REPOM: REPUTATION BASED OVERLAY MULTICAST

Bachelorarbeit
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Ulrich Bürgi
2009

Leiter der Arbeit:
Professor Dr. Torsten Braun

Betreuer der Arbeit:
Marc Brogle

Institut für Informatik und angewandte Mathematik

Contents

List of Figures

List of Tables

Acknowledgment

I would like to thank my supervising tutor Marc Brogle for the great support.

Chapter 1

Summary

Multicast is a very efficient way to distribute data. However, Internet-wide IP Multicast is hardly available. To still be able to profit from the benefits of multicast, Application Level Multicast (ALM) systems have been introduced. ALM systems are based on Peer-to-Peer (P2P) network protocols. Unfortunately, some participants of P2P networks are often selfish. They profit from other users but have no intent to provide any or only a limited service themselves. In an ALM network, selfish peers could try to minimize the amount of multicast data they have to forward. Therefore, other peers will not receive some parts of the expected data.

For this Bachelor thesis, we implement and evaluate a reputation system for a simple gossip-based ALM network. The participants of this network are able to rate the service provided by their multicast parents. The reputation system is used to distribute ratings and aggregate them into reputation values. Based on these reputation values, a participant should be able to connect to the possibly best parent available for a specific multicast group. The goal of the reputation system is to minimize packet loss that occurs when selfish peers participate in an ALM, in such a way that the network provides a good service to all of its participants.

To evaluate our approach, we perform some simulations. Therefore, we implement our reputation system REPOM using the OMNeT++ simulation environment. We simulate different scenarios to identify optimal parameters. First, we run tests without any malicious peers and without the reputation management system to see how the size of the peers' neighborhood affects the native P2P overlay multicast network efficiently. Then, we vary the amount of malicious peers to recognize how this impacts the receiving rate. Finally, we simulate the network with the determined parameters and with the reputation management enabled to see whether it is able to compete against the native version without any malicious peers.

Each simulation scenario is run for different network sizes reaching from 100 up to 2000 nodes, thirteen different distance matrices for each network size, and three initial random seeds. For the parameter estimating scenarios, the network sizes increase by 200 nodes per step, resulting in a total of 390 simulations per scenario. For the final comparative simulations, the increase is at 100 nodes per step resulting in 780 simulation runs.

The simulation results show that the presence of malicious peers significantly affects the amount of lost multicast messages in our ALM network. We can also see that the reputation system is capable of reducing the damage done by malicious peers. Having up to 30% of free-loading peers, the average percentage of received multicast messages did not drop below 94%.

Chapter 2

Introduction

2.1 Peer-to-Peer Networks

Peer-to-Peer (P2P) Overlay Networks [1] are self-organized systems without the need of centralized control or hierarchical organization. In a P2P network, all participants share a part of their resources, such as bandwidth, storage, CPU time, etc. [2]. Unlike client-server-based systems, the members of pure P2P networks do not have different roles. Each peer is client and server at the same time and all peers provide the same services to each other. However, there are many P2P systems that do not fulfill this strict definition because of the presence of more privileged nodes (e.g. supernodes in Kazaa) or central entities providing bootstrapping, reputation ratings or similar non-core functionality.

P2P Overlay Networks have some great advantages compared to client-server-based networks, such as robustness and stability. If in a client-server-based system, the server or the connection between client and server is broken, the client is no longer able to receive any service. In a P2P network, the lack or failure of single or multiple peers does not interrupt the service provided by the system. Another advantage is that no peer should get overloaded by requests or traffic. In a highly populated network, a server can reach the point where it is no longer able to process the incoming requests or provide the desired service due to the heavy traffic generated between clients and server. In P2P networks, the load can be balanced among the peers.

There are two different classes of P2P networks: structured and unstructured [1] P2P networks. Structured P2P networks organize their nodes by placing them in a design-specific topology and the data objects are not randomly distributed, but located at specific nodes. To store location information (e.g. a node address) for those data objects, such networks use a distributed hash table (DHT) [3]. The network assigns node IDs to a large identifier space. Out of this space, unique keys are assigned to data objects. This means that the DHT maps a data object represented by its key to a live peer. To access (i.e. get, put or remove) a data object, a peer sends a corresponding request containing the object's key to a neighbor. The request is then always forwarded to the nodes whose IDs are closer to the key, until it reaches the node that matches the key. Examples for structured P2P networks are provided in Section 3.1.

Unstructured P2P networks do not organize their nodes in a particular way. Therefore, the nodes are not aware of any topology, they just know some sparse number of peers. This setup

makes joining and leaving easier than in structured systems. To locate content, peers usually use flooding algorithms to distribute queries. A peer receiving a query sends a list of all its items that match the query to the interested peer. Obviously, such networks are rather ineffective for locating rare content. However, compared to structured networks, looking for popular content produces less overhead. Examples for unstructured P2P networks are provided in Section 3.2.

2.2 Multicast Schemes

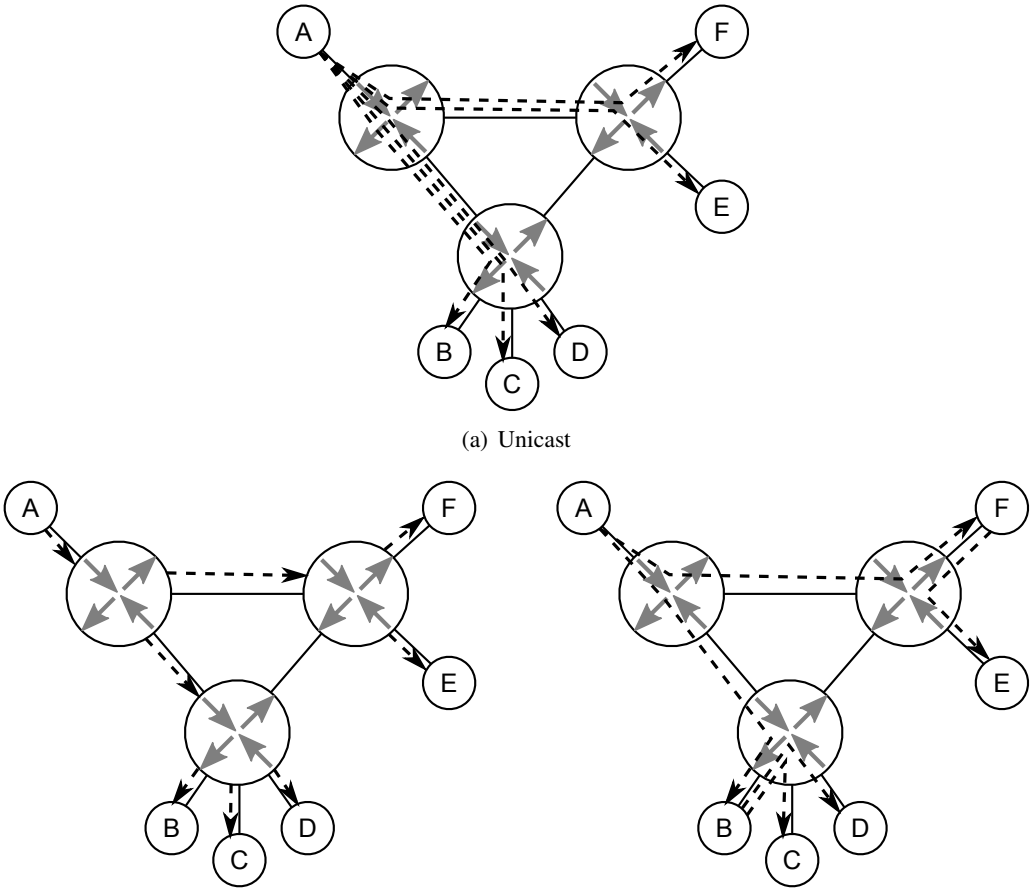
Nowadays, most network services are designed as one-to-one (called unicast) applications. A client requests specific information and a server responds with the desired data. This works well for services which, either are only used by a few users, or which provide user- or request-specific information, such as e-mail or HTTP services. However, if the same data has to be distributed to many users at the same time (e.g. web radio, TV streams), this method is very inefficient. Here, the server has to create and transmit the same data packets for every user, generating a high load on the server's resources, as Fig. 2.1(a) illustrates. Here, node *A* sends the same data to five other users, living in different subnets. As a result, the links between the subnet of *A* and the destination subnets are used multiple times, according to the amount of clients addressed living in the destination subnets.

IP Multicasting [4] is a method to overcome this problem. In IP Multicast enabled networks, a sender can address a packet not only to a single node, but to a group of nodes. The packet is then distributed in such way that each group member receives a copy, but only one copy per link has to be transmitted (compare Fig. 2.1(b)). Therefore, all routers in those networks must be capable of handling IP Multicast traffic, since they are responsible (among others) for group management, packet forwarding and packet duplication. Unfortunately, many Internet service providers (ISP) are not equipped with IP Multicast capable routers or do not enable IP Multicast. Therefore, Internet wide IP Multicast is rather unavailable, especially for home users.

Application Layer Multicasting (ALM) [5] follows the same idea as IP Multicasting, but multicasting is implemented as an application service instead of a network service. The participants of an ALM network build a virtual overlay network and act as end nodes and routers at the same time. The routing strategy in this overlay network is the same as in native IP Multicast: each packet is delivered to all nodes, but every link is used only once. However, since the overlay network is unaware of the real network structure, it is likely that routing is not as efficient as in IP Multicast. It is also possible that some packets using unique paths in the overlay network, use the same link in the physical network as shown in Fig. 2.1(c). But because ALM networks do not have need for special network hardware, they can easily be implemented for any kind of unicast-capable network, such as the Internet. The design of ALM protocols can be adapted to special needs of different applications.

2.3 Threats and Countermeasures

Despite the advantages we already discussed, the design of P2P networks involves some risks. P2P systems are based on large anonymous communities. Neither do the users know who the



(a) Unicast

(b) IP Multicast

(c) Application Layer Multicast

Figure 2.1: Routing a message from node *A* to all other nodes using multicasting

other participants are nor what intentions they have. All users have to rely on the others' cooperation. However, from the perspective of a single user cooperation means cost. Rational users joining the network to benefit from others, might not want to altruistically provide data and bandwidth, and therefore do not cooperate. Such users are called free-riders and can be a major problem for P2P networks [6]. If all users try to minimize their cost, P2P systems would not work at all or with very bad performance. Besides free-riders who are passively threatening the network, there is also the type of users, which behave actively malicious. Those users intent to harm the network, for example, by distributing corrupted or malicious data (e.g. viruses) or by responding with faked service messages. Obviously, these problems can also occur in ALM networks, with the difference that less malicious users can cause more damage. A free-rider in a multicast network, not only affects its known peers but all other peers, which live in the free-rider's subtree.

Due to the distributed design of P2P networks and the lack of knowledge about the participating users, it is quite difficult to identify and eliminate malicious users. However, there exist

several strategies, which target these problems. These strategies can be categorized as *incentive driven*, *reputation-based* and *trust-based* approaches. Of course, these categories are not specially designed for P2P networks. They can be adapted to different kinds of communities and can be described as follows:

Incentive-driven systems work completely passive. For certain actions, the user receives a bonus to encourage it to maintain this behavior. Other actions will cause some costs to make them unattractive if performed repeatedly. An example for costs is a time consuming process to acquire a unique user ID, which is required to join the community. This is for example required at various Internet services, such as forums, mailing lists, newsgroups, etc. For more details about P2P related approaches to incentive-driven systems, see Section 3.3.

Reputation-based systems are widely used in different scopes. The main purpose is to encourage a community by giving its participants the possibility to inform each other about experienced behavior. Based on these reported experiences, a participant is able to estimate whether an interaction with another participant will pay off or not. Well known examples for such communities are online trading platforms like Amazon or eBay. Here, the customers can rate the sellers (and vice versa) to inform others about the condition of the purchased product, shipping time, charges, etc. The results are made public as advice for other users. Similar evaluation strategies also work for special kinds of communities, such as P2P networks. Here, the peers rate each other according to the service received, such as sending rate. For further details about reputation-based systems, see Section 3.4.

Trust-based systems try to estimate the correctness of the information a user is providing. They are the counterparts to actively malicious users, which want to harm the network. Such malicious users could give negative ratings for well behaving users, and therefore lowering their reputation. Many trust-based systems rely or extend reputation-based systems. For examples of such networks, see Section 3.5.

2.4 Thesis Outline

This thesis is structured as follows: In Section 3, we present related work, such as different implementations of structured and unstructured P2P networks. We also introduce common approaches for incentive-driven P2P communities as well as specific implementations of protocols relying on reputation management and trust management. Section 4 describes the general design of our gossip-based P2P network, the multicast overlay and the reputation system. In Section 5, specific implementation issues are discussed. The simulation results are evaluated in Section 6. Finally, Section 7 concludes the thesis with a summary of the major results and gives an outlook to future work.

Chapter 3

Related Work

3.1 Structured P2P Networks

3.1.1 CAN

CAN [7] (acronym for Content Addressable Network) uses a multidimensional Cartesian coordinate space as an identifier space. This space is dynamically split, so each peer gets its own area. For routing a message, as example a get request for the data object represented by the key k, the target destination of the message is set as the point in the coordinate space, which represents k. Since each peer knows in which areas its neighbors live, it can send the message to the peer responsible for the area the destination point belongs to, if this peer is a direct neighbor. Otherwise the message is forwarded to the neighbor with the shortest distance to the destination. In the example in Fig. 3.1, peer A wants to send a message to the owner of key k. The neighbors of A are peers B and C, which both do not own k. But because B's area is more close to k than C's, A sends the message to B. B then forwards the message to E, which again is the neighbor closest to k. Finally, E knows the owner of k and sends the message to this peer.

The first step a joining node has to take, is to get its own area in the coordinate space. A bootstrap peer provides the joining node with an IP address of a random peer, via which the joining node can send a join request to a random point p in the coordinate space. The peer currently living in the same area as p splits its area and assigns one half of its area to the joining node. All objects now belonging to the area of the new peer are then transferred, as well as the list of peers that build its neighborhood.

When a peer leaves the network, a former neighbor takes over its area. All affected peers will then notify their neighbors about the change, so routing messages to this area will still work as expected.

3.1.2 Chord

Chord [8] uses a hash function to generate node IDs by hashing the node's IP addresses. The same hash function is used to create identifiers for the data, by hashing the data to keys. Both node IDs and data IDs now live in the same identifier space called Chord ring. Each ID n on this ring has a successor, which is defined as the first peer with an ID greater than or equal to n. If

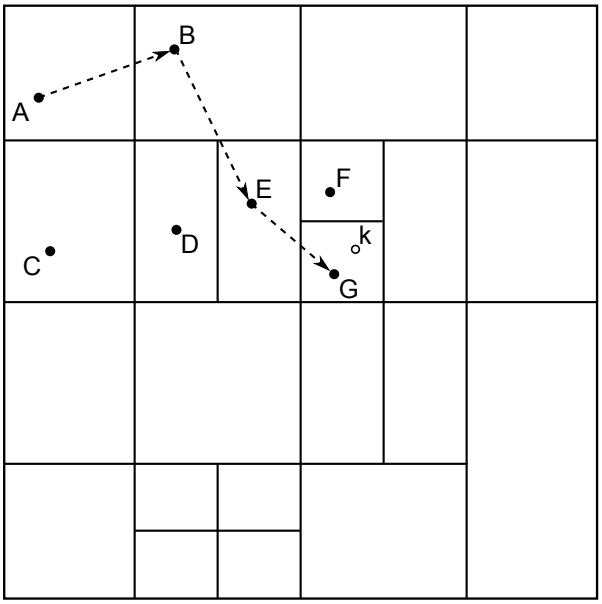


Figure 3.1: Routing a message from node *A* to the owner of key *k* in a 2-dimensional CAN

a peer with a such an ID does not exist, the peer with the lowest ID close to n will become the successor. The data IDs are assigned to their corresponding successors. As example, the data with ID 40 belongs to the first peer with an ID equal to or greater than 40. To be able to route messages through the network, each peer has to know its own successor in the Chord ring. For more efficient routing, each peer also maintains a set of further away peers, called finger table. The ith finger of the peer with ID n is the successor of the key identified by $n+2^{i-1}$. A Message with destination ID *k* can now be routed by forwarding it either to the finger with the greatest ID that is lower than *k*, or, if this finger does not exist, to the peer's successor. Obviously, only the receiver can verify whether itself is the target destination or not. Because of this and the fact that no node knows any of its predecessors, a message can only be forwarded to a peer with a greater ID than the key, if this peer is the successor of the current routing peer. The example in Fig. 3.2 shows the path a message has to take from peer *A* to the owner of ID *k*. According to the finger table of *A*, the peer would already know the destination. But *A* cannot know whether *E* is the owner of *k*, because unknown peers with the IDs 41 or 42 could exist. Therefore, *A* sends the message to *C*, which is the finger with the highest ID lower than *k*'s ID. Due to the same reason, *C* has to forward the message to *D*. Finally, the successor of this peer is *E* and the message can be sent to its destination.

Joining and leaving is quite simple. When a new peer joins the network, it acquires the keys corresponding to its ID from its successor. When a peer leaves, its keys are reacquired by its successor. To ensure that the peers always know their successors, Chord uses a stabilization protocol, which periodically checks the correctness of the successors and fingers. Since it is not possible for a peer with a broken successor entry to find its real successor, peers maintain a list of backup peers consisting of the successors' successor, its successor, etc. If a peer's successor does not respond, it can contact the first backup peer out of this list.

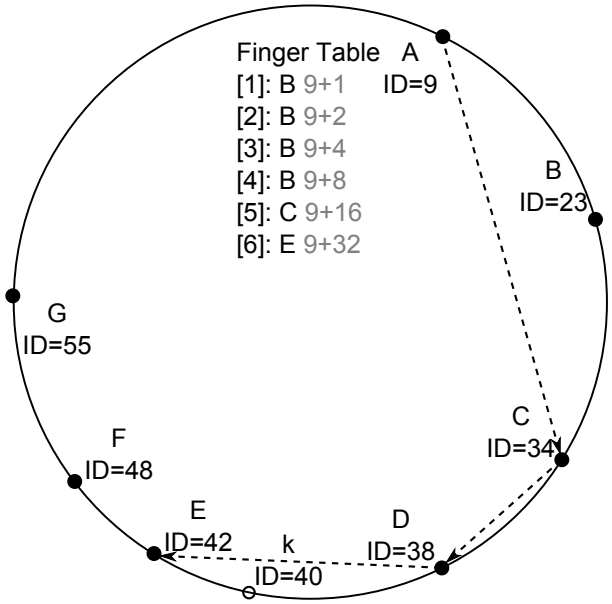


Figure 3.2: Routing a message from node *A* to the owner of key *k* in Chord

3.2 Unstructured P2P Networks

3.2.1 BitTorrent

BitTorrent [9, 10] is a centralized P2P system. One network is built for the distribution of a single file or a single set of files. To be able to join the network, the user needs a torrent file, which describes the distributed file. The torrent file contains information about the shared file, such as name, size, hash values, the URL to the tracker, etc. The tracker is a central entity, which keeps track of all peers participating in the file distribution. At request, it sends a list with random peers, to which the requester can establish a connection. Connected peers exchange lists, which contain hash values of the data parts they already have downloaded. This way, the peers know from which peer they can request further data.

3.2.2 Gnutella

Gnutella [11] is a decentralized P2P system. Messages sent in Gnutella are distributed either via broadcasting or back-propagation. Broadcasting means that a peer forwards the message to all peers to which it has an open TCP connection. A back-propagated message from peer *A* to peer *B* takes the same path that a broadcasted message takes from *B* to *A*, but reversed. To access data, a peer broadcasts a query message, which is forwarded using a limited number of hops. Peers having data matching the query will respond with a back-propagated message, containing necessary download information. The peer can then select one of the responding peers and directly download the requested data.

To be able to join a Gnutella network, a new peer needs knowledge of at least one peer in

the network. It opens a connection to this peer and initiates a broadcast of a ping message. Peers receiving this message respond with a pong message containing information, which is needed by the joining node to establish a connection. To stay updated about changes in their vicinity, peers periodically broadcast ping messages, even after joining.

3.3 Incentive-driven P2P Approaches

3.3.1 General Techniques

Feldman and Chuang [12] propose some theoretical strategies to give peers in a P2P network incentives for cooperation. The easiest way to do this is to use inherent generosity. The idea is that some people in P2P networks just share their resources without the need of any incentive. As long as a certain threshold of non-selfish peers is maintained, the network stays alive. Another strategy is to use monetary payment. Each peer should simply pay to its service provider for the service and the resources used. However, there are some difficulties with such micropayment systems. Sometimes, it is not possible to estimate the amount of resources used for providing a certain service. For instance, a service could require some computation time of which the client is not aware of. A peer could therefore not be sure if it really is paying the right price. Then, there is the problem of the payment transaction itself. The network has to provide a trusted accounting provider, which can make sure that the payments are accounted properly. A third strategy covers reciprocity-based schemes, such as tit-for-tat used in BitTorrent. Here, the peers keep a record of experienced behavior of other peers and treat them accordingly. For example, a peer could preferably choose to provide service to another peer, which previously provided a service to itself, rather than to someone that refused to provide service before.

Some further strategies to minimize the amount of profiting selfish nodes are provided by [6], including the following:

Transaction History Peers should maintain a list of all other peers with which they already interacted. This is used as a basis to be able to respond adequately to a request from an already known peer (e.g. to perform tit-for-tat in BitTorrent).

Shared History The transaction history, or parts of it, should be shared with other peers in the network. The peers gain knowledge about each other, even if they are (currently) not directly connected which leads to a level of cooperation.

Reputation System Feldman et al. [6] describe that a reputation system is used to be able to estimate the correctness of the shared history. Peers could act together and fake some shared information for their mutual benefit. This kind of reputation system could also be considered as some sort of trust management.

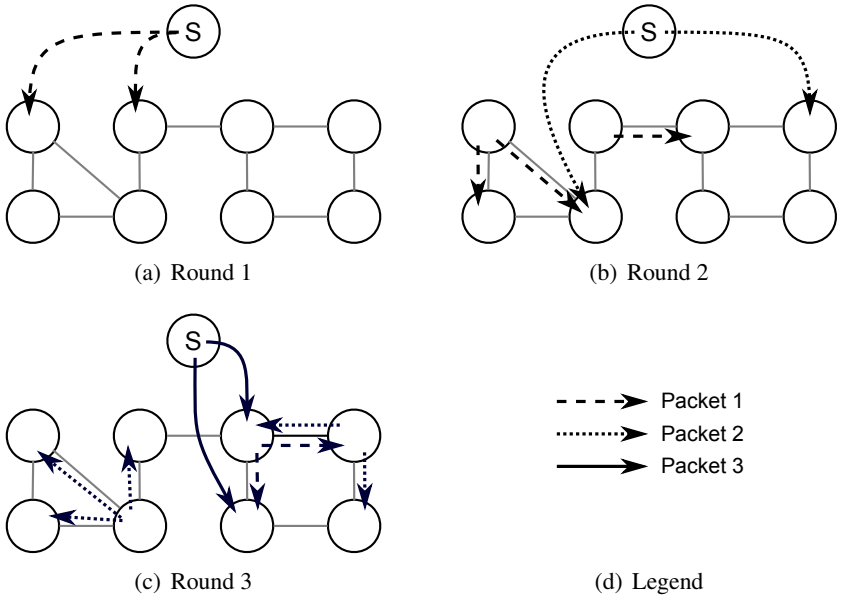
Adaptive Stranger Policy This targets the problem of zero-cost identities. To avoid that non-cooperative peers loose their penalties by generating a new identity, peers should treat unknown peers according to the behavior of previously unknown peers.

Short-term History Well behaving peers can build up reputation and could later on profit, while no longer cooperating with other peers. Therefore, a rating for a peer should not last forever.

3.3.2 EquiCast

EquiCast [13] is a P2P multicast protocol that targets to eliminate free-riders. The incentive for cooperation is that selfish peers will be excluded from the network. EquiCast uses a distribution server S that builds an overlay so that each peer has k neighbor peers. The data distribution is done in several rounds. In each round, S sends a fix amount of packets to k random peers and all peers notify their neighbors about the packets they received in the previous round. The peers can then request the packets needed (i.e. all packets they did not already receive) from their neighbors. As an example, Fig. 3.3 shows the first three rounds of this strategy without the notification and request messages. In round 1, S sends the fist packet to two random nodes. In round 2, the two nodes notify their neighbors about packet 1 and forward it at request. Meanwhile, S sends the second packet to two random nodes. In the same manner, in round 3, the nodes offer to their neighbors packets 1 and 2, which they received during the previous round either from one of their neighbors, or from S. Now, S initiates the distribution of the third packet.

Since in each round S randomly chooses the peers to which it sends the data, packets not transmitted in the same round traverse a different multicast tree. This means that on average during each round the peers receive the same amount of packets and therefore can offer an equal amount of packets to their neighbors.



(a) Round 1 (b) Round 2

(c) Round 3 (d) Legend

Figure 3.3: Data distributing in EquiCast

For the identification of free-riders, EquiCast peers monitor the sending balance of each

neighbor. This balance is the difference between the actual amount of packets sent by the neighbor and the expected sending rate. If a peer recognizes that one of its neighbors' balance is lower than a certain threshold, it terminates the connection to this neighbor. As a second incentive, a node with a negative balance has to send a *fine packet* to its neighbors each round. Neither does this packet contain any specific data, nor does it influence the balance. It is simply meant as a penalty for having a negative balance. If a node does not receive a *fine packet* from a neighbor with negative balance, it terminates its connection to this peer as well.

3.4 Reputation Systems

Many reputation systems have been developed and evaluated for problems in different kinds of networks. For example, in multihop wireless networks, routing packets from source to destination could be difficult if there are nodes in between that do not forward packets as intended. Milan et al. [14] propose a strategy where every node listens to its neighbors whether it is properly relaying packets or not. The result influences the current reputation of this node, and a node with bad reputation will be punished by not having delivered all requested packets to it.

We are not aware of any multicast protocol that uses a reputation system. However, there are protocols, like XRep, which provide reputation-based services for other types of P2P networks.

The XRep Protocol is an extension to Gnutella-based P2P networks proposed in [15]. The idea is that each peer maintains two experience repositories, which either map resources or servants to binary values. After a download, the node creates a new entry in the resource repository, which classifies the just downloaded data either as good, or as bad. This is done for the rating of the servant. However, the servant repository holds only one entry per servant, storing the amount of good and bad downloads provided. The difference between those two amounts are used to either classify the servant as good or as bad. The classification a peer made for an entity is called *vote* and is exchanged between peers.

The reputation-based resource selection and the exchange of the votes are implemented as additional steps in the standard querying process. The first step is unmodified: the peer broadcasts query messages and in return receives query responses from peers, which have the desired data. Now, the peer selects one of the responding peers, from which it would most likely want to download the data. It then broadcasts poll messages containing the IDs of the resource and a possible servant. A poll message is responded with the receivers vote corresponding to the two IDs. With all responses, the peer can now build the reputation values for both resource and servant, and decide whether it wants to download the data. If it does so, it has to decide whether the servant appears reliable enough. It can then either initiate the download or start a new poll for another candidate servant.

3.5 Trust Management

Due to the fact that most trust management systems rely on reputation management systems, we are not aware of any trust management designed for overlay multicast protocols. Therefore, in the following, we introduce two more generic approaches, which claim to provide some sort

of trust management for P2P systems, as well as a trust management system for mobile ad-hoc networks.

3.5.1 iComplex

Khambatti et al. [16] propose a solution where all peers aggregate and publish their own trust values in such a way that it can be verified by other users. It is based on the idea that each peer is interested in some particular topics and that peers, which are interested in the same topic (implicitly) form communities. It uses the notation *peer link* to describe an active relationship between peers that share a common interest. Links are assumed to be bidirectional and can always be terminated by one party.

The first links a peer establishes in a community are performed by either connecting to a bootstrap node or to an already known peer within the community. To determine the membership of a peer in a community and its rank, so called *link weights* are calculated. For each topic a peer is interested in, the link weight represents the number of links that reach other peers interested in the same topic, after at least one indirection. In other words, it is the sum of peers living in the 1-hop and 2-hop neighborhood that share at least one interest with the peer.

Trust values are now generated according to the link weights of a peer. This means that a peer has different trust values for its different interests, and the more 1-hop and 2-hop neighbors with common interest a peer has, the higher its trust values are. To propagate claimed interests and therefore aggregate link weights and form communities, the peers distribute special messages within their 2-hop neighborhood. These messages contain, amongst the sender's and receiver's ID, the list of interested topics and a digital signature of the message, created by the sender. Each message received is stored by the receiver on its publicly available blackboard. Since these messages are used to build the link weights, any peer can visit another peer's blackboard and recalculated the claimed link weights for verification. Furthermore, a peer can verify the authenticity of a publish message by validating its signature. Simulations showed that a peer only has to verify 10-20% of the messages to uncover the presence of fake messages with a high probability.

The different trust values can also be aggregated into one single value, the so called *iComplex*. How the values are aggregated does not matter, as long as all peers use the same aggregation function. For example, the iComplex value could be the sum of all link weights. When the iComplex value is placed on the peer's blacklist, other peers can easily identify if the peer is highly involved in different communities (high iComplex), and therefore is highly trusted or not.

The authors also propose a revocation mechanism to terminate links, because the behavior of peers could change over time. Therefore, a revocation list is added to the peer's blackboard. If a neighbor is no longer trusted, it is added to this revocation list. The message verification mechanism is extended in such a way that a peer not only verifies the signatures of some messages of peer A, but also if the message creator has revoked A. In this case, that particular message should not have been taken in account to calculate the iComplex value. It should also be mentioned, that if peer A puts peer B into its revocation list, it may no longer use the messages sent by peer B to build its iComlex, since links are bidirectional.

3.5.2 EigenTrust

The EigenTrust Algorithm for Reputation Management in P2P Networks [17] introduces a method that allows to compute global trust values for all peers of a P2P network. According to trust values, peers can choose which peer's service they want to use.

First, peers have to rate other peers. It should be mentioned, that the algorithm assumes the presence of a set P of pre-trusted peers, which could, for example hold the first few peers that joined the network. Every peer i is then assigned a pre-trust value p_i based on whether it is part of this set, where $p_i = \frac{1}{|P|}$ if $i \in P$ and $p_i = 0$ otherwise. For the actual rating, every transaction peer i has with peer j (e.g. peer i downloads a file from peer j) is rated by peer i and stored as $tr_{i,j}$. The sum of all transaction ratings of peer j performed by peer i builds this peer's *local trust value* $s_{i,j} = \sum tr_{i,j}$. In order to be able to distribute this value, a normalization has to be performed which creates the normalized local trust value $c_{i,j} = \frac{\max(s_{i,j},0)}{\sum_k \max(s_{i,k},0)}$. If the denominator should be zero (i.e. the peer never had a positive transaction) $c_{i,j}$ is set to p_j.

Now, to aggregate the normalized trust values, a peer could simply ask its neighbors about their opinion about other peers. The weighted sum of the received values builds $t_{i,j} = \sum_k c_{i,k}c_{k,j}$, which represents the trust peer i and its neighbors have in peer j. Another way to store this information is using matrix calculus. If values $c_{i,j}$ are stored in matrix C at their corresponding positions and store the peer's local normalized trust values in vector c_i, the vector $t_i = C^T c_i$ can be computed. This vector holds all aggregated trust values of peer i.

At this time, C only contains a few entries defined by the owning peer and its neighbors. However, peer i could also ask its 2-hop neighborhood for their normalized trust values. This would result in $t_i = (C^T)^2 c_i$. Of course, the peer could also aggregate the values up to its n-hop neighborhood, and therefore generate $t_i = (C^T)^n c_i$. For large n, the trust vector t_i is the same value for every peer: it converges to the eigenvector of C. In other words, t is a global trust vector representing how much the network trusts its individual participants and can iteratively be computed as illustrated using the following pseudocode:

```
n = |Peers|
t^0 = 1/n
while δ > ε {
        t^{k+1} = C^T t^k
        δ = ||t^{k+1} − t^k||
}
```

Obviously, it is very inefficient to let all peers compute t by themselves. This could also be done by a central entity or by the whole network in a distributed manner. The easier way is to use a central entity. All peers report their normalized local trust values to this server, which can then build C, and therefore compute the global trust vector. The peers can then query the server about the trust value of a certain peer.

To let all peers cooperatively compute the global trust vector, the first proposition is that each peer computes and stores is own global trust value. In other words, each peer iteratively computes one component of the global trust vector using $t_i^{k+1} = \sum_j c_{j,i} t_j^k$. Since a peer usually performs transactions to a limited set of other peers, most of the $c_{j,i}$ values are zero. For the others, the peer has to know the corresponding t_j^k. To reduce message overhead, instead of

requesting these vectors from the peers, a peer j automatically distributes its t_j^k to all peers it has provided service to. It then only has to wait for all incoming vector components in order to be able to perform the next iteration step.

However, it generally is not a good idea to let peers compute their own trust values, because malicious peers could easily fake them. Therefore, the authors provide a more secure method. A DHT is used to map peers to virtual coordinates. Each peer that covers the coordinate of another peer acts as score manager of that peer, which computes and stores its trust value. To prevent the distribution of false trust values, a peer is mapped to multiple DHT coordinates, and therefore has more than one score manager. It is now easy to detect if a score manager reports fake trust values.

Although our approach has some similarities to EigenTrust, there are substantial differences. For instance, we do not have to rely on the presence of pre-trusted peers. Furthermore, the peers of EigenTrust have a greater knowledge of the network, whereas our peers only know their 2-hop neighborhood.

3.5.3 Confidant

In mobile ad-hoc networks, malicious nodes could achieve certain advantages, such as reduced power consumption, by not forwarding packets. Confidant [18] is a protocol for mobile ad-hoc networks that targets to isolate such misbehaving nodes. Confidant is divided into the four components: *monitor*, *reputation system*, *path manager* and *trust manager*.

The monitor component is used to verify the behavior of neighboring nodes. This is done by listening to the transmission of a neighbor to which a packet has been forwarded. If for instance, no transmission takes place, the node can assume that its neighbor did not route the packet. The monitor also observes the communication taking place in its neighborhood, and is therefore able to detect uncooperative activity even if the node itself is not affected. Whenever a deviation of the routing protocol is detected, it is reported to the reputation system.

The reputation system manages rating for known nodes. Whenever a misbehaving node is reported, the reputation system checks whether it has to lower the node's rating. A node's rating is only adapted if a malicious behavior has been reported for a certain number of times. This ensures that coincidences or temporary failures do not lead to bad ratings. If a node's rating drops below a predefined threshold, the path manager is notified.

The path manager caches all routes that have been established from or through this node. It also checks whether incoming route requests contain a malicious node or are originated by one. In that case, the route request is ignored and the source of the request is notified. When the reputation system reports that a node became malicious, the path manager removes all cached routes containing this node and notifies the trust management component.

The purpose of the trust manager is to distribute *alarm* messages and evaluate received ones. Whenever this component is notified by the path manager that a node now has to be considered to be malicious, it sends an alarm message to the source of every route the malicious node was part of. Among other things, this message contains the ID of the malicious node, the type of the malicious behavior and the number of occurrences. When the message is received, the trust manager first checks whether the source of the alarm is trustworthy or not. This is achieved by looking up the source in a list of pre-trusted friends or by considering how much other nodes

trust the source. If the source is at least partially trusted, the alarm message is stored. When enough alarm messages for a certain node arrived, the node is reported to the reputation system.

Chapter 4

Design

To implement a working reputation system for an Overlay Multicast network, three requirements are needed: an ALM protocol, non-cooperative nodes in the network and a reputation model. Since we want the reputation system to be independent of the ALM protocol used, any protocol will do. Here we use a simple gossip-based approach. This is easier and more controllable than adapting an existing P2P protocol to our needs. The reputation system can then be used as a dedicated service by any node to distribute and acquire ratings for other participants. Both, gossip protocol and reputation system, are fully decentralized and no peer has knowledge about the complete network. The non-cooperative behavior of malicious nodes is achieved by blocking some of the outgoing messages to other nodes.

4.1 Gossip-based ALM Protocol

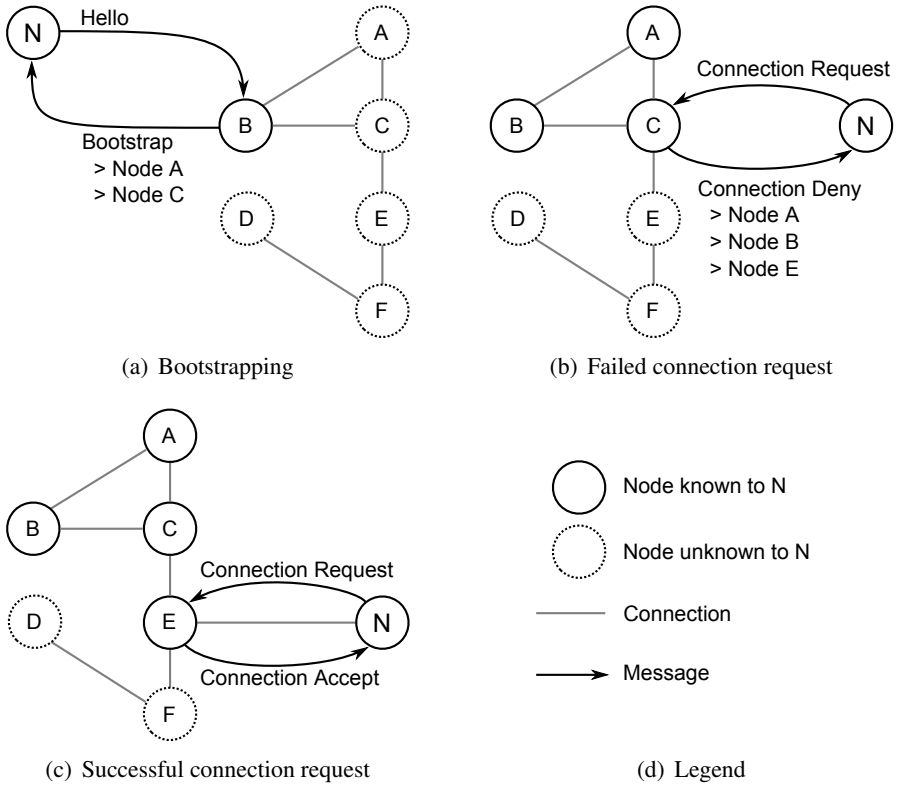
The design of the protocol can be split into the two parts *P2P Network* and *Multicast Overlay*. The P2P network part provides rules needed to link the peers and to establish the actual network, like joining and communication procedures. The multicast overlay part is responsible for the creation of a multicast tree and for the efficient distribution of multicast packets to all peers.

4.1.1 P2P Network

The peers of our P2P network maintain a set of at most k_{max} other live peers. This set is called the *neighborhood* of a peer. Peers and their neighbors are connected bidirectionally, meaning, if peer *A* is a neighbor of peer *B*, *B* is also a neighbor of *A*. As long as a peer stays in the network, it tries to have at least k_{min} neighbors. Therefore it actively sends connection requests to other known peers to join its neighborhood, until it has k_{min} connections. This limit exists to ensure that new peers joining the network will find other peers with non-full neighborhood sets, to which they can connect to. A peer receiving a connection request always answers with a response message, either accepting the request, if its neighborhood size has not exceeded k_{max}, or denying it otherwise. The response is needed to verify that the connections are established bidirectionally.

At a regular interval, each peer sends a *Gossip Message* to all of its neighbors. This message contains a copy of the sender's neighborhood set. This way, peers gain knowledge about other

peers (their 2-hop neighborhood), to which they can send connection requests. To be able to join the network, the joining node has to know at least one live peer in the network, to which it sends a *Hello Message*. As a reply, it receives a special Gossip Message. The nodes listed in this message are the first nodes, to which the joining node can send connection requests. Since it is possible that the neighborhood sets of all of these nodes are already full, the deny messages for connection requests are designed as Gossip Messages. Therefore, every peer will find other peers to which it could connect to.



(a) Bootstrapping

(b) Failed connection request

(c) Successful connection request

(d) Legend

Figure 4.1: Joining procedure

An example of such a joining procedure is illustrated in Fig. 4.1. We assume that a peer may only be connected to three other peers. When the new node *N* wants to join the network, it sends a *Hello Message* to the known node *B*, as shown in Fig. 4.1(a). *B* responds with a *Bootstrap Message* containing a list of *B*'s neighbors. Node *N* now knows nodes *A, B* and *C*. It randomly picks one of those to send a *Connection Request Message* to. If this message is sent to *C*, as shown in Fig. 4.1(b), the connection request is denied because *C* already maintains three connections. However, the joining node gets knowledge about an additional node *E* included in *C*'s response message. Node *B* sends an other *Connection Request Message*, but this time to *E* as shown in Fig. 4.1(c). This will succeed, since *E* is only connected to two other nodes. Therefore, *E* responds with a *Connection Accept Message*. The two nodes are now connected and node *B* successfully joined the network.

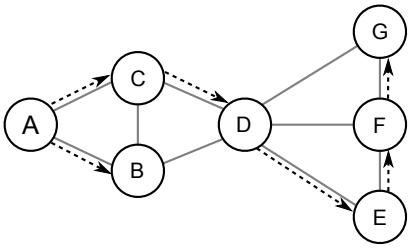


Figure 4.2: Example topology

The leaving process is just as simple. The leaving node only has to send a leave message to all of its neighbors to ensure that it no longer appears in their neighborhood sets. The peers do not have to respond to a leave message.

4.1.2 Multicast Overlay Network

The first peer of the network becomes the first multicast sender or the *root* of the core-based multicast tree. The network represents one multicast group and the root receives and forwards the data addressed for this group. The root does not have to be the same peer all the time: if the current root wants to leave the network, it can perform a root hand-over with one of its neighbors.

In order to receive multicast data, the peers have to subscribe to a multicast forwarder. This is either the root or another subscribed peer, but it has to be a neighbor, since these are the only peers a peer is linked with. To know to which neighbor a peer can subscribe, a peer receiving multicast data sends announcement messages to all of its neighbors. The peers can then pick one of their announcing neighbors and subscribe to it. A peer accepting a subscription is called the *parent* of the subscriber and its job is to forward all incoming multicast data to its subscribers. If multiple neighbors send announcement messages, the peers preferably select the one with the lowest round trip time (RTT) to the root. Later, other criteria, like the neighbors reputation, will also be taken into account for the selection of a parent.

To keep the design simple, no hand-over is performed when a parent (except the root, as already mentioned) leaves the network. Since a leaving peer has to notify all its neighbors, also its parent and its subscribers are notified, because they belong to the peer's neighborhood. A parent receiving a leave message from one of its subscribers simply stops forwarding the multicast data to this peer. When a subscriber receives a leave message of its parent, it has to look for a new parent in its neighborhood. If the peer has no longer a parent and is unable to find a new one, it notifies its subscribers, which can then unsubscribe and look for a new parent as well.

A problem with the selection of a new parent is that a peer must not subscribe to a neighbor, which is a member of the peer's subtree, since this would result in a loop. As an example, see Fig. 4.2. If peer *C* leaves the network, *D* can only subscribe to *B*, and it should notice that it must not subscribe to neighbors *E*, *F* or *G*.

To overcome this, the announcement messages contain the path to the root, which is a list of predecessors of the announcing node. Each node adds the IDs of the predecessors of their parents

(received by their parents' announcement messages) to the announcement messages, combined with the ID of their parents. The path to root of node G in Fig. 4.2 would be $\{A,C,D,E,F\}$. If node D now receives an announcement message from G, it could verify, that itself is part of G's predecessors and therefore not subscribe to this node. To preserve anonymity, we could also let the nodes add their hashed IDs or any other unique token to the list, since it is only used for self-identification.

4.2 Malicious Nodes

The goal of our reputation-based overlay multicast (REPOM) network is to provide a good service to nodes in a non-cooperative environment. Therefore, we do not focus on nodes, which actively damage the network by distributing fake or corrupt messages. Our malicious nodes are selfish and do not want to participate by not forwarding some or all multicast data. As illustrated in Fig. 4.3, we distinguish between different levels of severity, and therefore we have nodes, which

- behave as intended and are not malicious (Fig. 4.3(a)),
- occasionally do not forward a multicast message to all subscribers (Fig. 4.3(b)),
- occasionally do not forward a multicast message to some specific subscriber (Fig. 4.3(c)),
- do not forward any multicast messages to some specific subscribers (Fig. 4.3(d)),
- do not forward any multicast messages to all subscriber (Fig. 4.3(e)).

We consider the multicast root to always be non-malicious. First, the initial peer of the P2P network, which becomes the root, should have a natural interest that the network stays (or actually becomes) alive. Second, a selfish peer would always reject a root-handover request, and therefore never become root.

4.3 Reputation System

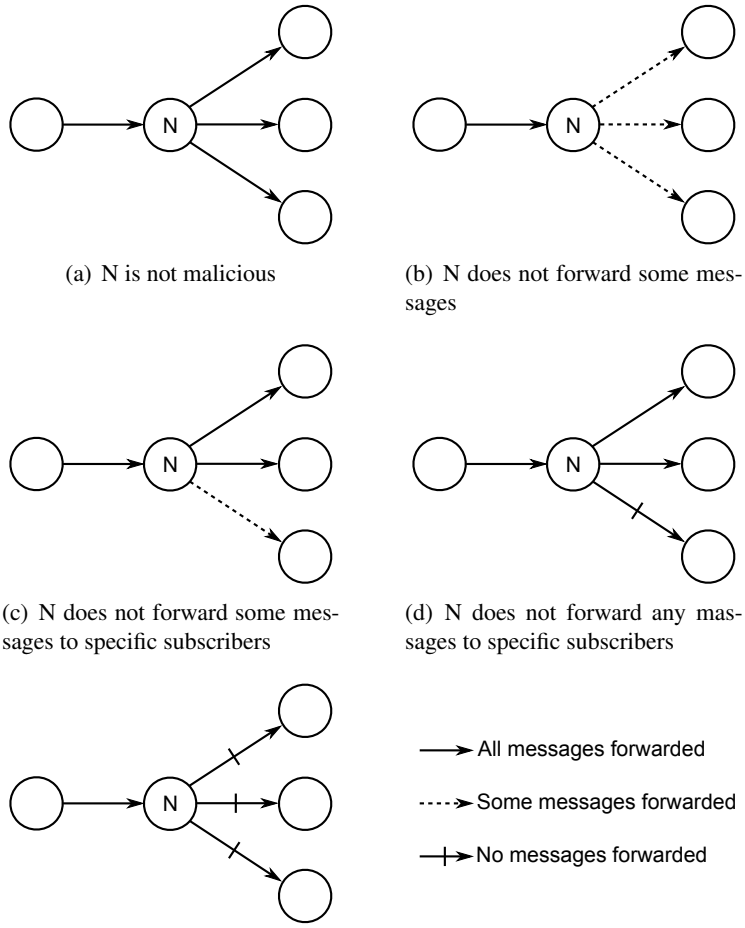
The reputation system is designed as a dedicated service peers can use independent of the multicasting procedures. Each peer can rate its current parent according to the estimated level of cooperation. The rating is represented by one of the following discrete values on the scale between -1 and 1, with the following meanings:

1.0 Parent forwards all messages

0.5 Parent occasionally drops some messages

-0.5 Parent drops most of the messages

-1.0 Parent never forwards any message



(a) N is not malicious

(b) N does not forward some messages

(c) N does not forward some messages to specific subscribers

(d) N does not forward any massages to specific subscribers

(e) N does not forward any massages

(f) Legend

Figure 4.3: Behavior types

The value zero is reserved as a representative for an unknown level of cooperation, and therefore can not be used as a rating.

At a regular interval, peers rate their parents as follows: We assume, that multicast messages are consecutively numbered. Therefore, when a peer receives a multicast message, it knows how many messages it has not received since the previous one. To support this, peers put the number of the last received multicast message in the announcement messages. Peers, which never receive any multicast message can use the announcement messages received from their neighbors, to check whether their parents behave properly. They can check if the message number in the announcement message equals the number of the last received multicast message. If the amount of received multicast messages is above a certain threshold, the parents behavior is considered to be good. Good behavior raises the current rating for the parent to the next higher stage, bad behavior provokes the contrary. As an example, if a peer's current parent rating is 0.5 and the

peer received all multicast messages since the last rating, the parent's rating will increase to 1.0.

After each rating step, peers send *Reputation Messages* containing the rating values and the parent's ID to their neighbors. These reputation messages are forwarded to the receiver's direct neighbors, so they reach the sender's 2-hop neighborhood, which covers the amount of nodes a peer usually has knowledge about due to the P2P gossip messages. Each incoming rating is stored together with the corresponding node ID, the sender's ID and the arrival time. If an older rating for an ID from the same sender already exists, it is overwritten by the new one. All received ratings, weighted by their arrival time, are combined to a reputation value (see Section 5.4.3 for further details). The weighting is used, because younger values should count more than older ones and values, which exceeded a certain age should be completely ignored. This is due to the same reason mentioned in Section 3.3 (short-term history). The reputation value and the peer's own parent rating are combined to an overall reputation value, which can now be used to influence the parent selection process. This can be either by encouraging a change to a potential better parent, or by preventing a subscription to a bad peer.

Chapter 5

Implementation

REPOM has been implemented using the OMNeT++ simulation environment [19] and uses the Overlay Multicast Quality of Service (OM-QoS) framework [20]. Among other things, the framework provides an overlay for the P2P network, which routes the messages sent from one peer to another. It uses a distance matrix containing the required times the packets need to travel from one peer to any other. According to the chosen matrix, the overlay delays the messages sent in the P2P network, creating delay times as in real networks.

5.1 OMNeT++

OMNeT++ is a discrete event simulation environment, which we used for the implementation and testing of REPOM. Its main purpose is the simulation of communication networks, but because of a very modular design, it is also suitable for other scenarios, such as protocol modeling, simulation of hardware architectures, etc [21]. It is written in C++ and works on different platforms, including Linux, Mac and Windows.

Besides documentation, utilities, etc., OMNeT++ provides a compiler for *NED*, the simulation kernel library, a GUI-based simulation environment (*Tkenv*) and a command-line-based simulation environment (*Cmdenv*). *NED* is a high-level topology description language used for the definition of the simulated network, the modules and the connectivity between each other. As already mentioned, the simulations can be run using either *Tkenv* or *Cmdenv*. *Cmdenv* just writes all generated status messages to the command-line but offers no interactivity. *Tkenv* is a graphical user interface, which additionally can draw an animated graphical representation of the network. The user can watch how the messages are transmitted within the network and can inspect the current parameters of any available object (messages, nodes, etc) at any time. It can also present collected statistical data as charts and plots. Additionally, both simulation environments can store statistical data as *vector files* and *scalar files*, so the user is able to do further analysis after the simulation.

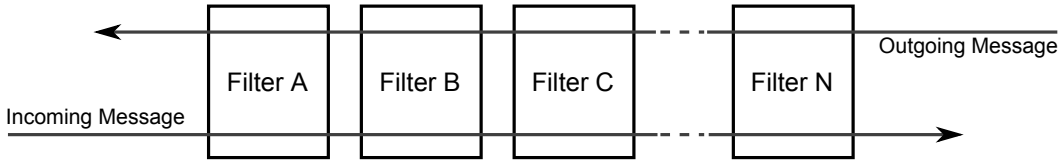


Figure 5.1: Filter Chain

5.2 Gossip Node

Participants of an OMNeT++ network are implemented as modules, which are objects that inherit from the OMNeT++ class *cSimpleModule*. In order to work properly, all modules have to override some of the following methods provided by cSimpleModule [22]:

```
void initialize()
void handleMessage(cMessage *msg)
void activity()
void finish()
```

The `initialize` method is called by OMNeT++ when the module is created, `finish` is called after a successful termination of the simulation and is typically used to record statistics. The methods `handleMessage` and `activity` can both be used to react to incoming and outgoing messages (see Section 5.3).

For the peers of our P2P network, defined in the module *GossipNode*, we use a modular approach for message handling and overall behavior. Each module contains a filter chain, set up in the `initialize` method. All messages are redirected by the `handleMessage` method to the filter chain and are processed as illustrated in Fig. 5.1. The filter chain is traveled sequentially: after a filter has successfully processed a message, it is taken to the subsequent filter. Incoming messages enter the chain at the first filter and leave it with the last filter, outgoing messages travel the other way around. Since incoming messages are no longer used after leaving the last filter, they are then deleted. The filter chain of a GossipNode consists of the filters described in Section 5.4.

5.3 Communication

OMNeT++ uses the class *cMessage* for the communication between nodes, as well as for scheduling events or other activities [22]. Different types of messages can be defined in *msg* files and the definitions in these files are automatically compiled into cMessages. Such a definition of a *msg* file looks as follows:

```
message OverlayMessage
{
        fields:
                unsigned int source;
                unsigned int destination;
}
```

The OM-QoS framework provides a special *OverlayMessage*, derived from cMessage, containing fields for source and destination node IDs. These fields are required by the P2P overlay for routing. To implement events or timers, messages can also be sent using `scheduleAt()`. This method takes a cMessage and a timestamp as parameters and does not deliver the message to the P2P overlay, but sends it back to the scheduling node at the specified time. Such a scheduled message is called *self-message*.

At initialization, each node receives a *NodeInitMessage*. This message contains general configuration parameters such as neighborhood sizes (k_{min}, k_{max}), as well as node specific parameters, such as the node's ID and an ID of a random live peer used for joining.

5.4 Filter Chain

A GossipNode filter chain consists of five filters. The Outgoing Address Filter and the Timeout Filter are generic filters, required by the OM-QoS framework. The Outgoing Address Filter is only responsible for setting the source field of outgoing overlay messages. Therefore, it has to be the first filter in the chain, to ensure all outgoing messages pass this filter last.

The Timeout Filter is used to block incoming messages for nodes, which are no longer part of the P2P network. After the node has left the network (by sending a *LeaveMessage* to any other node), the Timeout Filter discards all incoming messages, so they no longer reach the other filters, and responds with a *TimeoutMessage*. It is positioned at the second place, before all filters that handle incoming messages.

Gossip Filter, Malicious Filter and Reputation Filter implement the main functionality of REPOM. The positions of these filters are irrelevant, as long as they build the end of the chain.

5.4.1 Gossip Filter

The Gossip Filter implements the functionality a peer needs to live in our P2P network and it builds the multicast tree, as described in the Sections 4.1.1 and 4.1.2. The filter contains three sets of node IDs: a neighborhood set, a candidate neighborhood set and a blacklist. The candidate neighborhood set is used to store all peers living in the 2-hop neighborhood and the peer got knowledge about, due to *GossipMessages* from its neighbors. The neighborhood set holds all peers, to which this peer has successfully established a connection. Peers in the neighborhood set will no longer be listed in the candidate neighborhood set. Both sets map their entries against *GossipNodeInfo* objects. These objects store peer specific parameters like RTT to root, path to root, time of the last received *GossipMessage*, etc.

The blacklist is used to temporarily neglect nodes such that they are not chosen as a candidate neighbor or candidate parent. The reasons for blacklisting a node are described later on.

Joining

The joining procedure, illustrated in Fig. 5.2, follows exactly the logic described in Section 4.1.1. To initiate the procedure, a random node is contacted using a *HelloMessage*. As already mentioned, the ID of the random node is received as a parameter of the *NodeInitMessage*. In

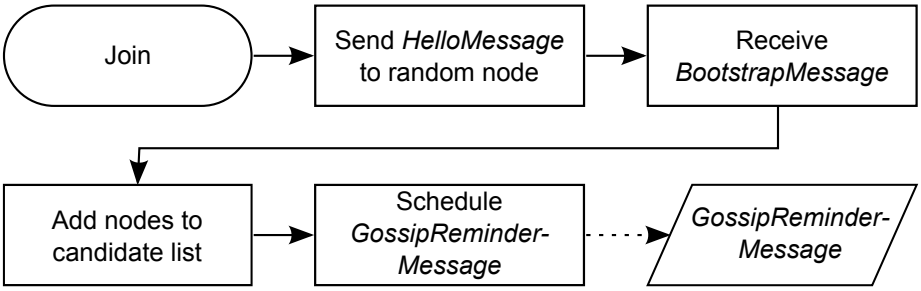


Figure 5.2: Joining Process

return, the node receives its first *GossipMessage* in the form of a *BootstrapMessage*. The peers stored in this message will be the first entries of the candidate neighborhood set. Immediately afterwards, the *GossipReminderMessage*, a special self-message, is scheduled.

Maintenance

The *GossipReminderMessage* triggers a general purpose maintenance process, as seen in Fig. 5.3. The cleanup is used to remove neighbors and candidate neighbors from their corresponding sets. A neighbor is removed if it did not send a *GossipMessage* within the last 10 seconds[1]. A candidate neighbor is removed after the same amount of time, if no incoming *GossipMessage* listed the candidate anymore.

The Refresh Neighborhood procedure is used to acquire new neighbors. As long as the node has less than k_{min} neighbors, every time this method is called, the node sends a *ConnectionRequestMessage* to a random candidate neighbor. If the candidate neighbor denies the request, it is blacklisted for 1 second and another candidate is contacted. The blacklisting occurs to guarantee that a node does not receive a connection request multiple times in a short period.

The maintenance process is also responsible for the distribution of *GossipMessages* and *AnnouncementMessages* to all neighbors. The *AnnouncementMessage* is a notification for the neighbors that this node is receiving (and therefore can forward) multicast messages. As already discussed in Section 4.1.2, the *AnnouncementMessage* also contains the RTT between sender and root, as well as the path to root, so the peers can decide, to which neighbor they should preferably subscribe to.

After completion, the *GossipReminderMessage* is rescheduled with a delay of 2 seconds.

Parent Rating and Reputation

As soon as the node has a parent, it schedules an *EvaluationReminderMessage* in a 2 second interval. This is used to rate the current parent's behavior. According to Section 4.3, the parent is rated with a value between 1.0 and -1.0, excluding zero. This value is reported to the Reputation Filter (Section 5.4.3) with a special self-message. In return, it receives self-messages containing the neighbors' reputation. As shown in the following, the reputation values are used for the

[1]The delay, as well as all other mentioned numeric values, are customizable using *NodeInitMessage* parameters

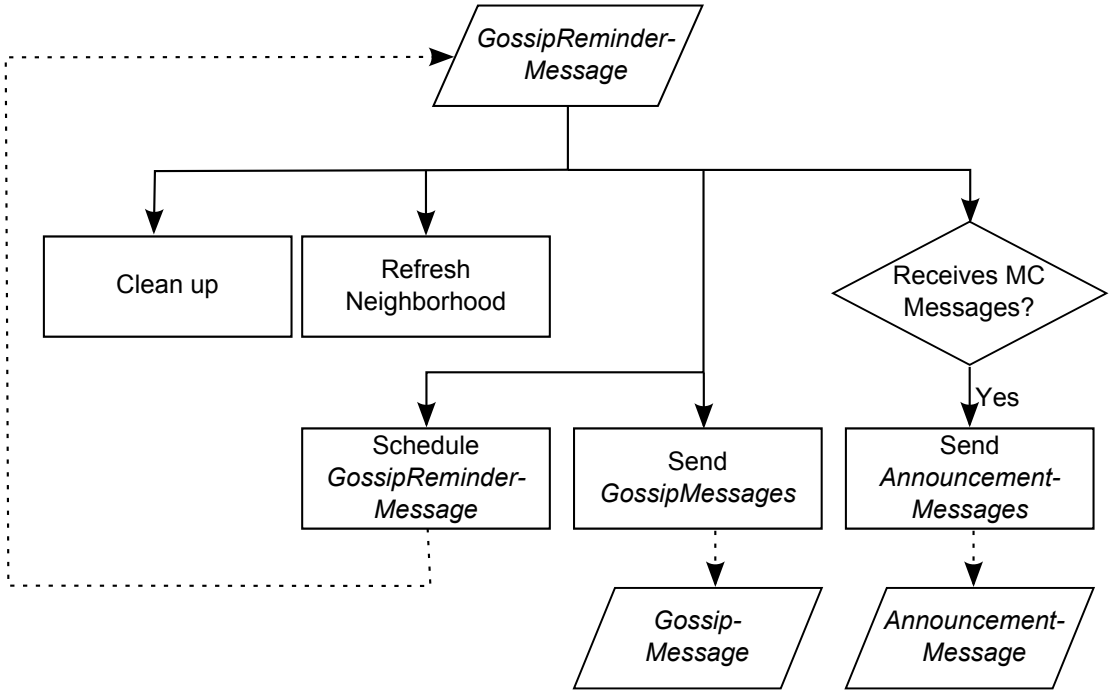


Figure 5.3: Maintenance Process

subscribing process and the generation of the weighted parent rating. The weighted parent rating is a combination of the parent's rating (created by the peer itself) and the parent's reputation (accumulated as described in Section 5.4.3).

Subscribing

Whenever the filter receives an *AnnouncementMessage*, it checks, whether its current parent is still the best available. Therefore, the neighborhood set is scanned for a parent candidate. To become a parent candidate, a neighbor has to fulfill the following conditions:

- The neighbor must not be blacklisted.
- The neighbor's reputation must be greater than or equal to the current parent's weighted rating.
- The neighbor's reputation may only be 0.05 lower than the highest reputation of all neighbors.
- The neighbor has to be a multicast forwarder.
- The node must not be part of the neighbor's path to root.

Should multiple neighbors pass these restrictions, the neighbor with the lowest RTT becomes a parent candidate. If a parent candidate is available, a *SubscribeRequestMessage* is sent to this

peer, which will either accept the request and become the node's new parent, or it will deny the request. In that case, the filter will blacklist the candidate for 1 second and look for another parent candidate.

5.4.2 Malicious Filter

At initialization, the Malicious Filter determines the node's level of maliciousness. First, it decides whether it is malicious in any manner, or not at all. This decision is based on a probability factor in form of a Node Init Message parameter. If the filter chooses to behave uncooperatively, it randomly assigns itself to one of the four malicious types, according to the definition in Section 4.2. The filter is then in one of five different states reaching from 0 (non-malicious) to 4 (totally uncooperative). This state is temporarily changed, only if the node becomes root, to avoid the complete breakdown of the P2P network. However, since the Gossip Filter sends root handover requests preferably to peers with a good reputation, this behavior should not result in statistical inaccuracy.

The Malicious Filter now processes all outgoing multicast messages according to its state. They either are passed to the next filter or are dropped. Some states only drop messages with a certain probability. Again, this parameter can be defined in the Node Init Message. The behavior of the filter in the different states can be described as follows:

0 All multicast messages are passed through.

1 A multicast message is dropped with a probability of 50%

2 A multicast message addressed to the first neighbor is dropped with a probability of 50%

3 All multicast messages addressed to the first neighbor are dropped

4 All multicast messages are dropped.

5.4.3 Reputation Filter

The Reputation Filter is responsible for the distribution of the parent ratings and the generation of peer reputation values. The Gossip Filter regularly sends the current parent's rating to the Reputation Filter. Besides that, the Reputation Filter also has to become notified about changes in the peer's neighborhood. Otherwise, the filter would not know, to which peers it has to send the estimated ratings.

Whenever a self-message containing the current parent rating arrives, it is sent to all neighbors, which forward it to their neighbors. This way, the rating reaches the 2-hop neighborhood. When the Reputation Filter receives such a rating, it stores the value together with the ID of the rating peer (i.e. the source of the message), the ID of the rated peer (i.e. the source's parent) and the time the message was received. If an entry from the specified source already exists for the rated peer, it is overwritten with the new value.

On a regular interval of 2 seconds, the Reputation Filter generates reputation values for all of its neighbors. Therefore, the collected ratings are decreased due to their age, using the following

formula:

$$r' = r \left(\frac{1}{2}\right)^{\frac{t-t_0}{t^*}}$$

Here, r represents the reported rating, t_0 the time in seconds the rating was received and t the current time. The (Node Init Message) parameter t^* is set to 1.71, so the rating looses half of its value approximately every 1.5 seconds. If $|r'|$ is lower than 0.01, the rating is considered as outdated and is discarded. All available r' values for a specific neighbor are summed and the sum is divided by the amount of available values. This builds the neighbor's reputation, which is reported to the Gossip Filter with a special self-message.

Chapter 6

Evaluation

The simulation to evaluate REPOM used thirteen distance matrices with delay values according to Table 6.3 and the three unique random seeds presented in Table 6.2. Therefore, 39 different networks were built for each network size. The different settings were tested in various network sizes, with 100 - 2000 nodes. We run different scenarios, which are summarized in Table 6.1. The first three scenarios were used to distinguish reasonable values for the parameters neighborhood size and amount of malicious peers. The final scenario was used to compare the performance of the network run natively, with malicious peers and with reputation management.

Table 6.1: Evaluated scenarios with Correlating Parameters

Scenario	**Neighborhood**	**Malicious Nodes**	**Reputation System**
Native	$k_{max} \in \{6, 8, 10\}$	0%	Disabled
Maliciousness	$k_{max} = 8$	{20%, 30%, 50%}	Disabled
Reputation	$k_{max} = 8$	{20%, 30%, 50%}	Enabled
Comparison	$k_{max} = 8$	{0%, 30%}	{Disabled, Enabled}

For the estimation of the parameters, the network size started at 200 nodes and was incremented by 200 nodes per run, resulting in a total of 390 runs per scenario (10 different network sizes, 13 distance matrices and 3 random seeds). For the final simulations used for the comparison, the network size started at 100 nodes and was incremented by 100 nodes per run. This results in 780 simulation runs for this scenario.

For better comparison, each of the following figures shows the 99% confidence interval. It covers the combined data of all graphs in the corresponding figure, because drawing one confidence interval for each graph would lead to overlaps, and would therefore not be distinguishable.

6.1 Native Gossip-based Network

We compared three different versions of the gossip-based overlay multicast network using different neighborhood sizes, with $k_{max} \in \{6, 8, 10\}$ and $k_{min} = \frac{k_{max}}{2}$. To test the network's perfor-

Table 6.2: Random Seeds

Seed	Value
Seed 0	1768507984
Seed 1	33648008
Seed 2	1082809519

Table 6.3: Delay Properties of Distance Matrices (in ms)

Matrix	min RTT	mean RTT	max RTT
Matrix 0	0.08	22.47	48.44
Matrix 1	0.09	30.35	90.48
Matrix 2	0.05	30.56	94.58
Matrix 3	0.05	29.76	90.23
Matrix 4	0.07	23.26	57.52
Matrix 5	0.09	22.78	51.82
Matrix 6	0.04	22.77	49.24
Matrix 7	0.08	23.27	52.30
Matrix 8	0.05	22.91	53.92
Matrix 9	0.05	23.27	50.83
Matrix 10	0.08	22.47	48.44
Matrix 11	0.05	22.91	54.00
Matrix 12	0.01	23.13	54.17

mance, we are interested in the amount of received multicast messages, time without multicast parent, hop count, fan-out and node to root RTT. These parameters are described in Table 6.4.

The average percentage of received multicast messages per node is shown in Fig. 6.1. The measurement of this value starts as soon as the node receives its first *AnnouncementMessage*. This means that all messages distributed before the node joined, are not counted as losses. As we can see, the majority of all nodes receives more than 98% of all multicast messages. The chosen neighborhood size affects this in such a way that higher populated networks benefit from a greater neighborhood size. However, the improvement from $k_{max} = 8$ to $k_{max} = 10$ is only marginal.

The reason why there are losses in a network without any malicious peers is because not all nodes always have a multicast parent. First, there is the time between the joining and the successful subscription to one of the neighbors, which leads to lost multicast messages. Second, there is a gap from the moment when the parent leaves the network until a new parent is found. Figure 6.2 illustrates how long peers continuously remain without multicast parents. Again, the smallest neighborhood is responsible for the worst values. This is because it is harder to find other peers, which still can accept new connections if the neighborhood size is kept at a low

Table 6.4: Measured Parameters

Parameter	Explanation
Percentage of received multicast messages	Total amount of multicast messages a node received in relation to the amount of multicast messages the root distributed during the node's live time
Time without multicast parent	Time a node stays without a multicast parent without interrupt
Hop count	Number of nodes the message passed until it reached the measuring node
Fan-out	Number of nodes to which the measuring node forwards the current multicast message (i.e. subscribers)
Node to root RTT	Required time for a message to travel from the root to the measuring node and back again

number. Therefore, it is also harder to find new parent candidates.

The fan-out shown in Fig. 6.3 represents the amount of neighbors to which a peer has to forward multicast data. It is registered every time a peer receives multicast data. On average, the fan-out is approximately 1 for all neighborhood sizes and network sizes. In other words, neither the neighborhood size nor the network size affect the fan-out significantly. The fan-out also indicates that in a network with more than 200 peers, no peer with a full neighborhood set has to serve all its neighbors, since the maximal fan-out is always lower than k_{max}.

The hop count illustrated in Fig. 6.4 shows by how many peers a multicast message has been forwarded, before it arrived at the measuring peer. The average hop count lies between four and eleven hops and is increasing the more peers live in the network. Again, it is noticeable that the network with the smallest neighborhood size performs worst. As also remarked for Fig. 6.1,

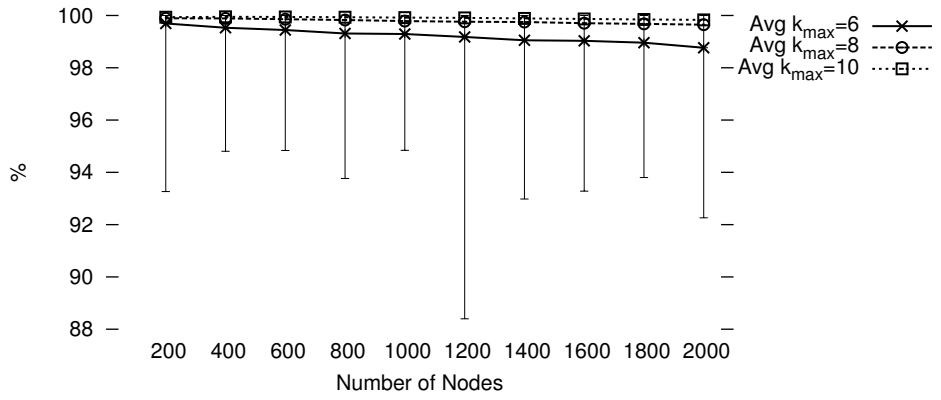


Figure 6.1: Percentage of received multicast messages

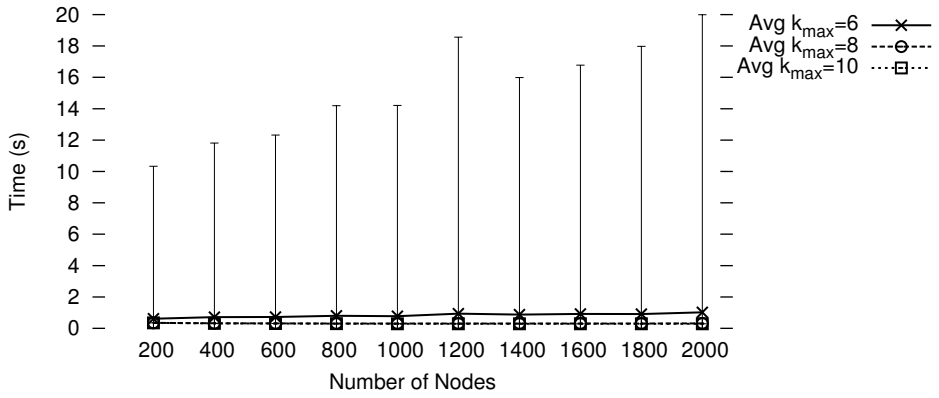


Figure 6.2: Time without multicast parent

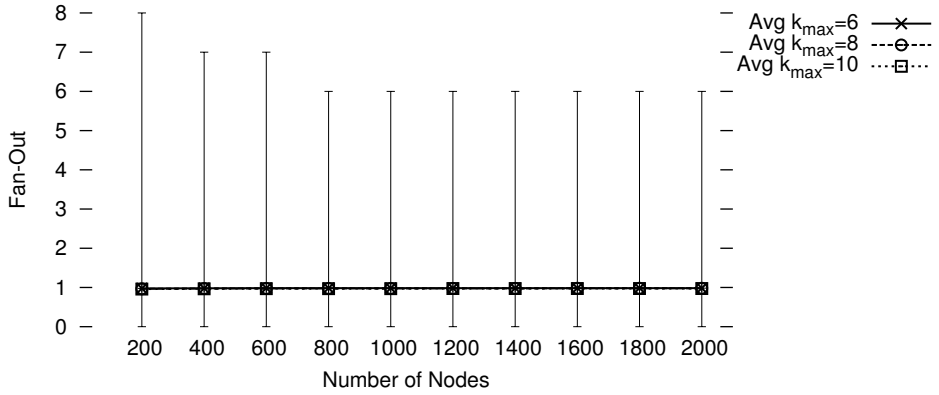


Figure 6.3: Fan-Out

there is only a low improvement from $k_{max} = 8$ to $k_{max} = 10$. The high maximum values derive from the peers that stay with their minimal neighborhood size of $\frac{k_{max}}{2}$. If peers with only three neighbors subscribe to each other, a high hop count is easily reached.

6.2 Malicious Peers and Reputation Management

To see, how the presence of free-loading peers affects out P2P network, we ran simulations of networks with different numbers of malicious nodes. We tested networks with 20%, 30% and 50% of malicious nodes, where the different behavior types of malicious nodes were uniformly distributed. For the neighborhood size, we set $k_{max} = 8$. As Section 6.1 showed, a network with $k_{max} = 6$ performs not well enough, whereas one with $k_{max} = 10$ performs only slightly better than one with $k_{max} = 8$.

Simultaneously, we run simulations with the same settings, but with peers that use the reputation system, in order to see the impact that malicious peers have on a system using reputation as well.

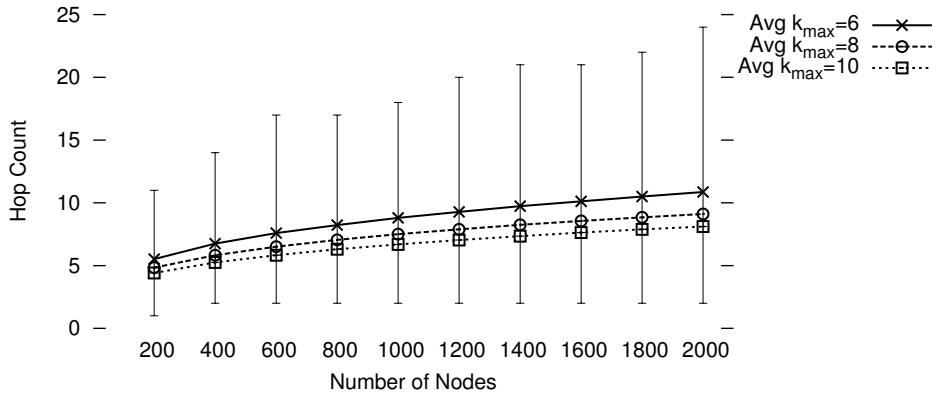


Figure 6.4: Hop Count

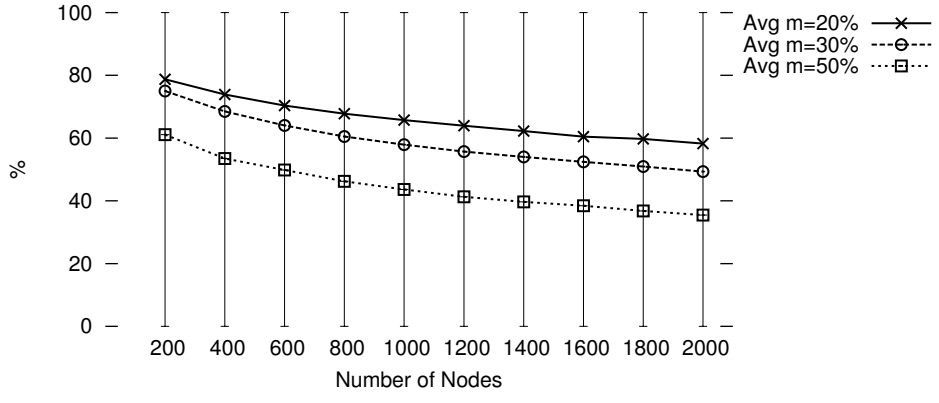
Of course, having free-loading peers in the network causes additional losses of multicast messages. This is confirmed by the results shown in Fig. 6.5. Without any counter measures, peers on average receive only 60% - 80% of all messages, when 20% of the peers are malicious. In the worst case, with 50% of malicious peers, the peers only receive 40% - 60% of messages. When peers use the reputation system, there are still losses of multicast messages. However, it is no longer that severe. As shown in Fig. 6.5(b), on average peers never receive less than 94% of messages, even in the worst case with 50% of all peers being malicious.

Another interesting parameter is the node to root RTT. Figure 6.6(a) reveals that the number of malicious peers does not affect this value. This result was expected, since changing the behavior type of some nodes should not change the network's topology. Peers still try to subscribe to a neighbor with the lowest RTT to root. Enabling the reputation system however, encourages the peers to preferably subscribe to a neighbor with a good reputation and a bad RTT to root rather than to a neighbor with complementary attributes. This is reflected in Fig. 6.6(b): the more malicious peers are participating, the higher the average RTT to root gets. However, whereas the RTT to root values increase in networks with 20% - 30% of malicious peers is quite low, 50% of malicious peers does considerably more damage.

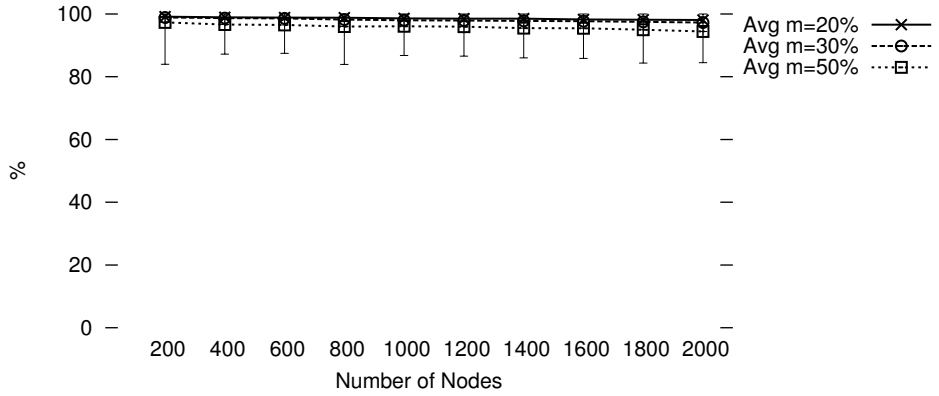
6.3 Performance Comparison

For the final comparison of the different scenarios (native, malicious and malicious with reputation management), we chose the amount of malicious peers of 30%. As Section 6.2 showed, with this setting nodes loose a considerable amount of multicast messages if they do not rely on the reputation system. If they do, however, the node to root RTT penalty is not as significant as with 50% of malicious peers.

First, we look again at the number of received multicast messages illustrated in Fig. 6.7. The results confirm a significant loss of data for the scenario, where free-loading peers are not recognized. It is also confirmed that the reputation system is capable of increasing the receiving rate up to over 94%. While this surely is an improvement, it does not reach the performance of the native version of the P2P network. Here, on average nodes receive more than 99% of



(a) without Reputation Management



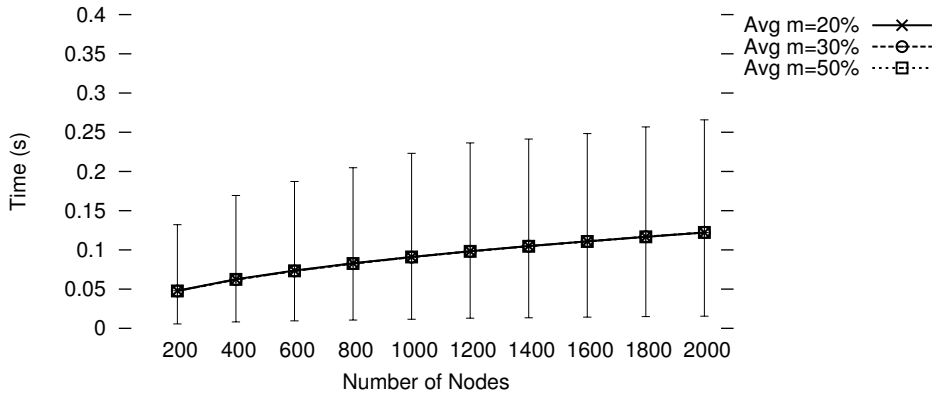
(b) with Reputation Management

Figure 6.5: Percentage of received multicast messages

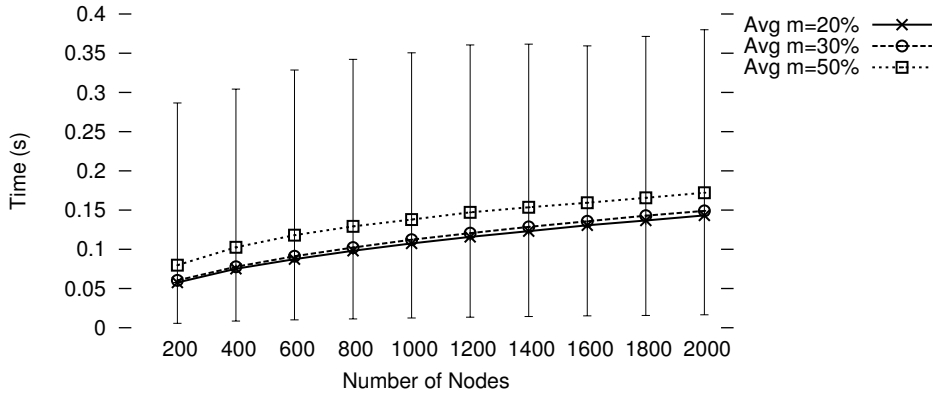
all messages. The major reason for this difference is the loss that occurs during the time nodes require to rate their (malicious) parents and connect to a new one. During this time, a node could encounter the situation that it lives in a neighborhood of only malicious peers. In that case, the node will disconnect and perform a complete rejoin. Until the rejoin is completed and a new multicast parent is found, the node is not subscribed to any other node, and therefore has no chance to receive multicast messages. This additional time of not having a multicast parent can be seen in Fig. 6.8. But, since living in a completely malicious neighborhood should be a rather rare event, the duration nodes additionally remain without a parent is usually short.

For completeness, we also compare node to root RTT, fan-out and hop count. Figure 6.9 confirms that the amount of malicious peers does not have influence on the node to root RTT. The native scenario performs as well as the one with free-loaders. However, a slight performance drawback for the reputation-based setting is still observable. This behavior was expected, since Section 6.2 revealed very similar results.

Surprisingly, Fig. 6.10 shows that the fan-out of malicious networks is slightly higher com-



(a) without Reputation Management



(b) with Reputation Management

Figure 6.6: Node to root RTT

pared to the native network and the reputation-based network. This is caused by the fact that the fan-out is measured each time a node receives and then forwards a multicast message. In this scenario, the probability for a leaf node to receive no multicast message at all is considerably higher than for the other scenarios. It is likely that a lot of such nodes never perform a fan-out measurement. Therefore, fewer zero-measurements are registered, which increases the average.

The same reasoning can be used to explain why the hop count illustrated in Fig. 6.11 is lower for the malicious scenario. The probability for a multicast message to reach the lower parts of the multicast tree decreases when the amount of malicious peers increases. Therefore, fewer high hop counts are measured, which lowers the average hop count. If the reputation system is enabled however, the average hop count increases again. In fact, it is higher than the average hop count of the native network. This result mirrors the outcome of the node to root RTT measurements. The nodes preferably subscribe to a node with a higher node to root RTT than to a malicious one with a lower node to root RTT. It is not unlikely that the higher node to root RTT is also connected to a higher hop count.

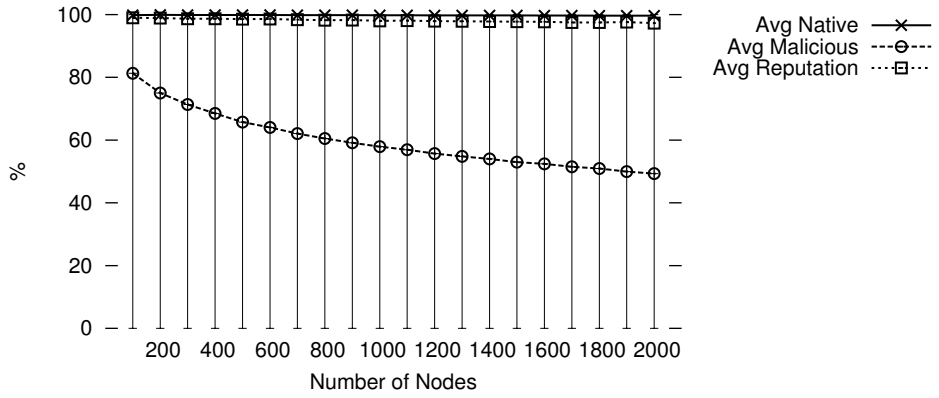


Figure 6.7: Percentage of received multicast messages

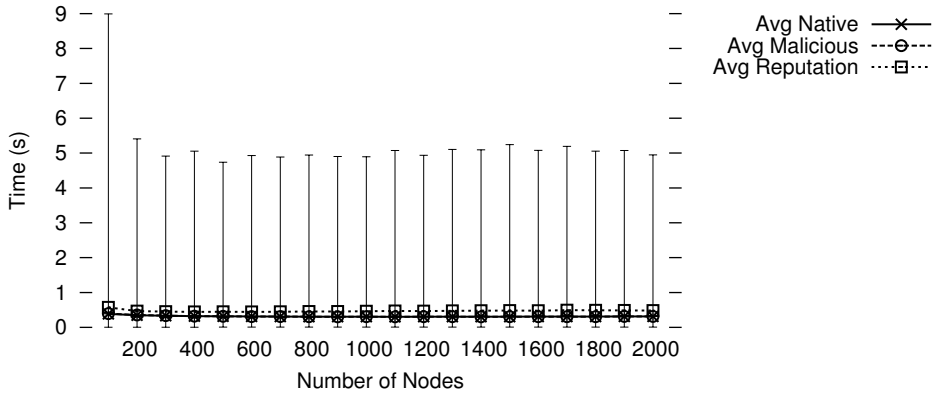


Figure 6.8: Time without multicast parent

6.4 Conclusion

In this Section, we presented the results of our simulations. We evaluated various scenarios to see, how different parameters affect the performance of our gossip-based P2P overlay multicast network. We saw that increasing the neighborhood size leads to a higher amount of received multicast messages. This is due to the fact that peers living in a large neighborhood can subscribe to other peers more efficiently. The neighborhood size also has an influence on the hop count. Increasing the neighborhood size increases the amount of subscribers a parent can have. Consequently, it lowers the hop-count.

The presence of malicious peers significantly affects the amount of received multicast messages. The more malicious peers there are, the more messages get lost. However, it does not have influence on the topology. Therefore, the other parameters remain the same.

Finally, we saw that the reputation system is capable of reducing the damage done by malicious peers. Although, the results are not as good as without any malicious peers, the percentage of received multicast messages is back at an acceptable level. The only drawback is a higher

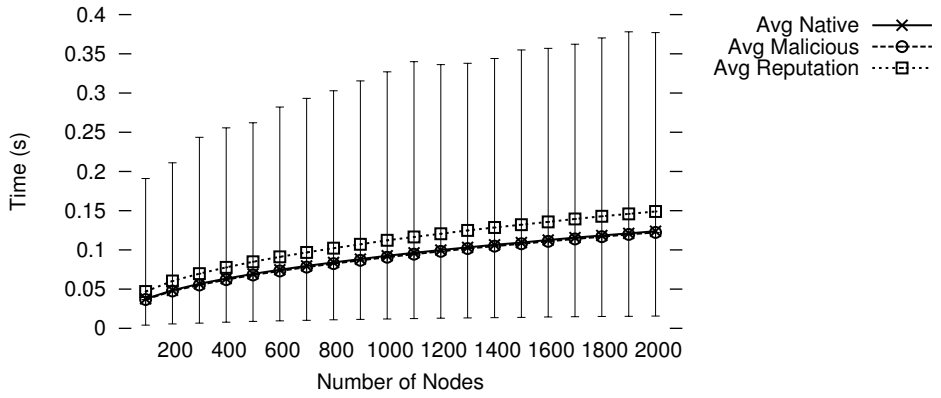


Figure 6.9: Node to root RTT

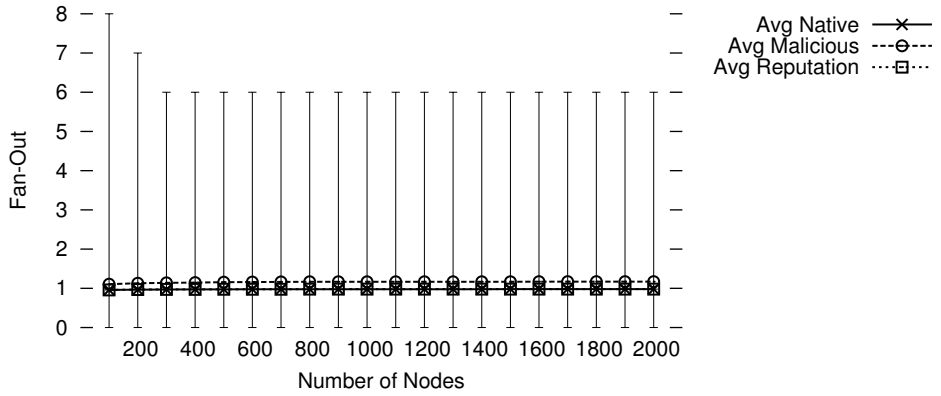


Figure 6.10: Fan-Out

node to root RTT. This results from the fact that peers change to a parent with a higher node to root RTT if its reputation is considerably higher than the current parent's reputation.

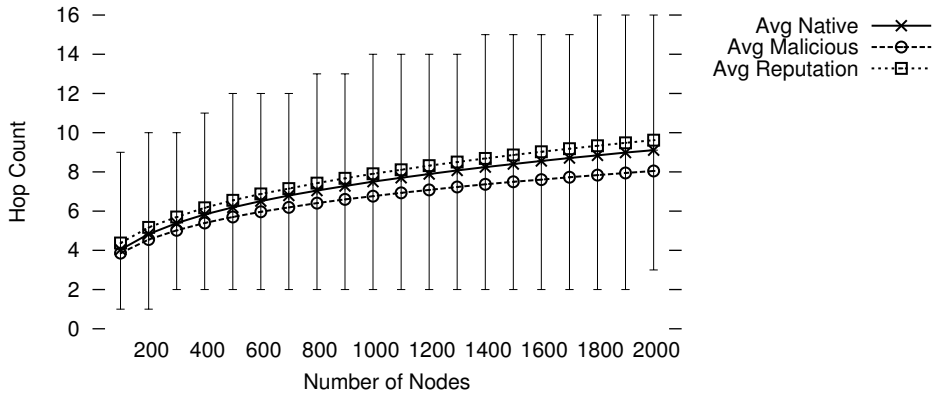


Figure 6.11: Hop Count

Chapter 7

Conclusion and Outlook

7.1 Conclusion

For this Bachelor thesis, we implemented a gossip-based overlay multicast network for the OMNeT++ simulation environment. We added the possibility to specify a certain number of peers to be malicious. Those peers do not forward all multicast messages to their subscribers. This is used to determine how the presence of free-loading peers affects the performance of an overlay multicast network. We also designed and implemented a reputation system, which allows peers to exchange information about their parents' behavior. This information influences the peers' parent selection procedure so that they do not subscribe to malicious peers.

We performed various simulations with different settings of the three network types (native, malicious, reputation-based) and compared the results with each other. This showed that free-loading peers have a great impact on the network's performance. With the native version, peers received more than 99% of multicast messages. With the malicious version, peers only received 50% to 80% of multicast messages (depending on the network size) if 30% of the peers are free-loading. It also showed that the reputation system is capable of competing against the loss of data. With the reputation system enabled, the peers' average receiving rate was at 94% for the worst case. Although this is not as good the native version, the improvement achieved compared to networks without reputation management is apparent.

7.2 Outlook

Peers that are actively malicious could harm the network. This was though not in the scope of this thesis. There would be several countermeasures that could be applied against them. For example, there could be peers, which create fake parent rating information to persuade other peers to subscribe to a malicious node. In the following, we briefly propose a possible enhancement to our reputation system, which could allow peers to detect fake reputation messages.

Each node generates a pair of keys for asymmetric digital signing and signature verification. Nodes then sign each reputation message they distribute and append the signature to the message. Now, nodes can verify every nth incoming reputation message for a particular parent. Therefore, it has to contact the creator of the message once, and request it's public key. With this

key, the receiver can verify the signature of every reputation message from this node. If a message should not be valid, the corresponding reputation value is dropped. The nodes now know with a certainty of $\frac{100}{n}\%$ that their calculated overall reputation value for that node is correct. Furthermore, a node could increase the certainty value by simply deciding to no longer accept reputation messages from a peer, which recently provided a fake reputation message.

This procedure reveals reputation messages, which have been altered by other nodes, or which have been created by nodes using a bogus identity. A malicious parent, for example, could try to impersonate another node and send good reputation messages for itself. However, a node can still create a fake reputation message for any node with a valid signature, using its real identity. To ensure that a node can only distribute reputation messages about its own parent, an additional feature has to be added. Whenever a parent forwards multicast messages, which are consecutively numbered, it adds the signature of the current message number to the message. The children nodes now have to add the last received number / signature pair as well as their parents' ID to the reputation messages they distribute. This way, a receiver of such a message can verify that the node currently is, or recently has been, subscribed to the specified parent. If this is not the case, the received reputation value has to be dropped.

What still remains is that peers could distribute fake reputation messages for their own parent. We did not find a satisfying solution for this problem yet. However, it is questionable, why a peer would want to do this. Neither does the peer profit from a parent with an inaccurate rating, nor does it harm the parent. Actually, the contrary is the case. If the peer prefers to distribute good reputation messages for its bad behaving parent instead of switching to another parent, it accepts to not receive all multicast messages.

Another still existing problem is that malicious peers could collaborate with each other. For example, a group of peers could exchange their multicast message number signatures. Hence, they are able to create valid reputation messages for each other without being in a parent-child relationship.

Bibliography

[1] K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *Communications Surveys & Tutorials, IEEE*, pp. 72–93, 2005. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1528337

[2] S. Androutsellis-Theotokis and D. Spinellis, "A survey of peer-to-peer content distribution technologies," *ACM Comput. Surv.*, vol. 36, no. 4, pp. 335–371, 2004.

[3] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web," in *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*. ACM Press, 1997, pp. 654–663. [Online]. Available: http://dx.doi.org/10.1145/258533.258660

[4] S. Deering, "Host extensions for IP multicasting," Internet Engineering Task Force, RFC 1112, Aug. 1989. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1112.txt

[5] M. Hosseini, D. T. Ahmed, S. Shirmohammadi, and N. D. Georganas, "A survey of application-layer multicast protocols," *Communications Surveys & Tutorials, IEEE*, vol. 9, no. 3, pp. 58–74, 2007. [Online]. Available: http://dx.doi.org/10.1109/COMST.2007.4317616

[6] M. Feldman, K. Lai, I. Stoica, and J. Chuang, "Robust incentive techniques for peer-to-peer networks," in *EC '04: Proceedings of the 5th ACM conference on Electronic commerce*. New York, NY, USA: ACM, 2004, pp. 102–111.

[7] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A scalable content-addressable network," in *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, vol. 31, no. 4. New York, NY, USA: ACM, October 2001, pp. 161–172. [Online]. Available: http://dx.doi.org/10.1145/383059.383072

[8] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, F. M. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Trans. Netw.*, vol. 11, no. 1, pp. 17–32, February 2003. [Online]. Available: http://dx.doi.org/10.1145/638334.638336

[9] "Another bittorrent protocol specification." [Online]. Available: http://wiki.theory.org/BitTorrentSpecification

[10] B. Cohen, "Incentives Build Robustness in BitTorrent," *In Proc. 1st Workshop on Economics of Peer-to-Peer Systems*, May 2003.

[11] The Gnutella Developer Forum, "The Annotated Gnutella Protocol Specification v0.4," July 2003. [Online]. Available: http://rfc-gnutella.sourceforge.net/developer/stable/index.html

[12] M. Feldman and J. Chuang, "Overcoming free-riding behavior in peer-to-peer systems," *SIGecom Exch.*, vol. 5, no. 4, pp. 41–50, 2005.

[13] I. Keidar, R. Melamed, and A. Orda, "Equicast: scalable multicast with selfish users," in *PODC '06: Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*. New York, NY, USA: ACM, 2006, pp. 63–71.

[14] F. Milan, J. J. Jaramillo, and R. Srikant, "Achieving cooperation in multihop wireless networks of selfish nodes," in *GameNets '06: Proceeding from the 2006 workshop on Game theory for communications and networks*. New York, NY, USA: ACM, 2006, p. 3.

[15] E. Damiani, D. C. di Vimercati, S. Paraboschi, P. Samarati, and F. Violante, "A reputation-based approach for choosing reliable resources in peer-to-peer networks," in *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2002, pp. 207–216.

[16] M. Khambatti, P. Dasgupta, and K. D. Ryu, "A role-based trust model for peer-to-peer communities and dynamic coalitions," in *IWIA '04: Proceedings of the Second IEEE International Information Assurance Workshop (IWIA'04)*. Washington, DC, USA: IEEE Computer Society, 2004, p. 141.

[17] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in p2p networks," in *WWW '03: Proceedings of the 12th international conference on World Wide Web*. New York, NY, USA: ACM, 2003, pp. 640–651.

[18] S. Buchegger and J.-Y. Le Boudec, "Performance analysis of the confidant protocol," in *MobiHoc '02: Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*. New York, NY, USA: ACM, 2002, pp. 226–236.

[19] OMNeT++ Community, "What is omnet++?" Website, August 2009. [Online]. Available: http://www.omnetpp.org/home/what-is-omnet

[20] L. Bettosini, "Quality of Service for Overlay Multicast Content Addressable Network (CAN)," Master's thesis, University of Bern, Switzerland, August 2009.

[21] R. W. Quong, "Omnet++ - manual," Website, August 2009. [Online]. Available: http://www.omnetpp.org/doc/omnetpp33/manual/usman.html

[22] “OMNeT++/OMNEST Simulation Library,” Website, August 2009. [Online]. Available: http://www.omnetpp.org/doc/omnetpp33/api/index.html