Preventing selfishness in open mobile ad hoc networks

Hugo Miranda Luís Rodrigues

Abstract

Mobile ad hoc networks are a new and challenging topic on mobile computing. A particular characteristic of ad hoc networks is their self-organization, what makes them highly dependable of the participants. Until recently, it was assumed that these networks would be composed of cooperative hosts, where their owners share a common goal. However, this assumption will probably be dropped due to the generalization of computers equipped with wireless network devices. This position paper shows how the selfishness of the participants in a ad hoc network can prejudice its overall functioning and sketches a protocol that discourages this kind of behavior.

1 Introduction

The research effort on mobile networks (infrastructured or ad hoc) has focused mainly on routing and usually assumes that all nodes are cooperative. These assumptions hold on military, or search and rescue operations, where all hosts belong to the same authority and their users share the same goals. The application of mobile ad hoc networks (MANETs) for the support of open communities has emerged recently. In such environment, different users, with different goals, share the resources of their devices, ensuring global connectivity. This sort of communities can already be found in wired networks, namely on peer-to-peer networks.

However, there is a significant difference between the fixed and the mobile environment. Some resources, namely battery power, are scarce in a mobile environment and can be depleted at fast pace with the device utilization. This can lead to a selfish behavior of the device' owner, that may attempt to take the benefit from the resources provided by the other nodes without, in return, making available the resources of his own devices.

In this scenario, open MANETs will likely resemble social environments. A group of persons can provide benefits to each of its members as long as everyone provides his contribution. For our particular case, each member of a MANET will be called to forward messages and to participate on routing protocols. A selfish behavior threatens the entire community. Optimal paths may not be available. As a response, other nodes may also start to behave in the same way. In the extreme, this can take to the complete decoupling of the network. This kind of problems was already noticed on peer-to-peer file distribution systems. It was observed in Gnutella that the number of sites providing valuable content to the network was a small part of the number of users retrieving information [1].

To provide the expected service, open MANETs should implement a protocol that discourages the selfish behavior. The protocol should apply some kind of penalty to the users that intentionally present selfish behavior and be tolerant to malicious attacks and to communication failures, that could penalize well-behaved hosts.

This paper is organized as follows. Section 2 presents a possible scenario that motivates the need of selfishness prevention protocols. Section 3 provides an overview of previous work realized on the subject. One protocol extending previous ones is presented on section 4. The future work and the conclusions are presented in section 5.

2 Motivation

Picture yourself in a conference room. The talk goes boring and you think that this is the perfect time for checking your e-mail using your new laptop with a 802.11 network card. The conference is hosting an Internet room holding a wireless base station. However, the room is too far from your current location and you are out of range. Your opinion about the speaker is shared by the majority of the audience and you notice that a few more laptops are being opened. Some of them really close to the door. You are sure that they are within the base station range. Suddenly, your desktop shows that an ad hoc network has been established and you have Internet access.

Your MANET utility application lets you know that you can reach two hosts within the wireless base station range and you start retrieving your e-mail. In the middle of the download you notice that the available bandwidth has decreased and a few moments latter you are unable to reach your e-mail server. However, your utility application keeps showing that those two nodes are active.

You lost connectivity when one of those nodes found that the battery power of his laptop and some of the available bandwidth was being used for forwarding your (and others) messages. When your laptop stopped receiving acknowledgments, it started a new routing protocol round that led him to the second computer. The user of this computer noticed a huge increase on the number of forwarded messages and, to prevent himself from depleting his battery, also ceased to forward messages on behalf of the other users. Because one of the users of the MANET was selfish, you have now the opportunity to get the most of the boring talk! Probably, next time you will apply the same policy to anyone attempting to use your network card.

As previously mentioned, MANETs where envisioned for search-and-rescue, military and law enforcement operations. In these examples, all users work together toward a common goal. Therefore, selfishness behavior is not expected since it would only prejudice the group. We envision that MANETs will rapidly expand to other domains, like the one presented above. In these Open MANETs, users do not share a common goal. Each user will agree to share his resources only if this brings him some benefit and not to the group as in Closed MANETs.

While being impossible to forbid selfish behavior, Open MANETs users should be discouraged to do it. This can only be achieved by applying some kind of penalty to users. The following section shows some proposals toward this goal.

3 Related work

Malicious networks nodes that participate in routing protocols but refuse to forward messages may corrupt a MANET. These problems can be circumvented by implementing a reputation system. In [5], the reputation system is used to instruct correct nodes of those that should be avoided in message's routes. However, as is, the system rewards selfish nodes, who benefit from not forwarding messages while being able to use the network.

On modern society, services are usually provided in exchange of an amount of money, previously agreed between both parts. The Terminodes project defined a virtual currency named *beans* used by nodes to pay for the messages. Those beans would be distributed by the intermediary nodes that forwarded the message [3, 4]. Implementations of digital cash systems supporting fraud detection require several different participants and the exchange of a significant number of messages [6]. To reduce this overhead, Terminodes assumes that hosts are equipped with a tamper resistant security module, responsible for all the operations over the *beans* counter, that would refuse to forward messages whenever the number of beans available are not sufficient to pay for the service. The modules use a Public Key Infrastructure (PKI) to ensure the authentication of the tamper resistant modules. This infrastructure can be used with two billing models. In the *Packet Purse Model*, the sender pays to every intermediary node for the message, while in the *Packet Trade Model* is the receiver that is charged. In both models, hosts are charged as a function of the number of hops traveled by the message.

The CONFIDANT protocol [2] implements a reputation system for the members of MANETs. Nodes with a bad reputation may see their requests ignored by the remaining participant, this way excluding them from the network. When compared with the previous system, CONFIDANT shows two interesting advantages. It does not require any special hardware and avoids the "self-inflicted punishment" that could be the exploitation point for malicious users. The system tolerates certain kinds of attacks by being suspicious on the incoming selfishness alerts that other nodes broadcast and relying mostly on its self experience.

These systems show two approaches that conflict in several aspects. The number of requests received by hosts depends of their geographical position. Hosts may become overloaded with requests because they are positioned in a strategical point in the MANET. A well-behaved node that temporarily supports a huge amount of requests should latter be rewarded by this service. CONFIDANT has no memory, in the sense that the services provided by some host are quickly forgotten by the reputation system. On the other hand, *beans* can be kept indefinitely by hosts. In MANETs, it is expected that hosts move frequently, therefore changing the network topology. The number of hops that a message must travel is a function based on the instant position of the sender and the receiver and varies with time. Terminodes charges the sender or the receiver of a message based on the number of hops traveled what may seems unfair since any of them will pay based on a factor that is outside his control.

4 The protocol

People tend to cooperate as long as they notice that there is a fair division of the tasks in a group. This problem is extended for MANETs where the unfairness of the service division may prevent users from using their own devices. Beside punishment, selfishness prevention protocols must perform an equally important role in the fair distribution of services by the overall community what is not guaranteed by all routing protocols. This can be achieved if hosts are allowed to present some "justified selfishness", by refusing part of the requests received, forcing the clients to search for alternatives.

Under some circumstances, like when two device' owners in a MANET are friends or when the network administrator attempts to send a message, selfishness may be acceptable. In the first case one host is willing to provide unlimited services to another but this is orthogonal to the community. The second shows one case of MANET acceptable selfishness.

This section presents a new selfishness prevention protocol. The protocol shares some of the properties of both protocols previously described: it presents a long time memory that allows hosts to be rewarded by services provided in the past and does not charge by the number of hops used. On the other hand, the protocol introduces some new concepts by tolerating justified selfishness and by encompassing trapdoors toward socially acceptable selfishness. The protocol uses only one message, that each host periodically broadcasts and allows that hosts not running it participate in the MANET.

4.1 System model

In the exposition of the algorithm, every node is assumed to have the same transmission power and is equipped with an omni-directional antenna. Like in other broadcast mediums, hosts are able to listen to messages that are not addressed to him. Each host has a unique network address which can not be changed and is included in the messages that he sends.

In order for the protocol to behave correctly two conditions are required: *i*) at least one route composed of only well-behaved nodes must exist between any two well-behaved nodes and *ii*) selfish nodes do not cooperate between themselves. Removing these constraints is an open issue that will be addressed in the future.

4.2 The algorithm

Outline The algorithm is based on a message that each node broadcasts periodically stating its view about the neighborhood. The message exposes a sufficient amount of information that allows the remaining nodes to detect a liar, this way defeating possible attacks.

The protocol introduces the fairness concept by allowing hosts to publicly declare that they refuse to forward messages to some hosts. It is up to the community to evaluate if the proportion of refused hosts reaches an unacceptable level of selfishness.

Data structures For each host q, each other host p will keep a data structure $status_p[q]$ with the following fields:

credits a signed integer increased for each message that p forwarded on behalf of q and decreased for each message that q sent on behalf of p. The initial value is 0.

maxCredits the upper bound to credits.

state One of the following constants: OK, SUSPECTED, OTHERS_SUSPECTED and SELFISH. The initial state of each new node is OK. A correct node p should continue to provide services to a node q in the SELFISH state as long as $status_p[q].credits < 0$.

notSelfishTo The list of hosts for which q is willing to provide services. This list is initialized with every node known by p;

selfishTo The list of hosts known by q to whom he is not willing to provide services. This list is empty when initialized.

dead an unsigned integer that will be set to zero whenever a message is listen from q. Periodically, a timer will increase this value. If it reaches some threshold, one can consider that the node is no longer in the neighborhood.

Each node participating in the algorithm will also keep two separate lists amSelfishTo and amNotSelfishTo where he keeps respectively the addresses of the hosts from whom he refuses to forward messages and of those that he will forward messages. The list pending_messages keeps the set of messages forwarded to other nodes and that are waiting to be acknowledged. Hosts also keep an integer variable Sc with the sum of the credit fields of all their status structures.

Each host p periodically broadcast a selfState message with the following fields:

notSelfishTo The list of hosts from whom p will accept service requests;

selfishTo The list of hosts to whom p will refuse to provide services;

knownSelfishNodes A list of nodes that p considers to be selfish but whose declarations of selfishTo omitted p;

Messages handling

Service request Whenever host p receives a message m from q, p will see if it is the final destination for m. If not, and q is not in the p.amSelfishTo list, p peeks the next host r such that $status_p[r].state \neq SELFISH$ and r is in a path toward m's destination. m is sent to r and $status_p[q].credits$ is incremented. The tuple (r, m) is added to pending_messages.

A selfState message will be sent if q is on the p.amSelfishTo list or if $status_p[q].credits \geq status_p[q].maxCredits$. In any case, $status_p[q].dead$ is reseted.

Message received If host p listens to a message issued by some other node q it will check if (q, m) is in pending_messages. This will be used as a confirmation that q did not become selfish. (q, m) is removed from pending_messages and $status_p[q].credits$ is decreased. Otherwise, p will check if a data structure for q is already defined and create one if not. In any case, $status_p[q].dead$ is reseted.

selfState message received Whenever host p receives a selfState message from host q it will reset the $status_p[q].dead$ variable and proceed to observe every field of the message.

notSelfishTo/selfishTo is used to replace the information in the local $status_p[q]$ structure. Unknown nodes are discarded. p checks the list where it is included, updating $status_p[q].state$ accordingly.

knownSelfishNodes is used by q to alert its neighbors of the hosts that have refused to provide him services while declaring to be not selfish for q. For each node r in the list, p should move q from $status_p[r].notSelfishTo$ to $status_p[r].selfishTo$. If $status_p[r].state = OK$, its state should be changed to OTHERS_SUSPECTED. The host should ignore unknown addresses in this field.

If p is in knownSelfishNodes but q is not in p.amSelfishTo, this may suggest a malicious attack from q. q should be added to $p.amSelfishTo$ and $status_p[q].state$ should be changed to $SELFISH$. The next selfState message broadcasted by p should include q in the knownSelfishNodes field.

4.3 Execution scenarios

The goal of this section is to show the overall application of the protocol in several different cases. Due to the lack of space, some cases have been omitted.

All cooperative hosts, lightly loaded network This is the most favorable scenario. Sc will be near zero for all participants. In this case, hosts will broadcast periodically one selfState message, where the fields selfishTo and knownSelfishNodes will be empty. The overhead introduced by the protocol is the periodic broadcast of one message. The case where some hosts do not implement the protocol is handled transparently: they will ignore the content of these messages. The nodes that are running the protocol will create a status structure for them, that will be mostly kept with the default values.

All cooperative hosts, heavily loaded network Starting from the previous example, Sc will become excessively positive for some hosts and excessively negative for others. Those that have been more loaded by requests from other hosts will start to move some of the nodes from the notSelfishTo to the selfishTo list. One can envision several criteria for the selection, using for example statistical information about the number of hops toward the usual destination of the messages of each host, or the number of negative credits. To keep network functionality, it is important that two or more loaded hosts attempt to define disjoint selfishTo lists. Note that well-behaved nodes should accept their movements to these lists and continue to provide services as long as their credits with the loaded hosts remain negative.

Nodes not running the protocol that are placed in selfishTo lists will start to notice that their messages are not being delivered. Their reaction to this event will depend on several factors, for example the routing protocol being used.

One selfish node Selfish nodes are always penalized, whether they run the protocol or not, because the decision is taken locally at each node based on the information provided by its pairs. When a host q refuses to forward messages, its address will be included in the knownSelfishNodes list of the selfState messages of the other hosts. Hosts listening to these messages will start to remove the entries from $status_p[q].notSelfishTo$ and placing them in $status_p[q].selfishTo$. Whenever the rate $\frac{\sharp status_p[q].notSelfishTo}{\sharp status_p[q].notSelfishTo + \sharp status_p[q].selfishTo}$ goes bellow some threshold, hosts will begin to set $status_p[q].state = SELFISH$. Eventually, q will exhaust its credits with every host running the protocol and will have its messages refused.

Previously selfish nodes may be reintegrated in the group by broadcasting a new selfState message with some hosts in the notSelfishTo list. To prevent repetitive behavior, non-selfish hosts may reduce maxCredits for the host. One alternative procedure, which also allows the reintegration of hosts not running the protocol is to become silent for a sufficient period of time so that their records can be erased by having the dead counter to reach the threshold.

Malicious selfish host In order to attempt to subvert the system, one malicious user must always have knowledge of the protocol. Changing the state of an host to SELFISH is an operation local to each protocol based on the following conditions: *i*) it exhausted the credits limit; *ii*) it refused to forward messages to some other host and *iii*) it was considered selfish by a sufficient number of hosts. The number of credits is a variable local to each host. Preventing one host from maliciously changing this value lies on the broader problem of computer security and is out of the scope of this paper.

The only possible attempt to subvert the protocol would be to fake the number of host that he is serving, by placing non-used addresses or by falsely declaring some existing hosts on the notSelfishTo field of its selfState messages. This would allow him to fool hosts that are not requesting services, postponing (possibly indefinitely) the problem of the credit exhaustion. These two possible turnarounds are separately handled in the protocol. Unknown addresses are ignored when each host accounts for the number of hosts that the nodes are serving, so the host has no advantage on providing this information. Declaring to be available to provide services to some hosts (that is, placing their address in the notSelfishTo field) will make these nodes to request him to forward messages. If he refuses, hosts will start to place the malicious host address in the knownSelfishNodes field which in turn, will make its rate of serving hosts lower, eventually (when a sufficient number of nodes detects the fraud) bellow the acceptable threshold for the hosts to consider him selfish. Note that invalid addresses are expected due to the limited transmission range of the network devices and can not, by themselves, be considered an attempt to subvert the protocol.

The definition of a proper threshold must be subject of further study and may vary depending on the density of the nodes in the MANET area.

4.4 Other considerations

Node movement MANETs expect their hosts to move frequently, changing the network topology. When moving, one host may become outside the range of another. Even in the presence of pending messages, the protocol does not falsely consider these hosts as selfish thanks to the `dead` timer that will erase all the information of the host if no messages are heard from him for some period of time.

Socially acceptable selfishness The two categories of socially acceptable selfishness presented above are handled differently by the protocol. Friendship can be tolerated by instructing nodes to never declare some address to be selfish. This way, even when the remaining members of the community start to declare the host has selfish, he will always be able to forward messages to that one. Note that the friends of a selfish node must still be able to request services from other nodes, and are accountable by the credits that should be payed for the service. MANET acceptable selfishness can be implemented with a Private Key Infrastructure where the certification authority makes available the public key of the hosts that can use the network without restrictions. All messages from that host should be signed and no credits accountability should be imposed to intermediary nodes.

5 Conclusions and future work

MANETs are particularly sensible to unexpected behaviors. The generalization of wireless devices will soon turn MANETs in one of the most important connection methods to the Internet. However, the lack of a common goal in MANETs without a centralized human authority will make them difficult to maintain: each user will attempt to retrieve the most of the network while expecting to pay as less as possible. In human communities, this kind of behavior is called selfishness. While prohibiting selfishness shows to be impossible over a decentralized network, applying punishments to those that present this behavior may be beneficial.

This position paper outlined a selfishness prevention protocol for Open MANETs. The protocol shows some novelties: its decentralization avoids the usage of complex payment systems and it introduces the concept of "justified selfishness" that makes the whole system more fair, not penalizing users by their network topological location. This work is on its early stages of development and lacks theoretical and experimental validation. Future work includes an experimental evaluation over a network simulator, where different topologies and host movements will be tested. Its integration with routing protocols may also shows to be beneficial.

References

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

[2] S. Buchegger and J.-Y. Le Boudec. Performance analysis of the CONFIDANT protocol: Cooperation Of Nodes - Fairness In Distributed Ad-hoc NeTworks. Technical Report IC/2002/01, Swiss Federal Institute of Technology, Lausanne, January 2002.

[3] L. Buttyán and J. Hubaux. Enforcing service availability in mobile ad-hoc wans. In *Proc. of the First IEEE/ACM Workshop on Mobile Ad Hoc Networking and Computing (MobiHOC)*, Boston, MA, USA, August 2000.

[4] L. Buttyán and J. Hubaux. Stimulating cooperation in self-organizing mobile ad hoc networks. Technical Report DSC/2001/046, Swiss Federal Institute of Technology, Lausanne, August 2001.

[5] S. Marti, T. Giuli, K. Lai, and M. Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *Proceedings of Mobicom 2000*, Boston, USA, August 2000.

[6] P. Veríssimo and L. Rodrigues. *Distributed Systems for System Architects*, volume 1 of *Advances in Distributed Computing and Middleware*. Kluwer Academic Publishers, Boston, January 2001.