pearlman. The performance of query control schemes for the zone routing protocol. In *SIGCOMM '98: Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 167–177, New York, NY, USA, 1998. ACM Press.

[13] David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. *Mobile Computing*, pages 153–181, 1996.

[14] Sung-Ju Lee, Elizabeth M. Belding-Royer, and Charles E. Perkins. Scalability study of the ad hoc on-demand distance vector routing protocol. *Int. J. Netw. Manag.*, 13(2):97–114, 2003.

[15] Mahesh K. Marina and Samir R. Das. Routing performance in the presence of unidirectional links in multihop wireless networks. In *MobiHoc '02: Proceedings of the 3rd ACM International Symposium on Mobile Ad Hoc Networking & Computing*, pages 12–23, New York, NY, USA, 2002. ACM Press.

[16] Vincent D. Park and M. Scott Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *INFOCOM '97: Proceedings of the INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution*, page 1405. IEEE Computer Society, 1997.

[17] Guangyu Pei, Mario Gerla, and Xiaoyan Hong. Lanmar: landmark routing for large scale wireless ad hoc networks with group mobility. In *MobiHoc '00*, pages 11–18, Piscataway, NJ, USA, 2000. IEEE Press.

[18] Guangyu Pei, Mario Gerla, Xiaoyan Hong, and Ching-Chuan Chiang. A wireless hierarchical routing protocol with group mobility. In *IEEE Wireless Communications and Networking Conference, WCNC1999*, number 1, pages 1538–1542, New Orleans, LA, USA, September 1999. IEEE, IEEE.

[19] C. Perkins. Ad-hoc on-demand distance vector routing, Nov 1997. In Internet draft RFC.

[20] Charles E. Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. In *SIGCOMM '94: Proceedings of the conference on Communications architectures, protocols and applications*, pages 234–244. ACM Press, 1994.

[21] Prasun Sinha, Raghupathy Sivakumar, and Vaduvur Bharghavan. Cedar: Core extraction distributed ad hoc routing. In *Infocom '99*, pages 202–209, New York, NY, USA, March 1999. IEEE.

[22] William Su, Sung-Ju Lee, and Mario Gerla. Mobility prediction and routing in ad hoc wireless networks. *Int. J. Netw. Manag.*, 11(1):3–30, 2001.

[23] Y.-B. Ko and N. H. Vaidya. Location-aided routing(lar) in mobile ad hoc networks. In *Proceedings of the Fourth annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 66–75, October 1998.

[24] Stefano Basagni, Imrich Chlamtac, Violet R. Syrotiuk, and Barry A. Woodward. A distance routing effect algorithm for mobility (dream). In *MobiCom '98: Proceedings of the 4th annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 76–84, New York, NY, USA, 1998. ACM Press.

[25] Brad Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In *Proceeding of the Sixth annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 243–254, 2000.

[26] H. Takagi and L. Kleinrock. Optimal transmission range for randomly distributed packet radio terminals. *IEEE Transactions on Communications*, 32(3):246–257, 1984.

[27] Lali Barriere, Pierre Fraigniaud, and Lata Narayanan. Robust position-based routing in wireless ad hoc networks with unstable transmission ranges. In *DIALM '01: Proceedings of the 5th International Workshop on Discrete Algorithms and methods for Mobile Computing and Communications*, pages 19–27, New York, NY, USA, 2001. ACM Press.

[28] J. C. Navas and T. Imielinski. Geographic addressing and routing. In *Proceeding of the third annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, September 1997.

[29] Seungjoon Lee, Bobby Bhattacharjee, and Suman Banerjee. Efficient geographic routing in multihop wireless networks. In *MobiHoc '05*, pages 230–241, New York, NY, USA, 2005. ACM Press.

[30] Yongwei Wang, Venkata C. Giruka, and Mukesh Singhal. A fair distributed solution for selfish node problem in mobile ad hoc networks. In *Proceedings of 3rd International Conference on AD-HOC Networks & Wireless (ADHOCNOW 2004)*, pages 211–224, 2004.

[31] Yongwei Wang and Mukesh Singhal. A light-weight solution for selfish nodes problem considering battery status in ad-hoc networks. In *Proceedings of IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob) 2005*, Montreal, Canada, August 2005.

[32] Yongwei Wang and Mukesh Singhal. On improving the efficiency of truthful routing in manets with selfish nodes. *Pervasive and Mobile Computing, Elsevier*, 3(5):537–559, October 2007.

[33] Luzi Anderegg and Stephan Eidenbenz. Ad hoc-vcg: a truthful and cost-efficient routing protocol for mobile ad hoc networks with selfish agents. In *MobiCom '03: Proceedings of the 9th Annual International Conference on Mobile Computing and Networking*, pages 245–259, New York, NY, USA, 2003. ACM Press.

[34] Yongwei Wang and Mukesh Singhal. Lstop: A light-weight scalable truthful routing protocol in manets with selfish nodes. In *Proceedings of 5th International Conference of Ad-Hoc Networks & Wireless (ADHOC-NOW 2006)*, pages 280–293, Ottawa, Canada, August 2006.

[35] Yongwei Wang and Mukesh Singhal. A light-weight scalable truthful routing protocol in manets with selfish nodes. *International Journal of Ad Hoc and Ubiquitous Computing*. to appear.

[36] Yongwei Wang and Mukesh Singhal. Truthful multipath routing for ad-hoc networks with selfish nodes. *Journal of Parallel and Distributed Computing*. accepted.

[37] Yongwei Wang, Venkata C. Giruka, and Mukesh Singhal. A truthful geographic forwarding algorithm for ad-hoc networks with selfish nodes. *International Journal of Network Security*, 5(3):252–263, November 2007.

[38] Jianfeng Cai and Udo Pooch. Play alone or together-truthful and efficient routing in wireless ad hoc networks with selfish nodes. In *Proceeding of The 1st IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS'04)*, 2004.

[39] Kai Chen and Klara Nahrstedt. ipass: an incentive compatible auction scheme to enable packet forwarding service in manet. In *Proceedings of The 24rd International Conference on Distributed Computing Systems (ICDCS'04)*, 2004.

[40] Weizhao Wang and XiangYang Li. Truthful low cost unicast in selfish wireless networks. In *Proceedings of 18th International Parallel and Distributed Processing Symposium (IPDPS'04) - Workshop 12*, 2004.

[41] S. Zhong, L. Li, Y. Liu, and Y. R. Yang. On designing incentive-compatible routing and forwarding protocols in wireless ad-hoc networks. Technical report, Yale University, 2004.

[42] Levente Buttyan and Jean-Pierre Hubaux. Enforcing service availability in mobile ad-hoc wans. In *Proceedings of MobiHOC'00*, 2000.

[43] L. Buttyan and J. Hubaux. Stimulating cooperation in self-organizing mobile ad hoc networks. *ACM/Kluwer Mobile Networks and Applications*, 8(5), 2003.

[44] N. B. Salem, L. Buttyan, J. Hubaus, and M. Jakobsson. A charging and rewarding scheme for packet forwarding in multi-hop cellular networks. In *Proceedings of MobiHoc'03*, 2003.

[45] E. Fratkin, V. Vijayaraghavan, Y. Liu, D. Gutierrez, TM Li, and M. Baker. Participation incentives for ad hoc networks. http://www.stanford.edu/ yl314/ape/paper.ps.

[46] Pietro Michiardi and Refik Molva. Prevention of denial of service attacks and selfishness in mobile ad hoc networks, January 2002. Research Report RR-02-063.

[47] Pietro Michiardi and Refik Molva. Core: A collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks. In *Proceedings of Communication and Multimedia Security 2002 Conference*, 2002.

[48] S. Buchegger and J. Y. Le-Boudec. Nodes bearing grudges: Towards routing security, fairness, and robustness in mobile ad hoc networks. In *Proceedings of EUROMICRO-PDP'02*, 2002.

[49] S. Buchegger and J. Y. Le-Boudec. Performance analysis of the confidant protocol (cooperation of nodes: Fairness in dynamic ad-hoc networks). In *Proceedings of MobiHOC'02*, June 2002.

[50] S. Marti, T. J. Giuli, K. Lai, and M. Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *Proceedings of Mobicom'00*, August 2000.

[51] Hugo Miranda and Luis Rodrigues. Preventing selfishness in open mobile ad hoc networks. In *Proceedings of 7th CaberNet Radicals Workshop*, October 2002.

[52] K. Paul and D. Westhoff. Context aware detection of selfish nodes in dsr based ad-hoc networks. In *Proceedings of IEEE Vehicular Technology Conference'02*, 2002.

[53] Jean-Pierre Hubaux and Levente Buttyan et al. Toward mobile ad-hoc wans: Terminodes. Technical report, Swiss Federal Institute of Technology, Lausanne, July 2000. Technical Report No. DSC/2000/006.

[54] J. Crowcroft, R. Gibbens, F. Kelly, and S. Ostring. Modelling incentives for collaboration in mobile ad hoc networks. In *Proceedings of WiOpt'03: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networksa*, 2003.

[55] Roger B. Myerson. *Game theory : analysis of conflict.* Harvard University Press, 1997.

[56] Tatsuro Ichiishi, Abraham Neyman, and Yair Tauman. *Game theory and applications.* San Diego :Academic Press, 1990.

[57] Charalambos D. Aliprantis and Subir K. Chakrabarti. *Games and Decision Making.* Oxford University Press, 2000.

[58] S. Zhong, Y. R. Yang, and J. Chen. Sprite: A simple, cheat-proof, credit-based system for mobile ad-hoc networks. In *Proceedings of Infocom 2003*, Mar 2003.

[59] V. Srinivasan, P. Nuggehalli, C. Chiasserini, and R. Rao. Cooperation in wireless ad hoc networks. In *Proceedings of IEEE Infocom'03*, 2003.

[60] Pietro Michiardi and Refik Molva. A game theoretical approach to evaluate cooperation enforcement mechanisms in mobile ad hoc networks. In *Proceedings of WiOpt'03: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, 2003.

[61] Stephan Eidenbenz, Giovanni Resta, and Paolo Santi. Commit: A sender-centric truthful and energy-efficient routing protocol for ad hoc networks with selfish nodes. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 12*, page 239.2, Washington, DC, USA, 2005. IEEE Computer Society.

[62] N. Nisan and A. Ronen. Algorithmic mechanism design. *Games and Economic Behavior*, 35:166–196, 2001.

[63] T. H. Clutton-Brock and G. A. Parker. Punishment in animal societies. *Nature*, 373:209–216, January 1995.

[64] http://pcl.cs.ucla.edu/projects/glomosim/.

[65] Tracy Camp, Jeff Boleng, and Vanessa Davies. A survey of mobility models for ad hoc network research. *Wireless Communication and Mobile Computing (WCMC): Special Issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, 2(5):483–502, 2002.

[66] Laura Marie Feeney. An energy consumption model for performance analysis of routing protocols for mobile ad hoc networks. *Mobile Networks and Application*, 6, 2001.

[67] David Parkes. Chapter 2 classic mechanism design. *Iterative Combinatorial Auctions: Achieving Economic and Computational Efficiency*, May 2001. Ph.D. dissertation, University of Pennsylvania.

[68] W. Vickrey. Counterspeculation, auctions and competitive sealed tenders. *Journal of Finance*, 16:8–37, 1960.

[69] E. Clarke. Multipart pricing of public goods. *Public Choice*, 11:17–33, 1971.

[70] T. Groves. Incentives in teams. *Econometrica*, 41:617–663, 1973.

[71] J. Feigenbaum, C. Papadimitriou, R. Sami, and S. Shenker. A bgp-based mechanism for lowest-cost routing. In *Proceedings of the 2002 ACM Symposium on Principles of Distributed Computing*, pages 173–182, 2002.

[72] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. stein. *Introduction to algorithms.* McGraw-Hill Higher Education, 2001.

[73] S. Zhong, L. Li, Y. G. Liu, and Y. R. Yang. On designing incentive-compatible routing and forwarding protocols in wireless ad-hoc networks - an integrated approach using game theoretical and cryptographic techniques. In *Proceedings of MobiCom'05*, 2005.

[74] Mineo Takai, Jay Martin, and Rajive Bagrodia. Effects of wireless physical layer modeling in mobile ad hoc networks. In *Proceedings of ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHOC'01)*, Long Beach, California, Oct 2001.

[75] Mineo Takai, Jay Martin, and Rajive Bagrodia. Effects of wireless physical layer modeling in mobile ad hoc networks. In *Proceedings of MobiHoc'01*, 2001.

[76] J. Raju and J. Garcia-Luna-Aceves. A new approach to on-demand loop-free multipath routing. In *Proceedings of the 8 th Annual IEEE International Conference on Computer Communications and Networks (ICCCN)*, pages 522–527, Boston, MA, Oct 1999.

[77] M. K. Marina and S. R. Das. Ad hoc on-demand multipath distance vector (aomdv) routing. In *Proceedings of the 9th International Conference on Network Protocols (ICNP)*, Nov. 2001.

[78] S. Zhong, Y. R. Yang, and J. Chen. Sprite: A simple, cheat-proof, credit-based system for mobile ad-hoc networks. In *Proceedings of Infocom'03*, Mar 2003.

[79] I. Stojmenovic. Position based routing in ad hoc networks. *IEEE Communication Magazine*, 7(40), 2002.

[80] Martin Mauve, J02rg Widmer, and Hannes Hartenstein. A Survey on Position-Based Routing in Mobile Ad-Hoc Networks. *IEEE Network Magazine*, 15(6):30–39, Nov 2001.

[81] L. Blazevic, L. Buttyan, S. Capkun, S. Giordano, J. Hubaux, and J. Le Boudec. Self-organization in mobile ad-hoc networks: the approach of terminodes. *IEEE Communications Magazine*, 2001.

[82] R. Morris, J. Jannotti, F. Kaashoek, J. Li, and D. De Couto. Carnet: A scalable ad hoc wireless network system. In *Proceedings of the 9th ACM SIGOPS European workshop: Beyond the PC: New Challenges for the Operating System*, september 2000.

[83] T. Hou and V. Li. Transmission range control in multihop packet radio networks. *IEEE Transactions on Communications*, 34(1):38–44, 1986.

[84] Evangelos Kranakis, Harvinder Singh, and Jorge Urrutia. Compass routing on geometric networks. In *Proceeding of 11th Canadian Conference on Computational Geometry*, pages 51–54, Vancouver, August 1999.

[85] Jinyang Li, John Jannotti, Douglas S. J. De Couto, David R. Karger, and Robert Morris. A scalable location service for geographic ad hoc routing. In *MobiCom '00: Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, pages 120–130, New York, NY, USA, 2000. ACM Press.

[86] Yuan Xue, Baochun Li, and Klara Nahrstedt. A scalable location management scheme in mobile ad-hoc networks. In *LCN '01: Proceedings of the 26th Annual IEEE Conference on Local Computer Networks*, page 102, Washington, DC, USA, 2001. IEEE Computer Society.

[87] Zygmunt J. Haas and Ben Liang. Ad hoc mobility management with uniform quorum systems. *IEEE/ACM Trans. Netw.*, 7(2):228–240, 1999.

[88] S. Giordano and M. Hamdi. Mobility management: The virtual home region, October 1999. EPFL, Lausanne, Switzerland, Tech. Rep. SSC/1999/037.

[89] J. Li, J. Jannotti, D. De Couto, D. Karger, and R. Morris. A scalable location service for geographic ad-hoc routing. In *Proceedings of the 6th ACM International Conference on Mobile Computing and Networking (MobiCom '00)*, pages 120–130, Aug 2000.

[90] Y. Xue, B. Li, and K. Nahrstedt. A scalable location management scheme in mobile ad-hoc networks. In *Proceeding of IEEE Conference on Local Computer Networks (LCN)*, 2001.

[91] Chansu Yu, Ben Lee, and Hee Yong Youn. Energy efficient routing protocols for mobile ad hoc networks. *Wireless Communications and Mobile Computing Journal*, 3(8):959 – 973, 2003.

[92] Ahmed M. Safwat, Hossam S. Hassanein, and Hussein T. Mouftah. Power-aware wireless mobile ad hoc networks. *The Handbook of Ad Hoc Wireless Networks*, pages 377 – 386, 2003.

[93] Suresh Singh, Mike Woo, and C. S. Raghavendra. Power-aware routing in mobile ad hoc networks. In *MobiCom '98: Proceedings of the 4th annual ACM/IEEE*

*International Conference on Mobile Computing and Networking*, pages 181–190, New York, NY, USA, 1998. ACM Press.

[94] C. Toh. Maximum battery life routing to support ubiquitous mobile computing in wireless ad hoc networks. *IEEE Communications Magazine*, June 2001.

[95] Qun Li, Javed A. Aslam, and Daniela Rus. Online power-aware routing in wireless ad-hoc networks. In *Mobile Computing and Networking*, pages 97–107, 2001.

[96] M. Maleki, K. Dantu, and M. Pedram. Power-aware source routing protocol for mobile ad hoc networks. In *International Symposium on Low Power Electronics and Design*, pages 72–75, August 2002.

[97] Ivan Stojmenovic and Xu Lin. Power-aware localized routing in wireless networks. *IEEE Transactions on Parallel and Distributed Systems*, 12(11):1122–1133, 2001.

[98] Joan Feigenbaum and Scott Shenker. Distributed algorithmic mechanism design: Recent results and future directions. In *Proceedings of Sixth International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (Dial-M'02)*, September 2002.

[99] J. Mitchell, R. Sami, K. Talwar, and V. Teague. Private communication, 2001.

[100] Lidong Zhou and Z. J. Hass. Securing ad hoc network. *IEEE network, special issue on network security*, November/December 1999.

[101] J. Hubaux, L. Buttyan, and S. Capkun. The quest for security in mobile ad hoc networks. In *Proceeding of the ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHOC01)*, pages 146–155, 2001.

**Vita**

1. Background.
   (a) Date of Birth: 10/04/1967
   (b) Place of Birth: Sichuan, P.R.China
2. Academic Degrees.
   (a) M.S., May 2006 Department of Computer Science, University of Kentucky, Lexington, Kentucky
   (b) M.E, July 1992 Department of Mechanical Engineering, Sichuan University, Sichuan, P.R.China
   (c) B.E., July 1989 Department of Mechanical Engineering, Sichuan University, Sichuan, P.R.China
3. Professional Experience.
   (a) 2002-2007, Research Assistant, Department of Computer Science, University of Kentucky
   (b) 2001-2002, Teaching Assistant, Department of Computer Science, University of Kentucky
   (c) Served as a reviewer for the following journals:
   IEEE Transactions on Computers (TC)
   IEEE Transactions on Parallel and Distributed Systems (TPDS)
   Elsevier Pervasive and Mobile Computing Journal (PMC)
   Computer Networks Journal (Elsevier)
   Ad Hoc Networks Journal (Elsevier)
   International Journal of Wireless Information Networks (IJWIN)
   Wiley's Wireless Communications and Mobile Computing Journal

(d) Served as a reviewer for the following conferences:
International Conference on Distributed Computing Systems (ICDCS)
IEEE International Conference on Pervasive Computing and Communications (PerCom)
IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS)
IEEE International Symposium on Reliable Distributed Systems (SRDS)
International Conference on Parallel Processing (ICPP)

4. Publications.

(a) Yongwei Wang, Venkata C. Giruka and Mukesh Singhal, "Truthful Multipath Routing for Ad-hoc Networks with Selfish Nodes", Journal of Parallel and Distributed Computing, accepted.

(b) Yongwei Wang and Mukesh Singhal, "A Light-weight Scalable Truthful Routing Protocol in MANETs with Selfish Nodes", International Journal of Ad Hoc and Ubiquitous Computing, to appear.

(c) Yongwei Wang and Mukesh Singhal, "On Improving the Efficiency of Truthful Routing in MANETs with Selfish Nodes", Journal of Pervasive and Mobile Computing, Elsevier, 3(5):537-559, October 2007.

(d) Yongwei Wang, Venkata C. Giruka and Mukesh Singhal, "A Truthful Geographic Forwarding Algorithm for Ad-hoc Networks with Selfish Nodes", Internal Journal of Network Security, 5(3):252-263, November 2007.

(e) Yongwei Wang and Mukesh Singhal, "LSTOP: A Light-weight Scalable Truthful Routing Protocol in MANETs with Selfish Nodes", In Proceedings of 5th International Conference of Ad-Hoc, Mobile, and Wireless Networks (ADHOC-NOW) 2 006, Ottawa, Canada, Aug 2006, (Springer LNCS 4104), pages 280-293.

(f) Yongwei Wang and Mukesh Singhal, "A Light-weight Solution for Selfish Nodes Problem Considering Battery Status in Ad-Hoc Networks", In Proceedings of IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob) 2005, Aug 2005, Montreal, Canada

(g) Yongwei Wang, Venkata C. Giruka and Mukesh Singhal, "A Fair Distributed Solution for Selfish Nodes Problem in Wireless Ad Hoc Networks", In Proceedings of 3th International Conference of Ad-Hoc, Mobile, and Wireless Networks (ADHOC-NOW) 2004, Vancouver, Canada, Jul 2004, (Springer LNCS 3158), pages 211-224.

(h) Venkata C. Giruka, Yongwei Wang and Mukesh Singhal, "A Secure Position-based Protocol Framework for Wireless Multi-hop Networks", In Proceedings of IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob) 2007, October 2007.

(i) Rendong Bai, Mukesh Singhal, Yongwei Wang, "On Supporting High-Throughput Routing Metrics in On-Demand Routing Protocols for Multi-Hop Wireless Networks", Journal of Parallel and Distributed Computing, 5(10):1403-1406, 2006.