

Creating Motives for Cooperation to Achieve High Throughput in Wireless Ad Hoc Networks

Vasilios A. Siris and Chariklia A. Athanasopoulou

Institute of Computer Science, FORTH and
Department of Computer Science, University of Crete
P.O. Box 1385, GR 711 10 Heraklion, Crete, Greece
{vsiris,athanas}@ics.forth.gr

Abstract

In wireless ad hoc networks, nodes that are not within the same transmission range communicate using intermediate nodes as relays. Due to their limited battery life, wireless nodes can decide to deny incoming relay requests, in which case the network's connectivity and aggregate throughput decreases. In this paper, we describe two algorithms that create motives for cooperation among nodes in an ad hoc network. The first algorithm, called neighbor GTFT (N-GTFT) is an extension of the GTFT (Generous Tit-For-Tat) algorithm presented in [1], according to which a node relays packets for a particular neighbor based on the amount of help it has received from that neighbor. The second algorithm maintains the difference between the amount of received and the amount of given help within bounds defined by a token bucket, and yields higher throughput in the case of bursty traffic. Both algorithms can be extended to a weighted version, which is appropriate when the destinations are not uniformly distributed, as in the case where most nodes send traffic to an access point.

1. Introduction

Ad hoc networks are attractive, since they can provide a high level of connectivity without the need of a fixed infrastructure. A mobile ad hoc network is a wireless multi-hop network that is formed by a set of possibly battery-constrained nodes. These nodes should self-organize themselves through cooperation, since the network lacks any pre-existing infrastructure, and nodes not within the same transmission range can communicate only using intermediate nodes as relays.

The limited battery life of mobile nodes can be a significant constraint, which affects the behavior of a

node as an intermediate relay. Hence, due to its battery constraint, a node may adopt a selfish behavior and save battery power by rejecting all relay requests it receives. If all nodes in the network adopt such a behavior, then the level of connectivity hence the total throughput of the network would be dramatically reduced. The only way to solve this problem is to apply methods that generate motives for cooperation among the nodes.

A number of researchers have investigated cooperation strategies among nodes in an ad hoc network, e.g. see [1,2,3,4,5,6,7,8] and the references therein. More specifically, in [1] the authors first determine the optimal throughput each node should receive given some lifetime constraints and the assumption of rational node behavior; this value corresponds to the rational Pareto optimal operating point. Then a distributed algorithm called Generous Tit-For-Tat (GTFT) is proposed to achieve the target throughput. Moreover, the authors show that GTFT results in a Nash equilibrium. In [2] the authors investigate the trade-off between the energy consumption and the blocking probability of a session, and study the ability of the network to guarantee low energy consumption for nodes that want or need to be selfish. Other work, such as [3], present solutions based on differentiation of the forwarding function using reputation techniques. Finally, the use of economic incentives is explored in papers such as [4,8].

In this paper, we first consider the GTFT algorithm described in [1]. According to this algorithm, a node decides whether to accept or reject a relay request based on the comparison between the total amount of help the node has received by all other nodes in the network and the total amount of help it has given to the other nodes. The amount of help received and offered is expressed as the percentage of relay requests that are accepted. An issue with this approach is that a self-

ish node, which does not accept relay requests, can influence the whole ad hoc network, and lead to a reduction of the throughput for the whole network. To address this issue, we propose a simple modification of the GTFT algorithm, whereby a node differentiates the amount of help it has received from different neighboring nodes. According to the modified algorithm, which we will refer to as neighbor GTFT (N-GTFT), a node accepts or denies a relay request from a neighboring node based on the total amount of help the node has received from this node and the total amount of help it has given to this node. Hence, the N-GTFT algorithm maintains more state compared to the original GTFT algorithm, which is used to identify and isolate selfish non-cooperating neighboring nodes.

The second algorithm we present maintains the difference between the amount of help received and the amount of help given within bounds defined by a token bucket, hence we will refer to it as the Bucket algorithm. According to the Bucket algorithm, a node adds tokens to a bucket each time it makes or receives relay requests if the percentage of its relay requests that have been accepted by other nodes is higher than the percentage of relay requests by other nodes that it has accepted. On the other hand, tokens are removed from the bucket each time it makes or receives relay requests if the percentage of its relay requests that have been accepted by other nodes is less than the percentage of relay requests by other nodes that it has accepted. The decision on whether to accept a relay request is based on the bucket contents: if there are tokens in the bucket, then the request is accepted; on the other hand, if the bucket is empty, then the request is rejected. The motivation for the Bucket algorithm is the following: if a node has received help from the other nodes, which is greater than the amount of help it has offered to other nodes, for some amount of time, then the bucket will obtain a positive value. If at some time instance, the percentage with which a node's relay requests are accepted by other nodes is smaller than the rate with which that node accepts relay request by other nodes, then the node can still accept relay requests, provided the bucket is non-empty. On the other hand, with the GTFT algorithm the node would not accept requests.

Finally, we present a weighted version of the GTFT and Bucket algorithm, where the percentage of help received and given is multiplied by a weight. Such an approach is appropriate when the destination of packets is not uniformly distributed, as in the case where a large percentage of packets are destined for an access point; in this case, nodes closer to an access point should relay packets at a higher rate compared to nodes far away

from the access point.

The rest of the paper is organized as follows. In Section 2 we describe the GTFT algorithm. In Section 3 we describe the the two new algorithms: the G-TFT and the Bucket algorithms. In Section 4 we present and discuss the results of simulations, and in Section 5 we conclude the paper.

2. Generous Tit-For-Tat (GTFT)

The Generous Tit-For-Tat (GTFT) algorithm [1], tends to equalize the percentage of requests that are served by a node j for all other nodes, with the percentage of requests that are served by other nodes for node j . The amount of help given and the amount of help received is expressed by the following ratios:

$$\text{help given} = \frac{\# \text{ of requests handled for others}}{\# \text{ of requests made to me by others}} \quad (1)$$

and

$$\text{help received} = \frac{\# \text{ of requests handled for me}}{\# \text{ of requests made by me to others}} \quad (2)$$

The condition that a node checks in order to decide whether to accept or deny a relay request is

$$\text{help received} + \text{gen} > \text{help given} \quad (3)$$

where gen is a small positive number used to avoid deadlocks during the network startup. Next we describe the actions performed by all nodes on reception of a relay request packet.

Relay Request packet arrival

When a relay request packet is received by node j , the node checks condition (3):

If condition (3) is true, the relay request is accepted and

- if there is a next relay after node j , then the relay request packet is forwarded to the next node.
- else, if there is no next relay after node j , then node j sends a positive answer reply to the node from which the relay request originated.

Else, if condition (3) is false, the relay request is not accepted and

- node j sends a negative answer reply to the node from which the relay request originated.

Relay Reply packet arrival

When node j receives a relay reply packet, then it checks the answer field and performs the following:

If node j is the source of the request:

- If the answer is positive (i.e. the request is accepted) node j increments the counter (*# of requests handled for me by others*) and initiates packet transmission.
- Else, if the answer is negative (i.e. the request is rejected), then node j updates its counters, without initiating data transfer.

Else, if node j is not the source of the request:

- If the answer is positive (i.e. the request is accepted), node j increments the counter (*# of requests handled for others*) by one and forwards the reply packet to the upstream node.
- Else, if the answer is negative (i.e. the request is rejected) node j just forwards the reply packet to the upstream node.

3. Algorithms for motivating cooperation to achieve high throughput

3.1. Neighbor GTFT (N-GTFT)

This algorithm is a simple extension of the GTFT algorithm. Specifically, a node differentiates the amount of help it has received from different neighboring nodes. A node accepts or denies a relay request from a neighboring node based on the total amount of help the node has received from this neighbor and the total amount of help it has given to this neighbor; as with the original GTFT algorithm, the amount of help received and offered is expressed as the percentage of relay requests that are accepted. Hence, with the N-GTFT algorithm a node maintains four counters for each of its neighbors, whereas in the original GTFT algorithm a node maintains a total of four counters. The additional information is used by a node to identify selfish non-cooperating neighboring nodes, and isolate them by not letting their behavior influence how the node behaves to its other (cooperating) neighbors. The N-GTFT algorithm is appropriate when nodes are not highly mobile, in which case a node's neighbors are relatively static.

According to the N-GTFT algorithm, if a node has n neighbors, then for each neighbor k , $1 \leq k \leq n$, it holds the following four counters:

(*# of requests handled for k*), (*# of requests made to me by k*)

and

(*# of requests handled for me by k*), (*# of requests made by me to k*)

When node j receives a relay request, it examines from which of its neighbors the request has come from. If we assume that the request has come from neighbor k , the relay request is accepted if the following condition is true:

$$HR_{j,k} + gen > HG_{j,k} \quad (4)$$

where the amount of help received $HR_{j,k}$ by node j from node k is computed using (2), and the amount of help given $HG_{j,k}$ by node j to node k is computed using (1).

At this point it is worth noting that when a node receives a relay request packet, the counters corresponding to the neighboring node from which the relay request was received are updated; the node does not maintain information about the source node from which the relay request originated.

The advantage of the N-GTFT modification over the original GTFT algorithm is that by differentiating its neighboring nodes, a node can detect and isolate nodes that behave selfishly. As a result, it punishes selfish non-cooperating neighbors by rejecting their relay requests, while accepting relay requests from cooperating neighbors.

3.2. Bucket algorithm

According to this algorithm, each node maintains a bucket. The contents of the bucket are increased or decreased whenever a new relay request is received, and depending on the amount of help the node has received and given up to that point. Specifically, when a relay request arrives at time t_i the bucket contents $B(t_i)$ are updated according to the following equation:

$$B(t_i) = B(t_{i-1}) + [HR(t_{i-1}) - HG(t_{i-1})](t_i - t_{i-1}) \quad (5)$$

where $HG(t_{i-1})$ and $HR(t_{i-1})$ are the amount of help given and received, estimated using (1) and (2) respectively, until the time t_{i-1} that the last relay request was received. Hence, according to (5) the bucket is filled when the amount of help received is greater than the amount of help given. Note that the amount by which the bucket contents increase or decrease also depends on the time that has elapsed since the last request $t_i - t_{i-1}$.

A relay request is accepted if the bucket is non-empty, i.e. $B_i > 0$. Hence, the behavior of the Bucket algorithm differs from the GTFT algorithm: Even if $HR + gen \leq HG$, a node can continue to accept relay requests as long as the bucket is non-empty. On

the other hand, with GTFT a node stops accepting relay requests when $HR + gen \leq HG$. The above behavior allows the Bucket algorithm to accept relay request for a longer period, hence can improve the aggregate throughput in the case of bursty traffic.

Similarly to the neighbor GTFT algorithm, the Bucket algorithm can be extended by having each node differentiate its neighboring nodes. This would require that a node maintains three parameters for each of its neighbors: the bucket parameter and the amount of help given to and received from the particular node.

3.3. Weighted versions of the N-GTFT and Bucket algorithms

Another modification to both of the above algorithms involves considering different weights for different nodes. These weights would multiply the amount of help received and given in order to determine whether a relay request is accepted. Hence, for the N-GTFT algorithm, the condition for accepting a new relay request made by node k to node j is the following:

$$W_j HR_{j,k} + gen > W_k HG_{j,k}$$

The weight for a node can depend on its position within a certain network topology. For example, consider the case where in an ad hoc network there is an access point to which most of the packets are destined. Nodes closer to the access point will have to accept more relay requests compared to nodes farther from the access point. Hence, if a node j is closer to the access point it would have a larger weight, compared to a node k which is farther from the access point. Hence, according to the last condition, node j would tend to accept relay requests at a higher percentage compared to the percentage of relay requests accepted by k .

Indeed, simulation results of the above weighted approach for the GTFT and Bucket algorithms show that when most traffic is destined to a specific node, higher total throughput is achieved if a node's weight is assigned based on the number of other nodes that can use that node as a relay. Hence, consider that nodes are located in different zones, where a different zone corresponds to a different distance from the access point. For simplicity assume that there are three zones A, B, and C, where zone A is the closest to the access point and zone C is the farthest. Then the weight for nodes within the three zones should satisfy the following:

$$\frac{W_A}{W_B} = \frac{N_{A+B+C}}{N_{B+C}} \quad \text{and} \quad \frac{W_B}{W_C} = \frac{N_{B+C}}{N_C} \quad (6)$$

where W_x is the weight for zone x , and N_x is the number of nodes located in zone x .

An issue we do not discuss in this paper is how to estimate the above weights in a distributed manner, and how a node knows which weight to use. The second issue can be addressed by using locationing information, or hop count information, similar to the Time To Live (TTL) mechanism used in IP routing.

4. Simulation results

The simulation results we present next were obtained using the OPNET simulator, where the algorithms described in the previous sections were implemented as an extension of the DSR (Dynamic Source Routing) algorithm [9]. In particular, a new module that performs the decision of accepting or denying a relay request was implemented before the function that is responsible for the packet transfer, and two new packet types are introduced: "Forward Request Packet Type" and "Forward Reply Packet Type", which carry the relay request (in the forward direction) and the relay request reply (in the backward direction), respectively.

In the simulated scenarios we considered 16 nodes randomly placed in a 350 meters X 350 meters area, except in the access point scenario where we considered a total of 28 nodes. The transmission range for each node was 100 meters. Each node generated packets of length 512 bits. The interval between successive packets was exponentially distributed, with an average of 0.25 seconds. In the experiments for bursty traffic, we assume that a source alternates between a non-idle and idle state, remaining in each for a random time between one and two seconds; while in the non-idle state, the node transmits packets in exponentially distributed time intervals with average 0.25 seconds.

4.1. Scenario 1: GTFT without selfish nodes

This scenario considered the total throughput of the GTFT algorithm, for which the parameter gen was set to 0.01. The results provided by a 10-minute simulation show that the amount of help given by a node and the amount of help received by the node both quickly converge to the value 1. The total throughput is 65 packets/sec, with each node transmitting with a throughput of approximately 4 packets/sec, see Table 1.

4.2. Scenario 2: GTFT with selfish nodes

Let us suppose that a middle positioned node becomes selfish, so it rejects all relay requests it receives. In this case for the GTFT algorithm the total throughput drops by approximately 70%, Table 1. The amount

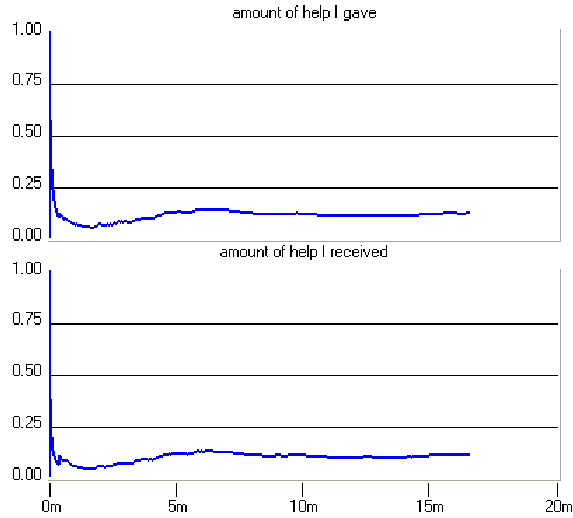


Figure 1. GTFT: Convergence of given and received help of a middle positioned node, in presence of one selfish node. The horizontal axis is the time in minutes.

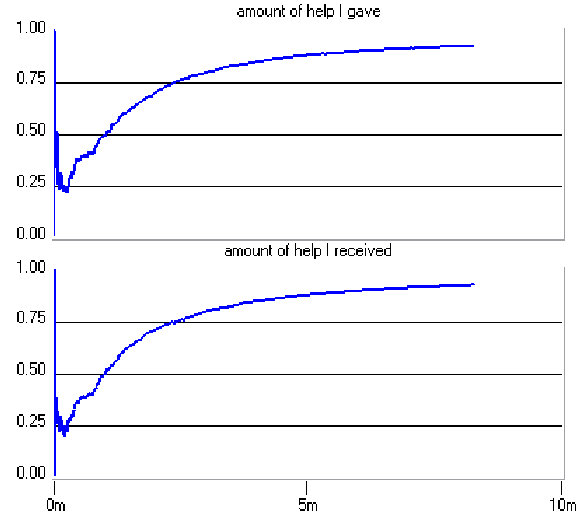


Figure 2. N-GTFT: Convergence of given and received help amounts of a middle positioned node, in presence of one selfish node. The horizontal axis is the time in minutes.

of help received and given for a node located in the center of the ad hoc network converges quickly to a very low value, as shown in Figure 1. The reduction of the aggregate throughput is higher when more nodes act selfishly; hence, in the presence of three selfish nodes positioned in the center of the network, the throughput drops by approximately 94%, Table 1.

4.3. Scenario 3: N-GTFT with selfish nodes

Keeping the same parameters as in scenario 2, in the case of one selfish node, results show that the N-GTFT algorithm results in a significant improvement in the total throughput (60 packets/sec), which is more than three times higher than the throughput achieved with the GTFT algorithm (19 packets/sec), Table 1. The received and given help of nodes far from the selfish node converge to a high value, Figure 2. Moreover, the throughput experienced by nodes close to the selfish node is less than the throughput experienced by the rest of the nodes. This is explained by the fact that nodes close to the selfish node are likely to send relay requests to the selfish node more often. Nevertheless, the nodes close to the selfish node manage to isolate its influence from the whole network, since they reject only the requests coming from the selfish node.

In the case of three selfish nodes, the N-GTFT algorithm again significantly improves the total throughput compared to the GTFT algorithm, increasing the to-

tal throughput to 13 packets/sec compared to 3 packets/sec, Table 1. As expected, the total throughput in this case is much smaller than in the case of one or no selfish nodes.

4.4. Scenario 4: bursty traffic

In this experiment, nodes transmit packets according to an exponential inter-packet time with average 0.25 seconds, for a random interval between one and two seconds, and remain idle for a random interval again between one and two seconds. From the results shown in Table 1, observe that the Bucket algorithm results in 37% higher total throughput compared to the throughput achieved using the GTFT algorithm.

The intuition of why the Bucket algorithm achieves higher throughput compared to the GTFT algorithm is the following. During scenarios with bursty traffic, the GTFT algorithm stops accepting relay requests when condition (3) is not true. On the other hand, the bucket algorithm continues accepting requests for some more time, until the moment its bucket becomes empty, i.e. it can still accept relay requests after (3) does not hold.

4.5. Scenario 5: bursty traffic with one selfish node

In this scenario, traffic is bursty and one node behaves selfishly by denying all relay requests. As shown

Algorithm	Throughput
GTFT without selfish nodes	65 pkts/sec
GTFT with one selfish node	19 pkts/sec
N-GTFT with one selfish node	60 pkts/sec
GTFT with three selfish nodes	04 pkts/sec
N-GTFT with three selfish nodes	13 pkts/sec
GTFT with bursty traffic	45 pkts/sec
Bucket with bursty traffic	61 pkts/sec
Bucket with bursty traffic & one selfish node	21 pkts/sec
N-GTFT with bursty traffic & one selfish node	48 pkts/sec
N-Bucket with bursty traffic & one selfish node	52 pkts/sec
GTFT with access point	46 pkts/sec
W-GTFT with access point	83 pkts/sec
W-Bucket with access point	85 pkts/sec

Table 1. Total Data Throughput Results

in Table 1, the total throughput for the Bucket algorithm is reduced by approximately 65% (21 packets/sec), compared to the case where there is no selfish node (61 packets/sec). The N-Bucket algorithm improves the throughput by almost 2.5 times (52 packets/sec). Moreover, the throughput with the N-Bucket algorithm is also higher than the throughput with the N-GTFT algorithm (48 packets/sec) by 8.3%.

4.6. Scenario 6: access point scenario

Let us consider an access point topology, where nodes are located in three zones around the access point. Starting from the inside, the internal zone A has 4 nodes, the second zone B has 8 nodes and the third (external) zone C has 16 nodes. We assume that all nodes send the 80% of the traffic they generate to the access point and the remaining 20% randomly to the rest of the nodes. The W-GTFT yields 82% higher total throughput compared to the throughput with the GTFT algorithm, Table 1. The improvement achieved with the W-Bucket algorithm is similar.

In case we repeat the same scenario assuming that one of the nodes of the internal zone A does not obey to the rules, and relays the traffic using weight with value one, we find that this node experiences around 58% lower data throughput, while the total data throughput of the network is decreased by almost 14%. This result shows that if a node uses a different weight than the one it should use, it is penalized by its neighbors which use its actual weight to decide whether to accept relay requests from that node; hence, all nodes have the incentive to use their assigned weights.

5. Conclusion

In this paper we presented two algorithms that create incentives for collaboration between the nodes of a wireless ad hoc network. The first algorithm, called N-GTFT, is a modification of the GTFT (Generous Tit-

for-Tat) algorithm presented in [1], where nodes differentiate between their neighboring nodes, and is appropriate in cases of low mobility. Simulation results show that the N-GTFT algorithm gives higher total throughput compared to the GTFT algorithm, in the presence of selfish nodes. The second algorithm considers a bucket, whose contents are updated based on the amount of help a node receives and gives. The Bucket algorithm achieves higher throughput in the case of bursty traffic. Both algorithms can be extended to weighted versions, which are appropriate when the destination of packets is not uniformly distributed, as in the case where most packets are destined to an access point.

An extension to the algorithms presented in this paper is to restart the corresponding counters periodically after some time interval. This would enable the algorithms to adapt faster to changes, such as the case where nodes that were previously cooperative become selfish, or the opposite. The duration of the time interval after which counters are restarted depends on how often changes occur in the network.

References

- [1] V. Srinivasan, P. Nuggehalli, C. Chiasserini, and R. Rao, "Cooperation in Wireless Ad Hoc Networks", In *Proc. of IEEE Infocom*, 2003.
- [2] V. Srinivasan, P. Nuggehalli, C. Chiasserini, and R. Rao, "Energy Efficiency of Ad Hoc Wireless Networks with Selfish Users", In *Proc. of European Wireless 2002*.
- [3] L. Mui, M. Nohtashemi, and A. Halberstadt, "A Computational Model of Trust and Reputation", In *Proc. of the 35th Hawaii International Conference on System Sciences*, 2002.
- [4] O. Ileri, S.C. Mau, and N. Mandayam, "Pricing for Enabling Forwarding in Self-Configuring Ad Hoc Networks", In *Proc. of IEEE WCNC 2004*.
- [5] L. Buttyan and J.P. Hubaux, "Stimulating Cooperation in Self-Organizing Mobile Ad Hoc Networks", *Technical Report No. DSC/2001/046*, August 2001.
- [6] L. Blazevic, L. Buttyan, S. Capkun, S. Giordano, J.P. Hubaux, and J. Le Boudec, "Self-Organization in Mobile Ad-Hoc Networks: the Approach of Terminodes", *IEEE Communications Magazine*, Vol.39 No.6, June 2001.
- [7] M. Conti, E. Gregori, and G. Maselli, "Towards Reliable Forwarding for Ad Hoc Networks", In *Proc. of Personal Wireless Communications 2003*.
- [8] J. Crowcroft, R. Gibbens, F.P. Kelly, and S. Ostring, "Modelling Incentives for Collaboration in Mobile Ad Hoc Networks", In *Proc. of Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt) 2003*.
- [9] B. Johnson and D. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks", *Mobile Computing*, Kluwer Academic Publishers, 1996.