Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Diana Senn

Reputation and Trust Management in Ad Hoc Networks with Misbehaving Nodes

Diploma Thesis DA-2003.27
31. March to 30. July 2003

Supervisor: Vincent Lenders
Professor: Bernhard Plattner

Abstract

In mobile ad hoc networks, every node has to act as a router for the network to function. However, what happens if some node misbehaves, either selfish or malicious? This DA proposes a reputation-based trust management scheme to detect misbehaving nodes and to exclude them from the network. This is done by monitoring the neighbors, which results in a direct trust value, and by asking others for reputation. Direct trust and reputation together form total trust which then is used for routing decisions. To be able to control routing decisions from source to destination, a source routing protocol is needed. Therefore we took the dynamic source routing protocol and extended it by our trust management. With this, we could show that our approach improves goodput, having 10% misbehaving nodes, by 20% , while causing less than 10% overhead.

In MANETs (mobile ad hoc networks) sind alle Knoten gleich. Damit ein solches Netz funktionieren kann, muss sich jeder Knoten als Router betätigen. Was passiert, wenn einer dies nicht tut? Diese DA schlägt ein reputation-based trust management scheme vor, mit welchem böswillige Knoten detektiert und vom Netz ausgeschlossen werden sollen. Durch das Beobachten der direkten Nachbarn wird ein direktes Vertrauen aufgebaut, welches durch Reputation von anderen zu einem totalen Vertrauen ergänzt wird. Dieses totale Vertrauen wird dann dazu verwendet Routing Entscheide zu fällen. Weil wir dazu die ganze Strecke zwischen Sender und Empfänger bestimmen wollen, müssen wir ein source routing protocol benützen. Darum haben wir DSR (dynamic source routing) um unser trust management erweitert. Durch Simulation und Auswertung dieses Ansatzes konnten wir zeigen, dass der Goodput in einem Setting mit 10% böswilligen Knoten um 20% verbessert werden kann. Der dabei generierte Overhead beträgt weniger als 10%.

Keywords: MANET, DSR, trust management, reputation

Contents

List of Figures

1 Introduction

Sometimes, there is an instant need for a network when all the infrastructure is missing. Mobile ad hoc networks (MANETs) serve this purpose. Lets have a closer look at the word MANET: MANETs consist of many small devices communicating spontaneously over the air (wireless). The topology of the *network* is changing frequently because of the *mobile* nature of its nodes. *Ad hoc* finally induces that there are no such things as fixed routers, therefore every node has to act as a router for its neighbors.
Different routing protocols have been proposed, but most of them assume that all the nodes will fully participate in the protocol. However, what happens if a node is not willing to forward packets for other nodes, either because he is selfish or malicious?
A routing protocol for MANETs should give incentives for acting correctly. Or at least it should be able to detect misbehaving nodes and punish them. Only this way it would be possible to really use MANETs in practice.
In MANETS no a priori trust relationships and no central trustworthy authorities exist. The goal of this thesis is to establish trust relationships by using a reputation-based trust management scheme. This can be done by getting reputation for a node and combining this with personal observations about its behavior.

1.1 Organization of the Paper

The remainder of this paper is organized as follows: Our problem statement is given in section 2 followed by an introduction to MANETs in section 3. Design, Implementation and Evaluation of our protocol can then be found in sections 4, 5 and 6 respectively. To end we do a comparison of our protocol to some related work, state future work and draw our conclusions in sections 7, 8 and 9 respectively.

2 Problem

As explained in the introduction, a reputation-based trust management scheme for MANETs is needed. The goal of this DA is therefore to design a protocol for the distribution of reputation data. To be able to give reputation for a node, we need to know something about him, so we should first establish some direct trust by monitoring our neighbors, which we then can use to give reputation. Reputation and direct trust together form our total trust we have about a node. These total trust values can be used for routing decisions. After the design of the model it should be implemented and tested to show its benefits and drawbacks.

3 MANETs – Mobile Ad Hoc Networks

MANETs where introduced shortly in section 1. This chapter represents a deeper introduction to some specific aspects of MANETs.

3.1 DSR – Dynamic Source Routing

DSR is one of the routing protocols used in MANETs and was first described in 1996 by Johnson and Maltz [8], who are still working on an IETF Internet Draft for DSR [9]. It is a reactive protocol, routes are searched when needed.

3.1.1 Sending a Packet

If a node S wants to send a packet to some other node D, he looks for a route in his route cache. If he finds one, he adds the route to the packet header, and then transmits the packet over its wireless network interface to the first hop X identified in the source route. X will then forward the packet to the next hop, and so on, until D is reached. If S does not have a route to D in his route cache he buffers the packet and does a route discovery first (see 3.1.2).

3.1.2 Route Discovery

Route discovery is used to find a route from a node S to another node D in the ad hoc network. Node S sends a *route request* – with a (for him) unique request id – by broadcast where the target is set to D and the route record is initiated by S. A host X receiving such a route request packet then processes it according to the following steps:

1. If the pair <initiator address, request id> for this route request is found in X's list of recently seen requests, then discard the route request packet and do not process further.
2. Otherwise, if X's address is already listed in the route record in the request, then discard the route request packet and do not process further (to avoid a loop).
3. Otherwise, if the target of the request matches X's own address (i.e. X == D), then the route record in the packet contains the route by which the request reached X from S. Return a copy of this route in a *route reply* to S.
4. Otherwise, append X's own address to the route record in the route request packet, and re-broadcast the request.

The route request thus propagates through the ad hoc network until it reaches D, which then replies to S. S then stores the received source route in his *route cache*.

3.1.3 An example

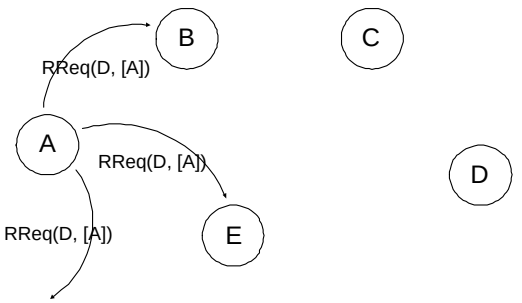


Figure 1: Route Request – 1

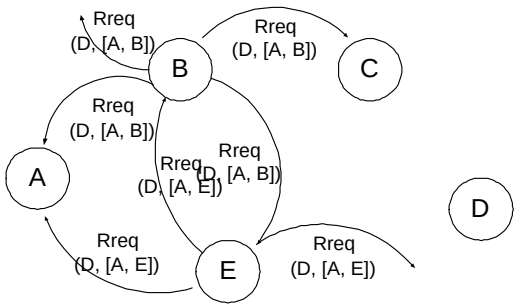


Figure 2: Route Request – 2

Node A does not know a route to node D. So he creates a route request where he sets D as the target and adds himself to the route record (Rreq(D, [A]) and broadcasts it (see figure 1).

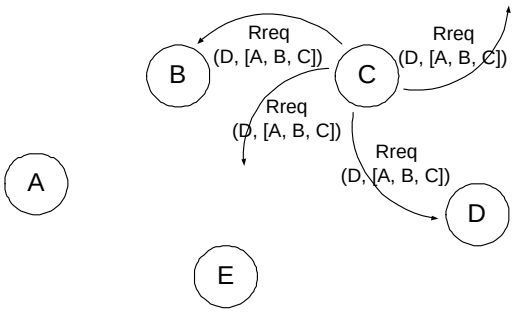


Figure 3: Route Request – 3

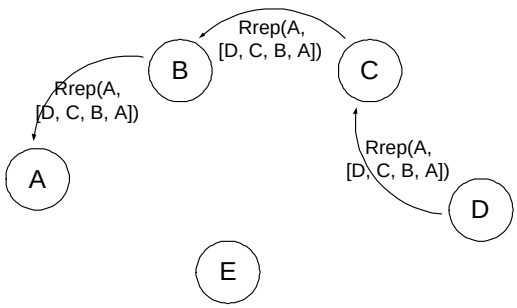


Figure 4: Route Reply

All the nodes who receive the route request from A process it according to the steps in 3.1.2. Node B for example adds himself to the route record and then re-broadcasts the packet (Rreq(D, [A, B] – see figure 2).
Now C is the only node who received the request for the first time, so he is the only one who will re-broadcast the packet after having added himself to the route record (Rreq(D, [A, B, C]) – see figure 3).
D now received the route request and therefore forms a route reply – where he sets the target to A, attaches the route record he received (as source route for A), and sets the actual route record (route which the packet has to take back to A) – and sends it to A (Rrep(A, [D, C, B, A]) – see figure 4).
When A gets the route reply from D, he extracts the source route ([A, B, C, D]) from the packet and adds it to his route cache.

3.1.4 Route Maintenance

Each node that sends or forwards a packet has to ensure that it reaches the next hop (of the route). This can either be done by hop-by-hop ACKs at the data link level or by promiscously sniffing to the transmission of the packet by the next hop (passive ACK).
If there occurs a transmission problem, when sending A's packet from B to C, from which the data link level cannot recover, B sends a *route error* packet (telling that B detected an error while attempting to transmit packets to C) to A (the original sender of the packet encountering the error).
When the route error packet is received by A, the hop in error (B-C) is removed from A's route cache, and all routes which contain this hop (B-C) must be truncated at this point.

3.1.5 Optimizations

- Hosts can learn routes from other route discoveries (route request and route reply packets), either if they are routed through them or if they are overhearing them (promiscuous mode).
- Hosts can send route replies if they know a route to the target in a route request (instead of forwarding the route request).
- Routes can be shortened. If for example Node B, having routes [B, C, D, ...] in his route cache, overhears a packet from D, he can shorten all these routes to [B, D, ...].
- Route discoveries initiated from any host for the same target could be limited with an exponential backoff.
- Negative information (host not reachable) can be cached.

3.2 Security Issues

There are a lot of security issues applying to all sorts of networks. We will state the security issues for wired networks - which apply to MANETs as well - and the MANET specific security issues. For each of the issues we will give a short explanation, for more detailed explanations please refer to a Security Glossary.

3.2.1 Security Issues in wired Networks

- Authenticity: the validity and conformance of the original information [2].
- Integrity: the need to ensure that information has not been changed accidentally or deliberately, and that it is accurate and complete [2].
- Confidentiality: the need to ensure that information is disclosed only to those who are authorized to view it [2].
- Availability: the need to ensure that the business purpose of the system can be met and that it is accessible to those who need to use it [2].
- Access control: ensures that resources are only granted to those users who are entitled to them [2].
- Non-repudiation: the ability for a system to prove that a specific user and only that specific user sent a message and that it hasn't been modified [2].

3.2.2 MANET specific Security Issues [5]

- Cooperation and fairness: without cooperation and fairness there is no routing in MANETS.
- Confidentiality on location: in some scenarios, e.g. in a military application, routing information can be very important.
- No traffic diversion: routes should truthfully reflect the knowledge of the network.
  - Routing
  - Forwarding

3.3 Active vs. Passive Attack

There are two different kinds of attacks: active and passive. To understand the difference between them, some definitions:

active attack

- attempts to alter system resources or affect their operation [11]
- involves an adversary who modifies or injects messages [4]

passive attack

- attempts to learn or make use of information from the system but does not affect system resources [11]
- involves an adversary who attempts to defeat a cryptographic technique by simply recording data and thereafter analyzing it [4]

For a detailed explanation of the types of passive (release of message contents, traffic analysis) and active (masquerade, replay, modification of messages, denial of service) attacks in all detail we recommend chapter 1.2 of [12].

3.4 MANET Attack Tree

Having discussed the security issues of MANETs, what kind of attacks exist?

active attacks

- incorrect forwarding
  - no forwarding
    - Data packets
    - Routing packets
      - Error packets
      - Route request packets
      - Route reply packets
  - too slow
  - replay
  - changing of packet (before forwarding)
    - route change
      - silent route change
      - route salvaging although no error has been observed (DSR specific)
    - data manipulation
  - forwarding messages to partners for analysis
- denial of service
  - bogus routing information
    - replay of old routing information
    - 'black hole routes'
  - distorting routing information
  - cause overload
    - sending route updates at short intervals
    - sending route requests at short intervals
- lack of error messages, although an error has been observed
- gathering information
  - unusual traffic attraction
    - advertising many very good routes
    - choose a very short reply time, so the route will be prioritized

passive attacks (eavesdropping)

- gathering information
  - use promiscuous mode to listen to traffic destined for other nodes

4 Design

In this section we will present the design of our protocol. The goal is to prohibit the *incorrect forwarding* attack from 3.4. For the moment we want to focus on the incorrect forwarding of data packets. We start with the assumptions we made, then we state how trust should be determined. Depending on this, we state which nodes should be looked at as misbehaving and how this will affect routing.

4.1 Assumptions

To proof the concept of our design we will first implement our protocol in a network simulator. We assume certain things:

- each node has a unique id
- links are bidirectional
- source IPs are not spoofed
- nodes do not have a priori "trust" relationships
- all nodes give correct reputation
- misbehaving nodes do not forward data packets, but act correctly for everything else (which is selfishness)

For the real world use of our protocol these assumptions do not hold. They will need to be assured, see section 8 for more information.

4.2 How to Determine Trust

4.2.1 Direct Trust

When we want to know if we can trust some node B, we can route some packets via B and see (by sniffing in promiscuous mode) if B forwards them correctly. The fraction of correctly forwarded packets in relation to the total amount of packets then gives us some idea on how trustworthy B is.
To be a bit more concrete: For every packet A sends to B, A puts a copy of it in a cache. If A sees B forwarding the packet correctly A promotes B for that. If A sees that B changed the packet or if A does not see the packet for some time, A punishes B. Then the packet is deleted from the cache.
A node can change its behavior over time, so we are only interested in its current behavior. Because of this we create an array of some size x (for B), where we can store the behavior of B for the last x packets we routed via B. A promotion gets a '1', a punishment gets a '0'. If we now sum up all the values and divide this sum by x (if we have only seen y < x packets we divide by y), we get the fraction of correctly forwarded packets by B. A can then use this value as his *direct trust* for B:

dt(A, B) = $\frac{\#forwarded}{\#sent}$

 # forwarded = number of packets (coming from A) correctly forwarded by B
 # sent = number of packets sent to B (by A)

All these arrays (one for each neighbor node) are managed in a sorted list. In this way all nodes establish some direct trust values for their (direct) neighbors.
What if we want to take other measures (than forwarding) into account? We can create an array per measure and node - i.e. a matrix - (at each node) and then get one trust value per measure in the way we did it before. These different values can then be combined to one by giving each measure a weight and adding up the weighted values (the result should be between 0.0 and 1.0). This value can then be used as our direct trust in the same way than before.

4.2.2 Reputation

General Idea
What happens if there is a new node? We can either watch it and establish some direct trust values or we can ask our neighbors for references. Asking for references is done in everyday life and works quite well. So why should we not do it in MANETs? The question is how do we ask for references, what answers will we get, and how do we combine them?
So lets create two new types of packets *reputation request* and *reputation reply* analogous to the existing route request and route reply packets of DSR.
If A now wants to get references for B, he creates a reputation request, sets himself as source, sets B as target and broadcasts it to his neighbors (ttl = 1). Every node N receiving this request then looks if he has a direct trust value for B and if yes creates a reputation reply (from him to A) which is carrying this value. After some time A can then combine the reiceived values to a reputation value for B:

$$Reputation\, r(A,B) = \frac{\sum_{i=1}^{n} dt(A,N_i) * dt(N_i,B)}{n}$$

This reputation depends on when it is calculated and how many answers (route replies) have been received (and from whom). It can be completely different at some other node and/or time!

Whom to Ask
The question is who do we ask for reputation? Only our neighbors (as described above) or everybody or trusted nodes? If we ask only our neighbors we can determine trust only for nodes near us (worst case: maximal 2 hops away). If we ask everybody we can get trust values for nearly every node. But in this case we have two problems: First, reputation replies are routed over other nodes (we do not get them directly) and thus can be manipulated on their way. Manipulation of packets - except dropping - could be prevented by using cryptography, but this is not easy to achieve (establishing a public key infrastructure in MANETs is a problem of it's own). Second, we get a big overhead on the protocol by flooding the net with reputation requests. Because of this it is better to send reputation requests only to neighbors. In this way we will get a local but "correct" view (we cannot exclude spoofing) with not so much overhead. And when moving around we can expand this local view in a more complete one.

How to combine reputation replies
The question is how do we combine all the direct trust values from the reputation replies together to one reputation value. One possibility is to weight them with the direct trust values we have (as described above). Another possibility is to look at the answers and compare them. If we have a lot of nearly identical values we can treat them as correct and the few "differing" values we can treat as incorrect. We then can either just average the correct values to get our reputation. Or we can remember which nodes gave a correct answer and which ones did not and use this information to weight future answers.
The question is if a value of 0.1 really is incorrect if all the other nodes answer with 0.9? Maybe 0.9 just tells us that the forwarding is perfect and 0.1 tells us that delay is very bad. Or maybe there is a node dropping all the packets from one node but correctly forwarding all the packets of all the other nodes. This will also result in very different direct trust values.
If we assume that a node is either good or bad, i.e. either behaves correctly for everything or nothing, answers of nodes can be weighted with the direct trust values we have for them. Therefore the first possibility (as described above) seems fairer.

Reputation Request and Reply Packets
Reputation request and reply are very similar to route request and reply. The following information is contained in these two packet types:
reputation request: sender, target, node reputation is requested for, TTL
reputation reply: sender, target, node reputation is given for, reputation value

4.2.3 Total Trust

Now we have some direct trust values and some reputation values. They can be combined in the following way:
Total Trust tt(A, B) = a * dt(A, B) + (1-a) * r(A, B) 0 < a < 1

4.3 Which Nodes are Misbehaving?

First we need to observe that it is not possible for us to differentiate the different types of misbehavior. We cannot say if a node is misbehaving because he is malicious, just selfish, has no battery left and so on. We - in the following - just try to somehow determine which nodes are misbehaving without too many false positives.

In 4.2 we calculated trust values. But how to use them? When do we trust a node for routing packets?
The idea is to exclude misbehaving nodes from the net. Nobody wants to send his packets via a misbehaving node where one can not be sure if it reaches its destination (unchanged), but when nobody sends packets via misbehaving nodes they are relieved from the burden of forwarding packets, and therefore rewarded for their misbehavior. Many proposed protocols work like this. But we do not want to encourage misbehavior, we want to enforce cooperation. This is achieved when packets of misbehaving nodes are dropped by the other nodes (instead of forwarded). In this way, misbehaving nodes are completely excluded from the network. Because we want to give misbehaving nodes a chance of changing their behavior we will route some of our packets through them (so that we can monitor their behavior), but we will not forward packets for them.
How do we determine if a node is misbehaving? A trust value can be small if a node dropped packets, but also if they never reached him or if we have not seen the correct forwarding. For the forwarding of packets it does not matter why a node has a small trust value. We therefore choose nodes with high trust values to maximize the probability of reaching the destination. In the other case we want to drop packets of misbehaving nodes only.
All this can not be achieved at 100%, but the errors should be minimized. So we need some threshhold β, where all nodes with total trust < β will be treated as misbehaving. A good value for β needs to be determined by analyzing simulations with different values for β.

4.4 Which Route to Choose

Now, which one is the best route? We can take the route where the possibility of reaching the destination (product of the total trust values of all the nodes on the route =: route value) is the biggest. But if we have a route with only x hops and another one with y hops, y » x, should we take the one with y hops if its route value is better? So what we want to do is to find the shortest path in a graph based on several metrics.
So what actually is done here is, that we exchange the *shortest path metric* with a *trust-based metric*. Therefore the following idea (using "sequential filtering"):
For sending a packet from A to B:

1. Set *lr1* to the length of the shortest route from A to B (using dijkstra - where each link has cost 1).
2. Search for the shortest route from A to B, where the cost of a link from N1 to N2 is set to 1 / tt(A, N2). Set *lrc* to its length.
3. If lrc - lr1 < α (some threshhold) use the route found in step 2, else ...

So here we do not exclude misbehaving nodes - to give them a chance of changing their behavior - but the routes with misbehaving nodes on them will not be chosen very often.

5 Implementation

Ns-2 [1] is a network simulator in which the dsr protocol is already implemented. Therefore it was choosen for the implementation of this dsr extension. Because of this there will be a short introduction to ns-2 before coming to the "real" implementation.

5.1 History of ns-2 (from [1])

Ns is a discrete event simulator targeted at networking research. Ns provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks.
Ns began as a variant of the REAL network simulator in 1989 and has evolved substantially over the past few years. In 1995 ns development was supported by DARPA through the VINT project at LBL, Xerox PARC, UCB, and USC/ISI. Currently ns development is supported through DARPA with SAMAN and through NSF with CONSER, both in collaboration with other researchers including ACIRI. Ns has always included substantal contributions from other researchers, including wireless code from the UCB Daedelus and CMU Monarch projects and Sun Microsystems.

5.2 The DSR Implementation of ns-2

We were only interested in the dsr part of ns, so we handled the rest as a blackbox. But to understand our extension to dsr one has to know certain things about the dsr implementation of ns, which consists of several files in the dsr-subdirectory of ns. We will give a short survey of the files we used, the others (add_sr.cc, cache_stats.h, constants.h, dsr_proto.{h,cc}, flowstruct.{h,cc}, mobicache.cc, path.{h,cc}, requesttable.{h,cc}, simplecache.cc, sr_forwarder.{h,cc}, srpacket.h) are handled like the rest of ns, as a blackbox.

hdr_sr.{h,cc}

This includes the definition and some handling of the dsr packet header and the different dsr packet types like route request, et cetera.

routecache.{h,cc}

This is an implementation of a cache for storing routes (basic stuff).

linkcache.cc

Linkcache is an extension of routecache, which divides routes into links and stores these links. These links are then used to compute routes (shortest path, using dijkstra).

dsragent.{h,cc}

The DSRAgent does the routing and handling of the various packets. We will distinguish four different cases:

1. I am the sender of a packet: First create a packet and fill the header with the needed information. Then - if you do not want to broadcast it - get a route for it and add this route to the header. And last but not least send the packet out.
2. I am the receiver of a packet: If the packet is
   - a route request: answer with a route reply (return the source route)
   - a route reply: add the source route to the cache

  - a route error: delete the link in error from the cache
  - data: ignore it (data packets are not implemented yet)

3. I am on the route of a packet destined for somebody else: Do the same as in the case above (2), but additionally forward the packet to the next hop in the source route.

4. I see a packet (not destined to me) by promiscously sniffing: If the packet is

  - a route request: ignore it (one could send route shortenings if applicable)
  - a route reply: add the source route to the cache
  - a route error: delete the link in error from the cache
  - data: ignore it

For the visualization of cases 2, 3 and 4 please refer to appendix B.

5.3 Our Changes

hdr_sr.h

We added two new packet types - rep_request and rep_reply packet - for handling reputation. For details please refer to Appendix C.

linkcache.cc

Instead of computing the shortest path (smallest number of links) from a node A to some other node B, the most trusted path (the probability of a packet arriving at B being maximal) is computed. This is reached by setting the costs of a link from node N1 to node N2 to '1 / tt(A, N2)' and then running dijkstra with these linkcosts (instead of using 1 as linkcosts for computing shortest path).

routecache.h

A "nodelist" was added for storing trust values (for using them to compute linkcosts).

dsragent.cc

There are several things that were added to the DSRAgent.
First there now is a SendCache where all data packets are stuck in before sending out. If a packet is then seen correctly forwarded by the one I sent it to, he will be promoted for that. If I do not see him forward it (correctly) then (after some time) he will be punished for that. So that is how we get direct trust values for neighbors.
For each node I want to communicate with, I ask my neighbors for reputation by sending a reputation request. Reputation requests are sent when I have no reputation for a node or when it is too old (last reputation longer ago than 'update_rep').
For evaluation purposes the behavior values (direct trust, reputation and total trust) are printed (every 10s and) at the end of the simulation.

So if we have a look at the four cases from above again what is additional?

1. I am the sender of a packet: Sending of packets is the same as before. The only change is that there are two additional types of packets (reputation request and reply) that can be sent out.

2. I am the receiver of a packet: If the packet is

  - a reputation request for node B: If I have a direct trust value for B, send a reputation reply. Else, do nothing.

  - a reputation reply for B: add the reputation to B's behavior list.

3. I am on the route of a packet destined for somebody else: This will currently not happen for reputation request and replies, because they are - in the moment - sent only to direct neighbors.

4. I see a packet (not destined to me) by promiscously sniffing: If the packet is

  - a reputation request: this should be case 1
  - a reputation reply: ignore it
  - data: See if it is in my SendCache. If yes, promote the sender if he correctly forwarded it and punish him if not. If it is not in the SendCache, ignore it.

behavior.cc (new file)

The 'Behavior' class is used to store and handle the various trust values like 'Forwarding', 'DirectTrust', 'Reputation', 'TotalTrust, ...
For more information on the variables (NEUTRAL, minp, windowsize, update_rep) please refer to appendix F.

sorted-list.{h,cc} (new file)

Implementation of a double-linked list where the elements are sorted sorted according to their ID. It is possible to 'insert', 'find' and 'remove' single elements, to 'empty' the whole list, and to iterate over the list. Instances of this list are used in 'behavior.h' and 'dsragent.cc' to store trust and reputation values for the different nodes.
Disclaimer: This is a 100% handwork (and not using STL deque) because gcc-3.2.3 has some problems with templates.

6 Evaluation

To evaluate our protocol, we run some simulations with our implementation, which we did as a DSR-extension in ns-2 (see 5 for details). We will first give some definitions of terms used in the evaluation, then all the parameters used for the simulations are listed, followed by some evaluations of the results.

6.1 Definitions

- false negative: When a test incorrectly gives a negative result.
- false positive: When a test incorrectly gives a positive result.
- good classification: total trust >= beta
- misbehaving classification: total trust < beta
- neutral classification: Classification of nodes we know they exist, but do not know enough about to classify them. They are treated as good.
- Goodput: # received data packets / # sent data packets
- Routing overhead: # all routing packets / # received data packets
- Reputation overhead: Routing overhead + (# all reputation packets / # received data packets)

6.2 Simulation Parameters

If not stated otherwise we run our simultations with the following parameters:

- NEUTRAL = 0.1
- beta = 0.1
- update_rep = 10.0
- windowsize = 100
- minp = 10
- simulation time = 1000s
- # nodes = 75
- simulation area = 1000m x 1000m
- pause time = 5s
- maximal speed = 10m/s
- maximal number of connections = 30
- sending interval = 0.25 (sending rate = 4)

For explanations on the parameters please refer to appendix F. For explanations on the generation of scenarios please refer to appendix E.

6.3 Categorisation

The idea here was that misbehaving nodes get categorised as misbehaving by the others and good nodes get categorised as good. Because neutral nodes are treated as good, there should not be too many neutral classifications of misbehaving nodes. The faster nodes get classified correctly, the better our goodput gets. The following classifications are evaluations for two randomly picked nodes (a good and a misbehaving one) of one run with 5 misbehaving nodes (nodes 70 - 74). They shall give some idea on how such classifications look like.

6.3.1 Of a Good Node

In figures 5 and 6 we can see how a good node is categorised by the others using direct trust (dt) and using total trust (tt) respectively (over 5000s of simulation). For details please refer to appendix D, where you can see the classification a node X, X between 0 and 74, has got for node 5 at some time t, where t is between 0 and 1000s. E.g. after 650s of simulation, node 18 classified node 5 as good.

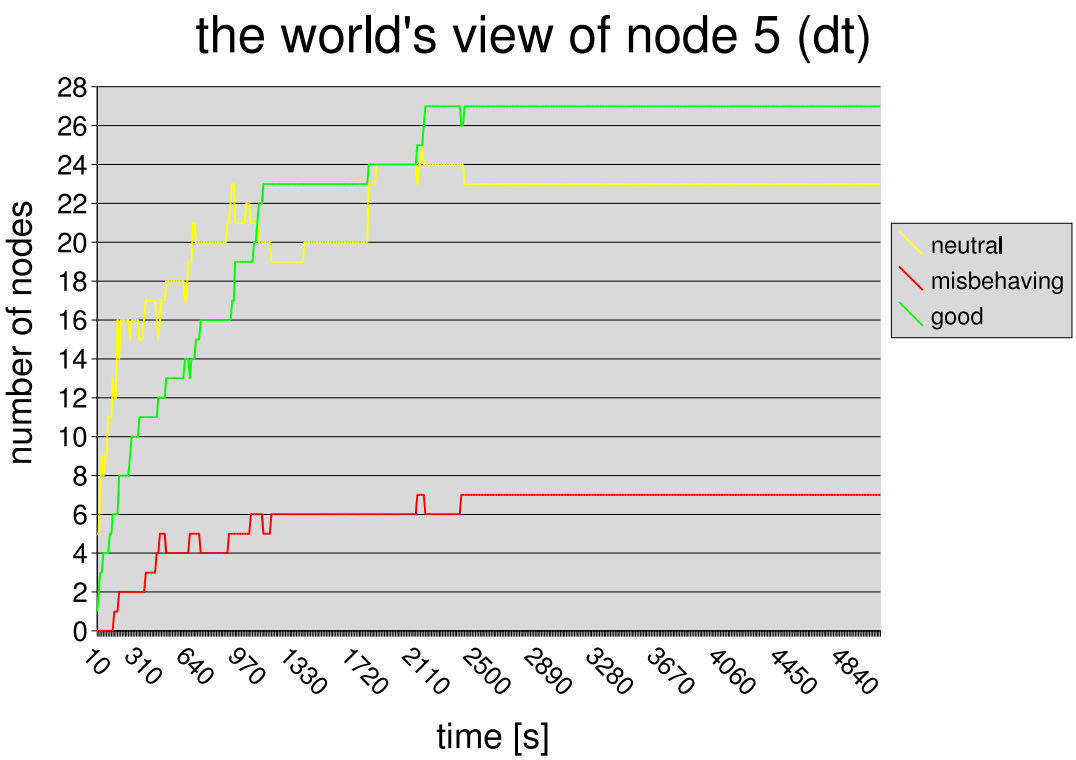


Figure 5: The world's view of node 5 according to direct trust

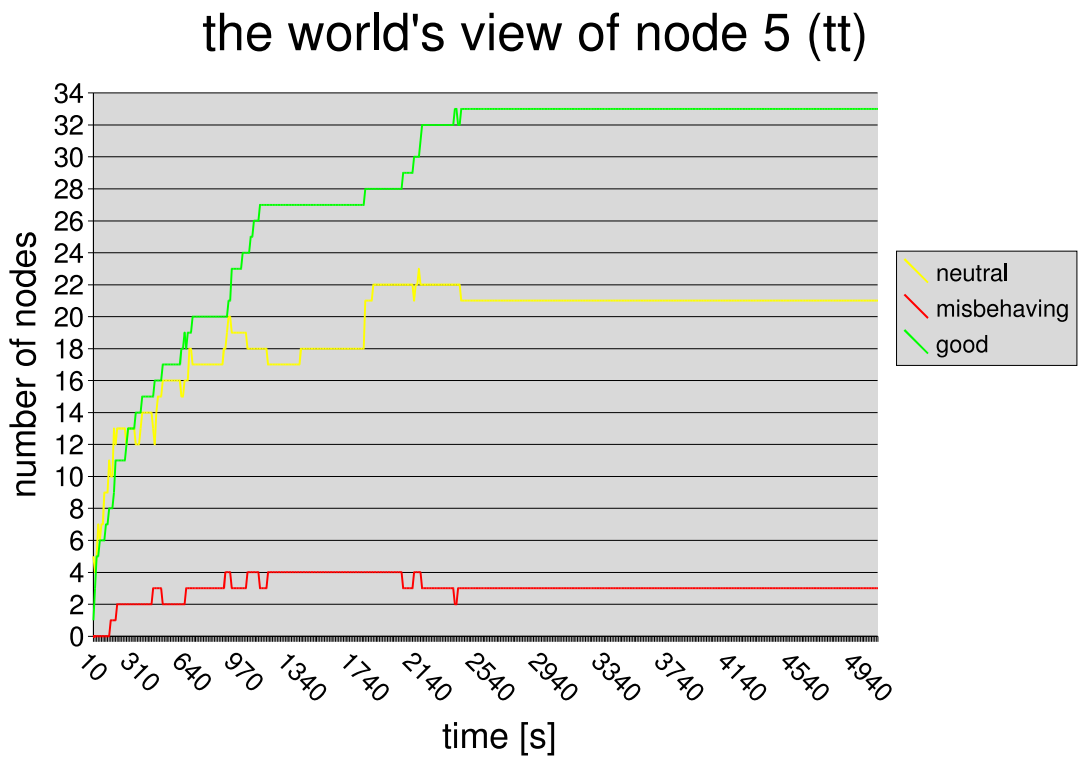
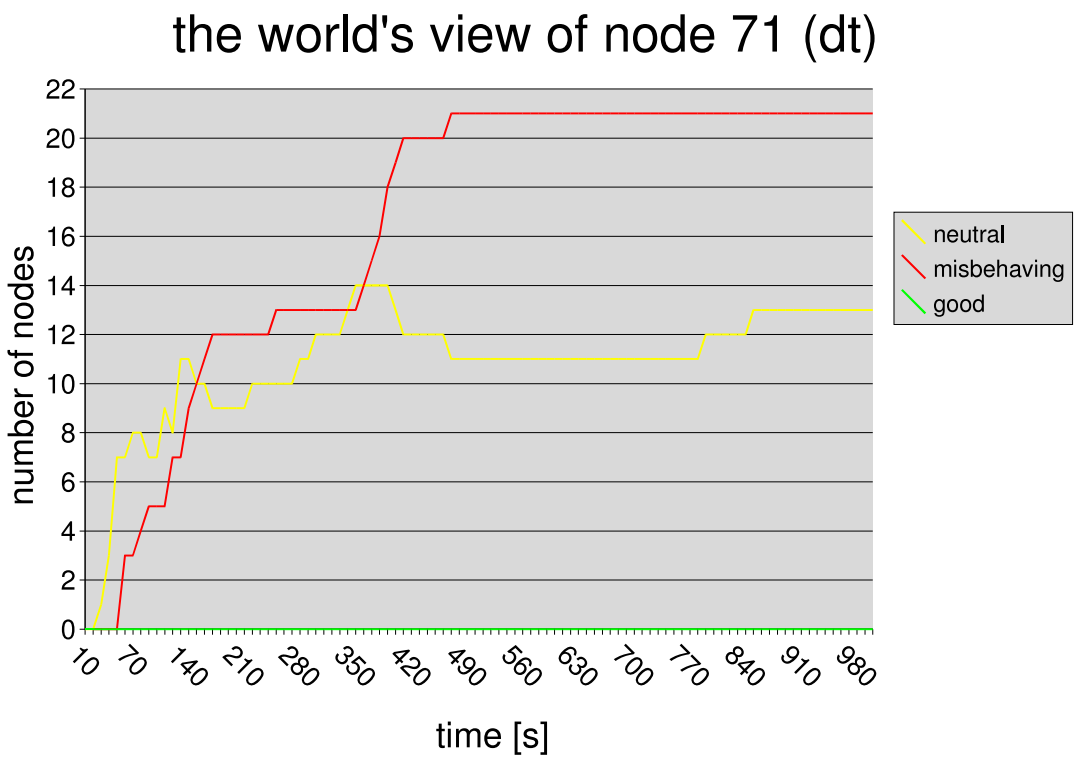


Figure 6: The world's view of node 5 according to total trust

We see that there are a lot of good, a lot of neutral and some misbehaving classifications. Comparing the two figures, we see that using total trust instead of direct trust increases the number of good classifications. The number of neutral classifications is less using total trust than using direct trust, but there are still a lot of them. Because they are treated as good, this does not harm. This shows that node 5 is classified correctly by most of the others, which was the goal. Looking at the misbehaving classifications we see that using total trust helps to decrease them, but still there are quite a lot of them. There are two causes for this: First, many of the correctly forwarded packets are not seen, because of overlay and therefore correct behavior is punished instead of promoted. Second, every node has reputation only for a couple of other nodes (and not of all of them). But only with reputation we can correct the wrong classification (by direct trust) of nodes. As long as we have that large number of false positives, dropping packets of misbehaving nodes isn't a good idea, because too many packets of correctly behaving nodes will be dropped as well.
So without dropping packets of misbehaving nodes our protocol on one hand enhances goodput but on the other hand misbehaving nodes are rewarded for their behavior by taking load away of them. So the next step really should be to lessen the number of false positives (see section 8), for being able to punish misbehaving nodes by dropping their packets, without dropping too much packets of correctly behaving nodes.

6.3.2 Of a Malicious Node

In figure 7 we can see how a misbehaving node (node 71) is categorised by the others using direct trust (dt) and using total trust (tt) respectively.



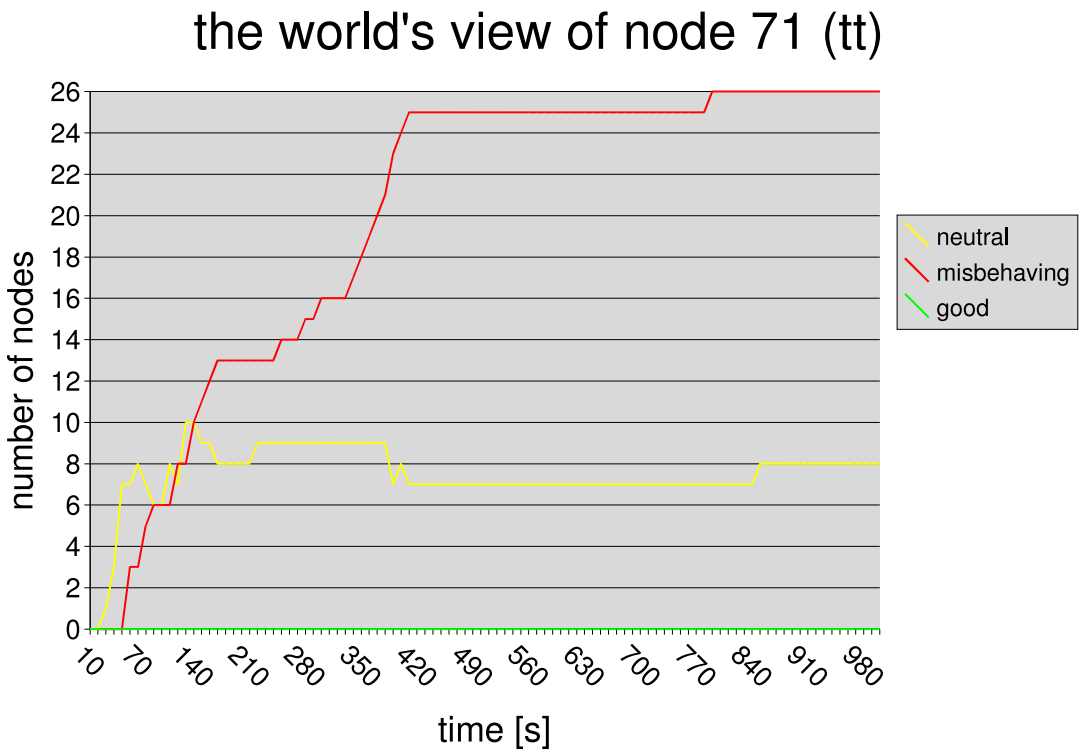
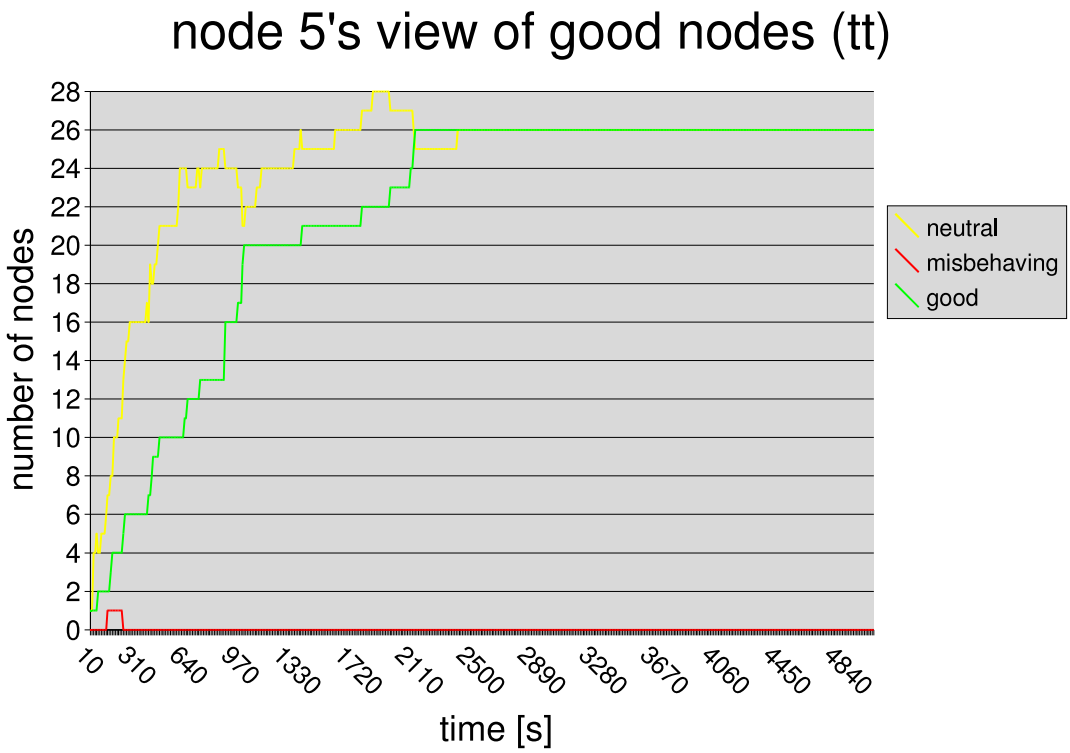


Figure 7: The world's view of node 71

We can see that we have no false positives which is great. But we have a large number of neutral classifications, even if they are fewer of them in the total trust case. Because neutral classifications are handled as good, this large amount of neutral classifications should be decreased. Therefore a future step should be to get more reputation - not only from the neighbors - as explained above - to decrease the number of classifications as neutral and increase the number of classifications as misbehaving.

6.3.3 Of the World

In figure 8 we see how a good node (node 5) classifies the world. The world has 5 misbehaving and 70 good (one of them is node 5) nodes, therefore there should be a stabilisation at 5 misbehaving classifications and 69 good classifications.



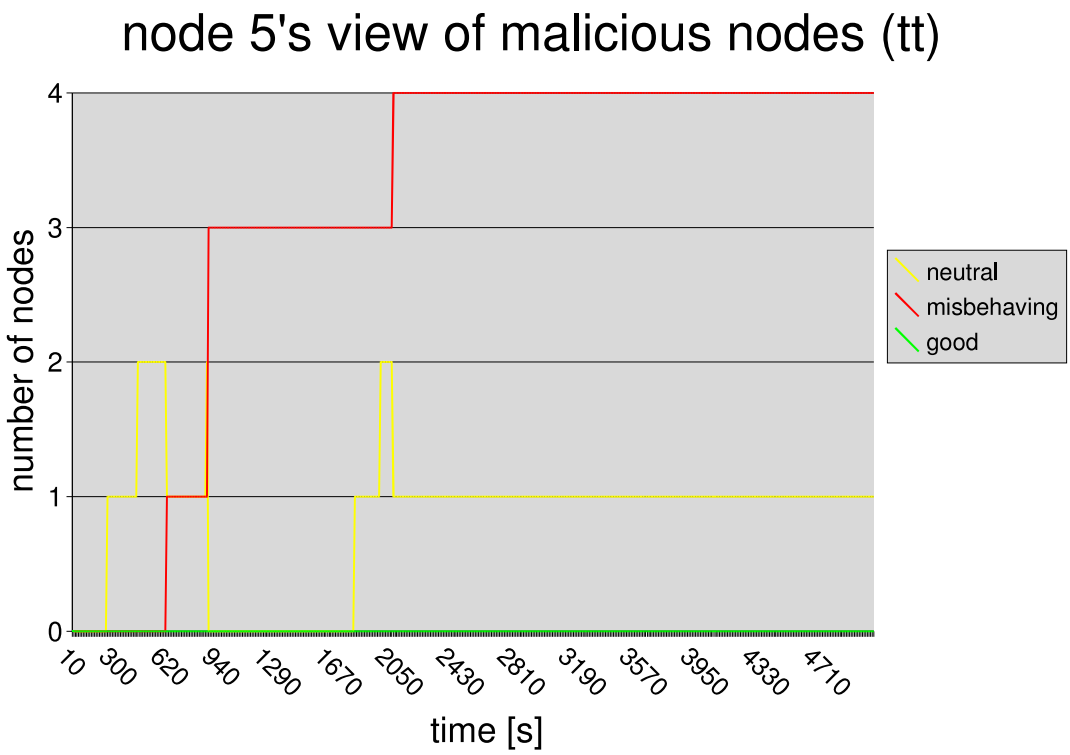


Figure 8: Node 5's view of the world

We see that the classification from one node for the others looks very similar to the classification

of all nodes for one. The same things than before hold here as well, there is nothing more to add. There remains just to say that there is no difference in monitoring between good an misbehaving nodes.

6.4 Goodput

One goal of our extension was to improve goodput, therefore we simulated four different things:

- no: DSR without our extensions
- t(dt): DSR with direct trust
- t(tt,r0): DSR with total trust, reputation from neighbors only
- opt: Optimum - knowing the misbehaving nodes

For each of these four, we did 9 simulation runs (with 9 different seeds), a run consisting of 16 simulations, one for 0, 5, ..., 75 misbehaving nodes each. The resulting goodput is shown in figure 9.

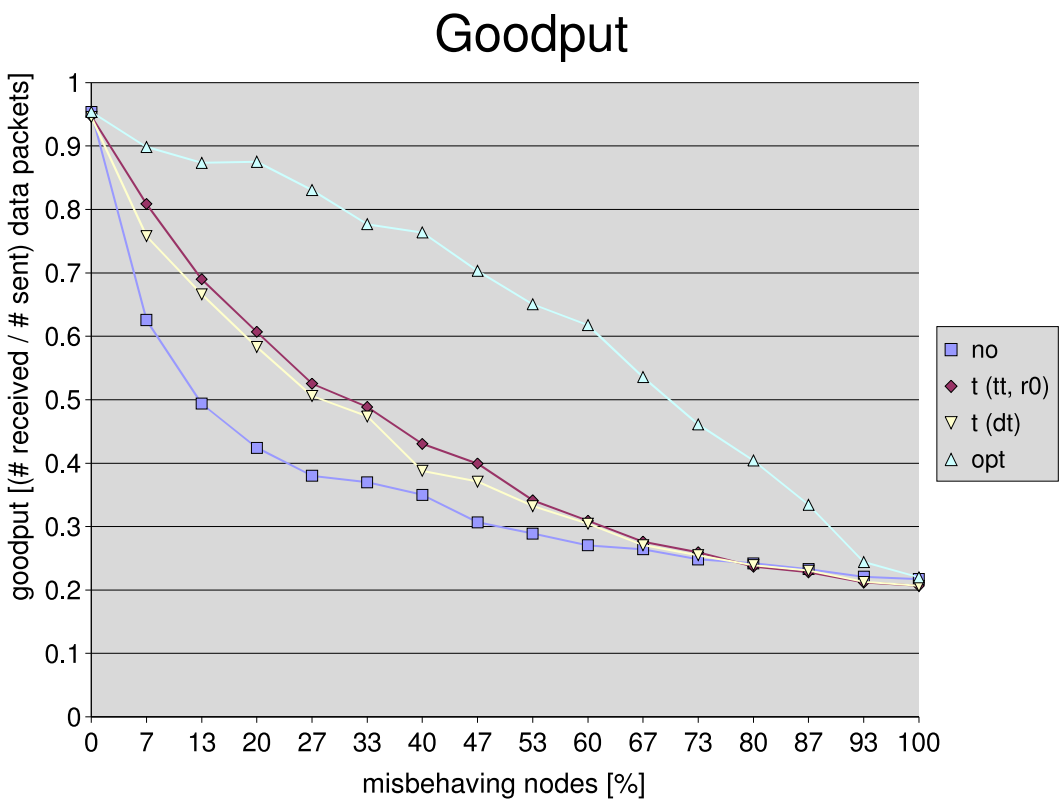


Figure 9: Goodput

For 10% misbehaving nodes we get a 20% better goodput using Direct Trust than without. Using Total Trust does not improve that very much. This is because we get reputation from our neighbors only, which results in nodes having reputation for only half of the other nodes. If we could ensure that all nodes have reputation for all the others - by asking more than just our neighbors for reputation - we suppose that this will improve goodput again. So even if we are still far away from the optimum, we already did a great step towards it, with more work on this issue another step will be possible.

6.5 Overhead

Always when adding a new protocol, the overhead caused should not be too large. In figure 10

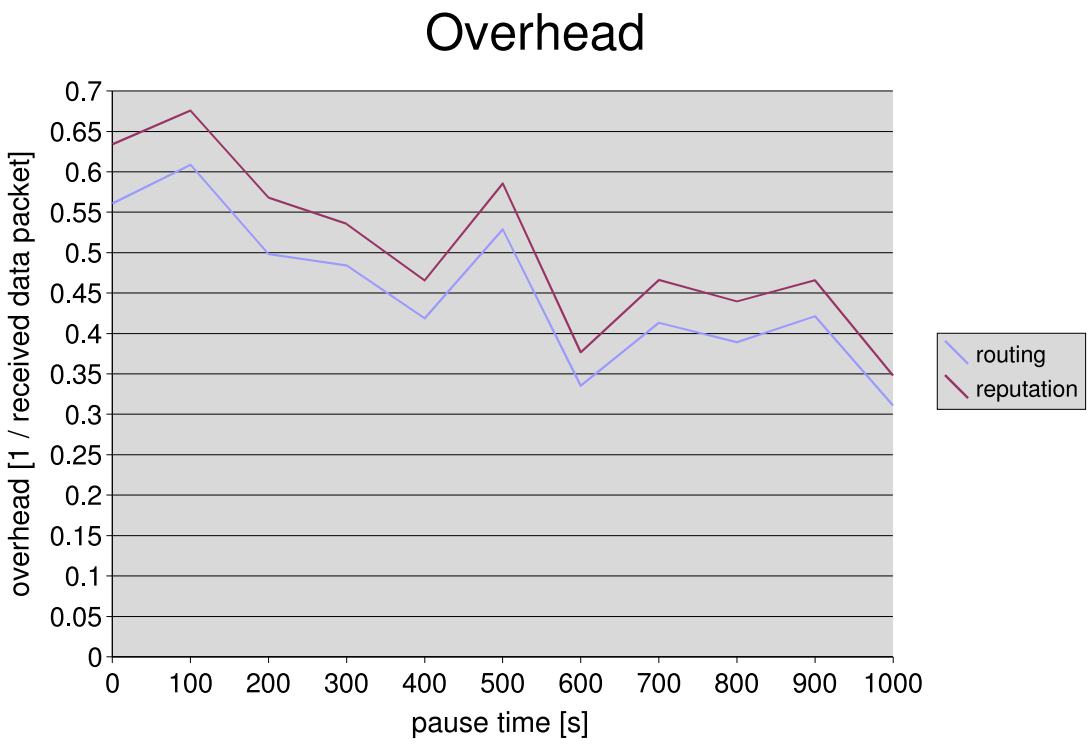


Figure 10: Overhead

it is shown how much overhead we generate (evaluating one simulation run with 5 misbehaving nodes - 70 to 74 - for different pause times). The routing overhead of DSR (route request, route reply, route error) is shown as one line, the other line is the whole overhead we generate with our extension (reputation overhead). For definitions of these overheads please refer to section 6.1. We can see that the additional overhead we generate is less than 10% of the received data packets, whereas we gain 20% in goodput (see section 6.4). So it is worth it.

6.5.1 Prevented Attacks / Misbehavior

Which attacks from 3.4 can be prevented by our enhancement of DSR? Direct trust values are established by monitoring the neighbors. At the moment only forwarding is looked at. In this way nodes who do incorrect forwarding will get a small trust value, and will therefore be looked at as misbehaving by the other nodes. At the moment we are not dropping packets of misbehaving nodes, because of the large number of false positives we have. With dropping packets of misbehaving nodes, which is still the goal, the *no forwarding* attack (see 3.4) can be prevented.
By taking other metrics into account for direct trust, all *incorrect forwarding* attacks (see 3.4) can be prevented.

6.5.2 Attacks We Do Not Resist

At the moment we just detect selfish but not malicious behavior. If a node really wants to attack the system we are not prepared for that. Also we cannot detect allies. By taking other metrics into account for direct trust, as discussed above, it can be detected when a node forwards a packet to a partner for analysis, if this node is not his successor on the path. This would help at least a bit to detect allies. But if the allies exchange some information in a different way we will not be able to detect them.

7 Related Work

7.1 The Quest for Security in Mobile Ad Hoc Networks

In [7] Jean-Pierre Hubaux, Levente Buttyán and Srdan Capkun try to secure ad hoc networks with a self-organized public-key infrastructure. Their system is similar to PGP in the sense that public-key certificates are issued by the users. But, opposed to PGP, certificates are stored and distributed by the users. To find a certificate chain from some user u to some other user v they use some trust graph in which the vertices represent users and the edges represent public-key certificates. Their algorithm `Shortcut Hunter` selects a subgraph consisting of two distinct paths: an out-bound (from u to v) and an in-bound (from v to u). A slight modification of this algorithm results in `Star Shortcut Hunter` which takes a constant c and constructs c out-bound and c in-bound paths (all distinct). They showed that the performance of Star Shortcut Hunter is more than 0.95 when the size of the selected subgraph is around two times the square root of the size of the trust graph.

7.2 An Evidential Model of Distributed Reputation Management

In [13] Bin Yu and Munindar P. Singh evaluate the trustworthiness of a correspondent by combining the local evidence of an agent with the testimonies of other agents regarding the same correspondent.
The idea of this paper is very similar to ours. They took a very mathematical approach and other evaluation metrics, therefore we cannot compare our results.

7.3 Nodes Bearing Grudges: Towards Routing Security, Fairness, and Robustness in Mobile Ad Hoc Networks

In [5] Sonja Buchegger and Jean-Yves le Boudec analyze the special security issues for MANETs. From the different types of misbehavior they then design their protocol, with which they want to achieve that only good behavior pays off in terms of service and reasonable power consumption.
Their system consists of a `monitor`, a `reputation system`, a `path manager` and a `trust manager`. Every node watches its neighbors and sends an ALARM to his friends if he detects a misbehavior. The trust relationships and routing decisions (of a node) then are based on experienced, observed, or reported routing and forwarding behavior of other nodes.
The idea of this paper is very similar to ours as well, the difference is push versus pull. In our protocol, reputation is sent on demand, where in CONFIDANT (the protocol from this paper) ALARM messages are just sent always a misbehavior is detected. This causes a lot more overhead than we have. Also their protocol introduces a new attack to the system (sending false ALARMs) which should be avoided.

7.4 Performance Analysis of the CONFIDANT Protocol Cooperation Of Nodes - Fairness In Dynamic Ad-hoc Networks

In [5] Sonja Buchegger and Jean-Yves le Boudec proposed a protocol for making misbehavior unattractive (see 7.3). In [6] they then do a performance analysis for CONFIDANT. It is shown that a network with CONFIDANT and up to 60% of misbehaving nodes behaves almost as well as a benign network (70 - 80% goodput). In a static network (pause time = simulation time) they even claim to reach 100% goodput for 30% maliciuos nodes.
The goodput they reach with CONFIDANT looks similar to what we got as optimal goodput. Because they used a different setting and a different simulator we cannot comment better on these results. The other result does not say much.

7.5 Mitigating Routing Misbehavior in Mobile Ad Hoc Networks

In [10] Sergio Marti, T.J. Giuli, Kevin Lai and Mary Baker try to solve the problem of malicious nodes with two mechanisms: a `watchdog`, in charge of identifying the misbehaving nodes, and

a `pathrater`, in charge of defining the best route circumventing these nodes. The paper shows that these two mechanisms make it possible to maintain the total throughput of the network at an acceptable level, even in the presence of a high amount of misbehaving nodes.
The problem of this solution (as seen by many other authors) is that malicious nodes are not punished, but rather relieved of the burden of forwarding for others, whereas their messages are forwarded without complaint. This way, the malicious nodes are rewarded, and reinforced in their behavior.
This solution represents what we have now. But contrary to us they do not aim to get rid of the problem that malicious nodes get rewarded for their behavior, which should be the goal.

7.6 Managing Trust in a Peer-2-Peer Information System

In [3] Karl Aberer and Zoran Despotovic try to manage trust in a Peer-2-Peer information system by analyzing earlier transactions of agents and deriving the reputation of an agent from that. In their system agents perform transactions t(p, q). If some agent q cheats within a transaction with some other agent p, p can file a complaint c(p, q) which is stored at a number of agents. When an agent wants to evaluate the trustworthiness of another agent it starts to search for complaints on it. If he receives the same data from a sufficient number of replicas he needs no further checks. If the data is insufficient or contradictory he continues to check (the trustworthiness of the agents that sent him the complaints). With this system they can identify honest and cheating agents with very high probability.
In this solution complaints need to be filed and therefore other agents diskspace and cooperation is needed. When searching for complaints one never can be sure how many of these have been modified. In this solution a lot of time is needed to search for all possible complaints about a node and to verify them. We, in our solution ask a node for its actual view of some other node. Like this we get more information (the number of cheated transactions alone does not say much without knowing when they happend and how many transactions in total there were) with less work.

8 Outlook / Future work

This diploma thesis actually represents a proof of concept. There are certain things we just assumed - see 4.1 - but not ensured. And there are also some problems left we did not solve until now. For reaching even better values and for being able to use the protocol in practice further work needs to be done. This includes:

- Ensure that each node has a unique id.
- Get reputation from trusted nodes only.
- Sign or encrypt packets (for ensuring that the source IP is not spoofed and that the contained data is not modified).
- Take other metrics (delay, ...) into account for direct trust.
- Simulate other misbehavior (than just no forwarding).
- Test if changes in behavior are noticed.
- Run longer simulations, e.g. 5000s, and evaluate goodput in the time when trust values are stable (e.g. 2000 - 5000s).
- Initialize trust values (correctly) and see what happens.
- Analyze different ways to compute the "optimal" path. Try to do some load balancing (to avoid congestion).
- Evaluate why many correctly forwarded packets aren't seen by the "predecessor" (and therefore "punished" instead of "promoted"). Is it really just the "overlay" of packets?
- Implement it (outside the simulator) for use in practice.
- Test it in practice (instead of in a simulator).

Some of the problems we encountered could be solved by using cryptography. For that there first needs to be a solution on how to establish a PKI in MANETs, which is a problem of its own.

9 Conclusions

Summarized, we have the following results:

+ 10% misbehaving nodes -> 20% better goodput
+ less than 10% overhead
+ no false negatives (no misbehaving node is classified as good by any other node)
+ reputation decreases the number of neutral classifications
- too many false positives (some good nodes are classified by other nodes as misbehaving)
- too many neutral classifications (even with total trust)
- stabilisation takes up to 2000s
- when routing data only via good nodes this can sometimes lead to an "overload" which results in nearly no goodput at all (which could be even worse than in the "normal" dsr case)
- at the moment reputation does not help improve goodput much

We showed that by using DSR with a reputation-based trust management, goodput in a setup with 10% misbehaving nodes can be improved by 20%, causing less than 10% overhead. This is a great result and represents a proof of concept.
Still there are a lot of things that need to be improved. Until now, reputation does not help much in improving goodput, direct trust does most of it. We showed that reputation helps in getting better classifications, the problem is that we have reputation for a small number of nodes only. Another problem is that the system takes up to 2000s to stabilise, but our simulations for the evaluation of goodput were run for 1000s only. For getting a further improvement of goodput, when using total trust instead of direct trust, more reputation is needed. Therfore, the next step will be to get reputation from more nodes than just our neighbors. This results in the question whom we trust for giving correct reputation, and how we protect reputation replies from spoofing.

There is still a lot of work needed, before being able to use our approach in practice. This includes at least the following:

- the number of false positives and neutral classifications should be lessened (by getting more reputation)
- packets should be signed or encrypted (to protect from spoofing)
- other metrics for direct trust should be taken into account

10 Acknowledgement

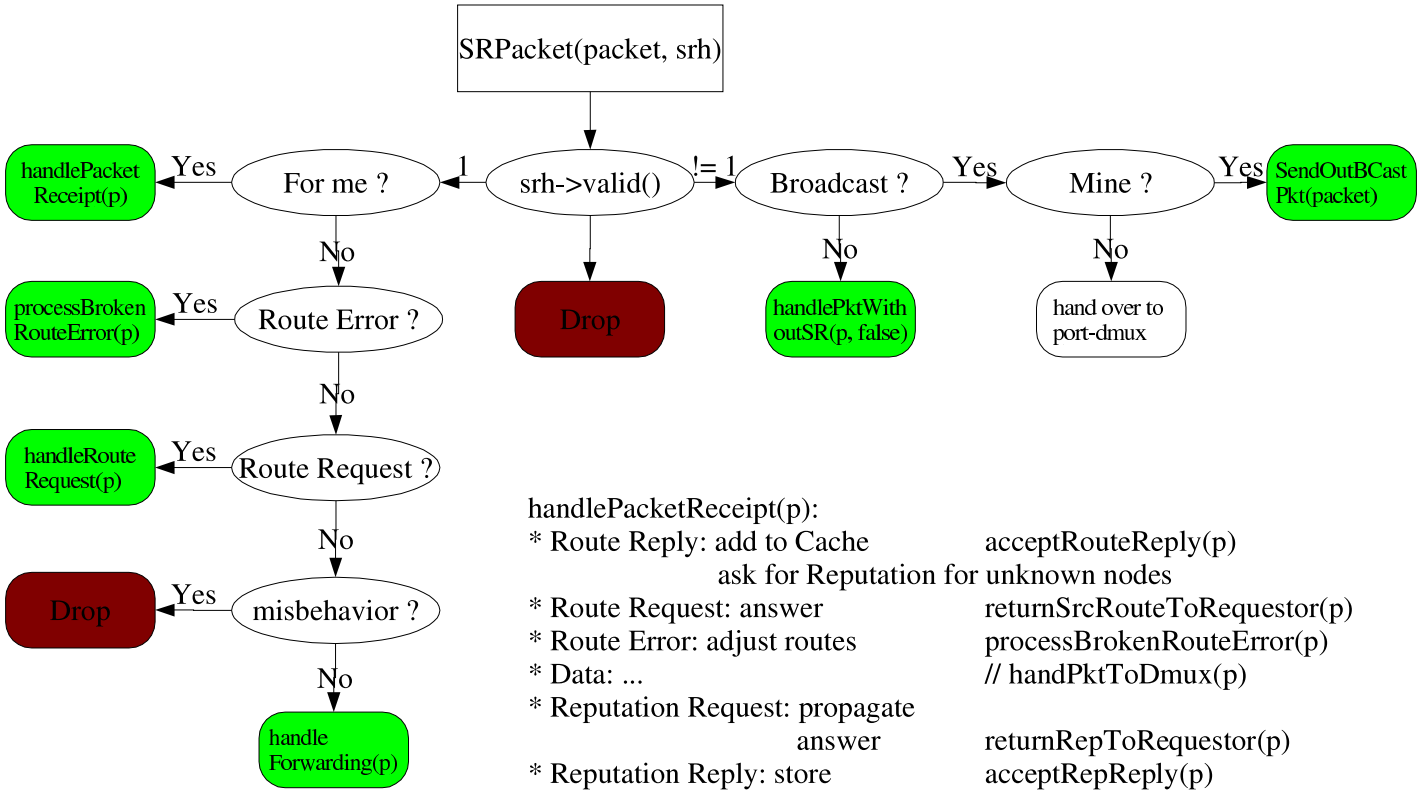
I would like to thank

- my supervisor Vincent Lenders for all the high-producing discussions, his suggestions and the time he had for me whenever I needed him.
- Sonja Buchegger for the links (to MANET papers).
- Adrian von Bidder for his assistance, for the time he spent reading my thesis, for the corrections he gave me and for all the discussions we had about the topic.
- my parents to make my studies at ETH possible.
- all the people at TIK for their help they gave me.
- all the other people who somehow helped me.

A Timetable

	31. 03.	07. 04.	14. 04.	21. 04.	28. 04.	05. 05.	12. 05.	19. 05.	26. 05.	02. 06.	09. 06.	16. 06.	23. 06.	30. 06.	07. 07.	14. 07.	21. 07.	28. 07.
Installation	X																	
Reading related work	X	X																
Getting familiar with ns		X	X	X	X													
Attack tree			X															
Design of the protocol				X	X	X												
Implementation							X	X	X	X	X	X						
Evaluation													X	X	X	X		
Finishing																	X	X
Discussion									X									X

B Flow of a SRPacket in DSRAgent




```
handlePacketReceipt(p):
* Route Reply: add to Cache            acceptRouteReply(p)
                ask for Reputation for unknown nodes
* Route Request: answer                returnSrcRouteToRequestor(p)
* Route Error: adjust routes           processBrokenRouteError(p)
* Data: ...                            // handPktToDmux(p)
* Reputation Request: propagate
                        answer         returnRepToRequestor(p)
* Reputation Reply: store              acceptRepReply(p)
```

Figure 11: Flow of a SRPacket in DSRAgent

C The DSR Header

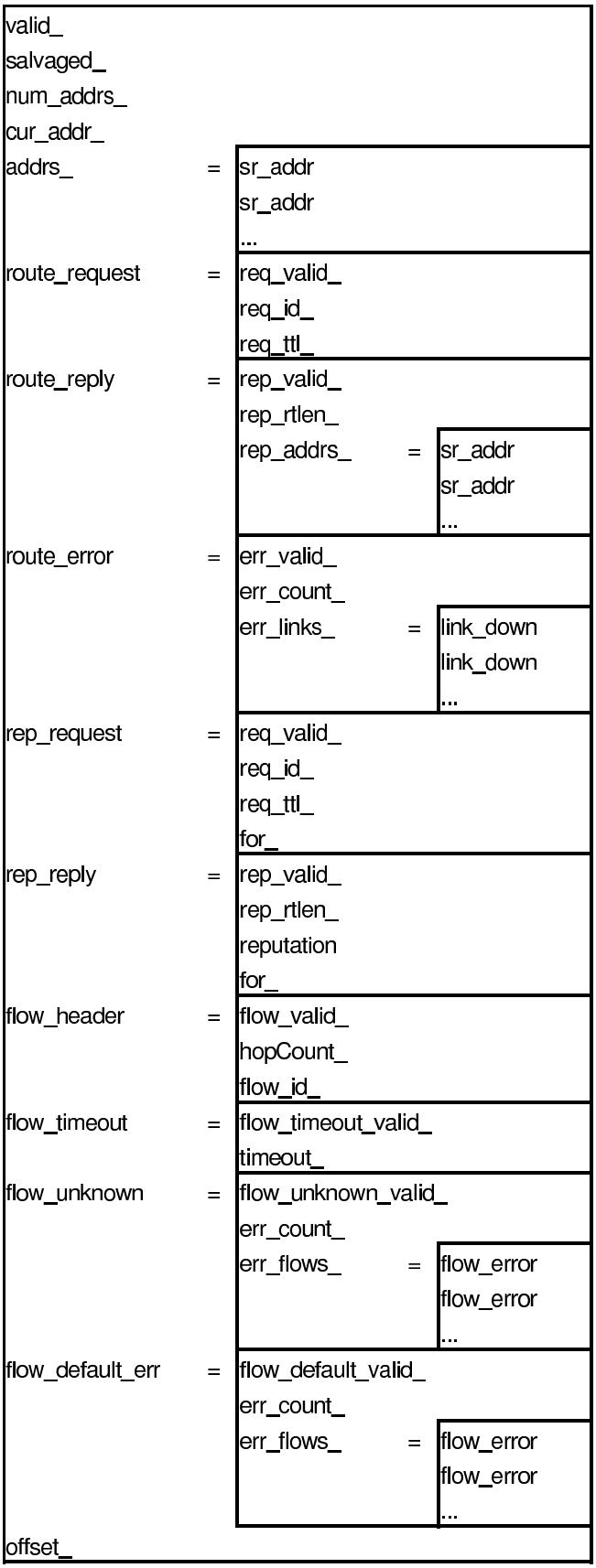


Figure 12: The extended DSR header (hdr_sr)

D The World's View of Node 5 - Details

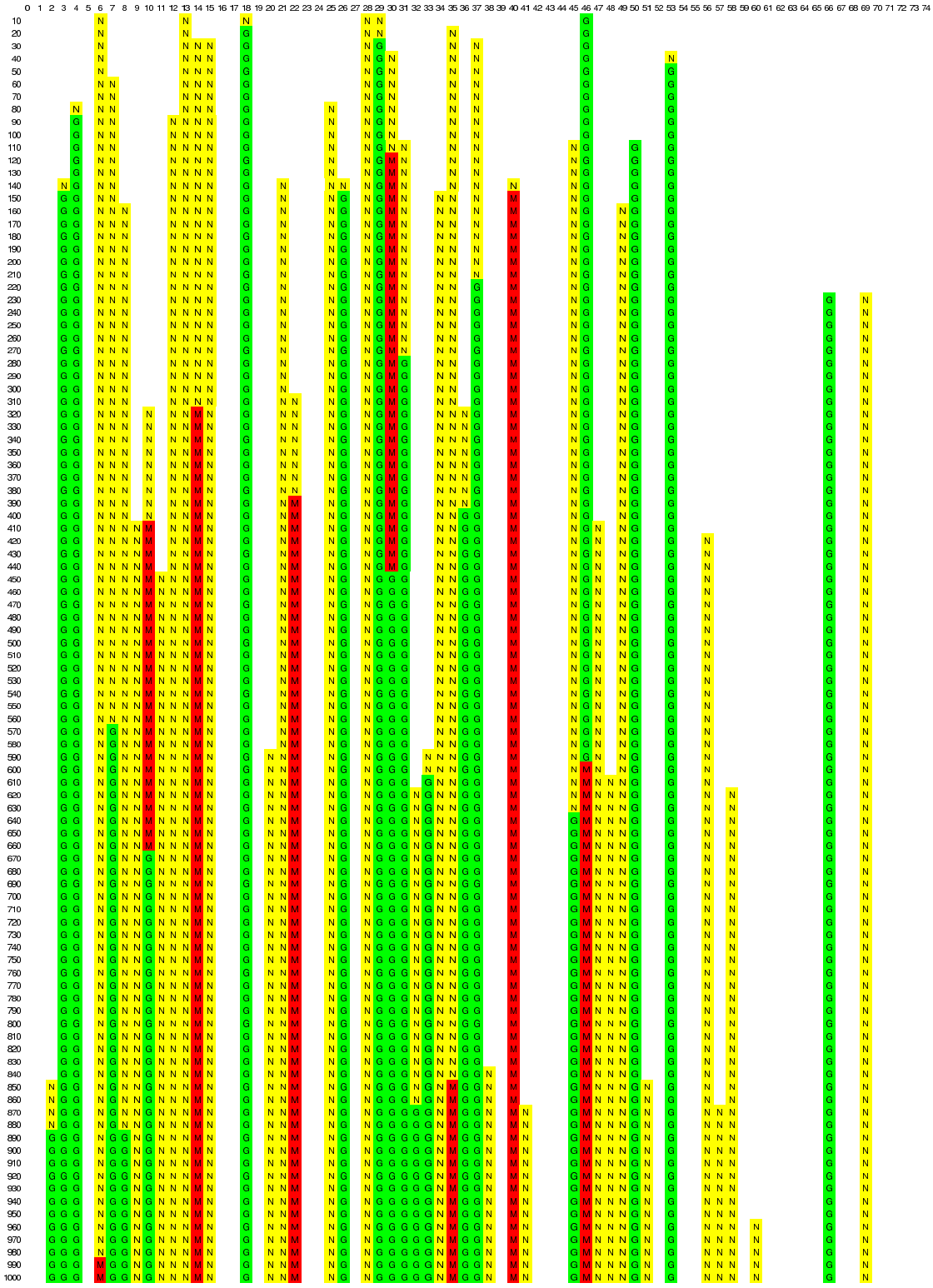


Figure 13: Categorisation of node 5 (by the world) between 0 and 1000s

E How to Setup the Environment

The following points will explain you how to setup the environment we used, how to run your first simulation and how to evaluate it. The scripts used can be found in **DA-Diana/Testing/** on the CD if not stated otherwise. A short explanation of every file in that directory can be found in appendix G.

- Download and install nam, otcl, tcl, tclcl an tk (please refer to "Getting ns and avoiding problems" on [1]). We used nam-src-1.9.tar.gz, tcl8.3.2.tar.gz, tclcl-src-1.0b13.tar.gz, otcl-1.0a8.tar.gz and tk8.3.2.tar.gz.
- Compile our modified version of ns-2.
  - Copy the ns-2 directory from the CD to your disk.
  - Change to this directory (the ns-2 directory on your disk).
  - Build the Makefile according to your machine by typing *./configure*
  - Configure ns according to your needs - see appendix F.
  - Make the binary by typing *make*.
- Take the scenarios from the CD. They can be regenerated the following:
  - For the placement and movement of nodes use **ns-2/indep-utils/cmu-scen-gen/setdest/setdest**, e.g. *./setdest -n 75 -p 5 -s 10 -t 1000 -x 1000 -y 1000* and redirect stdout to a file, e.g. *> scenarios/scen-n75-p5-s10-t1000-x1000-y1000*.
  - For the connections between the nodes use **ns-2/indep-utils/cmu-scen-gen/cbrgen.tcl**, e.g. *../../ns cbrgen.tcl -type cbr -nn 75 -seed 1 -mc 30 -interval 0.25* and redirect stdout to a file, e.g. *> scenarios/cbr-nn75-seed1-mc30-rate4*.
- Run a simulation
  - Rename your ns binary.
    - to **ns_nobeh** if you took configuration F.1
    - to **ns_beh** if you took configuration F.2
    - to **ns_behrep** if you took configuration F.3
    - to **ns_behdroprep** if you took configuration F.4
    - to **ns_x**, where x is the first misbehaving node, if you took configuration F.5
  - Edit the run-Skript belonging to your ns binary. E.g. *run-nobeh* for *ns_nobeh*. You can choose the used scenarios (by seed), the number of nodes (nn) and the first malicous node (mb).
  - Start your simulation(s) by executing the run-Skript edited in the last step.
- Evaluate your simulation
  - Use *./auswerten2* to get a summary of the number of sent and received data packets. If you want to have these numbers for every node separately, use *./auswerten*.
  - If you want to know more about classifications over time, use *./trust-auswerten*. For explanations on how to use this script, please refer to the file itself.

For evaluating goodput, just run different simulation runs (with different scenarios), evaluate all your log-files with *./auswerten2* and compute goodput with these numbers (taking averages).
For evaluating overhead, run your simulations with different pause times - using *./run-overhead* - and evaluate them with *./auswerten-overhead* afterwards.
If you want to do something else, you will need to write your own tcl script, or change one of ours (*_1000.tcl). For this please refer to a ns Tutorial or the ns Manual [1].

F Configuration of the Trust Management

There are different variables (in dsragent.cc and behavior.cc) that can be used to "tune" the trust values in some ways:

- NEUTRAL: this is the value set as direct trust for nodes we know "nothing" about
- beta: a node having a total trust value below beta is looked at as misbehaving.
- update_rep: how often (interval in seconds) a node should ask for new reputation.
- windowsize: number of monitoring results held per node.
- minp: minimal number of monitored packets before computing direct trust (and no longer using NEUTRAL for it).

F.1 "Normal" DSR

In dsragent.cc: Comment '#define USEBEH'.

F.2 DSR using Direct Trust

In dsragent.cc: Use '#define USEBEH'.
In behavior.cc: Return 'getDirectTrust()' in 'getTotalTrust()'.

F.3 DSR using Total Trust (Direct Trust and Reputation)

In dsragent.cc: Use '#define USEBEH'. If you want reputation request sent further (ttl > 0) than your neighbors, you should use your ttl when calling 'sendOutRpReq(reply_route[i], 0)' instead of 0.
In behavior.cc: Return '(getDirectTrust() + getReputation()) / 2' in 'getTotalTrust()'.

F.4 DSR using Total Trust with Dropping

In dsragent.cc: Use '#define USEBEH' and '#define DROPMAL'.
In behavior.cc: Return '(getDirectTrust() + getReputation()) / 2' in 'getTotalTrust()'.

F.5 "Optimal" Routing (Knowing the misbehaving nodes)

In linkcache.cc: Set 'v->ln_cost = 100' for misbehaving nodes v (see 'XXX use this for computing optimal goodput' comments).

G READMEs

DA−Diana/README

```
/**
 * README for my Diplomarbeit
 * 2003 by Diana Senn
 * /cvsroot/DA−Diana/README,v 1.2 2003/07/21 13:11:46 diana Exp
 */

This file together with my report should be enough to understand what I did
in my diploma thesis, what can be found where, ...


Code
====
In the Implementation−chapter of my report you can read in which files you can
find my code. For understanding it, the comments in the code (together with
my report) should be enough. There are 4 types of comments:
* normal        explaining what the code is for
* TODO          urgent things to correct
* XXX           things one should think about
* FFF           here further work needs to be done (see report)
Besides the fourth, all types of comments are used as usual.


Directory structure (without 'CVS' directories)
===============================================
|−− latex
|   '−− template                templates for my report
|−− DA−Diana                    report (use 'make' to generate it)
                                and presentation of my DA
|   |−− Doku                    tutorials, ...  (mostly on ns−2)
|   |   '−− nstutorial          the same as ns−2/nstutorial
|   |       |−− examples
|   |       '−− images
|   |−− Testing                 from/for the evaluation of my protocol
                                see DA−Diana/Testing/README for details
|   |   |−− log
|   |   |−− scenarios
|   |   '−− some−stuff
|   |−− graphics                all figures for my report
|   |−− ps−files                Postscript−files for Presentation, ...
|   '−− related−work            related work (for resumes see DA)
|       '−− unread
|−− ns−2                        source code of ns−2
|   |−− common                  definitions of packet, node, ...
|   |−− dsr                     all the DSR stuff
|   |−− indep−utils
|   |   |−− cmu−scen−gen        scenario−generation
                                (see report for explanations)
|   |   |   '−− setdest
|   |   |       '−− bump−scen
|   |−− nstutorial              ns−2 tutorial
|   |   |−− examples
|   |   '−− images
|   [..]                        treat the rest as blackbox!!!
```

DA−Diana/Testing/README

```
/**
 * README for files in DA−Diana/Testing
 * 2003 by Diana Senn
 * /cvsroot/DA−Diana/Testing/README,v 1.4 2003/07/28 08:29:37 diana Exp
 */
*_1000.tcl      files used for running simulations with run−*

run−nobeh       run simulations with "original" DSR
                creates log−files log/1002_nobeh_*_seed*.{tr, nam}
run−behrep      run simulations using total trust
                creates log−files log/1002_behrep_*_seed*.{tr, nam}
run−beh         run simulations using direct trust
                creates log−files log/1002_beh_*_seed*.{tr, nam}
run−opt         run simulations knowing the misbehaving nodes
                creates log−files log/1002_opt_*_seed*.{tr, nam}
run−overhead    the same as run−behrep but for different pause times
                creates log−files log/1002_nobeh_*_seed5_p*.{tr, nam}
                (new and forwarded route_request, route_reply, route_error
                rep_request and rep_reply should be written to the trace−file)

auswerten       for the log−files generated by run−*
                data packets sent and received per node
auswerten2      for the log−files generated by run−*
                total number of data packets sent and received
auswerten−overhead      for the log−files generated by run−overhead
                counts then total number of data, routing and reputation packets
trust−auswerten outputs the number of classifications as dG, ... per time
                (for details see explanations in the script itself)

dt_over_time*.sxc       some sample classifications
                (data is gained using trust−auswerten)

Goodput1000.sxc Goodput for simulations run by run−* (without run−overhead)
                (data is gained using auswerten2)
Overhead.sxc    Overhead for simulations run by run−overhead
                (data is gained using auswerten−overhead)

scenarios/      the scenarios used by *1000.tcl
log/            the log−files generated by run−*

some−stuff/     files I used, which can be regenerated or are irrelevant
```

Institut für
Technische Informatik und
Kommunikationsnetze

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo

Communications Systems Research Group

Summer 2003

Diploma Thesis

for

Diana Senn (D-INFK)

Main Tutor: Vincent Lenders
Supervisor: Prof. Dr. Plattner

Issue Date: 31th March 2003
Submission Date: 30th July 2003

Reputation and Trust Management in Ad Hoc Networks with Misbehaving Nodes

1 Introduction

In recent years, a paradigm shift can be seen in distributed computing. Moving from quite static networking systems with client-server architectures, more and more peer-to-peer or ad hoc networking systems are appearing. Peer-to-peer or ad hoc networks have a completely decentralized architecture. A *mobile ad hoc network* is such a system consisting of small devices communicating spontaneously over the air. Devices are mobile and thus, the interconnections between those are changing frequently. Furthermore, instead of having dedicated fixed routers, all nodes are relaying the network traffic for other peers.

The dynamic and open nature of *mobile ad hoc networks* raises many new challenges, specially in security. In many reallife settings, nodes will not necessarily want to cooperate with one another. Nodes might be selfish to save own resources or malicious and try to harm others. Therefore, one goal is to establish trust relationships with nodes that are correctly behaving and thus trustworthy and to exclude nodes from the system that are not willing to cooperate. In this thesis, the trustworthiness of a node is considered as the expectation of cooperative behavior from that node.

However, unlike common practice in fixed networks, in a large network of numerous autonomous and heterogenous nodes, it is inadvisable to assume that there are universally accepted trustworthy authorities who can declare the trustworthiness of other nodes. Even if such trust relationships exist and the identity of a correspondent can be elicited this way, it does not justify that the correspondent can be trusted for cooperation. Furthermore, it is not always possible to build trust based on a priori knowledge since communicating parties may see each other for the first time.

To evaluate the trustworthiness of nodes and to ensure cooperation, trust in a node can be modeled as the reputation of a node combined with the outcome of its direct interactions. Such a reputation-based trust management scheme for ad hoc networks should be examined in this thesis.

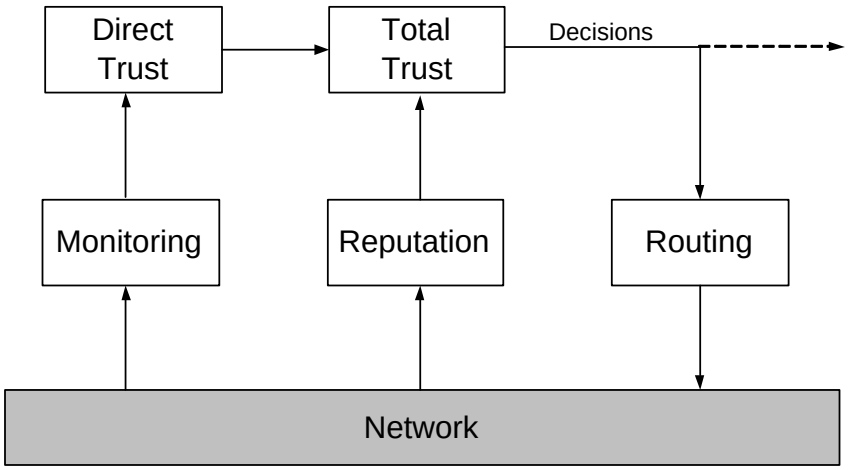


Figure 1: Trust model of a single node

The main goal of this thesis is to develop a protocol to distribute reputation data in mobile ad hoc networks in a decentralized way. The protocol should then be implemented in a network simulator for evaluation purposes.

2 Motivation

In a reputation-based trust management scheme, reputation is used to assess the level of trust a node puts into another node. The reputation itself is derived from direct knowledge on earlier interactions of nodes. Earlier reputation-based trust management systems (e.g. ebay) were based on central mechanisms for storing and distributing reputation data. In this thesis, a new approach must be taken to distribute and store reputation data in a decentralized way.

The trust model of a single node is pictured in Figure 1. *Direct trust* is derived from the *monitoring* module. This module monitors the behavior of neighboring nodes. For example, when a data packet is sent along a path to a destination node, the sender can listen in promiscuous mode on the network to see if the next hop is forwarding the packet correctly. Depending on the outcome, the sender can update his trust towards the corresponding node. The node will update its trust towards the node for forwarding packets. Every node differentiates between different classes of trust. For instance, a node can be trusted for forwarding packets but not for distributing reputation data.

The *reputation* module collects reputation data from the nodes in the network. The reputation data is combined with the direct trust of a single node to establish the total trust of a node. *Decisions* are based on the current *total trust* of a node. For example, nodes can use total trust for routing decisions. As an effect, network traffic is routed through a trusted path which improves network reliability.

To distribute reputation data a protocol must be developed in this thesis. This protocol must allow the direct exchange of reputation values between neighboring nodes. The main challenge is develop a protocol with the least possible overhead caused by exchanging reputation data but where the total trust converges quickly.

3 Assignment

3.1 Objectives

The goal of this thesis is to design a protocol for distribution of reputation data in a complete decentralized way. The protocol should be implemented in a network simulator. The goal is to show the impact

of misbehaving nodes on routing in ad hoc networks with and without the protocol. A source routing protocol (DSR) will be used to evaluate the concept. In a source routing protocol, the sender has explicit control on choosing the nodes along routing paths which is required in this context.

3.2 Tasks

- Learning on the following fields:
  - Dynamic Source Routing (DSR) - a source routing protocol for mobile ad hoc networks [5].
  - Security issues of mobile ad hoc networks [4, 6]
  - Analyze different reputation-based trust management systems for distributed systems [3, 2, 7]
- Make a security analysis of mobile ad hoc networks. Define threats and possible attacks in a mobile ad hoc network. Model the identified threats with an attack tree.
- Design a protocol for distributing reputation data in ad hoc networks.
- Implement the protocol in ns-2 [1].
- Adapt the DSR implementation in ns-2 to choose routing paths based on trust metrics.
- Evaluate the protocol with your implementation in the simulator. Show the impact of the introduced protocol on network performance in the presence of misbehaving nodes.

3.3 Deliverables

- At the end of the second week, a detailed time schedule of the semester thesis must be given and discussed with the advisor.
- At half time of the semester thesis, a short discussion of 15 minutes with the professor and the advisor will take place. The student has to talk about the major aspects of the ongoing work. At this point, the student should already have a preliminary version of the written report, including a table of contents. This preliminary version should be brought along to the short discussion.
- At the end of the semester thesis, a presentation of 20 minutes must be given during the TIK or the communication systems group meeting. It should give an overview as well as the most important details of the work.
- The final report may be written in English or German. It must contain a summary written in both English and German, the assignment and the time schedule. Its structure should include an introduction, an analysis of related work, and a complete documentation of all used software tools. Three copies of the final report must be delivered to TIK.
- Software and configuration scripts developed during the thesis must be delivered to TIK on a CD-ROM.

References

[1] The Network Simulator - ns-2. http://www.isi.edu/nsnam/ns/, 2002.

[2] Karl Aberer and Zoran Despotovic. Managing Trust in a Peer-2-Peer Information System. In *Proceedings of the Tenth International Conference on Information and Knowledge Management*, pages 310–317, Atlanta, Georgia, USA, 2001. ACM Press.

[3] Sonja Buchegger and Jean-Yves Le Boudec. Performance Analysis of the CONFIDANT Protocol (Cooperation of Nodes: Fairness in Dynamic Ad-hoc Networks). In *The Third ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, Lausanne, Switzerland, June 2002.

[4] Jean-Pierre Hubaux, Levente Buttyan, and Srdan Capkun. The Quest for Security in Mobile Ad Hoc Networks. In *Proccedings of the ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHOC 2001)*, Long Beach, CA, USA, 2001.

[5] David B. Johnson, David A. Maltz, Yih-Chun Hu, and Jorjeta G. Jetcheva. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR). IETF Internet Draft, draft-ietf-manet-dsr-07.txt, February 2002.

[6] Sergio Marti, T.J. Giuli, Kevin Lai, and Mary Baker. Mitigating Routing Misbehaviour in Mobile Ad Hoc Networks. In *Proccedings of the Sixth Annual International Conference on Mobile Computing and Networking (MOBICOM)*, August 2000.

[7] Bin Yu and Munindar P. Singh. An Evidential Model of Distributed Reputation Management. In *First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Bologna, Italy, July 2002.

31th March 2003

Prof. B. Plattner

References

[1] The Network Simulator - ns-2. http://www.isi.edu/nsnam/ns/, 2002.

[2] Sans glossary of terms used in security and intrusion detection. http://www.sans.org/resources/glossary.php, 2003.

[3] Karl Aberer and Zoran Despotovic. Managing Trust in a Peer-2-Peer Information System. In *Proceedings of the Tenth International Conference on Information and Knowledge Management*, pages 310–317, Atlanta, Georgia, USA, 2001. ACM Press.

[4] Scott A. Vanstone Alfred J. Menezes, Paul C. van Oorschot. *Handbook of Applied Cryptography*. CRC Press, October 1996.

[5] Sonja Buchegger and Jean-Yves Le Boudec. Nodes bearing grudges: Towards routing security, fairness, and robustness in mobile ad hoc networks. In *The Tenth Euromicro Workshop on Parallel, Distributed and Network-based Processing, Canary Islands, Spain.*, pages 403–410, Lausanne, Switzerland, January 2002. IEEE Computer Society.

[6] Sonja Buchegger and Jean-Yves Le Boudec. Performance Analysis of the CONFIDANT Protocol (Cooperation of Nodes: Fairness in Dynamic Ad-hoc Networks). In *The Third ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, Lausanne, Switzerland, June 2002.

[7] Jean-Pierre Hubaux, Levente Buttyán, and Srdan Capkun. The Quest for Security in Mobile Ad Hoc Networks. In *Proccedings of the ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHOC 2001)*, Long Beach, CA, USA, 2001.

[8] David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks, 1996.

[9] David B. Johnson, David A. Maltz, Yih-Chun Hu, and Jorjeta G. Jetcheva. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR). IETF Internet Draft, draft-ietf-manet-dsr-07.txt, February 2002.

[10] Sergio Marti, T.J. Giuli, Kevin Lai, and Mary Baker. Mitigating Routing Misbehaviour in Mobile Ad Hoc Networks. In *Proccedings of the Sixth Annual International Conference on Mobile Computing and Networking (MOBICOM)*, August 2000.

[11] R. Shirey. RFC 2828, May 2000.

[12] William Stallings. *Network Security Essentials*. Prentice Hall, 2000.

[13] Bin Yu and Munindar P. Singh. An Evidential Model of Distributed Reputation Management. In *First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Bologna, Italy, July 2002.