Friends and Foes: Preventing Selfishness in Open Mobile Ad Hoc Networks

Extended Report*

Hugo Miranda Luís Rodrigues
Universidade de Lisboa - Departamento de Informática
Edifício C5 - Campo Grande - 1749-016 LISBOA, Portugal
{hmiranda,ler}@di.fc.ul.pt

Abstract

Technological advances are leveraging the widespread deployment of mobile ad hoc networks. An interesting characteristic of ad hoc networks is their self-organization and their dependence of the behavior of individual nodes. Until recently, most research on ad hoc networks has assumed that all nodes were cooperative. This assumption is no longer valid in spontaneous networks formed by individuals with diverse goals and interests. In such environment, the presence of selfish nodes may degrade significantly the performance of the ad hoc network. This paper proposes a novel algorithm that aims to discourage selfish behavior in mobile ad hoc networks.

1 Introduction

The research effort on mobile networks (infra-structured or ad hoc) has focused mainly on routing and usually assumes that all nodes are cooperative. These assumptions hold on application such as military or search-and-rescue operations, where all nodes 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, like battery power, are scarce in a mobile environment and can be depleted at fast pace with the device utilization. In this scenario, open MANETs will likely resemble social environments: a group of persons can mutually benefit from cooperation as long as every participant contributes with approximately the same share. In this paper, we are concerned with the resources invested in forwarding messages and in the participation on routing protocols. To extend the life of their devices, users may fell compelled to

*Selected sections of this report will be published in the IEEE International Workshop on Mobile Distributed Computing (MDC). Providence, Rhode Island USA, May 2003 (in conjunction with ICDCS 2003).

exhibit a selfish behavior, by benefiting from the resources provided by the other nodes without, in return, making available the resources of their own devices.

Selfish behavior threatens the entire community. Optimal paths may not be available and cooperative nodes may become overloaded and be forced to abandon the community. In the worst case scenario, the network may become partitioned. This kind of problems as been already observed on peer-to-peer file distribution systems. For instance, it was observed in Gnutella that the number of sites providing valuable content to the network is a small part of the number of users retrieving information [1].

In order to effectively support open and spontaneous communities, MANETs should complement routing protocols with additional mechanisms and algorithms that discourage selfish behavior. This paper presents a novel algorithm that meets this goal. The algorithm penalizes users that are intentionally selfish. Additionally, the algorithm is tolerant to malicious attacks and communication failures.

The paper is organized as follows. Section 2 motivates the need of selfishness prevention algorithms using a concrete scenario. Sections 3 and 4 provide an overview of previous work. Our proposal is then described in Section 5. Future work and conclusions are presented in Section 6.

2 Motivation

Consider a campus wide wireless network with incomplete coverage. In order to preserve Internet access while moving within the campus, students are motivated to cooperate. If nodes accept to forward messages when located close to base stations, connectivity for the whole MANET may be insured (with the contribution of some underlying MANET routing protocol).

However, if no mechanisms is present to reward cooperation and discourage selfishness, some users may instruct their devices not to relay incoming messages, as this would extend their battery power and increase the available bandwidth. Such behavior may cause a snow-ball effect: cooperative nodes that are willing to forward messages for third parties obtain no benefit for their behavior and realize that selfish nodes are implicitly rewarded. This motivates them to become also selfish. Message forwarding may be disrupted to the point the MANET collapses.

The disadvantage of selfish behavior may be exacerbated by the operation of the routing protocols used in the MANET: some protocols use caches to accelerate route discovery. Routes containing cooperative nodes will be more frequently present in the caches and, therefore, will be used more seldom. Thus, the batteries of cooperative users are likely to be drained at a fast pace while other potential intermediary nodes do not spend any of their batteries forwarding messages.

As previously mentioned, most algorithms for MANETs where envisioned for search-and-rescue, military and law enforcement operations. In these applications, all users work together toward a common goal. Therefore, selfishness behavior is often not expected. 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 the resources of his node only if this brings him some benefits. While it is impossible to prevent self-

ish behavior, it is possible to design algorithms that discourage such behavior. This can only be achieved by applying some kind of penalty/reward to users. The following section provides an overview of previous proposals toward this goal.

3 Related Work

Malicious nodes that participate in routing protocols but refuse to forward messages may disrupt the operation of the MANET. These nodes may be circumvented by implementing a reputation system. In [5], the reputation system is used by correct nodes to avoid malicious nodes when selecting message routes. However, as is, the system rewards selfish nodes, who benefit from not forwarding messages while being able to use the network.

On modern societies, services are usually provided in exchange of an amount of money, previously agreed between both parts. Several *e*-currency models have been proposed as a reward mechanism for cooperating nodes. However, most of them assume that protocols are trusted and will not subvert the system. Trustos [6], for example, defines a model where users open "emotional bank accounts" on their service providers.

The Terminodes [3] project proposes the use of a virtual currency named *beans*. This currency is used by nodes to pay for the forwarding of their own messages. The payment is distributed by the intermediary nodes that forwarded the message. Unfortunately, current implementations of reliable digital cash systems require several different participants and the exchange of a significant number of messages [8]. To reduce this overhead, Terminodes assumes that nodes are equipped with a tamper resistant security module, responsible for all the operations over the *beans* counter. The security module refuses to forward messages whenever the number of beans available are not sufficient to pay for the service. To ensure the authentication of the tamper resistant modules, a Public Key Infrastructure (PKI) is used. 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, nodes 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 participants, which in practice excludes them from the network. CONFIDANT does not require any special hardware and tolerates certain kinds of attacks by being suspicious on the incoming selfishness alerts that other nodes broadcast and relying mostly on its self-experience.

In a MANET, the number of requests received by each node depends of its geographical position. Nodes may become overloaded with requests because they are positioned in a strategical point of the topology. A well-behaved node that temporarily supports a huge amount of requests should latter be rewarded by this service. The previous approaches do not address this aspect adequately. CONFIDANT has no memory, in the sense that the services provided by some node are quickly forgotten by the reputation system. Terminodes does not suffer from this problem, since *beans* can be kept indefinitely by nodes. However, the charges for packets depends on the number of hops on the message's path, a

factor that is outside the control of the participants.

4 MANET Routing

In order to evaluate the effect of cooperative/selfish behaviors on the resource consumption of nodes, one needs to understand the underlying MANET routing protocol. Although the mechanisms proposed in this paper could be adapted to other routing protocols, to make our discussion concrete, we assume that nodes use the Dynamic Source Routing protocol (DSR) [4]. For self-containment, we provide here an overview of the DSR protocol.

In DSR, multi-hop routes can be learned either by the replies to the broadcast of a RouteRequest message or by inspecting packets flowing on the network. All operational routes are kept in a route cache. When receiving a RouteRequest message, nodes should verify if they are themselves the destination of the RouteRequest or if some route for the destination is available in their route cache. If a route toward the destination is available in the node's route cache, it will prefix its entry in the route cache with the payload of the RouteRequest and send it in a RouteReply message to the initiator. When none of the previous conditions apply, and the node has not received a previous copy of the same request message, it will append its address to the RouteRequest payload and forward the request to all neighbors. When a RouteRequest reaches its destination, its payload will be copied into the RouteReply message, because it contains a valid route between the initiator and the destination.

The protocol includes provisions for non-bi-directional links: while the destination of a RouteRequest message can invert the route in the request to forward the RouteReply, it can also start a new route discovery and piggyback the reply in it.

Routing is performed as follows. The header of data messages include the route to be followed. Each hop will look into the message header to find the address of the next hop. Additionally, each hop also adds this route to its route cache. This avoids a new RouteRequest to be issued by the intermediary hop whenever it sends a message to one of the nodes in that path.

Route errors can occur due to the movement of the nodes or from radio interference. When one link finds that the next hop is unreachable, it should send a RouteError message to the source. If no further paths are available in the route cache, a new route discovery should be initiated. Otherwise, the source should use a new route that avoids the missing link. A complete description of the DSR protocol can be found elsewhere [4].

5 Friends and Foes

People tend to cooperate as long as they notice that there is a fair division of tasks in a group. This observation can also be applied in MANETs where the unfairness of the service division may prevent users from using their own devices. Therefore, any selfishness prevention algorithm must also include some mechanisms to foster a fair distribution of resource consumption. This can be achieved if nodes are allowed to present some "justified selfishness", by refusing

some of the requests received, forcing the clients to search for alternatives. The algorithm should equally provide hooks to allow nodes to be selective in the way they apply selfishness prevention. For instance, two friends may provide unlimited service to each other while resources are available. One may also be willing to unconditionally accept messages from particular users, such as administrators.

This section presents a new selfishness prevention algorithm which embodies some of the properties of the protocols described in the related work section: it presents a long lived memory that allows nodes to be rewarded by services provided in the past but also does not charge by the number of hops used. Additionally, the algorithm introduces new features by tolerating justified selfishness. The algorithm uses only one type of message, that each node periodically broadcasts. Furthermore, nodes that do not participate in the algorithm may be still allowed to benefit from the MANET.

5.1 System Model

In the exposition of the algorithm, every node is assumed to have the same transmission power and to be equipped with an omni-directional antenna. Like in other broadcast media, nodes are able to listen to messages that are not addressed to them. Each node has a unique link-level address which cannot be changed and that is included in all the messages that it sends (this address serves as the node unique identifier).

In order for the protocol to perform 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 conspire to disrupt the network. Removing these constraints is an open issue that will be addressed in the future.

5.2 The Algorithm

The algorithm supports the management of fairness by allowing nodes to publicly declare that they refuse to forward messages to some nodes.

Each node p maintains three variables:

friends$_p$ The set of nodes to which p is willing to provide services.

foes$_p$ The set of nodes to which p is not willing to provide services.

selfish$_p$ The set of nodes which are known to act as if p is a foe.

Note that, in our protocol, it is acceptable for participants to have some foes. It is up to the community to evaluate if the number of foes a given node declares reaches an unacceptable level of selfishness. Each node p advertises the content of these three variables in a control message with the following format:

$$\textsc{SelfState}[p, \mathit{friends}_p, \mathit{foes}_p, \mathit{selfish}_p]$$

This is the only type of message exchanged by the protocol. In order to evaluate the degree of selfishness of other nodes, and to detect inaccuracies in the information advertised by other nodes, each node keeps a record of received SelfState, as described below.

For each other known node q, node p will keep a number of status variables, as follows:

credits$_p^q$ 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$_p^q$ the upper bound to $credits_p^q$.

friends$_p^q$ The set of nodes for which q is willing to provide services. This set is initialized with every node known by p.

foes$_p^q$ The set of nodes to whom q is known not to provide services. A node r can be inserted in this set either because q explicitly advertises r as its foe or because r declares q to be selfish. This set is empty when initialized.

liedTo$_p^q$ A set of nodes to which q is known not to provide services, even though it had advertise them as friends. The set $liedTo_p^q$ is always a subset of $foes_p^q$.

msgpending$_p^q$ The set of messages forwarded by p to q waiting to be acknowledged.

msgsuspected$_p^q$ An unsigned integer increased for each message forward by p to q, but whose forwarding by q was not listened by p (e.g. due to collisions).

deadbeat$_p^q$ 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.

These variables are updated in response the following relevant events: *i)* when the node wants to transmit a data message; *ii)* when the node forwards a data message on behalf of other nodes; *iii)* when the node receives a data message addressed to it; *iv)* when a node is able to listen on the shared medium a message exchanged among two other nodes and, finally; *v)* when the node receives a SelfState control message. The actions taken in each of these cases for some node p are depicted in Figure 1.

When a node sends a message m, it looks for a route such that the next hop q is its friend. It then forwards the message m to q and stores m in $msgpending_p^q$ (l.04). The message m is removed from $msgpending_p^q$ when p detects that q has forward m along the route (l.18–22).

When requested to forward a message m on behalf of q, p only does so if q is its friend (l.07) and if the next hop is not believed to be a foe of p (l.09). If the message is forwarded, node p accounts credits for providing the service and, as in the previous case, stores m to later check the behavior of the next hop.

Slightly more elaborate steps are executed when a SelfState message is received. First, the message is purged from information about nodes unknown by p (l.25–27). If node q declares r to be a liar, the status of q with regard to r is updated accordingly (l.28–31). Next, if q is declaring p itself as a liar, p further treats q as a foe (l.32–34). Then, p updates its record of q's friends and foes (l.35–37). Finally, every peer r of q that has not previously declared itself as a friend of q is simply assumed to be a foe of q (l.38–40).

```
01  upon event service request for message m do
02      find an adequate route such that next hop q satisfies p ∈ friends_p^q;
03      send m to q;
04      msgpending_p^q := msgpending_p^q ∪ {m};
05  upon event forward message m from q to r do
06      deadbeat_p^q := 0;
07      if q ∈ foes_p then
08          discard m;
09      else if p ∈ foes_p^r then
10          reply with a RouteError to q;
11      else
12          forward m to r;
13          if r is not the last hop for m then
14              msgpending_p^r := msgpending_p^r ∪ {m};
15          credits_p^q := credits_p^q + 1;
16  upon event receive message m from q do
17      credits_p^q := credits_p^q + 1;
18  upon event listen message m sent by q to r do
19      deadbeat_p^q := 0;
20      if m ∈ msgpending_p^q then
21          credits_p^q := credits_p^q − 1;
22          msgpending_p^q := msgpending_p^q \ {m};
23  upon event receive message SelfState[q, friends_q, foes_q, selfish_q] do
24      deadbeat_p^q := 0;
25      friends_q := friends_q ∩ (friends_p ∪ foes_p ∪ p);
26      foes_q := foes_q ∩ (friends_p ∪ foes_p ∪ p);
27      selfish_q := selfish_q ∩ (friends_p ∪ foes_p ∪ p);
28      forall r ∈ selfish_q : q ∈ friends_p^r do
29          friends_p^r := friends_p^r \ {q};
30          foes_p^r := foes_p^r ∪ {q};
31          liedTo_p^r := liedTo_p^r ∪ {q};
32      if p ∈ selfish_q ∧ q ∉ foes_p then
33          friends_p := friends_p \ {q};
34          foes_p := foes_p ∪ {q};
35      friends_p^q := friends_q \ liedTo_p^q;
36      foes_p^q := foes_q;
37      liedTo_p^q := liedTo_p^q \ foes_q;
38      forall r ∈ friends_q ∪ foes_q do
39          if q ∉ friends_p^r then
40              foes_p^r := foes_p^r ∪ {q};
41  upon event timeout for message m ∈ msgpending_p^q do
42      msgsuspected_p^q := msgsuspected_p^q + 1;
43      msgpending_p^q := msgpending_p^q \ {m};
44      if msgsuspected_p^q > threshold then
45          selfish_p := selfish_p ∪ {q};
46          foes_p := foes_p ∪ {q};
47          friends_p := friends_p \ {q};
48          msgsuspected_p^q := 0;
```

Figure 1: Variable update.

Nodes detect selfish behavior when they verify that, after some time, their pending messages have not been forwarded by q. When a sufficient number of messages is not forwarded by q, the node is appended to the *selfish* list to alert its neighbors (l.41–48).

5.3 Execution Scenarios

We now illustrate the operation of the algorithm in some concrete scenarios. Due to the lack of space, only the most representative cases are presented.

All Cooperative Nodes, Lightly Loaded Network This is the most favorable scenario. Nodes will broadcast periodically one SELFSTATE message, where the fields $foes_p$ and $selfish_p$ will be empty. The overhead introduced by the protocol is minimal: the periodic broadcast of one message. The case where some nodes 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 variables for them, that will be mostly kept with the default values.

All Cooperative Nodes, Unbalanced Load Network The nodes that have been more loaded by requests from others will start to move some of the nodes from the $friends_p$ to the $foes_p$ set. 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 node, or the number of credits. To preserve network connectivity, it is important that two or more loaded nodes attempt to define disjoint $foes_p$ sets. Note that well-behaved nodes should keep providing services to peers, even if these peers consider them as foes.

Nodes not running the protocol that are classified as *foes* will notice that their messages are not being delivered. Their response to this event will depend on several factors, as 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 peers. When a node q refuses to forward messages, its address will be included in the $selfish_p$ set of the SELFSTATE messages of the other nodes. Nodes listening to these messages will start to remove the entries from $friends_p^q$ and placing them in $foes_p^q$. Whenever the rate $\frac{|friends_p^q|}{|friends_p^q|+|foes_p^q|}$ goes bellow some threshold, node p will begin to mark q as selfish and reciprocally become its foe. Eventually, q will have all other nodes to refuse its messages.

Previously selfish nodes may be reintegrated in the group by broadcasting SELFSTATE messages first acknowledging that they have been selfish for some nodes (to remove their entries from the *liedTo* sets) and latter with some nodes in the $friends_p$ set. Two measures are implemented to prevent selfish nodes from abusing this mechanisms: *i*) to be accepted, a liar must consume additional energy by broadcasting two SELFSTATE messages, and *ii*) non-selfish nodes reduce $maxCredits_p^q$ for the node, reducing the number of messages that it can send without providing useful work for the community. Another mechanism, which also allows the reintegration of nodes not running the protocol, is to let the node voluntarily prevent itself from using the community resources by becoming silent for a sufficient period of time so that its records can be erased by having the *deadbeat* counter reach the threshold.

Malicious Selfish Node Malicious nodes will attempt to hide its selfishness from the remaining participants. Classifying a node as selfish is an operation

local to each protocol based on the following conditions: *i*) it exhausted the credits limit or *ii*) it refused to forward messages to some nodes.

The protocol provides enough information to let each node to detect the envisioned attacks. A malicious node could try to defeat the system using one of the following methods: *i)* by declaring to be friend to a large number of fake nodes or; *ii)* by omitting to be foe of some nodes. Fake nodes are ignored when each node accounts for the number of nodes that each peer is serving, so the node gain no advantage from advertising dummy friends. Note that the limited transmission range of the network devices will make some nodes mark correct addresses as invalid, so their existence can not be considered an attempt to subvert the protocol. Additionally, nodes can not hide their selfishness because information about the nodes known by each other node is crossed: if some node p mentions some other node q in is SelfState message, and q omits p in is own message, then it is assumed that q is selfish for p.

Malicious nodes may also attempt to corrupt the MANET by falsely declaring other nodes as foes. However, in order to consider q as selfish, correct node p will use the information provided by all of p's neighbors. If the malicious node acts alone, its information will not be sufficient to achieve its goal.

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

5.4 Other Considerations

In MANETs nodes are expected to move frequently, changing the network topology. When moving, one node may become outside the range of another. Even in the presence of pending messages, the protocol does not falsely consider these nodes as selfish thanks to the *deadbeat* timer that will erase all the information of the node if no messages are heard from it for some period of time.

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 addresses to be selfish. This way, even when the remaining members of the community start to declare some node as selfish, it will always be able to forward messages to that node. 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 nodes that can use the network without restrictions. All messages from that node should be signed and no credits accountability should be imposed to intermediary nodes.

5.5 Integration with Routing Protocols

The selfishness prevention protocol must work close to the routing protocol for the MANET to keep its efficiency for justified selfishness cases. This section shows how the protocol can be integrated with the Dynamic Source Routing (DSR) protocol [4].

Message forwarding penalizes each intermediary hop by reducing its available power and bandwidth. A desirable feature of MANETs would be to distribute evenly the overhead of the network maintenance tasks (like message forwarding) across all participants. While optimizing the route discovery process, the DSR

route cache hurts load balancing by making all nodes in the neighborhood of some path to share it. [7] proposes a new set of metrics for routing protocols that consider the available power at each node. However, such protocols can hardly be used in Open MANETs where it is assumed that nodes can lie.

By applying to RouteRequest and RouteReply messages the same rules defined above for selfishness prevention, nodes that were previously considered selfish would see their route requests and replies ignored and would become unable to send their messages. This rule would also benefit load-balancing: a node p that based on justified selfishness criteria refuses to forward RouteRequest messages from a node q is forcing an alternative path to be established. For efficiency, all nodes should remove from their route cache all entries pointing to selfish nodes. A side effect of this protocol is that bi-directional routes may be harder to achieve due to the arbitrary selection of nodes to whom some other node will be selfish. However, DSR can handle unidirectional links efficiently by piggybacking the RouteReply message in the RouteRequest issued by the intended destination.

The load-balanced routes may suffer from an excessive inefficiency that could be avoided if overloaded nodes agree on the set of nodes to whom they will become selfish.

6 Conclusions and Future Work

MANETs are particularly sensible to uncooperative 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 paper outlined a selfishness prevention protocol for Open MANETs. The protocol includes a set of new interesting features: 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 physical location. By using just one message that is periodically broadcasted by every node, it does not introduce a substantial overhead on the network or in the nodes. Additionally, it is scalable since each node is only supposed to keep information about the nodes in the range of its network card. This work is on its early stages of development. Future work includes an experimental evaluation over a network simulator, where different topologies and node movements will be tested.

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] D. Johnson, D. Maltz, and J. Broch. *Ad Hoc Networking*, chapter DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks, pages 139–172. Addison-Wesley, 2001.

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

[6] J. Seigneur, J. Abendroth, and C. Jensen. Bank accounting and ubiquitous brokering of trustos. In *Proc. of the* 7^th *CaberNet Radicals Workshop*, Bertinoro, Italy, October 2002.

[7] S. Singh, M. Woo, and C. Raghavendra. Power-aware routing in mobile ad hoc networks. In *Proc. of the 4th annual ACM/IEEE Intl Conf. on Mobile Computing and Networking*, pages 181–190. ACM Press, 1998.

[8] 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.