

SocioCast, the Social Extension to Tribler

Wouter L. de Vries



SocioCast, the Social Extension to Tribler

Master's Thesis in Computer Science

Parallel and Distributed Systems group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

W. L. de Vries

25th August 2008

Author

Wouter L. de Vries

Title

SocioCast, the Social Extension to Tribler

MSc presentation

August 22, 2008 at 16:00

Room 09.130, Mekelweg 4, Delft

Graduation Committee

prof. dr. ir. H. J. Sips (chair) Delft University of Technology

ir. dr. J. Pouwelse Delft University of Technology

dr. P. H. Westendorp Delft University of Technology

Abstract

High quality metadata and spam prevention has proven to be a difficult challenge in completely distributed P2P systems. With the SocioCast subsystem for Tribler we try to solve this challenge. Tribler is such a completely distributed P2P client based on the Bittorrent protocol, developed at the Delft University of Technology. Metadata available in torrent networks consists of only filenames. The ModerationCast subsystem has been introduced in Tribler to increase the amount of available metadata. These moderations are however based on wiki style last-write authority and prone to abuse.

We present SocioCast, a completely decentralized Tribler subsystem, that estimates the quality of moderations based on social connections and user feedback. We show that SocioCast scales well by utilizing the scale-free property of social networks where a few peers serve as hubs for the entire network.

Preface

Many years ago I worked for MediaDesign as a programmer on the social networking site cu2.nl. The social relations between its users intrigued me and I wondered if they could be put to use. Mathijs de Weerdt introduced me to the subject of my research: incentivation methods in multi agent systems. Something as intangible as reputation and trust between inanimate objects inspired me. This thesis is a logical followup to my research assignment and my work on a social networking site. In this thesis I introduce a hybrid between a trust system and a social network to calculate a relevance index for moderations in the Tribler peer-to-peer client.

I would like to thank my supervisor Johan Pouwelse for his support and positive attitude during the work on my thesis. Also I would like to thank Freek Zindel, my second supervisor, for his support and advise. He made sure my work stayed on track and helped me keep up the pace. Furthermore I would like to thank Jack Ha for reviewing my thesis. Finally I would like to thank my parents for their endless patience and their monetary support during the many years of my study.

Wouter L. de Vries

Delft, The Netherlands
25th August 2008

Contents

Preface	v
1 Introduction	1
1.1 Scope	1
1.1.1 Tribler	2
1.1.2 Metadata	2
1.1.3 ModerationCast	3
1.2 Research Objective	3
1.3 Report Structure	5
2 Related Work	7
2.1 Trust	7
2.1.1 Prisoners Dilemma	8
2.1.2 Tit-for-tat	9
2.1.3 Correlated Opinions	10
2.2 Reputation	11
2.2.1 Majorities rule	12
2.2.2 Weighted opinions	13
2.3 Conclusion	16
3 SocioCast Design	19
3.1 SocioCast Protocol	20
3.2 Social Relevance Index	20
3.2.1 Breadth First Search	21
3.2.2 Max-flow	22
3.2.3 Bootstrapping	23
3.3 Conclusion	23
4 Constructing the SocioCast Model	25
4.1 Social Network Analysis	25
4.2 Network Generation Algorithms	26
4.2.1 Small World	26
4.2.2 Scale-Free	27
4.2.3 Combination	28

4.2.4	Comparison of Peer Influence in Generated Networks . . .	29
4.3	Modeling Setup	31
4.4	Conclusion	33
5	Simulations of Speed and Scalability	35
5.1	SRI Implementations	35
5.1.1	Breadth First Search	36
5.1.2	Edmonds-Karp Max-flow	36
5.1.3	Max-flow Heuristic	37
5.2	Social Network Exploration Speed	38
5.3	Network Exploration with Restricted Message Length	40
5.4	SocioCast Scalability	41
5.5	SocioCast Scalability with Link-weight based Pruning	42
5.6	Directed Exploration in Social Networks	43
5.7	Conclusion	44
6	Simulations of Attack Resistance	45
6.1	Sybil Collusion Resistance	45
6.2	Traitor Collusion Resistance	46
6.3	Worst-case Traitor Resistance	47
6.4	Conclusion	48
7	Implementation	49
7.1	Tribler Modifications	49
7.2	SocioCast Protocol	50
7.3	Database Layout	50
8	Conclusions	53
8.1	Summary and Conclusions	53
8.2	Future Work	54

Chapter 1

Introduction

Social relations have a big impact on how people act. People are likely to trust their friends and have the same interests as their friends. This overlap in interest and trust in friends can be combined to gain more knowledge. Based on the knowledge of friends and trust in these friends their knowledge can be used to form opinions on things like music, digital cameras or video clips.

For example, when someone is in the process of deciding which camera to buy objective information is needed about all the available models to make an informed choice. There are several ways of getting more information:

A shop assistant usually knows a lot about all the cameras for sale, however he may be biased by the profit margin on the cameras. On the other hand friends are less likely to provide biased information, but they may not be informed enough to provide good information about all the cameras available.

The best source of information would be a shop assistant that is also a friend, but this is not very likely to happen often. More likely a friend of a friend knows about camera models and is willing to give unbiased information free of charge.

For the Tribler P2P file sharing client we use a similar strategy to provide users with opinions about video content using the knowledge of their friends.

1.1 Scope

With the research presented in this thesis we will give users of the Tribler P2P file sharing client a method to estimate the quality of moderations about video content. These quality ratings are based on the social relations of users in the Tribler network.

1.1.1 Tribler

Tribler[19] is a peer-to-peer (P2P) file sharing client, based on the Bittorrent[5] protocol, developed at the Delft University of Technology. Its main focus is the exchange of video content combined with social cohesion in a distributed context. This seems like a trivial problem since sites like YouTube[24] exist as a platform to share videos and make friends. However the emphasis of the research is on making a completely distributed system, a key advantage over the old client-server model.

What makes P2P file sharing different from the old fashioned client-server download is the role of peers in these networks. In contrast to the client-server model data exchange is not one way. Peers in P2P networks are all equal and each peer can send and receive data to and from every other peer. Despite the name people usually log into these P2P file sharing networks not to share, but to retrieve files.

P2P is a cost effective way of exchanging data since it does not require servers to be present. Hosting data on servers costs money and a distributed solution enables anyone to share content with the world using a normal PC. A P2P network scales inherently better than traditional client-server systems. Both the bandwidth supply and demand grow by the same factor as the network grows.

Also completely distributed systems do not have a single point of failure. When a server shuts down, is censored or is disconnected the entire network notices the effect. When however several peers in a distributed network are disconnected the effect is hardly noticeable.

Files are shared in bundles called torrent networks. These networks are identifiable by *.torrent* files which are listed on public websites. These *.torrent* files contain the filenames of the files shared in the torrent network and their checksums. Also included is information about the tracker, a central server that keeps track of all peers in the torrent network. A Bittorrent client needs this information to initiate the download because otherwise it is impossible to find peers that are part of the torrent network.

1.1.2 Metadata

Normal Bittorrent clients require a central server to list available video content. Operators of these sites are the central authority and have complete control of the torrents listed on the site. People trust the quality of these torrents because they trust the operators of these sites due to previous experience and word of mouth.

Tribler does not require a central storage of *.torrent* files. A subsystem of Tribler called BuddyCast employs an epidemic protocol to allow Tribler peers to exchange these torrent with each other. This removes the dependency on a central server, a potential single point of failure. However since each and every peer is able to insert *.torrent* files into the system users are not able to trust the system as a whole and

therefore there is no guarantee for quality.

Currently the only information available about torrents in the Tribler network is the filename of the *.torrent* and the filenames contained within the torrent networks. This information is not sufficient for users to choose between several versions of the same video content. Filenames may be incorrect or insufficient and can not be changed after injection into the system.

1.1.3 ModerationCast

In an upcoming version of Tribler peers will be able to add moderations to torrent networks and spread them around the network using the ModerationCast[10] protocol. These moderations are packets of metadata created by users and assigned to torrent networks to better inform users about the contents of torrents.

Since Tribler is a completely distributed system there is no central trust authority and therefore the challenge is to determine which moderations are of good quality. To prevent the spreading of bad moderations peers do not automatically forward moderations created by other peers. Only when a user specifically approves the moderations of another user will these moderations be forwarded by the Tribler client. Another option is for users to ignore moderations from a specific user.

This forwarding strategy makes propagation of moderations that have been approved by multiple peers faster, but it is easily matched by people with malicious intent. A colluding network of peers can easily propagate more metadata than a normal moderator can, bandwidth is relatively cheap. Also, the option to ignore moderations does not limit the propagation speed of these malicious moderations. It is essentially a trade-off between the cost of bandwidth to operate a collusion network versus the income generated by a spam attack. This income may be derived from promoting content by making it appear to be something else.

1.2 Research Objective

The key research challenge of this thesis is to design a subsystem, SocioCast, for the Tribler P2P client that allows users to determine the quality of moderations accompanying a torrent network. There are several requirements for the solution to this challenge. The combination of adherences of SocioCast to these requirements should make it credible that SocioCast can successfully improve the quality of moderations in the Tribler network.

1. Using SocioCast peers can create a view of the Tribler social network.
2. SocioCast scales with a limited cache size to store social relations.
3. Socially relevant information equals good information.

4. SocioCast is resistant against attacks.
5. SocioCast requires as little interaction with the user as possible.
6. SocioCast gives preference to propagating good moderations over bad moderations.

The first requirement is the most basic, a peer with social connections should be able to create a view of the social network by exchanging relationship information with other peers in the network.

The second requirement is the most important requirement. SocioCast should provide *unbound scalability*. It should enable quality moderations on millions of Internet-connected television sets given a fixed size for the social relation cache. The ability of a P2P system to scale to extremely large network sizes is of the utmost importance, P2P systems that do not scale will never succeed in acquiring a large user base.

The third requirement consists of proving that high correlation exists between social relevance of peers and quality of their information. Using a partial view of the social network, SocioCast can calculate the social relevance of other peers. Social relevance depends on how peers relate to each other socially. Peers that deliver information of high quality to other peers are usually altruistic. Altruistic peers will probably also be socially active, have a lot of friends, and thereby be socially relevant for many peers. There exists correlation between relevance and providing good information. However the fact that a peer has many friends does not imply that it delivers information of high quality.

The fourth requirement, resistance against attacks, is important because malicious peers should not be able to trick other peers into believing the malicious peer can provide good moderations without making a social effort.

The fifth requirement is important because people do not like intrusive behavior of computer programs. By keeping SocioCast simple while improving the quality of moderations we aim to please the user.

The last requirement, giving preference to propagating good moderations over bad moderations is important because bad moderations will clog available storage space and network bandwidth when exchanged in abundance.

Based on these requirements our research objective is formulated as follows:

In what way can a *social network* formed by the friendships of users of the *Tribler* client contribute to the *quality of moderations* of *.torrents*?

1.3 Report Structure

- In chapter 2 we will first describe several trust and reputation methods that might help discern good from bad moderations. These methods and more have already been discussed in our previous report[20].
- Based on these methods we will describe the design for SocioCast in chapter 3.
- To test the performance of SocioCast we have looked at various algorithms to generate network structures which will be compared in chapter 4.
- Using networks generated by one of these algorithms we have looked at the performance of SocioCast in chapter 5.
- The resistance of SocioCast against attacks, will be treated in chapter 6.
- Based on the outcome of these models we have designed an implementation for SocioCast in chapter 7.
- We will draw our final conclusions in chapter 8.

Chapter 2

Related Work

A social network is a specific type of Multi Agent Network (MAS). Each agent in such a network has its own goals which it tries to achieve. These goals can usually be described as obtaining specific resources. Agents in a MAS are the same as peers in a P2P network. In P2P networks resources can be several things ranging from video content files to moderations. These resources come in various levels of quality, but providing resources of good quality is usually not one of the goals of agents.

Altruistic peers will always try to provide the best resources. Agents that make a strategic choice when deciding whether to provide good or bad resources need an incentive. In this thesis we try to give moderators incentive to provide good resources.

We will first explain several terms that will be used in this chapter to explain the incentivisation methods.

Sybil Attack A single agent introduces many helper agents into the network under different pseudonyms.

Collusion Multiple agents cooperate in a group with a single goal.

Whitewashing An agent reconnects to the network under a different pseudonym, clearing its track record.

Traitor An agent that acted altruistic before radically changes its behavior and exploits its reputation by influencing other agents.

In this chapter we will first look at trust based methods and after that several reputation based methods will be discussed.

2.1 Trust

An incentive for agents to provide services is trust that another agent will do good in the future. People forge friendships with each other which means that they

	a_1 is silent	a_1 betrays
a_2 is silent	R_2, R_1	S_2, T_1
a_2 betrays	T_2, S_1	P_2, P_1

Table 2.1: Prisoner’s dilemma

will help each other out and that they will not deliberately harm each other. For example, you need to remove a large tree from your garden and it is too heavy to lift by yourself. Your friends are probably willing to help you carry it out of your garden, because they trust you and they might call upon you later to fix their car. The idea of trust in a MAS is like friendship. Agents that trust each other provide resources to each other because they trust that some day that other agent will repay his debt. Trust can be a very good incentive for agents to act altruistic, but agents must first learn how to start a trustful relationship with each other. According to Stranders[17] trust can be defined as:

Definition 2.1.1. [*Trust*] *The expectation that one will not be deceived when relying on an entity one does not control completely.*

When an agent enters a MAS it knows no other agent. A trust-based method that provides incentives should have some sort of guideline to start resource exchange with unknown agents. It should also have an algorithm that determines the trust metric based on an interaction history. Lastly an agent needs a decision function to determine whether the trust metric is high enough to warrant another transaction.

In this section we will first discuss the prisoners dilemma, the basis for trust between agents. For a single non-repetitive interaction between peers it is the best strategy to betray each other. Trust systems can give agents incentive to cooperate in the case of repeated interaction.

The first trust system discussed in this section is tit-for-tat, the basis for the Bit-torrent protocol. The final method, correlated opinions, bases trust on the overlap of taste between peers.

2.1.1 Prisoners Dilemma

The outcome of an interaction with an unknown agent can be modeled by the *prisoner’s dilemma*, which can be explained as follows. Two prisoners, a_1 and a_2 , are caught doing a crime and their jail time depends on whether they betray each other or not. When both prisoners remain silent they get the *reward*. However when one betrays the other the betrayer gets the *temptation* and the other is played for a *sucker*. When both betray each other they are both *punished*.

Let R_i be the event that agent a_i gets a reward, T_i the event that agent a_i receives the temptation, S_i the event that agent a_i is played for a sucker and P_i that agent a_i is punished.

The jail time they get is based on the outcome of the statements of both prisoners and the duration is ordered as following $T_i < R_i < P_i < S_i$. Even though the combined jail time is shortest when they both keep silent, it is most beneficial to betray each other. When a_2 remains silent it is most rewarding for a_1 to betray him ($T_i < R_i$). Even if a_2 betrays a_1 should betray as well ($P_i < S_i$). In both cases a prisoner is better off betraying the other. Things get more interesting when agents interact with each other more and strike a deal to both keep silent because in the end $2 * R_i < T_i + S_i < 2 * P_i$.

2.1.2 Tit-for-tat

A fairly recent and very popular P2P file exchange system is Bittorrent[5] which employs the tit-for-tat strategy for the repeated prisoner dilemma. In each round both agents have to decide whether they give a resource to the other. Keeping silent in the prisoner dilemma is the same as providing a resource and betraying the other is the same as not providing a resource. When agents do not know each other nor trust each other the most rewarding strategy is to betray. When every agent does this the trend will become that a lot of agents will also betray and resource exchanges will come to a grinding halt.

To promote resource exchange tit-for-tat is introduced. In this scheme agents will always choose to give the resource in the first round. In the second round the tit-for-tat player will do what the opponent did in the round before. When a tit-for-tat agent plays against an agent that plays unfair it will not receive a resource in the first round. From then on it will do as the unfair agent did in the previous round and will thus in total it will lose only one resource. When it plays against a tit-for-tat agent however each agent will provide a resource in the first round and in the second round they do as the opponent did, which means they will cooperate with each other indefinitely. This means that an agent will lose at most one resource to unknown agents, but it will never be betrayed by other agents that employ tit-for-tat.

Agents select other agents in the network at random for resource exchange. To allow newcomers to join the network each agent reserves 20% of its upload bandwidth to establish connections with unknown agents. This also allows agents to discover agents that perform better than agents that it is connected to at the moment. However, there exist a modification of the bittorrent client that exploits this part of the protocol to augment download performance. Bittyrant agents use less than 20% of their upload bandwidth to connect newcomers to the network when it downloads at a reasonable rate already. Bitthief is even more malicious, it abuses this altruism towards newcomers by reannouncing that it is a new peer every round. By doing this it takes advantage of tit-for-tat agents that always provide a resource in the first round.

Tit-for-tat works very well for data exchange in torrent networks. It is however insufficient for providing trust in the quality of moderations of other peers. There

is no clear exchange between peers of metadata nor is there an objective method to decide of what quality received moderations are.

2.1.3 Correlated Opinions

Walsh et al. propose Credence[21, 22] which allows agents to calculate a value for trust in unknown agents based on the correlation of opinions. Credence is a system designed to order resource search results in the LimeWire Gnutella client by their quality rating, which is computed from votes issued by other agents. Trust between agents is defined as the correlation of their voting behavior, their opinions. Votes are propagated by a gossip protocol supported by query flooding, but a Distributed Hash Table (DHT) could be used as well.

Votes consist of a reference to a resource and a claim concerning metadata using set operators. For example, the vote $\langle \mathcal{R} : madonna \subseteq name, mp3 = type \rangle$ claims that *madonna* is a valid name and *mp3* is the type of resource \mathcal{R} . The claim $\langle \mathcal{R} : mp3 \notin type \rangle$ is in contradiction with the preceding vote because it states that \mathcal{R} is not of type *mp3*. Resources that are identified as completely unwanted can be described with the vote $\langle \mathcal{R} : name = \emptyset \rangle$ which refutes any other claim concerning the name of \mathcal{R} .

The trust agent \mathcal{A} places in votes cast by agent \mathcal{B} is computed using a slightly modified Phi coefficient as follows. Let a and b be the fraction of votes where \mathcal{A} respectively \mathcal{B} voted positive and p be the fraction where both agents agree with both vote having positive intentions. Then θ is the coefficient of correlation, taking on values in the range $[-1, 1]$. When insufficient voting history for an agent is available agent \mathcal{A} picks an arbitrarily low value of trust.

$$\theta = \frac{p - ab}{\sqrt{a(1-a)b(1-b)}} \quad (2.1)$$

Agents discover transitive correlations by building a model of the P2P network. Every agent keeps track of a list of correlation coefficients of known agents. These values are propagated through the network using a gossip protocol. By building a model of the P2P network an agent can discover agents to which it is transitively correlated. It will then exchange votes with these transitively correlated agents to see whether a direct correlation can be found.

Currently a similar system is in place in the Tribler software called TasteBuddies. Each peer propagates which files it has downloaded and by calculating the overlap of these lists buddies are found. Though TasteBuddies is used for recommendations, not to distinguish good from bad moderations.

The problem with this system is that agents do not have a direct relation with each other and can not reciprocate. The trust an agent places in another agent is not based on interaction between the two agents. An agent can easily fabricate votes or copy votes from other agents. This means that an agent can fabricate a high trust level by copying generally held opinions. This fabricated high level of

trust can then be used to advertise their own resources or discredit other resources. Another weak point is the amount of user interaction Credence requires to be able to calculate correlation with other peers.

The main problem with trust systems is that agents may not be able to find a resource exchange partner. For example, agent A wants a resource from agent B but not the other way around. Agent B has no trust incentive to provide services to A and therefore agent A is dependent on B to be altruistic and to have free resources to spare.

2.2 Reputation

When an agent is active in a P2P network other agents will learn about the behavior of that agent and determine to what degree they trust it. The experience of all these other agents is a distributed track record. These values for trust can be combined into a single value, the reputation of an agent. Reputation is defined by the Oxford dictionary as following:

1. the beliefs or opinions that are generally held about someone or something.
2. a widespread belief that someone or something has a particular characteristic.

Reputation thus not only describes trustworthiness of something or somebody, it also describes particular characteristics and opinions. However, to predict the outcome of transactions with unknown agents only the trustworthiness is of importance. Therefore we defined[20] reputation building upon the definition for trust (2.1.1) as following:

Definition 2.2.1. [*Reputation*] *Trust that one should place in a dynamic entity according to entities one does not control.*

The problem lies in that an agent can not blindly trust the opinions of other agents; without good incentives agents will lie about other agents for various reasons. For example, an agent can discredit competitors to better its own position or an agent can dishonestly recommend its cheating friends.

There is a thin line between trust and reputation systems and this is why the word *dynamic* was added to the definition of reputation. Consider the following two situations:

- Agent A decides whether to trust an unknown resource r depending on the opinions and trustworthiness of agents B and C .
- Agent A decides whether to trust an unknown agent \mathcal{R} depending on the opinions and trustworthiness of agents B and C .

These seemingly similar scenes differ in one important point, the quality of a resource is fixed while the quality of an agent is dynamic. A resource does not change over time and therefore neither the trust of rational agents in it. However, an agent can change their strategy towards other agents over time. While the quality of agents is represented by a trustworthiness notion, the quality of a resource is a fact. Therefore the first scenario is a trust system, while the second scenario is a reputation system.

2.2.1 Majorities rule

The idea that the majority must be right is quite appealing. Democracy which is based on the same principle is the most widely used government form. XRep[6] is a system based on this principle, it allows agents to collect the public opinion about other agents and resources through polling. XRep is built as a reputation extension to Gnutella and uses a flooding communication model to collect trust information. The general idea is to let every agent vote and base resource and agent selection on the amount of votes received. Before explaining how XRep works Gnutella will be treated in more detail.

Gnutella is a pure P2P network, meaning that the network of peers operate without the need for a central server. All peers connect to a predefined amount of peers and all messages a peer sends are routed through these neighbors. The Gnutella protocol comprises of only 5 message types: PING and PONG facilitate peer discovery, QUERY and QUERYHIT facilitate resource discovery and PUSH messages are used when normal resource retrieval fails. PING and QUERY messages are flooded through the immediate neighbors spreading out to neighbors of neighbors and further; PONG and QUERYHIT messages are sent directly back to to originating agent to indicate agent and resource discovery.

XRep adds the message types POLL, POLLREPLY, TRUEVOTE and TRUEVOTEREPLY and every agent generates a public/private key generation upon entering the network. Peers can ask the opinion of other agents in the network about a resource or an agent. Note that a majority poll about a resource is a trust mechanism and that a majority poll about an agent is a reputation mechanism. Such a poll is initiated by flooding a POLL message into the network. Let \mathcal{A} be the agent that wants to poll the network. The POLL message includes the public key of \mathcal{A} and a list of resources and peers. Every peer that receives the POLL messages sends a POLLREPLY back containing the votes it cast encrypted by the public key of \mathcal{A} . This encryption prevents tampering with votes and preserves the confidentiality of votes. When enough POLLREPLY messages have been received \mathcal{A} starts to evaluate the votes. Based on the assumption that colluding agents are usually operated from a small subnet of ip addresses, only one vote is considered from each subnet.

From the resulting set of remaining votes \mathcal{A} randomly selects peers to check the authenticity of their vote. \mathcal{A} sends a TRUEVOTE message containing their vote to

these randomly selected peers, who in turn send a TRUEVOTEREPLY back. The purpose of this exchange of messages is to check whether the vote received was actually sent by that peer. Agents on the internet are able to forge messages with an arbitrary source ip address. When the authenticity of the audited peers has been noted, \mathcal{A} can select resources and peers according to majority vote. After having interacted with the selected peers and resources it takes note of their quality to be able to reply with better opinionated POLLREPLY messages to other agent.

Even though majority voting is based on the well proved government form democracy it is inadequate for MAS. Collusion and Sybil detection in XRep is solely based on the conviction that colluding agents operate from a subnet, which discriminates against virtuous companies and university dorms operating from a single subnet. XRep is also vulnerable to whitewashing, agents with a bad reputation are able to rejoin the Gnutella network without any repercussions. The biggest disadvantage of majority voting is that very well opinionated and trustworthy agents have the same voting power as ignorant agents. Several ways of weighing opinions will be looked at in the next subsection.

2.2.2 Weighted opinions

The idea behind the methods in this subsection is that trust is transitive. This means that when an agent \mathcal{A} completely trusts agent \mathcal{B} , and agent \mathcal{B} completely trusts agent \mathcal{C} , agent \mathcal{A} can completely trust agent \mathcal{C} . There are however different methods to determine the reputation of \mathcal{C} from the perspective of \mathcal{A} . First a method based on probability theory is discussed where values of trust are multiplied to calculate a value for reputation. After that a method based on the max-flow algorithm is discussed.

EigenTrust

EigenTrust[11] is a method designed to determine which peers inject bad files into the network based on trustpaths. In EigenTrust agents calculate a value for trust according to the ratio of satisfactory and unsatisfactory transactions with other agents. It can however be used for any other notion of trust and reputation. The value c_{ij} is the amount of trust agent i places in agent j . These values are then normalized to make sure that malicious agents can not assign arbitrarily high or low values to skew the results. These values, c_{jk} , are then propagated and weighted by value of trust, c_{ij} , the receiving agent i places in the agent that provided it, agent j .

$$t_{ik} = \sum_j c_{ij}c_{jk} \quad (2.2)$$

The resulting value t_{ik} is the reputation of agent k in the perception of agent i based on the opinion of others weighted by his opinion of those others. This first value is actually the result of a majority vote method scaled by the trustworthiness

of the voters. This step can also be described in matrix notation when C is the matrix $[c_{ij}]$.

$$\vec{t}_i = (C^T)^2 \vec{c}_i \quad (2.3)$$

The vector \vec{t}_i contains values of trust for all agents based on the weighted opinions of everyone about each agent weighted by the trust agent i places in them. This aggregation step can be done multiple times producing the following vector for n steps.

$$\vec{t} = (C^T)^n \vec{c}_i \quad (2.4)$$

When this step is done often enough the vector \vec{t} will converge to the same vector for every agent, the left principal eigenvector of C . This final vector \vec{t} contains the global values of trust, the reputation of each peer. This method is however susceptible to collusion and therefore pre-trusted agent are introduced. This results in a web of trust around these pre-trusted agents excluding colluding agents, because those agents are not trusted by any agent transitively trusted by pre-trusted agents.

In the actual implementation of EigenTrust each agent is assigned to several agents as a score manager. As a score manager an agent will take care of calculating the reputation for other agents. Score managers are necessary because convergence of reputation values is dependent on where the algorithm is started. When agent \mathcal{A} is connected through a long string with agent \mathcal{B} convergence will take many steps. Another reason is that agents are capable to commit fraud when they administer their own reputation. To further prevent score managers from cheating it is crucial that they are not able to pick the agents they audit.

A score manager first requests the local values of trust of the agent it audits, the daughter, and stores these in the vector t_d^0 . It will then annotate which agents have trust information about its daughter in a vector B_d and which agents the daughter has trust information about in A_d . Every agent knows or receives the vector p_d which contains trust values pertaining to the pre-trusted agents in the network and a scalar a that indicates the influence of the pre-trusted agents.

The algorithm works in rounds indicated by k . In each round every score manager of agent i forwards $c_{id}t_i^{k-1}$ to the score managers of the agents in B_i and thus each score manager for agent d will receive $c_{id}t_i^{k-1}$ from the score managers of the agents in A_d .

$$t_d^{(k+1)} = (1 - a)(c_{1d}t_1^{(k)} + c_{2d}t_2^{(k)} + \dots + c_{nd}t_n^{(k)}) + ap_d \quad (2.5)$$

The resulting vector $t_d^{(k+1)}$ is then used in the next round until the result has converged far enough.

$$\left| t_d^{(k+1)} - t_d^{(k)} \right| < \epsilon \quad (2.6)$$

Malfunctioning agents will never get a high reputation value, because already in the first round these agents will not receive any reputation from other agents. The pre-trusted agents make sure that all agents are connected to each other through a trust path containing these pre-trusted agents. Colluding agents do not provide resources and are assigned very low trust values by agents not in the colluding swarm. In the end their reputation will converge to 0. EigenTrust is however vulnerable to agents that provide good service and collude. This creates a trust path from well performing agents to agents in a colluding network, boosting the ratings of all the agents within it. Also, EigenTrust does not provide protection against agents that whitewash their history and return as a new user.

Max-flow

Feldman et al[8] introduced a reciprocative decision function to calculate values of trust. This reciprocative decision function depends on some definitions. The *generosity* of agent j is calculated according to what j has provided (p_j) divided by what it has consumed (c_j). Generosity can therefore not be applied in systems where trust is not based on resource consumption.

$$g(j) = p_j/c_j \quad (2.7)$$

The outcome of this generosity function is then used to calculate the *normalized generosity* of agents. The generosity of each agent is divided to the generosity of the agent itself.

$$ng_j(i) = g(i)/g(j) \quad (2.8)$$

When the normalized generosity for an agent is higher than 1 that agent is more altruistic and has been more generous. This means that peers will consider interaction with lowly reputed agents when it is itself lowly reputed. To make this method more scalable these values for generosity can be made available to other agents by storing them in a DHT. A shared history should be used with precautions because it is vulnerable to collusion and other malpractice.

Therefore the idea of normalizing generosity is applied to a distributed and collusion proof method based on the maxflow algorithm. Agents keep track of the amount of successful services provided by other agents. It is assumed that an agent can trust the opinions of agent that have provided many services in the past and this trust is transitive. The maxflow algorithm finds the path with the largest capacity in a flow network. Agents are vertices in the flow network and the capacity of edges are formed by the amount of service agents have received from other agents. Unfortunately the maxflow algorithm has $\mathcal{O}(V^3)$ complexity and therefore a heuristic is used that runs in constant time at the cost of not always finding a flow when it does exist.

$$c_{ij} = \min \left(\frac{\text{maxflow}(j \text{ to } i)}{\text{maxflow}(i \text{ to } j)}, 1 \right) \quad (2.9)$$

When a reputation value is 1 more resources have been provided towards the inquiring agent than the other way around. Colluding networks that do not provide services are therefore never chosen for the maxflow path because they choose not to provide services. The usage of maxflow has however the same weakness as EigenTrust: agents that perform good, but collude as well.

2.3 Conclusion

In a MAS system agents can choose to behave altruistic or not. When there is no repercussion to acting in a selfish way the most profitable way to act is the selfish way. Therefore a MAS needs incentive methods to make it more worthwhile for agents to act altruistic. In this section we have looked at incentivisation methods based on trust and reputation.

We have discussed two trust-based methods, tit-for-tat and correlated opinions. Tit-for-tat is based on the exchange of services between agents, which is not very common in the setting of moderation exchanges. Since providing moderations is a mostly one way exchange there is little chance for reciprocation.

The second trust method is based on the correlation of taste between agents. The problem with this method is that trust is based on wrong information. Agents that lie about their preferences can easily trick other agents into believing their opinion about something.

Reputation systems are discussed due to the limitation of trust-based systems that repeated interaction between the agents is required to build up trust. In reputation systems trust information is shared among agents which enables agents that have never interacted with each other to form opinions about each other.

We first looked at majority voting which is very similar to a democracy, each agent in the network gets a vote. Votes in such a system without a central authority gives rise to Sybil attacks which are very easy to initiate in P2P networks.

Another way to calculate reputation is to add weights to the votes of agents. With eigentrust reputation is calculated as the eigenvector of the trust matrix. What makes this method hard to implement is the amount of synchronized steps and the amount of communication required.

With the last reputation method agents use the maxflow algorithm to calculate the reputation of other agents regarding bandwidth sharing. Each agent accounts for each other agent how much bandwidth it has spent uploading and downloading to and from other peers. Using this information a flow network is generated and to get an indication of the generosity of other agents the amount of flow from and to that agent is compared.

Of all these methods we prefer maxflow because it does not require communication while calculating the reputation for an agent. It is calculated relative to

each agents making it immune to collusion. We will now discuss our design for SocioCast building upon the maxflow algorithm designed by Feldman et al.[8].

Chapter 3

SocioCast Design

We present the design of the SocioCast subsystem of Tribler. It enables peers to calculate the SRI for each peer in their social network which is a relative quality estimator for their moderations. We will first reiterate our research objective:

In what way can a *social network* formed by the friendships of users of the *Tribler* client contribute to the *quality of moderations* of *.torrents*?

The idea of this thesis is to augment ModerationCast with a reputation system that allows peers to form their opinion about torrents based on the beliefs of their friends. SocioCast builds upon the previously designed ModerationCast and the already implemented ability to add friends in Tribler. Currently these friendships are used to boost downloading speed with the 2Fast[9] protocol. 2Fast enables faster downloads by utilising the Internet bandwidth of idle online friends, they conduct the tit-for-tat protocol on their behalf.

These friendships are social ties between users and when these ties are linked together a view of the social network is created. Peers and friendships are the vertices and respective edges of the social network in Tribler. We will use this view of the social network to order peers in the network by their social relevance.

Throughout this thesis we will be talking about good and bad moderations. There is no real difference between the two, appreciation of quality differs from person to person. In general good moderations are beneficial to the user and bad moderations are either wrong or of poor quality.

SocioCast is designed as a simple addition to already existing features of Tribler. SocioCast makes the following additions to Tribler.

Approval relation Peers receive the ability to approve of moderators and propagating their opinion to their social network.

SocioCast protocol The SocioCast protocol facilitates the distribution of social network information across the Tribler network.

SRI The social relevance index ranks peers according to their relevance.

In this chapter we will first discuss the approval relation and the protocol that enables peers to build a view of the social network in Tribler. The SRI which calculates the relevance of peers will be introduced in section 3.2.

3.1 SocioCast Protocol

The SocioCast protocol serves to propagate peer relations across the network. Users have the ability to add each other as a friend. Friendship only counts as a relation for SocioCast when both users have added each other.

The other relation between users is the moderator approval. An approval signifies that a user approves of the moderations another user has created. The approvals mechanism is a simple thumbs up, an unobtrusive method of user feedback. The simplicity will hopefully make the approval relationtype more ubiquitous in the network than moderations. Having multiple approvals for a single moderation will help the popularity of good moderators.

The SocioCast subsystem will contact other peers every so often to exchange social relation information. When a peer has just joined the network the frequency of contacts will be high, but the rate of contacts will slow down. The reasoning behind it is that it helps newly connected peers discover their social network faster while keeping the available bandwidth unclogged by throttling peers that have been online a while.

During each contact a peer sends a list of all its friends and approvals to another peer and in exchange it will receive those lists from that other peer. By mainly contacting peers that are already connected in the social graph peers will expand their view of the social network.

3.2 Social Relevance Index

With SocioCast we introduce a heuristic, the *social relevance index* (SRI), to determine the relevance of peers. The relevance of peers is used for several purposes in the Tribler client.

- The SRI will help distinguish good from bad moderators. Moderators are ranked based on the relevance of peers approving their moderations.
- The SRI will decide for which moderator moderations are forwarded and stored on disk.
- The SRI will direct the social network exploration by choosing exchange partners.

According to Christianson et al.[4] trust can not be calculated transitively when only one aspect of trust is considered, a friendship relation in the case of Socio-Cast. There are several aspects of trust for peers based on several abilities, their ability to write good moderations and their ability to befriend peers that write good moderations. The quality of these abilities may be correlated but not all good moderators have good taste in friends. Therefore the SRI of peers is an *indicator* for social relevance and not an indicator for trust or reputation.

We have chosen to distinguish good moderators. We do not distinguish bad moderators, because blacklists and bad reputation can easily be circumvented by creating new identities. The goodness of moderators is based on the appreciation it receives from relevant peers, not on measurable merits. Relevance is therefore very subjective and its performance can not be measured objectively.

Subjectiveness with regard to moderations is a benefit. EigenTrust formulates a global opinion distilled from the average of opinions weighted by trustworthiness. However there may not exist a unique global opinion. Each and every user is unique and therefore peers in these networks should suggest moderations on personal taste. Moderations suggestions are therefore tailored for each and every peer.

Because there is no real distinction between good and bad, there is no specific *threshold* that determines when a moderation is spam. The SRI can only be used to give an indication of the relative quality of moderations compared to other moderations in the system.

We will now discuss two versions of the SRI. The beauty of this index is that it depends on readily available information and does not require interaction with other peers during calculation of relevances. Therefore each peer can in theory implement its own version of the SRI. In the following sections we will discuss two designs for the SRI.

3.2.1 Breadth First Search

The first implementation is based on the *shortest path* between peers. This logic is based on the idea of transitive trust: the trust one places in a friend of a friend is based on the trust one places in the friend that defines the two relations in the social path. Since relationships are either existant or nonexistant the trust represented by these relationships is binary.

Trust can be described as the chance the other party will succesfully cooperate. For example peer a has experience with peer b and peer b has had experience with peer c . The trust peer a places in peer b is defined as t_{ab} , which is the chance peer b will cooperate with other peers according to previous experiences. If t_{bc} is calculated in a similar way, the transitive trust peer a places in peer c is calculated as the chance that both t_{ab} and t_{bc} happen:

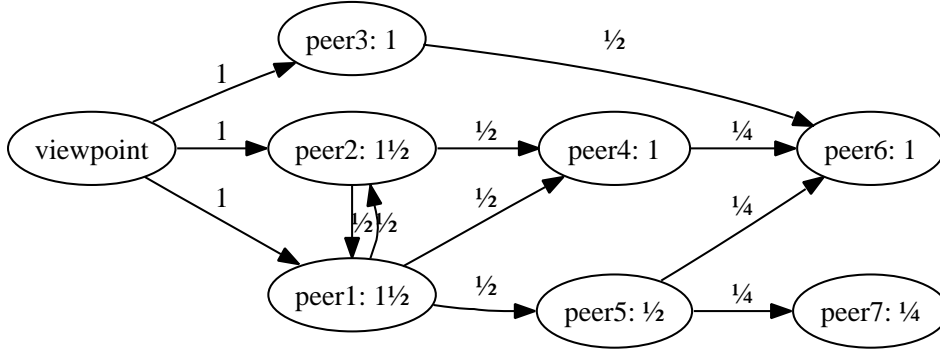


Figure 3.1: Example of SRI flowing through a social network. The capacity of relations is noted alongside the edges. The MAX-FLOW of peers is indicated on the vertices.

$$t_{ac} = t_{ab} \cdot t_{bc} \quad (3.1)$$

This same principle is used in reputation systems like EigenTrust[11] and PGP[?]. When a friendship relationship between peers in Tribler implies the chance for cooperation is 1, any peer in the social network is equally trustworthy. This seems improbable and when for example the trust peers place in their friends is 0.9 trust will diminish deeper down the social network.

The requirement this places on the calculation of the SRI is that peers found deeper in a social graph should have a lower SRI. The SRI of all peers in the social network can then be determined in a singleBFS (breadth first search) sweep through the social relationships in the local cache. The most simple algorithm for SRI is to credit peers with relevance according to its depth in the social network. The SRI for peer n is calculated as follows where $depth_n$ is the shortest path to peer n :

$$sri_n = \frac{1}{2^{depth_n-1}} \quad (3.2)$$

3.2.2 Max-flow

We designed a version of the SRI based on MAX-FLOW algorithm designed by Feldman et al[8]. to give more weight to popular peers. Highly connected peers are more popular than other peers, but not all their connections should count towards the trust score they receive. MAX-FLOW only considers connections that are linked to the originating peer. All possible paths to the target peer are found and the maximum flow is calculated to signify social relevance of that peer.

The flow through each link is bound by the capacity of that link. The capacity of the link between peers i and peer j is calculated as follows:

$$c_{ij} = \frac{1}{n^{depth_i}} \quad (3.3)$$

The variable $depth_i$ indicates the length of the shortest path from the viewpoint peer to peer i . An example of how trust flows through the edges of a social network can be seen in figure 3.1 where the variable n is 2. By decreasing n preference is given to the connectedness of peers and by increasing n preference is given to peers with short social paths. The optimal value of this parameter should be established with an empirical study of an actual SocioCast network.

3.2.3 Bootstrapping

Both algorithms to determine the SRI are dependent on having a connected social network to work with. When this is not available or when no path to an appropriate approval can be found other measures should be taken.

BarterCast[2] is a subsystem of Tribler that does bandwidth accounting. In the current version of Tribler (4.1.9) it uses an epidemic protocol to aggregate how much each peer has uploaded and downloaded. In future versions it will work similar to the algorithm by Feldman[8] et al. discussed before in section 2.2.2. The MAX-FLOW of bandwidth from peers is an indication of their altruism and is an indication that they probably provide good approvals.

Through the BuddyCast protocol peers receive information about what other peers have downloaded. Using this information the BuddyCast subsystem calculates taste buddies. These taste buddies are active peers and have sacrificed bandwidth to share their taste. Such a sacrifice is a reason to trust approvals of such peers to some degree and therefore taste overlap could be used as weak relations in the social network.

A last resort is to bootstrap new peers into the system using a network of pre-trusted peers which was proposed as part of the EigenTrust[11] system. We strongly discourage any use of centralized systems and therefore using pre-trusted peers to bootstrap peers should be dismissed.

The goal of this thesis is however to see if there is a way to infer trust in peers based on social connections. Therefore we will not look into the problem of bootstrapping socially unconnected peers into the Tribler network.

3.3 Conclusion

In this section we explained how the SocioCast protocol works. Upon entering the Tribler network peers exchange lists of friends and approvals. By combining received social information a view can be created of the social network.

We have introduced two versions of the SRI, which gives scores to peers in the network. These scores are calculated relatively from each peer based on the information that is available on the social network.

- The BFS variant of the SRI is designed with the idea that the trust someone places is based on the length of the shortest path
- The second variant of the SRI based on the MAX-FLOW algorithm gives higher scores to peers that are highly connected from the viewpoint peer.

Finally we discussed several methods to bootstrap the SocioCast subsystem of socially unconnected peers. We concluded that we will not look further into the problem of bootstrapping because the goal of this thesis is to determine how social relations can contribute to the availability of higher quality moderations.

Chapter 4

Constructing the SocioCast Model

In order to test the validity of our SocioCast design we have implemented a simulation environment for social networks. We need representative networks to test our model of SocioCast. This section examines the properties of real social networks. We use this information to compare several algorithms that generate social networks.

4.1 Social Network Analysis

Most social networks have properties in common. The statements below are shown by the outcome of an analysis by Mislove et al.[12] on the Flickr, Orkut, Youtube and Livejournal social networks, unless otherwise cited.

Relations between people in social networks are very symmetric, friendships between people are usually both ways. Another property is that the outdegree has a great correlation with the indegree, people that receive many friendship requests also befriend many friends themselves. These effects are due to the tendency of people to reciprocate friendship requests from other users.

The amount of friends people make in a social network varies a lot. According to the research by Mislove et al.[12] the link distribution of users on the Flickr, Youtube and LiveJournal networks could be fitted well by a power law. This means there are very few users with very many friends while most users have only few friends. Most social network operators have instituted a cap to the amount of links allowed per user which affects the most connected people and causes a less good fit with a power law distribution.

On the other hand, a study by Pennock et al.[13] shows that the distribution of links shows a better fit with a normal distribution when links in specific category of interest are considered.

The average shortest path length between people in all four social networks is about 5 which is partly due to their power law distribution. The other aspect of these networks that causes this is the *scale-free* behavior. A network is scale-free when highly connected people have a tendency to befriend other highly connected people more frequently than normal people. This forms a highly clustered sub-network in the social network consisting of highly connected peers that act as a shortcut for social paths in the network.

Not only are highly connected people densely clustered, the entire social network has a high clustering coefficient. This can be explained by the nature of social networks. Friends of a person are more likely than normal to also know each other or even be friends. This property for social networks is called the *small-world* property.

Social networks have a *densely connected core*, removing the highly connected people will break the network in many small clusters. Removing the 10% most highly connected people is enough to break the entire network in small clusters in all networks analyzed. This can be explained by the scale-free property of these networks, highly connected peers are the main providers of connections between other peers in these networks.

We have looked at a dataset extracted from YouTube by the Tribler group. The problem of this dataset is that it is a subset which may not exhibit the properties of the entire network. According to research by Stumpf et al.[18] most subsamples of scale-free network do not exhibit scale-free behaviour as well. Also the number of friends per peer in the Tribler YouTube dataset distribution did not correspond to the analysis of Mislove et al.[12].

4.2 Network Generation Algorithms

For testing purposes we have made a generic implementation to create network representations. We used this implementation to test the properties of networks generated by several algorithms. We have looked at an algorithm to create *small-world* networks and another to create *scale-free* networks. These are the most important properties of social networks and therefore we will compare the networks they create.

4.2.1 Small World

To generate small-world networks we looked at the algorithm *random rewiring procedure* designed by Watts et al.[23] to create a small world social graph. A small world network is highly clustered, the friends of people have a very high

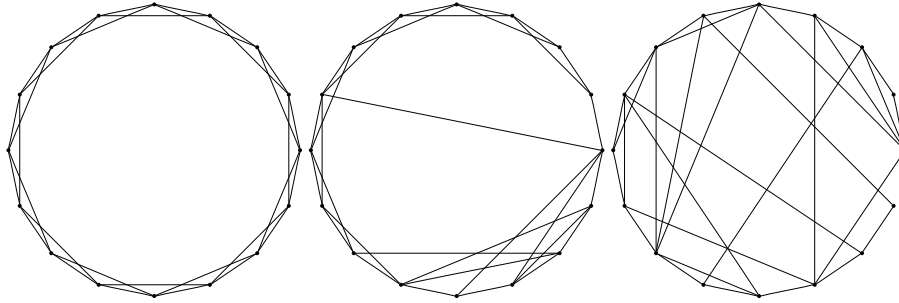


Figure 4.1: Small world networks with increasing randomness. The left network is not rewired at all with $p = 0$. The links in the second network have been rewired with $p = 0.2$, while the right network has been completely rewired with $p = 1$.

chance to also be friends of each other. Another property is the relatively low average shortest path lengths between peers.

This algorithm starts by creating a ring network consisting of n peers connected to their k nearest neighbours by undirected edges. A peer is chosen and a link in the clockwise direction is chosen and randomly reconnected with a chance p . This step is repeated by rotating clockwise around the ring until each link has been considered.

This algorithm has several drawbacks: the network is incapable of growing after creation which prevents the modeling of a dynamic network. Once the ring has been created and reconnected it is impossible to add additional peers. Also the structure of the network is not scale-free which causes the average shortest path between peers to be longer than desired.

4.2.2 Scale-Free

We looked at the *preferential attachment* algorithm by Barabasi et al.[1] that creates networks with the scale-free property.

The algorithm starts out with a fully connected graph of m peers. Peers are added to the network until the network contains n peers in total. Whenever a peer is added to the network it will receive k connections to peers already present in the network. The peer to which this newly created peer is connected is chosen using preferential attachment. The chance a peer is chosen is $p_i = c_i / \sum_j c_j$ where c_i is the connectivity of a peer and j iterates over all previously added peers.

The distribution of friends per peer in networks generated by the preferential attachment algorithm follows a power law in contrast to small-world. We plotted the distribution of friendships per peer in networks generated by the scale-free algorithm and the small-world algorithm in figure 4.2.

Unfortunately the network produced by this algorithm is not a small world net-

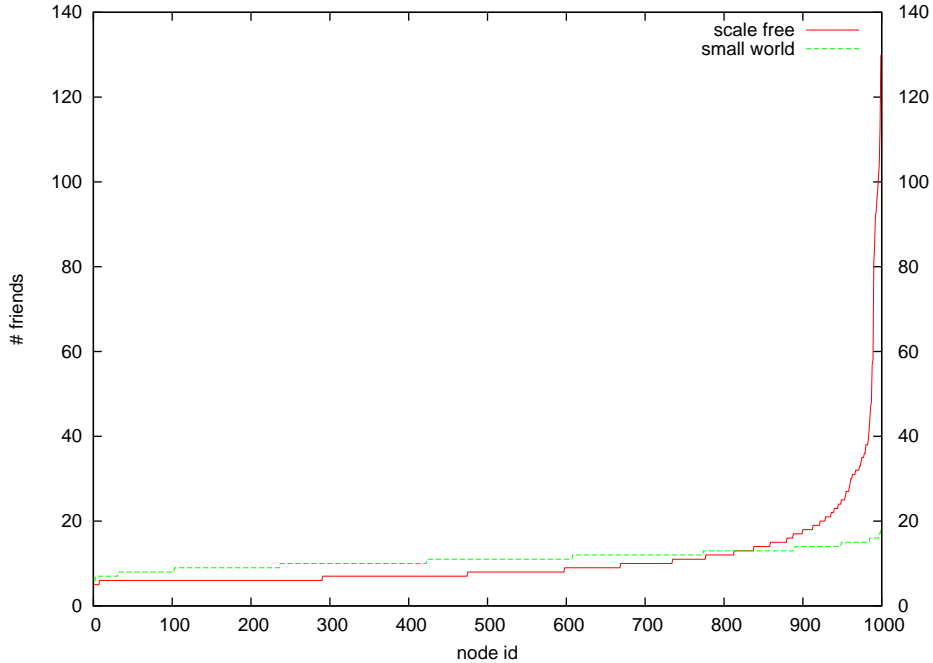


Figure 4.2: Friendship distribution in synthetic small-world and scale-free networks. Only the friendship distribution of the scale-free network can be fitted with a power law.

work, the clustering of peers is far lower than networks produced by the algorithm of Watts et al.[23]. Due to lower clustering in network generated by *preferential attachment* algorithm the MAX-FLOW version of the SRI will underestimate the relevance of close friends in comparison to highly connected peers.

4.2.3 Combination

Chakrabarti et al. designed R-MAT[3], an algorithm that can generate graphs of different types including a combination of scale-free and small-world. To generate a network of size n a matrix of size $n * n$ is created where the entry $[i, j]$ indicates peer i added peer j as a friend. The size of the network needs to be of the form 2^x . During each addition of a relation several steps are taken. For each step the matrix is divided into 4 quadrants and for each step a quadrant is chosen according to preset chances to be the matrix for the next step.

Unfortunately many peers remain unconnected due to the random nature of adding random edges which makes it hard to make a sparse network of peers. To make the network fully connected the algorithm needs to continue adding relations even though the requested amount of average number of friends has been reached, which results in too many friends per peer. Therefore we decided not to use this network generation algorithm.

4.2.4 Comparison of Peer Influence in Generated Networks

Highly connected peers are interesting for other peers because of their knowledge of the network. This knowledge can comprise of actual moderations or useful approvals of other moderators. For other peers to benefit of this knowledge the influence of these highly connected peers should be bigger than the influence of average peers.

It takes a lot of effort for users to create a highly connected peer because of the amount social interaction it requires. An easier strategy is to introduce many slightly connected peers to work together in a so called collusion network. The single peers in such a collusion network have few friends on their own, but together they might have more influence than a single highly connected peer.

In a social network the potential influence a peer can exert on another peer is determined by the social links in the shortest path between the two. Intuitively a friend of a friend is more trustworthy than a friend of a friend of a friend. The aim of this comparison is to see whether the influence of a single highly connected peer is indeed larger than the influence of a combination of many peers and what the tipping point is.

We will compare the influence of different types of peers in scale-free networks and in small-world networks. Using the algorithms discussed previously we generated a scale-free network and a small-world network both containing 25.000 peers with on average 20 social relations per peer. The amount of peers was chosen arbitrarily, the average number of friends is slightly higher than the amount of friends found in the social network claws¹ by Mislove et al.[12].

All peers in these networks are passive and social relations are stable. We use a simplified model to test our hypothesis that the potential influence of highly connected peers is higher than a colluding network. Such a collusion network of size n consists of n peers randomly selected from the tail in the friendship distribution graph, peers with few social connections.

The experiment is in effect a breadth first search from a seed of potentially multiple peers. The graph that is generated from this breadth first search is called a *hop-plot*. In figure 4.3 the discovered peers on the y-axis are cumulative. Peers discovered at depth 2 are the total of peers found at depth 1 and 2 including the seed. Since all peers in these network are connected the discovered peers will always end up at 25.000 peers.

The resulting hop-plots 4.3 and 4.4 show that on average the amount of hops to cross the network is smaller in scale-free networks than in small-world networks. However not only is the average of hops different, the influence of highly connected peers is different between the two networks.

In figure 4.3 it can be seen that in a small-world network the influence of 25

¹The average number of friends per network: 12.24 on Flickr , 16.97 on LiveJournal, 106.1 on Orkut and 4.29 on YouTube.

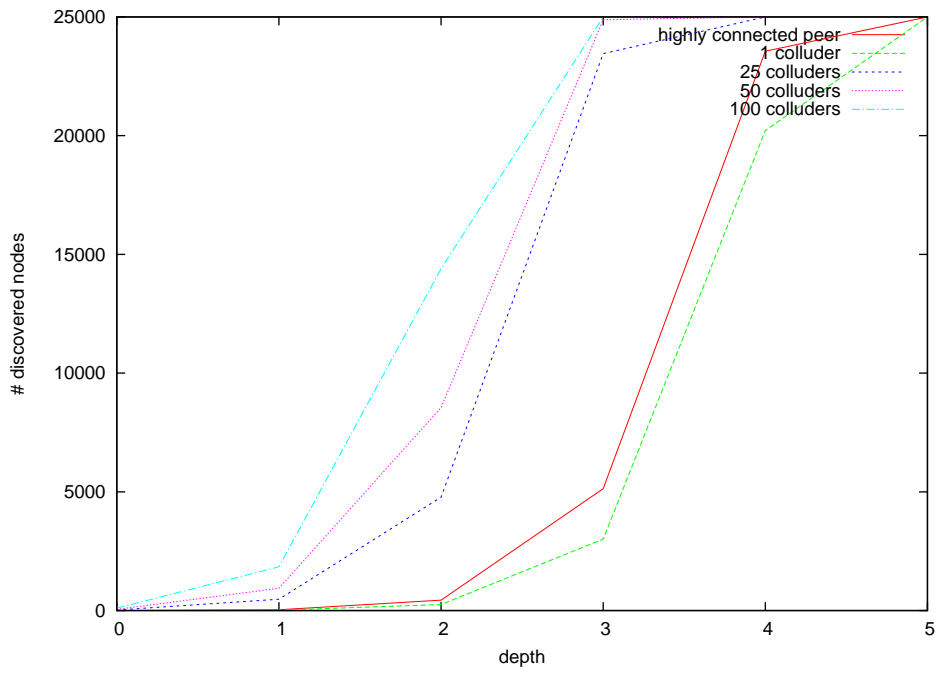


Figure 4.3: Hop-plot of a small-world network, an overview of shortest paths to the entire network from a specific seed.

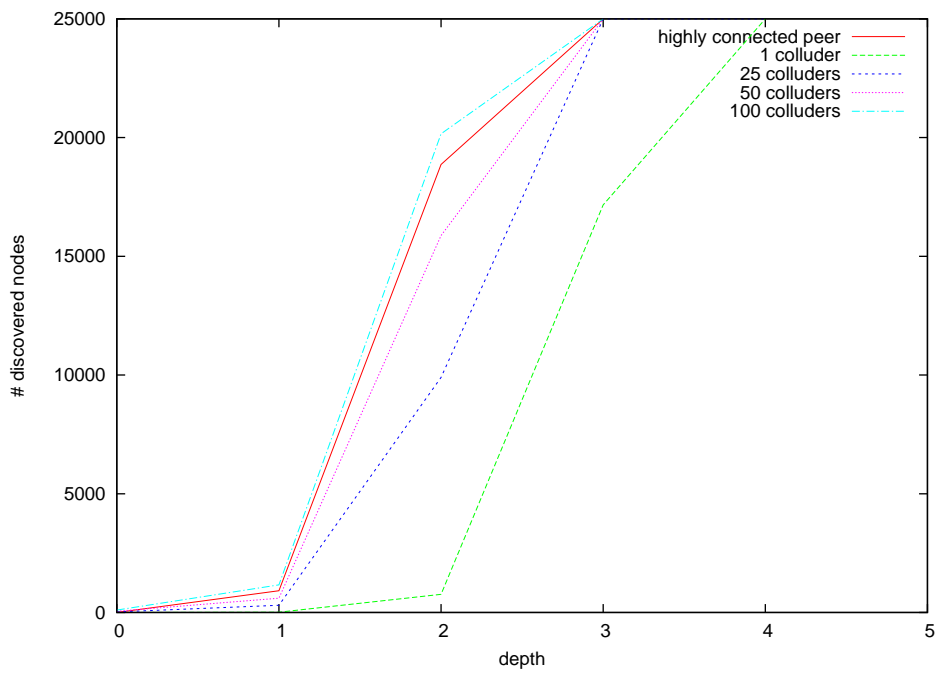


Figure 4.4: Hop-plot of a scale-free network, an overview of shortest paths to the entire network from a specific seed.

colluders and more is substantially higher than the influence of a single highly connected peer. In contrast to this figure 4.4 shows that a highly connected peer in a scale free network has far more influence on the network than a colluding peers. The influence of 100 colluders is however slightly higher than the highly connected peer, which is not bad per se. For the scale-free network the single highly connected peer has more friends than 50 colluders combined which makes the breadth first search from the highly connected faster than collusion networks of size 50 and smaller. The exact numbers do not matter as the influence of peers depends on the size of the network, what is important is that the influence of highly connected peers is far greater in scale-free networks than in small-world networks.

Real social networks are a combination of a scale-free and a small world network. The defining property of a scale free network is the far shorter average path length. Therefore it can be concluded that in real social networks the influence of highly connected peers is substantially greater than the influence of a collusion network.

However, the problem of this first explorative test is that the influence is concluded from a breadth first search from a source. This does not take into account the amount of relationships per message and the size of the cache that holds a global view of the network in each peer. Most importantly the influence of a peer is not defined by how it can reach other peers in the network, but how other peers can reach the peer in question. The score calculated by the SRI heuristic is determined from the viewpoint of other peers and that is what we will analyse in chapter 5.

4.3 Modeling Setup

To do proper simulations a generalised implementation of SocioCast was made using the Python[14] programming language. Each peer is an autonomous component capable of communicating with known peers. Time is discrete and at each point in time an active peer has a chance to connect to a peer to exchange relationship information. We made a simulation of SocioCast using the SimPy[16] library which helps create a network where all peers are active. These peers can activate themselves at any point in time to conduct relationship exchanges. Unfortunately due to simulation scaling issues with the MAX-FLOW version of the SRI we were forced to do experiments with only one active peer throughout the simulation. With only one active peer it was no longer necessary to use SimPy.

Analysis on real social networks points out that they exhibit small world and scale free properties. Due to not having a proper dataset and not being able to generate a network with both these properties we need to choose between a scale-free and a small-world network. We choose to use *scale-free* networks in our experiments, because the smaller shortest paths in these networks are more essential for

the MAX-FLOW algorithm than heavily clustered small worlds. Due to the lack of high clustering MAX-FLOW will overestimate the importance of highly connected peers. In real social networks peers with shorter paths will receive higher scores due to their clustering even though they are not as connected

A property that is hard to model is the irregular behavior of users in a real social network. Peers in real social networks come and go which results in network thrashing. It would be interesting to see how this affects the speed at which peers discover the social network. Unfortunately due to time constraints we were not able to test the effect of network thrashing on social network discovery.

Also our simulation is done on a static network where all friendships links are generated and active from the start. In the real world peers can add and possibly remove friends at any point in time. We have not looked into this because it is hard to model link dynamicity that is comparable to real world networks. Instead of guessing what would be closest to real behaviour we have chosen to keep our model static and simple for now.

The BuddyCast protocol in Tribler, responsible for peer contact information and taste, has a blocking system and a throttling system that limits the amount of messages sent in a specific timeframe. Peers will at first exchange BuddyCast messages at a high rate, but the protocol slows down after a preset online time.

We have envisioned such a system for the SocioCast implementation. We have not modeled such a throttling system, because it only affects exploration speed slightly since the effects of it are mostly on incoming messages from long time active peers which we don't model at all. Also what is most important is exploration speed and scaling behaviour of peers that have just joined the network and thus are not affected by throttling.

Another issue is the connectivity of peers. Peers behind a nat router are unable to receive messages unless they have setup their router in a specific way. Up to 50% of the peers in the current Tribler network are not connectible. This seriously affects the possible connections between peers and can cut of parts of the social network.

Our model consists of 1 active peers and incoming connections are non-existent. This means the peer in our model can not contact 50% of the peers in the network and can not exchange social information with these peers.

If all peers in our network model were active and the monitored peer is connectible the availability of social information is better. Unconnectible peers can contact the monitored peers and communication is initiated the other way around. In our model only 1 peer is active and therefore we have chosen to ignore connectibility. The effects of connectivity problems is left for future work.

4.4 Conclusion

Due to computation restraints it is very costly to do experimentations on networks with sizes comparable to real social networks. Subsets of datasets should not be used for simulations because they may exhibit different network properties than the entire network. Therefore the we will do experimentation on artificially created networks. Due to the shorter average paths of scale-free networks we choose the *preferential attachment* algorithm by Barabasi et al.[1] to generate networks for our simulation.

Chapter 5

Simulations of Speed and Scalability

We will look at the influence of different SRI algorithms on the network exploration of a new peer. After that we will see how SocioCast scales by limiting the size of the relationship cache. We will compare two different pruning algorithms for the cache. Finally we will see how well both SRI algorithms discover highly connected peers.

5.1 SRI Implementations

In section 3.2 we introduced two versions of the SRI which determine the relevance of peers. This index drives the social network exploration by choosing exchange partners and it distinguishes good from bad moderations.

In this section we introduce three separate implementations of the SRI. The first implementation for the SRI is based on a breadth first search (BFS). The second and third implementation are both based on the MAX-FLOW algorithm.

Each implementation is tested on a scale-free network consisting of 1000 peers. The average amount of friends in this network is 10 and the most connected peer has 130 friends. The SRI is calculated from the perspective of a single peer, the *viewpoint peer*. This is necessary because the SRI is always calculated relatively to each peer.

In our experiments we choose the viewpoint peer as following: we order all peers based on the amount of friends they have with the peers with the most friends towards the end. The peer at $\frac{1}{3}$ of this ordered list is chosen as the viewpoint peer. This peer is part of the tail in the distribution of friends and stands for the average peer in a social network.

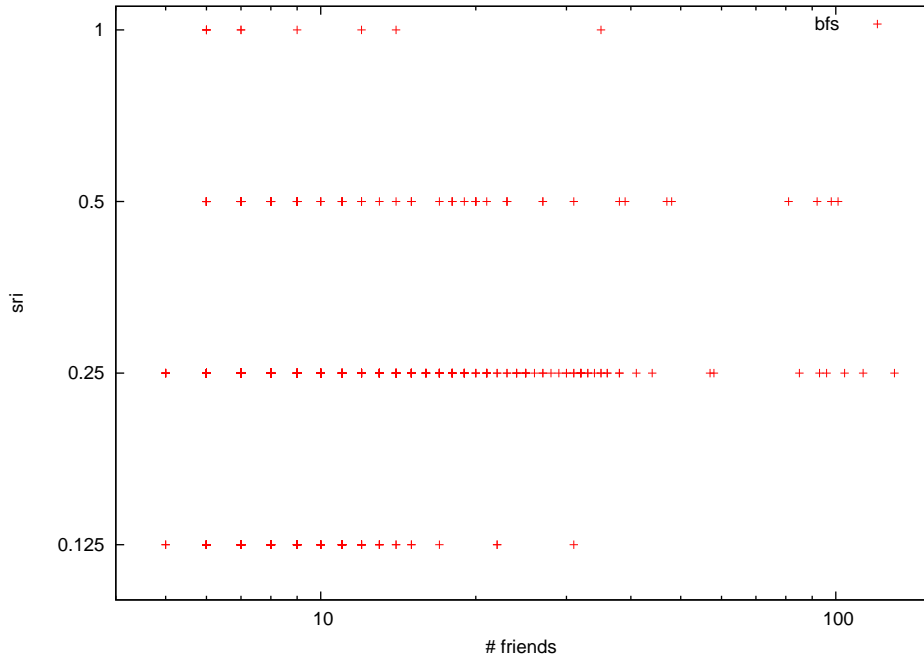


Figure 5.1: Scatterplot of the BFS SRI of peers and their number of friends. The correlation coefficient is 0.12.

5.1.1 Breadth First Search

The first implementation we tested is based on the shortest path in a single BFS sweep. The resulting scatterplot is figure 5.1 with on the x-axis the amount of friends and on the y-axis the SRI of the peer. There is no correlation between the SRI of a peer and its number of friends. The Pearson product-moment correlation coefficient is 0.12 when the SRI is determined by the shortest path to peers.

$$sri_n = 1/2^{depth_n-1} \tag{5.1}$$

The four lines in figure 5.1 correspond to the various lengths of the shortest paths. The peers with the lowest SRI, 0.125, are connected with a shortest path of length 4 to the viewpoint peer.

5.1.2 Edmonds-Karp Max-flow

Our second implementation is a MAX-FLOW version of the SRI. The MAX-FLOW is calculated using the Edmonds-Karp[7] algorithm which has complexity order $\mathcal{O}(VE^2)$ with V representing the vertices and E representing the amount of edges in the social network. Although the Edmonds-Karp algorithm is optimal for sparse graphs it is significantly slower than the BFS implementation.

To illustrate this further, the BFS implementation is of complexity order $\mathcal{O}(V + E)$ and it calculates the score for every peer in a single sweep. The complexity

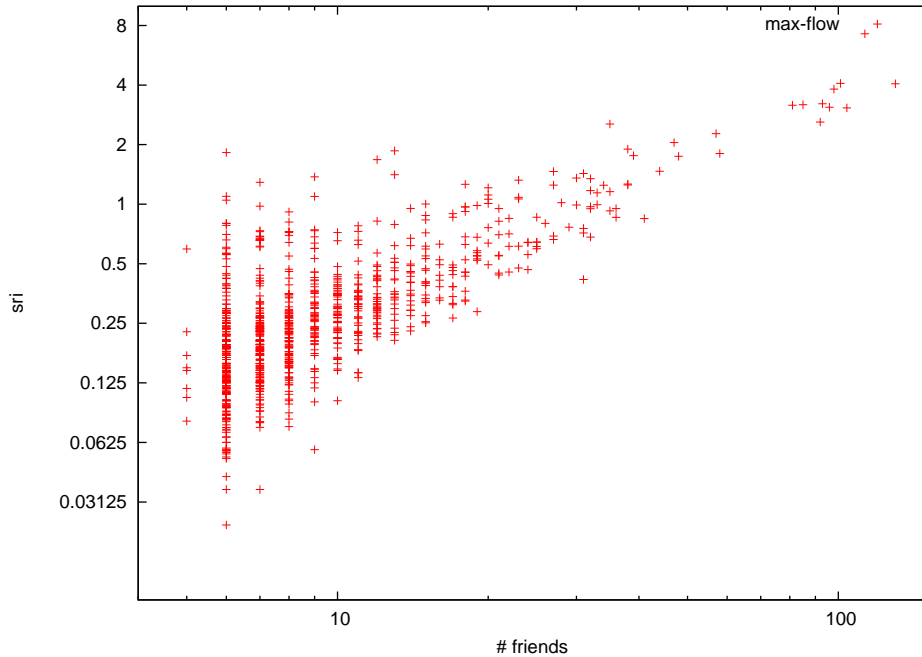


Figure 5.2: Scatterplot of MAX-FLOW SRI of peers and their number of friends. The correlation coefficient is 0.87.

order of Edmonds-Karp is much higher and it needs to calculate the SRI for each peer individually increasing the complexity to $\mathcal{O}(V^2E^2)$.

A scatterplot of the SRI and the number of friends is shown in figure 5.2. The Pearson product-moment correlation coefficient between the SRI and the number of friends of peers is 0.87, which means there is a clear correlation.

It is not clearly visible in figure 5.2, but the maximum relevance of peers is bound at the number of friends of the viewpoint peer. The maximal flow coming out of the viewpoint peer corresponds to the sum of the capacities of the links to its direct friends and these links have capacity 1. Therefore the distinction a viewpoint peer can make between highly connected peers is dependent on the amount of friends it has.

5.1.3 Max-flow Heuristic

We designed a heuristic to calculate the MAX-FLOW in a social network. It calculates the MAX-FLOW to all peers in one breadth first sweep which means it is of complexity order $\mathcal{O}(V + E)$ like the BFS SRI implementation. The capacity of links is calculated ad-hoc disregarding existing flow. This makes the heuristic of MAX-FLOW prone to exploitation by routing social paths in parallel through an

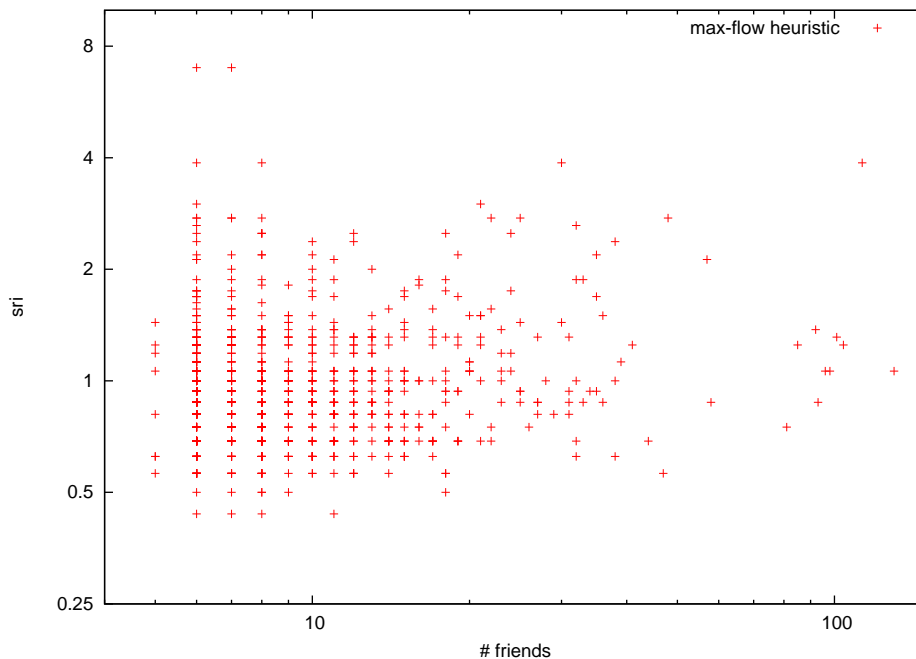


Figure 5.3: Scatterplot of the MAX-FLOW heuristic SRI of peers and their number of friends. The correlation coefficient 0.16.

entire collusion network toward one peer.

Not only does this make the algorithm prone to exploitation, it also does not work as can be seen in figure 5.3. It overestimates the SRI because it does not adhere to normal flow. The Pearson product-moment correlation coefficient is 0.16, only slightly higher than the SRI based on BFS. Therefore we decided not to use this heuristic in further experiments.

5.2 Social Network Exploration Speed

Using our model of SocioCast we will now take a look at how peers discover the social network. What is most important here is that with only limited interaction with the user a Tribler peer can create an extensive view of the social network.

Since the social network is static all friendship links are created before the first step. The viewpoint is selected as described in section 5.1 and it starts exploring the network. The viewpoint peer will only have knowledge about its own friends initially. During each step it will contact a peer which is selected using several strategies. We have tested SocioCast with two strategies to select targets: select the peer with the highest SRI and select a random peer.

The usage of MAX-FLOW is similar to the hill-climbing graph search algorithm[15].

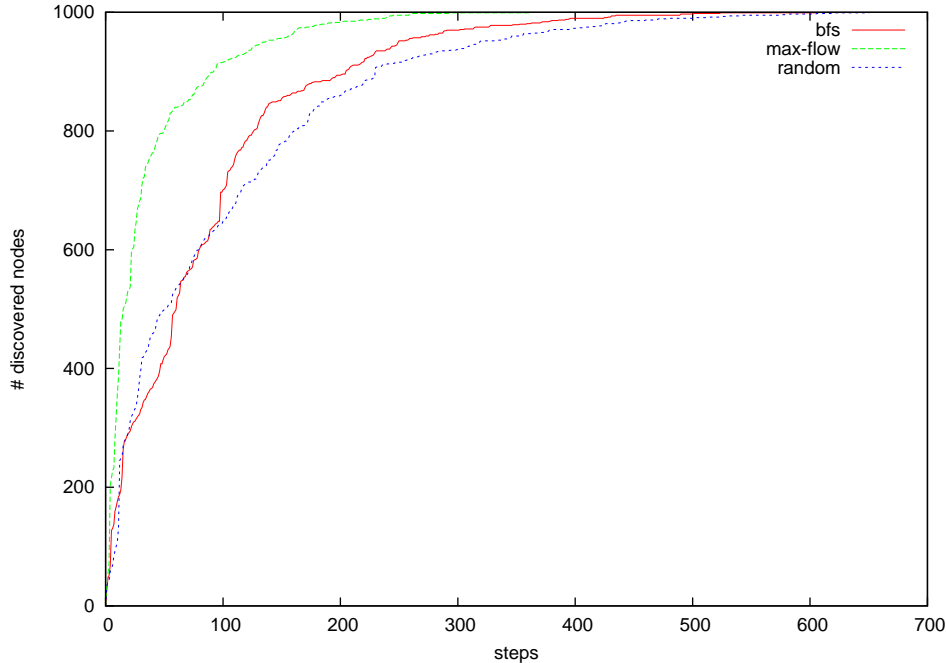


Figure 5.4: Network discovery speed using BFS and MAX-FLOW based SRI

Both algorithms are directed towards peers with a high score. The difference between the two is that our graph discovery algorithm is not a search algorithm.

We have compared the exploration speed of MAX-FLOW, BFS and random target selection strategies. To compare these strategies we created a scale-free network consisting of 1000 peers with on average 10 friends per peer.

Peers are discovered at greater speed when peers are selected using the MAX-FLOW algorithm as can be seen in figure 5.4. The number of steps taken is shown on the x-axis, which corresponds to the amount of nodes contacted at that time. The amount of nodes reached by the discovered social graph are shown on the y-axis.

This difference in performance can be explained by the way the SRI is calculated. As shown earlier in figure 5.2 the SRI of peers is correlated to the amount of friends they have. This means that when a peer selects exchange partners based on their SRI they will have a preference to highly connected peers. Because the network is of the scale-free type these highly connected peers have a high chance to be connected to other highly connected peers. Even at greater depth highly connected peers will gain preference over peers at lower depth because of their connectedness.

It took the MAX-FLOW algorithm 30 steps to discover 687 peers, which is 22.9 peers per step. After 30 steps the discovery process slows down due to overlap between peers already discovered and peers received in each step. For larger networks highly connected peers will have more friends and the MAX-FLOW algorithm

directs its exploration towards these peers. Therefore we expect that the amount of peers discovered per step increases as the network size increases. Given this increased rate of peer discovery, our social network exploration algorithm scales as the network increases.

Based on this experiment MAX-FLOW SRI is the superior algorithm because it generates a broader view of the social network quicker. For all versions end-game discovery could be sped up slightly with more push exchanges from unknown peers. Since this experiment was done with only the viewpoint peer active in initiating exchanges the real world SocioCast implementation will be faster.

5.3 Network Exploration with Restricted Message Length

By not restricting the size of SocioCast the Tribler network becomes prone to abuse. Although friendships need to be acknowledged both ways, a malicious peer can generate many fake relationships by introducing fake peers in a Sybil attack. Many fake relationships can clog the relationship cache and will consume bandwidth. We will therefore look at how the SocioCast protocol could be altered when the message size is limited. We will also discuss whether it should be limited.

The main objective of peers in SocioCast message exchanges is to gather social information and therefore peers should retain information used for gathering and not specifically for spreading. Peers could send each other bloom filters to see which relations are not yet known to the other. Such a bloom filter can also be used to check if other peers still have a relationship in their cache that has been deleted.

When the message size is fixed an extra piece of information should be sent during exchanges, the amount of social relations not yet exchanged. This allows peers know which peers have relationships that are not yet known and should be contacted at a later point in time.

Another way to limit the size of messages is to limit the maximum amount of friends peers are allowed to have which indirectly limits the maximum size of SocioCast messages. Such a limit is quite common in social networks[12], but it could affect the scaling of SocioCast by increasing the average shortest path in the network.

Message length should not be limited until abusive behaviour is detected for several reasons. Synchronization of relations using bloom filters makes exchanges more complicated and requires more messages. More importantly only sending a subset of friends hampers the speed at which peers explore the social network when highly connected peers have too many friends to fit in a SocioCast message.

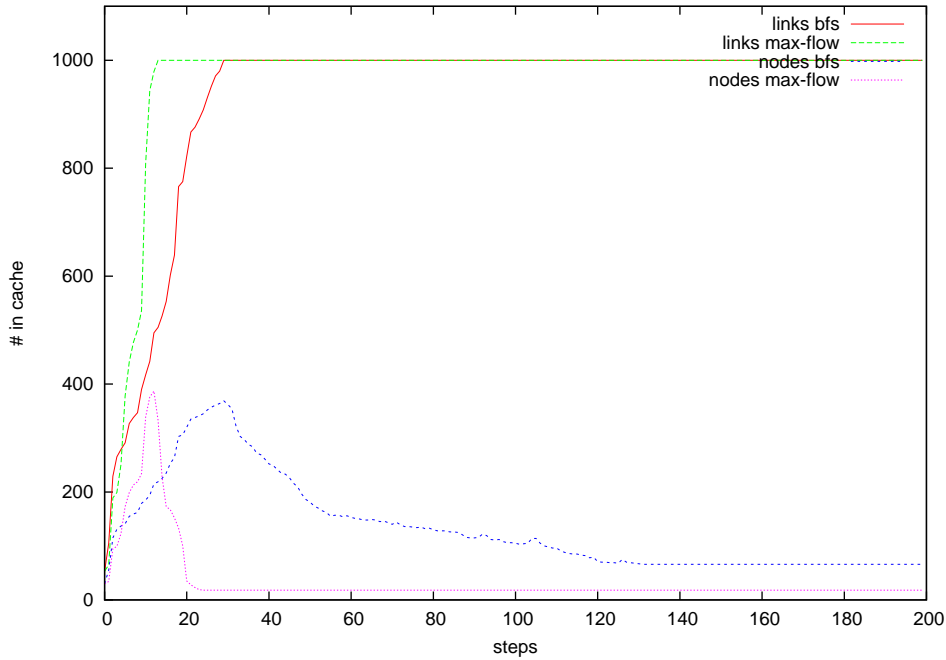


Figure 5.5: SocioCast scaling with fixed cache size

5.4 SocioCast Scalability

It is important for SocioCast to scale to larger network sizes. We will test how many peers can be discovered in a network when the cache that stores social connections has a fixed size. This is inevitable because limitations must be placed on the usage of disk space by Tribler on the computer of its users. It is important that SocioCast scales because when it only spans a part of the social network not all information is available to the user.

For this experiment a network of size 1000 was generated with on average 10 friends per peer. The viewpoint was picked as described in section 5.1 to initiate network discovery. We tested the scalability of SocioCast with SRI based on BFS and MAX-FLOW given a fixed size for the cache that stores social relations. Social relations are pruned from this cache based on the SRI of the peer they connect to.

The cache size for both tests was limited to 1000 connections, which under optimal conditions would mean a peer could discover the entire network. By using a spanning tree n vertices can be connected by n edges and thus n potential places in the cache can span a social network containing n Tribler peers. The scalability of our two SRI heuristics depends on how many cycles are formed by the way they traverse and store the social network.

The resulting graph of this experiment is shown in figure 5.5. The number of

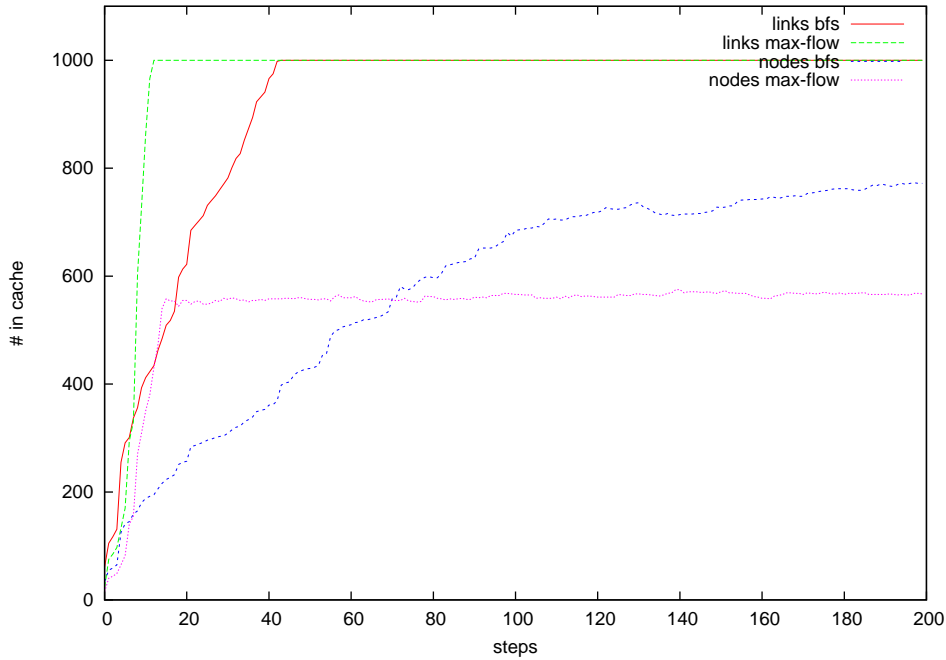


Figure 5.6: SocioCast scaling with link-weight based pruning

steps and nodes contacted is shown on the x-axis again. The amount of links in the cache and nodes reached by the social graph spanned by the links in the cache are shown on the y-axis.

Node discovery using MAX-FLOW is again much faster. However due to aggressive cache pruning the amount of nodes in the cache diminishes after the cache fills up. In step 11 the MAX-FLOW cache is filled and relations are thrown out. Connections to peers that have many connections will be stored in favor of connections to new peers due to their high SRI. This means the cache will fill up with only paths to the most interesting peers.

During the experiment with BFS the amount of peers in the cache only started to drop at step 32. This decline is due to peers with high SRI with multiple incoming paths. When there exists a peer with a path of length 2 and of length 3 it will receive an SRI of $\frac{1}{2}$, meaning its path of length 3 will prevail over all other peers at depth 3 even though it does not add any information.

The scaling of SocioCast is limited for either implementation of the SRI due to excessive pruning and storing of unnecessary link information.

5.5 SocioCast Scalability with Link-weight based Pruning

Social links should be pruned based on amount of SRI they add to peers and not on the SRI of the end peer. When the score of a peer is hardly affected by removing

a link towards it the link is superfluous. For example a highly connected peer at depth 2 with a high SRI does not gain much score from an additional separate path of length 4. For the BFS variant of the SRI only one relation needs to be stored for each peer.

The MAX-FLOW algorithm still discovers peers faster than the BFS algorithm as can be seen in figure 5.6. Both algorithm are able to keep social paths to far more peers with the link-weight based pruning algorithm. Both tests show however that discovered peers are thrown out of the cache again.

The pruning algorithm has an undesired property for the BFS SRI resulting in counterintuitive behavior. For example: when a peer has 2 connections at depth 3 of which 1 is redundant it can happen that both these connections are pruned, removing the peer from the discovered list altogether. Otherwise the BFS algorithm will create a spanning tree containing as many peers as there are links in the cache.

For MAX-FLOW the amount of peers in the cache decreases because extra connections to interesting peers brings more difference in SRI than making connections to new peer. A link to an undiscovered peer gets the same priority as an additional path to an already connected peer at the same depth. Both links provide the same difference in SRI making the choice which peer to link up arbitrary.

SocioCast scales well with the link-weight based pruning algorithm. This experiment indicates that SocioCast makes efficient use of the available cache. It has the ability to quickly and efficiently discover moderators with the most relevant moderations To further improve scaling for MAX-FLOW the pruning algorithm should give a slight priority to links that connect unconnected peers to allow more peers to be discovered.

5.6 Directed Exploration in Social Networks

Highly connected peers have more information about the network and therefore peers should have social paths to these peers in particular. More important than the total amount of peers in the social cache is the amount of highly connected peers in the social cache. Even more important is the amount of highly connected peers contacted over time.

For this experiment we have generated a scale-free network consisting of 1000 peers with on average 10 friends per peer. We selected the top 25% most connected peers in the network as the target set. For both MAX-FLOW and BFS the amount of peers from the target set contacted and discovered over time were tracked and plotted in figure 5.7.

Even though the BFS algorithm lets a peer discover more peers, it discovered about the same amount of interesting peers. The important result of this test is however that the MAX-FLOW algorithm contacted interesting peers faster. This is important because contacting interesting peers is the only way to receive their

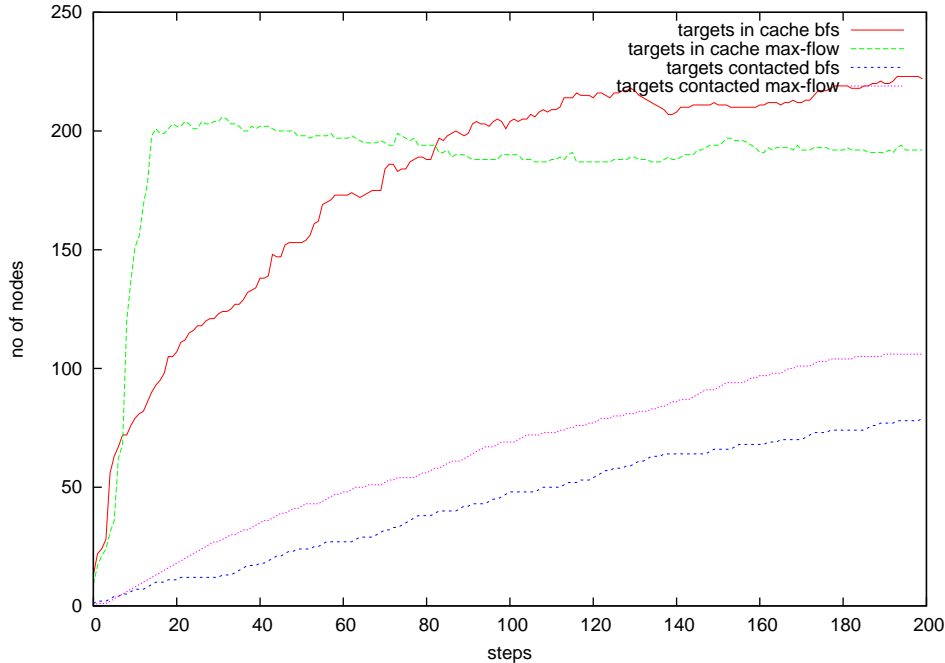


Figure 5.7: Effect of scalability on interesting peer discovery

approvals and moderations. Pruning links based on the added weight they provide to the SRI of peers has proven to be far more effective than pruning based on the total amount of SRI of a peer.

5.7 Conclusion

In this section we looked at exploration speed and scalability of our SRI implementations using a model of SocioCast on a scale-free network.

Exploration of social networks is much faster using the MAX-FLOW algorithm than with the BFS algorithm. The disadvantage to using MAX-FLOW turns out however that it scales less good than the BFS algorithm.

Pruning based on link-weight instead of peer-weight causes both algorithms to scale a lot better. What is more important is that peers using the MAX-FLOW algorithm contact highly connected peers more quickly. These highly connected peers have more friends than other peers, speeding up exploration. They are also providers of above average quality moderations and approvals, thereby improving the knowledge of other peers about torrents.

We can conclude that MAX-FLOW is the superior algorithm albeit far more demanding on processing power. Peers using the MAX-FLOW based SRI contact more highly connected peers in a shorter timespan and discover more of the network in less time than peers using the BFS based SRI.

Chapter 6

Simulations of Attack Resistance

There are several techniques that can be used to cheat in multi agent systems. In this section we will take a look at 3 combinations of these techniques that try to influence the social network. All experiments were done with the MAX-FLOW version of the SRI. We will first look at a combination of the *sybil* attack in a *collusion* network. The second combination is a *collusion* network of moderately connected *traitors*. Finally we will be discuss a *traitor* attack initiated by a highly connected peer.

6.1 Sybil Collusion Resistance

A sybil collusion attack is a combination of introducing fake peers as part of a Sybil attack and connecting them together in a collusion network. This network is then connected via a single peer, the gateway. The phantom peers behind the gateway obviously receive a lower SRI than the gateway peer, but there is a reasoning to this attack. The user operating this gateway peer can then inject bad metadata using its phantom peers without endangering its own friendships.

To simulate a sybil collusion attack using our model we generated a scale-free network of size 1000. We then proceeded by selecting the 10% least connected peers to serve as the collusion network. The least connected peer serves as the gateway and the rest of the peers is stripped of their friendships. The whole collusion network is then transformed into a fully connected sub network. Figure 6.1 was generated as following: for each peer ordered by the amount of friends the SRI is calculated from the viewpoint peer. The peers with id in the range [901 – 1000] form the collusion network. For each node on the x-axis its SRI is shown on the y-axis.

SocioCast is fairly resistant against this type of attack. The SRI of the peers in the sybil collusion network have a low SRI comparable to the least connected peers in the network. Using sybil peers bad approvals can be injected into the system

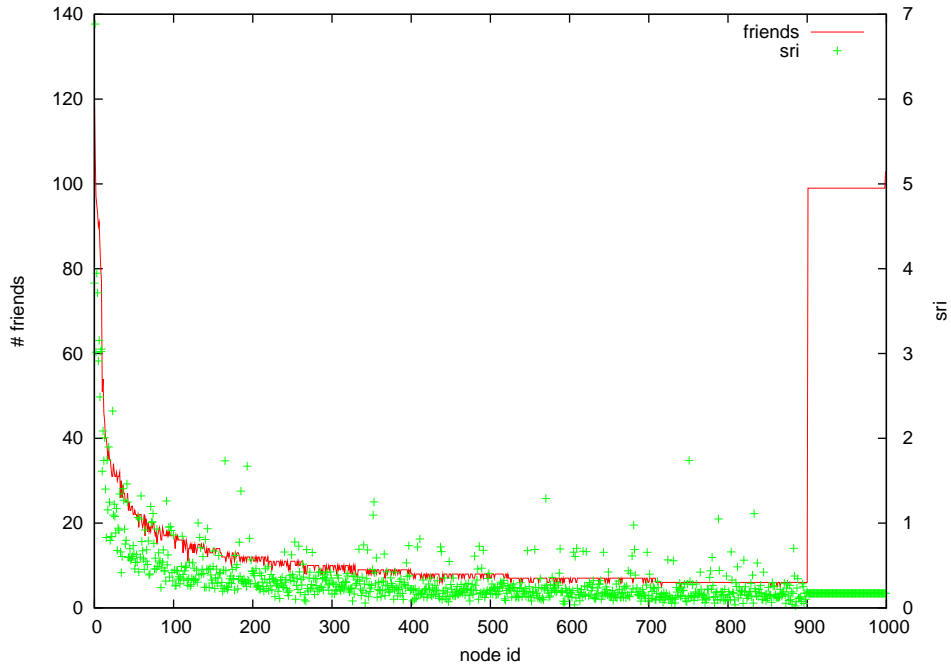


Figure 6.1: SRI for peers in a network with a colluding subnet of Sybil peers

without reciprocity albeit with a low SRI. A counter for such an attack is to punish the gateway peer. This would require a local trust system that punishes friends of peers that gave bad approvals or moderations.

6.2 Traitor Collusion Resistance

The traitor collusion attack is an effect of the friend-spam behaviour of some users in social networks. These users try to befriend every other peer in the network to boost their own popularity.

We will create a traitor collusion network in a similar fashion like the sybil collusion network in the previous subsection. A scale-free network of size 1000 is generated and the 10% least connected peers is selected to form the collusion network. Links are added to the collusion network to form a fully connected sub network. Their original friends are not stripped, the traitor part in this attack means that these peers are actually socially active in the network but betray their friends by colluding with likeminded peers to boost their popularity. In figure 6.2 the peers with id in the range [901 – 1000] form the collusion network.

This strategy gravely boosts the influence of colluding peers on the network. This is unfortunate, since this gives peers incentive to befriend every peer they encounter to boost their own SRI. Peers are unable to discern 'fake' from real

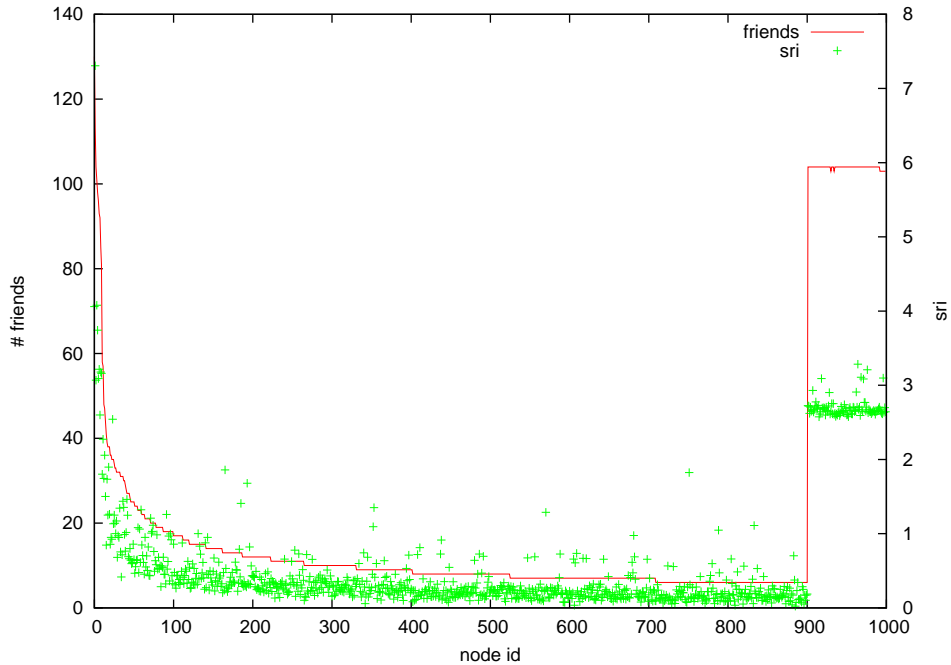


Figure 6.2: SRI for peers in a network with a colluding subnet of traitors

friendships. Without negative consequences to 'fake' friendships, SocioCast is not resistant against this type of attack. Making many friends for the sake of boosting popularity should be discouraged. Friendships should have more consequences than boosting the SRI score. Friendship could for instance require peers to give unlimited access to torrent resources. The same solution to sybil collusion could also be applied, punishing peers for bad behaviour of their friends by lowering their trust rating.

6.3 Worst-case Traitor Resistance

The last attack we will discuss is the worst-case traitor attack. In this scenario the best connected peer acts as a traitor and misbehaves by sending around bad moderations and approvals. Such a situation could arise when a malicious user invests a lot of time by communicating with other users to build a large social network. Another possible route, although rather unlikely, is to hack the account of a highly connected user and abuse it.

SocioCast is not resistant against such an attack because it requires action from the users of Tribler. Users can reciprocate against traitoring friends by cancelling their friendship. Such reciprocation requires users to act which causes delay. Even after removal of the friendship it still needs to propagate into the network. Due to unwillingness or laziness of its friends and the time it takes to propagate the

friendship cancellation, a traitor may benefit from its popularity for a very long time.

6.4 Conclusion

We have discussed three different types of attacks where peers try to boost their SRI score. The MAX-FLOW algorithm is resistant against Sybil attacks. When a collusion network is formed with unconnected peers the SRI of the colluding peers is not affected.

SocioCast is not resistant against attacks involving traitors. The weakpoint of the MAX-FLOW algorithm is that there is no negative consequence to having many 'fake' friends. Peers can boost their SRI by trying to befriend each and every peer in the network without repercussions. In future work possible repercussions should be looked into.

Chapter 7

Implementation

A partial implementation of the SocioCast in Python has been made and integrated into Tribler. The implementation details of the SocioCast design in chapter 3 are here.

7.1 Tribler Modifications

There are several modifications that need to be made to Tribler to be able to be able to integrate the SocioCast subsystem.

The Tribler P2P client already includes a feature that allows peers to add friends to be used by the 2Fast subsystem. Unfortunately these friendships are one-way at the moment. The friendship relation must be made verifiably two-way.

Peers in the Tribler system all have a public and a private key. A *friendship agreement* will be signed by both parties. The contents of such an agreement is shown in table 7.1. When a peer receives such a friendship agreement it knows the relationship is confirmed both ways. These friendship agreements include an expiration timestamp which signifies when the agreement is no longer valid. These are included, because otherwise friendships can not be terminated. Peers should keep track of the lifetime of their friendship agreements and repropagate new friendship agreements in due time.

Part	Field	Size
Payload	public key 1	84 bytes
	public key 2	84 bytes
	expiration timestamp	4 bytes
Signatures	signature 1	64 bytes
	signature 2	64 bytes
Total		300 bytes

Table 7.1: Friendship Certificate

Part	Size	Field	Size
Friends	300 bytes * entries	friendship certificate	300 bytes
Moderators	32 bytes * entries	md5(id moderator)	32 bytes

Table 7.2: SocioCast message layout

The approval relation of SocioCast is not present in SocioCast. In the interface it could be added as a simple *thumbs up* button accompanying moderations. These relations do not need to be signed since peers only distribute their own approval relations.

7.2 SocioCast Protocol

The SocioCast protocol is built as a piggyback system on the BuddyCast subsystem. SocioCast messages are appended to BuddyCast messages. Currently BuddyCast selects peers from a list of peers that are known to be online. With a chance of 50% that it will select a taste buddy from the list. This makes sure *.torrent* files are distributed to peers with similar taste.

The modifications needed for SocioCast to work is that the peer selection is changed to a strategy where online peers with high SRI have a preference over other peers. This makes sure relevant *.torrent* files and moderator approvals are acquired. Another advantage is that this will make social network discovered a lot faster.

SocioCast messages consist of 2 parts, a list of friendship certificates and a list of approvals. The contents of SocioCast messages is shown in table 7.2.

The reason that the identifiers of moderators are encrypted with MD5¹ are twofold. Firstly it saves a lot of bandwidth and secondly to help moderators remain anonymous. Peers need only know about approvals that concern moderators they have moderations from.

There is no limit on the size of these messages to keep the protocol simple. When limits are imposed synchronization methods need to be introduced. These could be based on timestamps and sending oldest relations first or by using a bloom filter.

7.3 Database Layout

The Tribler P2P client uses SQLite as the database backend. Within this database each client stores connection information about other peers and their taste in torrents. SocioCast requires 2 additional SQL tables to keep track of friends and moderators. These SQL tables are listed in tables 7.3 and 7.4.

¹A cryptographic hash function

Field	Type	Size
peerid_from	integer	8 bytes
peerid_to	integer	8 bytes
expiration timestamp	integer	4 bytes
Total per entry		20 bytes

Table 7.3: SocioCast Friends SQL Table

Field	Type	Size
peerid_from	integer	8 bytes
peer_to	integer	8 bytes
Total per entry		16 bytes

Table 7.4: SocioCast Moderator SQL Table

The maximum size of the combined SocioCast database should be no larger than 10MB. Each relation costs 16 bytes and thus 524.288 relations can be stored excluding index tables. Since relations are verifiably two-way only half of all relations need to be stored. If for example data is stored for a network where peers have on average 20 friends the cache can hold relations for 52.428 peers.

Chapter 8

Conclusions

In this chapter we will give a short summary of the scope of this research, the results and the results. Finally we will list several areas to inspire future work on SocioCast.

8.1 Summary and Conclusions

Tribler[19] is a P2P file sharing client, based on the Bittorrent[5] protocol, developed at the Delft University of Technology. Its main focus is the exchange of video content combined with social cohesion in a distributed context. This seems like a trivial problem since sites like YouTube[24] exist as a platform to share videos and make friends. However the emphasis of the research is on making a completely distributed system, a key advantage over the old client-server model.

With the research presented in this thesis we give users of the Tribler P2P file sharing client a method to estimate the quality of moderations about video content. These quality ratings are based on the social relations of users in the Tribler network.

In what way can a *social network* formed by the friendships of users of the *Tribler* client contribute to the *quality of moderations* of *.torrents*?

To answer the research question we have first looked at several incentivitation methods in chapter 2. From all the methods listed we prefer MAX-FLOW because it does not require communication with other peers while determining the reputation for an peer. Also it is calculated relative to each peers making it immune to simple collusion. We therefore decided to design SocioCast with an incentivitation method based on MAX-FLOW.

We discussed the design for the SocioCast extension to Tribler in chapter 3. This design consists of two parts: One part is concerned with propagating friendship information to other peers so that each peer can create a personal view of the social network. The second part is the social relevance index (SRI) of which we

designed two versions, one based on breadth first search (BFS) and one based on MAX-FLOW. This index gives an estimate of the trustworthiness of peers by calculating their social relevance. The SRI algorithm based on BFS bases this score on the shortest social path to peers. The SRI algorithm based on MAX-FLOW bases this on the amount of flow that can reach peers given a certain capacity to social links.

Based on our simulation results in chapter 5 we conclude that MAX-FLOW is the superior algorithm compared to our reference implementation BFS, albeit far more demanding on processing power. Peers using the MAX-FLOW based SRI contact more highly connected peers in a shorter timespan and discover more of the network in less time.

In chapter 6 we discuss how resistant SocioCast is against three specific attacks. The overall weakpoint of the MAX-FLOW algorithm is that there is no negative consequence to having many friends. Peers can boost their SRI by trying to befriend each and every peer in the network without repercussions.

SocioCast is vulnerable to the so called traitor collusion attack, the result of people spamming the network with friendship requests. Therefore a deterrent should be added to make it less worthwhile to indiscriminately add everyone as a friend.

Our research question can thus be answered with: the social network formed by friendships between users can assist users in finding socially active peers in the network that are more likely to provide moderations of good quality.

8.2 Future Work

When the MAX-FLOW algorithm is used, the SRI of peers is limited by the amount of friends of the viewpoint node. Peers are not able to compare the relevance of peers whose SRI is capped by this limit.

A possible solution is to assign infinite capacity to links towards friends at depth 1. It is however not very elegant, because it removes the ability to order friends at depth 1 by their relevance and it only shifts the problem.

Another option is to alter the parameter of MAX-FLOW that determines the capacity of links. SocioCast could dynamically change this parameter dependent on the network structure. When the SRI of peers exceed the upper bound the parameter can be increased to lower the SRI of highly connected peers under limit. This would allow the viewpoint node to compare the relevance of the most relevant peers. Note however that the order of SRI of peers changes by tweaking the MAX-FLOW parameter.

Social groups should be introduced to Tribler to strengthen the cohesion of peers with similar interest. This will make it easier to find relevant approvals and mod-

erators.

Due to the complexity order of the MAX-FLOW algorithm by Edmonds-Karp SocioCast may not scale due to processing power constraints. If this is found to be the case the MAX-FLOW heuristic by Feldman et al.[8] should be evaluated. It should first be shown that when the heuristic by Feldman et al. is used there still exists correlation between the SRI of peers and their connectedness.

Finally SocioCast should be implemented in the Tribler client to see how it performs with a real and dynamic social network.

Bibliography

- [1] Albert-Laszlo Barabasi and Reka Albert. Emergence of scaling in random networks. *Science*, 286:509, 1999.
- [2] BarterCast. <http://www.tribler.org/BarterCast>.
- [3] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. R-mat: A recursive model for graph mining. In *Proceedings of the fourth SIAM international conference on data mining*, 2004.
- [4] Bruce Christianson and William S. Harbison. Why isn't trust transitive? In *Proceedings of the International Workshop on Security Protocols*, pages 171–176, London, UK, 1997. Springer-Verlag.
- [5] Bram Cohen. Incentives build robustness in bittorrent, 2003.
- [6] Ernesto Damiani, De Capitani di Vimercati, Stefano Paraboschi, Pierangela Samarati, and Fabio 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*, pages 207–216, New York, NY, USA, 2002. ACM Press.
- [7] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2):248–264, 1972.
- [8] Michal Feldman, Kevin Lai, Ion Stoica, and John Chuang. Robust incentive techniques for peer-to-peer networks. In *EC '04: Proceedings of the 5th ACM conference on Electronic commerce*, pages 102–111, New York, NY, USA, 2004. ACM Press.
- [9] Pawel Garbacki, Alexandru Iosup, Dick Epema, and Maarten van Steen. 2Fast: Collaborative downloads in p2p networks. In *P2P '06: Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing*, pages 23–30, Washington, DC, USA, 2006. IEEE Computer Society.
- [10] Vincent Heinink. Metadata infrastructure for peer-to-peer video. Master's thesis, Delft University of Technology, 2008.
- [11] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 640–651, New York, NY, USA, 2003. ACM Press.
- [12] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the 5th ACM/USENIX Internet Measurement Conference (IMC'07)*, 2007.
- [13] David Pennock, Gary Flake, Steve Lawrence, Eric Glover, and C. Lee Giles. Winners don't take all: Characterizing the competition for links on the web. In *Proceedings of the National Academy of Sciences*, 2002.
- [14] Python. <http://www.python.org>.

- [15] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*, chapter Informed Search and Exploration, pages 111–115. Prentice Hall, 2003.
- [16] SimPy. <http://simpy.sourceforge.net>.
- [17] Ruben Stranders. Trust models in multi-agent systems, 2004.
- [18] Michael P. H. Stumpf, Carsten Wiuf, and Robert M. May. Subnets of scale-free networks are not scale-free: Sampling properties of networks. *Proceedings of the National Academy of Science*, 102:4221–4224, Mar. 2005.
- [19] Tribler. <http://www.tribler.org>.
- [20] Wouter de Vries. Review of incentivization methods in mas, 2007.
- [21] Kevin Walsh and Emin Gün Sirer. Fighting peer-to-peer spam and decoys with object reputation. In *P2PECON '05: Proceeding of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems*, pages 138–143, New York, NY, USA, 2005. ACM Press.
- [22] Kevin Walsh and Emin Gün Sirer. Experience with an object reputation system for peer-to-peer filesharing. In *NSDI'06: Proceedings of the 3rd conference on 3rd Symposium on Networked Systems Design & Implementation*, pages 1–1, Berkeley, CA, USA, 2006. USENIX Association.
- [23] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, June 1998.
- [24] Youtube. <http://www.youtube.com>.