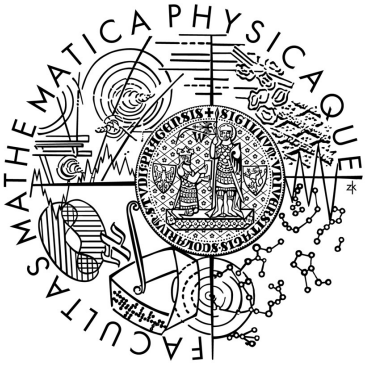
Charles University in Prague
Faculty of Mathematics and Physics

DOCTORAL THESIS



Miroslav Novotný

Trust Management Systems in P2P Networks

Department of Software Engineering

Supervisor of the doctoral thesis: RNDr. Filip Zavoral Ph.D.

Acknowledgements

I would like to thank all those who supported me in my doctoral study and the work on my thesis. I appreciate the help and guidance of my supervisor RNDr. Filip Zavoral, Ph.D., for inspiring leadership and valuable comments that helped me solve many problems. Further, I would like to thank all my colleagues from the Department of Software Engineering, Faculty of Mathematics and Physics, Charles University, for friendly atmosphere and help.

This work and the related research was partially supported by the Ministry of Education, Youth and Sports projects MSM0021620838, the Grant Agency of the Czech Republic projects 201/09/H057, 202/10/0761 and by the Grant Agency of the Charles University under grant numbers 2010/28910, SVV-2010-261312, SVV-2011-263312 and SVV-2012-265312.

Above all, I am in debt to my parents and my girlfriend Marcela for their support and endless patience.

I declare that I carried out this doctoral thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In…..... date..... .....

Název práce: Řízení důvěry v P2P sítích

Autor: Miroslav Novotný

Katedra / Ústav: Katedra softwarového inženýrství

Vedoucí doktorské práce: RNDr. Filip Zavoral Ph.D.

Abstrakt: V dnešní době se architektura určitých typů služeb jako jsou distribuované výpočty, distribuovaná úložiště nebo sítě pro distribuci obsahu, posouvá od tradičního modelu klient-server k více škálovatelnému a robustnějšímu P2P modelu. V takto složitém, anonymním a otevřeném systému je ale velice komplikované zajistit alespoň základní míru zabezpečení. Největší hrozbu představují útočníci, kteří dokáží spolupracovat a s použitím sofistikovaných strategií se snaží obejít stávající bezpečnostní systémy. Jako obrana proti těmto uživatelům vznikly takzvané systémy na řízení důvěry v P2P sítích. Nicméně jejich účinnost právě proti sofistikovaným strategiím není dostatečně ověřena.

V této práci jsme navrhli nový systém pro řízení důvěry s názvem BubbleTrust a vyvinuli simulační framework P2PTrustSim pro testování různých systémů na řízení důvěry a libovolné strategii používané útočníky. Navržený framework definuje několik kriterií, která pomohou vyhodnotit úspěšnost dané strategie oproti zkoumanému systému. V rámci simulací jsme testovali čtyři systémy jež reprezentují současné hlavní přístupy k řízení důvěry a BubbleTrust.

Klíčová slova: P2P sítě, řízení důvěry, distribuované aplikace, bezpečnost.

Title: Trust management systems in P2P networks

Author: Miroslav Novotný

Department / Institute: Faculty of Mathematics and Physics

Supervisor of the doctoral thesis: RNDr. Filip Zavoral Ph.D.

Abstract: The architecture of certain class of services, such as distributed computing, distributed storages or content delivering networks shifts from the traditional client-server model to more scalable and robust peer to peer networks. Providing proper protection to such complex, open and anonymous systems is very complicated. Malicious peers can cooperate and develop sophisticated strategies to bypass existing security mechanisms. Recently, many trust management systems for P2P networks have been proposed. However, their effectiveness is usually tested only against simple malicious strategies. Moreover, a complex comparison of resistance of a particular method is missing.

In this thesis, we (1) propose a new trust management system called BubbleTrust and (2) develop a simulation framework for testing trust management systems against various malicious strategies. Our simulation framework defines several criteria which determine the success of each malicious strategy in the network with a given system. We present results of four trust management systems that represent main contemporary approaches and BubbleTrust.

Keywords: P2P network, trust management, distributed applications, security.

Contents

1 Introduction

In recent years, the traditional client-server model of certain class of services is being replaced by globally interconnected distributed systems which are able to satisfy our requirements on scalability and performance. The most progressive distributed systems are based on peer-to-peer architecture (P2P). This architecture does not have a notion of clients or servers but only peers which can work in both roles. The load connected with providing services is equally distributed among all members of the network. Therefore, the P2P network is an abstract overlay network built on the top of the physical network. This overlay is used for indexing and discovering peers and it makes the P2P system independent on the physical network topology.

The major advantage of this architecture is the elimination of the need of high-perfomance servers which increase the total cost and represent undesirable single point of failure. However, there are several drawbacks which the developers of the P2P applications have to take into account, especially if the application allows open and anonymous access. Except for the peers or network failures, the P2P applications have to deal with treacherous peers that try to deliberately subvert their operation. There is not any central authority that watches the peer behaviour and expels misbehaved peers from the network. The peers have no other possibility than trust that the remote party works as expected.

The P2P architecture is very attractive due to its low operating cost but the unresolved security issues can be discouraging for other types of applications. Imagine full decentralized auction application similar to eBay. Such application cannot exist without proper security mechanisms because the vision of financial profit is very attractive for potential attackers. Other example can be a social network like Facebook built above the P2P network. The access to private information or unauthorized modification of personal information can be also a strong motivation for attackers. We can continue with other applications whose transformation into the P2P architecture its currently difficult to imagine due to security reasons.

The general goal of the thesis is to address the challenges and issues mentioned above - i.e. ensuring at least some level of security in a dangerous environment represented by the open and anonymous P2P networks. We focus primary on the security on the application level jeopardized by the misbehaved users. Because there are not any trustworthy components in P2P network, the peers are forced to manage trust themselves. The simplest method is to remember the outcomes of the past transactions and avoid the cooperation with the peers which behaved incorrectly. The main disadvantage is that the trustworthy of the remote peer can be established until after the first transaction. In the typical P2P application, the peers often have to cooperate with a great number of others and cannot test each one

individually. The more complex methods use experience of other peers in the network to find out whether the remote peer is trustworthy without any transaction with it. Unfortunately, this opens the possibility for malicious peers to report false experience and manipulate with the trust towards any peer. These represent the main reasons why managing trust represents the biggest challenge in the current P2P networks.

In this thesis, we analyse all possible threats in the P2P applications and the current defence mechanisms, which are mainly based on some form of trust management system. We propose a new trust management system which addresses the deficiencies of the current systems and we create a simulation framework to verify its efficiency.

The rest of the thesis is organized as follows: Section 2 enumerates the main threats in P2P network. We distinguish threats on the overlay and application layer. In section 3, we summarize state of the art of the trust management systems to mitigate the threats on application layers. The attacks directed to these trust management systems are discussed in section 4. Section 5 focuses at describing a novel trust management system called BubbleTrust. In section 6, we present our P2P trust simulation framework and define criteria that the quality trust management system should meet. The results for five selected trust management systems, including BubbleTrust, are stated in section 7. Section 8 concludes the thesis and proposes a direction for future research.

2 Security threats in P2P networks

In this section, we present a general overview of security threats related to the P2P networks. These threats can be classified into two basic groups: attacks against overlay network and attacks on the application level. In our work, we focus mainly on the application level but it is necessary to summarize threats related with overlay layer, because they significantly influence the layer above.

2.1 Attacks against overlay network

The overlay network is a network built on the top of another network, typically on the top of the IP network. The overlay network offers its own system of identifications and routing protocols and it abstracts the physical network. The structured P2P networks use a mechanism called distributed hash tables (DHT) which can be considered as a generalization of the classical hash tables.

The distributed hash tables stores (key;value) pairs and any participating node is able to efficiently retrieve the value associated with a given key. Each node is responsible for an assigned subset of keys and has records in its routing table which allows to locate all other keys. The important feature of the DHT is a capability of joining and leaving nodes with a minimal amount of disruption of the lookup services. There are several implementations of this concept which differ mainly in the routing algorithm: CAN [1], Chord [2], Pastry [3], Kademlia [4] or P-Grid [5].

Except for the general attacks applicable to all network systems such as denial-of-service or exploitation of implementation bugs, DHTs provide some specific weaknesses. There are three most discussed DHT attacks in literature: (1) the Sybil attack, where the attacker creates a large number of false identities, (2) the Eclipse attack, where the attacker corrupts the routing tables of honest peers by filling them with incorrect routing information, and (3) routing and storage attacks, where the attacker does not follow the routing or storage protocol correctly, e.g. routing to incorrect nodes or wrongfully modification of stored data. We discuss these weaknesses in detail below.

DHTs also have to deal with other issues which may be the result of the regular operations. One of them is churn - the continuous process of node arrival and departure, which makes great demands on DHT algorithm to efficiently handle continuous restructuring the routing table and migration data. The churn has been studied in [6], [7] and [8], and Ou et al. proved that DHT Kademlia is highly resistant against this issue [9]. The load balancing problem is other issue which has been intelsively studied in literature [10], [11], [12], [13] and [14]. It is connected with inappropriate popularity of some resources and inadequate capacity of the nodes offering these resources. Even if node and item identifiers are randomly chosen, there is $\Theta(\log N)$ imbalance factor in the number of items stored at a node [10].

There are several surveys that discuss the DHTs security in general. The most comprehensive survey is given by G. Urdaneta et al. [15]. It summarizes some well-known security threats faced by DHT and reviews techniques proposed to solve them. The critical review on the security in the DHT and proposed solutions is provided by Dahan and Sato [16], their conclusion is that DHT should not be used to create secure systems.

2.1.1 Sybil attack

The Sybil attack exploits the fact that P2P network is open to anyone and uses virtual identifiers which are only loosely connected with physical entities. The attacker is easily able to create a large amount of virtual identities with a relative small number of physical nodes. This attack does not damage the DHT itself, but can be used as a prerequisite for other attacks on both overlay and application layer. The most of the security mechanisms assume that there is only a limited fraction of malicious peers in the network and the Sybil attack can break this assumption.

The Sybil attack is not specific to DHTs, but it is extraordinarily dangerous in this environment. In DHTs, first analysed by Doucer [17] with the conclusion that the only practical way to limit the number of virtual identities related to one physical entity is the existence of a logical central authority which issues the identifiers. This central authority needs to have a reliable way of identifying physical entities and this is hard to achieve. Other difficulty is that it is unacceptable for a distributed environment like DHT to rely on any kind of central entity.

The central authority can be replaced by its decentralized variant. For each newcomer several peers are chosen. These peers are responsible for the validation of its identity and can refuse to join then into the network. The suitable candidates for this job are bootstrapping nodes used by newcomers to initialize their routing tables. But these bootstrapping nodes can be already under control of the attacker and allow connection of more malicious peers. The prevention of this should be ensured by the limitation of usable bootstrapping nodes. For instance, Dinger and Hartenstein [18] propose an algorithm where the bootstrapping nodes are computed using the hash of node's IP address.

A bigger challenge is the validation of the identity itself. First problem is how to define the physical identity in this context. Is it a single computer, a user sitting behind this computer, or the whole criminal organization which possesses a large botnet of computers with different IP addresses? The current validation algorithms use several techniques.

Castro et al. [19] proposed using a set of trusted certification authorities to produce signed certificates that bind a random node identifier to a public key and IP address. In order to prevent Sybil attack, they suggested to charge money for each

certificate or to bind them to the real-world identities. This includes considerable administration and processing overhead and for binding to the real-world identities requires reliable authentication procedures.

The hash of IP address and port is used as for identification in [18]. But this solution causes trouble to users behind NAT or mobile users changing IP address continuously. Additionally, it is ineffective against botnets possessing many IP addresses. A similar solution was proposed by Wang et al. [20]. They count on other network characteristics like default router IP address, MAC address and RTTs measured by randomly selected nodes within the sub-network of these routers (landmarks). The authors introduced a concept called *net-print* containing this data and representing self-certifying data, which can be directly verified by other nodes. However, the capability of verification of this data is limited, the MAC addresses can be verified only by nodes in the same local network and the verification of the default router is based on the ICMP message with IP Route Record, which is filtered in most networks. The only really verifiable information is RTTs between the node and a set of designated landmarks. Other disadvantage of this approach is that changes in the network conditions cause subsequent identification tests fail and it is not possible to support mobile nodes with this system.

Other solutions use network coordinates to group nodes. The system described in [21] measures round trip time between nodes and uses a triangle inequality to place them into *d*-dimensional euclidean space. If two nodes are reasonably far from each other in this space, we can assume that they represent different identities. If they are close, we cannot assume anything. Therefore, this system can be used to detect distinct nodes to ensure that the critical network functions are distributed among distinct identities, but cannot prevent connection of multiple virtual identities which operate on a single node. The algorithm proposed by Bazzi et al. in [22] uses a similar idea; the distance between two nodes is represented as hop count and its measurement is cryptographically protected.

A computational puzzle is other method how to protect DHT network against Sybil attack. A general difficulty in such systems is enforcing that puzzle solutions are not reused by attacker over time. Borisov [23] proposed to add computational puzzle into Chord. The system uses periodic ping messages, used by Chord to maintain the structure of distributed overlay. They modified these messages to include the instructions for computation which are different each time. Rowaihy et al. [24] suggested a hierarchical system based on computational puzzle. The system creates a tree where the root must be trusted and reliable. Unfortunately, this presents another form of central authority.

The fundamental problem with computational puzzle in the P2P network is that the network can consist of many different types of nodes with different computation capacity. The puzzle must be complex enough to prevent joining

malicious nodes and at the same time simple enough not to obstruct regular users on slow computers.

All previous methods try to assign a virtual identity to a single physical node, represented by a single IP address, a position in the network, or a computation capacity. If the attacker controls a large amount of physical nodes, geographically distributed and with sufficient computation power, these methods do not present obstacles. The approaches based on social networks try to deal even with this dangerous situation.

The first of them was SybilGuard [25] which uses human-established trust relationships. The relationships between honest region (i.e., the region containing all the honest nodes) and Sybil region (i.e., the region containing all the Sybil identities created by malicious users) are called attack edges. The basic assumption is that the attacker can create any number of relationships between Sybil identities, but it is limited in the number of attack edges. The SybilGuard partitions nodes into groups such that the number of groups that include at least one Sybil identity is bounded by the number of attack edges, independently of the number of Sybil identities. In the 2010, the SybilLimit [26] was published which improves the SybilGuard in several ways.

Table 1 summarizes previously discussed defences against Sybil attacks. As we can see, the entity identification methods belong to one of the following categories: (1) real-world identification, (2) costly identifiers - money or computation time, (3) underlying network - IP address or measured position, (4) social networking. Some of the methods put barrier for Sybil identities to entering the system, others just detect already joined Sybil nodes.

All these methods significantly reduce the number of Sybil identities in the network; however, they are not able to suppress them completely; except the methods using real-world identification, which are difficult to implement. The main conclusion is that the developers of the security mechanisms on the application level have to suppose that the attacker can posses a large number of fake virtual identities without additional cost.

At this point, we should already mention other problem, which does not relate directly with Sybil attacks, but is closely related to identification generation. In the DHT the node identification (nodeId) does not serve only for identification purpose but it also determines which objects will be stored on the node and which neighbours will be in its routing table. Attacker who can chose nodeId arbitrarily, can control access to target object or compromise the integrity of a structured P2P overlay.

Authors	Entity identifications	Verified by	Prevent/Detect Sybil attack
Doucer2002 [17]	Real-world identification.	Central authority.	Prevent.
Castro2002 [19]	Charge money or real-world identification.	Set of trusted certificate authorities.	Prevent.
Hertenstein2006 [18]	IP address and port.	Bootstrap nodes.	Prevent.
Wang2005 [20]	Network characteristics.	All nodes, but some of them have more possibilities.	Detect.
Bazzi2005 [21]	Network coordinates.	All nodes.	Detect.
Bazzi2006 [22]	Network coordinates.	All nodes.	Detect.
Borrisov2006 [23]	Computational puzzle.	Neighbours in DHT.	Prevent.
Rowaihy2007 [24]	Computational puzzle.	Hierarchical system rooted by trusted authority.	Prevent.
Yu [25], [26]	Social network.	All nodes.	Detect.

Table 1: Defences against Sybil attack.

The solution for this problem was already offered by Castro in [19]. It is based on crypto puzzle and distributed identity generation. The newcomer needs to cooperate with several nodes which provide restrictions for its new identity and guarantee that these restrictions ware satisfied. This solution can be easily built in Sybil defence techniques which used trusted authorities or bootstrap nodes but it is difficult to integrate it into other methods. On the other hand, the techniques using IP address or network characteristics as nodeId already have a natural defence against arbitrarily chosen identities. In the rest of the networks, it can be solved using asymmetric cryptography. Each peer generates a pair of private/public keys and uses a hash of public key as its nodeId. But this method does not prevent tampering with nodeId completely. The attacker can generate key pairs as long as it finds proper keys giving them the nodeId close enough to the desired value.

Except the last issue with repeated generating key pairs, we consider the problem of arbitrarily chosen identities as acceptable solved. In the application level we suppose that the attacker cannot place itself into strategic position into under-laying DHT network. But it is necessary to check that the under-laying network implements corresponding countermeasures.

2.1.2 Eclipse attack

The Eclipse attack is also known in literature as routing table poisoning. It consists in tampering the routing table of the honest nodes. The aim is either the disruption of the communication in the network or the redirection of the lookup queries to malicious nodes. The easiest way to perform this attack is through incorrect routing updates. Sit and Morris [27] stated that systems which do not have special verifiable requirements on the records in the routing tables are most vulnerable to this type of attack. For instance, in the DHT Pastry [3] one item in the top level in the routing table can contain a large number of different identifiers. The attacker can easily supply a desired identifier during the routing updates and the target peer accepts it because it is a valid identifier. The systems like Chord [2] have stronger restrictions on identifiers in the routing table and make this attack more complicated. On the other hand, the loose restrictions on the routing table entries allow routing optimization according to the network proximity [28]. However, such optimization presents another possible attack scenario. Hildrum and Kubiatowitz [29] showed that attackers can reduce their apparent distance from a target node and enforce itself into its optimized routing table.

The basic defence against Eclipse attack consists in introducing additional constrains into the routing table. Castro [19] proposed a solution which applies this basic strategy and allows preserving proximity-aware optimization. He suggests using two routing tables. One table exploits the potentially vulnerable network proximity information (called optimized routing table) and the other contains only entries which can be verified (called verified routing table). In a normal operation the optimized routing table is used. The system switches to the verified routing table in a case of a routing failure.

A problem of the previous solution is that the poisoning in the optimized routing table can increase over time and shortly it can be unusable and the system degrades into non-optimal routing. The defence proposed by Condie et al. [30] deals with this problem by periodical resetting the optimized routing table to the content of the verified routing table. At each reset, every node gets a new random identifier. This should prevent attacks exploiting the knowledge of how the routing table are updated over time. The authors stated that if good nodes move continuously, then it is difficult to attack them in the same way after every reset.

Hildrum and Kubiatowicz [29] stated that the network proximity optimization can be also used to help to prevent the Eclipse attack. However, they assumed a trusted mechanism for measuring network distance. Unfortunately, the authors did not mention how this trusted mechanism should be realised in practice. They assumed that if the fraction of the malicious nodes is reasonably small, it is difficult for the malicious nodes to be closest in the network distance to a majority of honest

nodes.

Other defence proposed by Singh et al. [31] is based on the observation that a node that mounts an Eclipse attack must have a higher than average node degree. The authors proposed a mechanism in which the nodes anonymously audit each other's connectivity and showed that enforcing a node degree limit is an effective defence against Eclipse attack. The results showed that the system is effective only if the degree limit is small which has a negative impact on the lookup time in the absence of attacks.

Awerbuch and Scheideler [32] introduced the concepts of regions in [0,1) identifier space. Each new node that joins the network is securely assigned to a single region and routing is done from region to region. To prevent malicious nodes to continuously join and leave the system until it receives the desired region, the protocol called *cuckoo rule* is implemented. This protocol establishes that when a new node joins the system, all nodes in the certain region must leave the system and rejoin with a new random identifier. This prevents the attacker from concentrating many malicious nodes in one or a small number of regions.

Table 2 summarizes previously discussed methods. None of them provides sufficient defence against this attack. The methods which do not preserve a stable node identifier are unacceptable from our point of view. It is causing a significant overhead because it is necessary that data migrate each time the node identifier changed. Additionally, the security mechanisms on the application level require stable node identification.

As we can see, the defence against the Eclipse attack involves a trade-off between performance (optimized routing table) and security (constrained routing table). Therefore, these techniques are not able to guarantee proper routing in the DHT and must be combined with other mechanisms such as redundant routing, routing failure tests and recovering from routing and storage failures (these methods will be described in section 2.1.3).

Author	Techniques	Disadvantage
Castro2002 [19]	It uses two routing tables, optimized and verified.	Optimized routing table can be easily poisoned and the verified routing table is used most of the time.
Condie2006 [30]	It uses two routing tables, optimized and verified. The content of the verified table is periodically reset and churn is induced.	The induced churn introduces a significant overhead. The node identifier is not stable.
Hildrum2003 [29]	It is based on the trusted measuring of the network proximity.	The trusted network distance measurement is difficult to implement in practice.
Singh2006 [31]	It limits the number of node degree via anonymous auditing.	It has a negative impact on the lookup time in the absence of attack.
Awerbuch [32], [33]	It uses a region-based redundant routing table and *cuckoo rule*.	Complex algorithm. The node identifier is not stable.

Table 2: Defences against Eclipse attack.

2.1.3 Routing and storage attacks

The last group, routing and storage attacks, presents the most serious threats. The malicious node can refuse to forward lookup requests or it can forward them to incorrect or malicious nodes. These attacks are generally classified as routing attacks and can lead into increasing the lookup time or in a worst-case scenario into the routing failures. In the storage attacks, the malicious node responsible for the key can deny the existence of the key or provide invalid data as response. The malicious node on the path of the lookup query can only pretend to be the node responsible for the key and compromise a significant number of keys in the system.

A fundamental defence against these attacks is a replication, but it is not sufficient regarding the possibility of the Sybil identities or arbitrarily chosen identities [16]. The artificially created identities can control all replicas or be on the path towards all replicas. For instance, several mechanisms [19], [29], [34] use replicas stored on numerically closed locations. This increases the chance that the malicious nodes will be able to control all replicas or paths towards them. As a defence, Harvesf and Blough [35] proposed to place replicas at equally spaced locations in a Chord ring. They proved that this method can produce d disjoint routes

if 2^{d-1} replicas are placed in a fully populated chord ring.

Other defences try to enhance routing protocols to either support independent routing paths between every two nodes or allow the querier node to control routing progress. The mechanism called Cyclon [36] is an example of the first group. This protocol was proposed by Artigas et al. and extended the Chord. The authors divided the system into $r = 2^{m-p}$ independent Chord rings, each contains only nodes that share p rightmost bits of the m-bit identifier. The successor lists do not have this restrictions and can be used as the first or last hop in a lookup. The system provides r independent path, because routing is realized through the r independent rings.

The second approach was first used by Sit and Moris [27] who proposed iterative routing as a defence against the routing attack. In the iterative routing, the requester controls the lookup progress and detects potential routing anomalies. But this approach has been rejected due to significant overhead. Xiang [37] proposed to use tracer routing where the requester only observes the lookup progress and together with node-ID based signature is able to detect the malicious nodes. The author proposed the protocol which bypasses the malicious nodes and allows to establishing secure paths.

Other possibility was proposed by Ganesh and Zhao [38]. They proposed a protocol which allows querier to assure that the results from a lookup are correct. Their solution is based on signed certificates that prove the existence of nodes in some range of the identifier space. These certificates are placed at randomized node subsets (proofs managers). After completing the routing request, the querier can trigger verification procedure and it determines whether the better root node exists by searching the existence of the proofs. A disadvantage is that this solution requires the existence of offline certification authority which distributes the public-private key pairs.

Most of the security mechanisms do not rely on the one of the strategies described above but they implement a combination of them. For instance, Artigas et al. in their next work [39] stated that the independent routing paths implemented in the Cyclon are not enough to provide sufficient probability of success and the proposed routing protocol called Bypass. This protocol is a combination of two strategies: A redundant routing algorithm that, at each step, routes to a set of nodes instead of just one; and a filter that avoids selecting nodes that are not reliable as next hop. Their analysis shows that Bypass can potentially achieve a significant improvement.

All discussed methods are summarized in table 3. This table does not contain all published methods it but only illustrates commonly used approaches to solve these issues. Authors of all techniques agree that the data replication is a basic means of preventing routing and storage attacks but as many studies show it is not sufficient and must be completed with some sort of redundant routing or verified routing

protocol. Also the protection against Sybil attacks and arbitrarily chosen identities makes these attacks difficult and many security mechanisms count on it.

Author	Techniques	Disadvantage
Castro2002 [19]	It uses two routing tables, optimized and constrained. The constrained routing table is used in case of routing test failure. Replicas are placed at numerically closed locations.	There is no guarantee that the routing paths are disjoint.
Hildrum2003 [29]	It uses redundant routing table entries based on network proximity. Replicas are placed at numerically closed locations.	There is no guarantee that the routing paths are disjoint.
Fiat2005 [34]	Modification of Chord that uses swarm of nodes instead of single nodes as a basic construct.	The swarm contains numerically closed nodes.
Harvesf2006 [35]	Replicas are placed at equally spaced locations in a Chord ring.	Difficult replicas maintenance.
Artigas2005 [36]	Independent chord rings providing independent paths.	Addresses only routing issue.
Artigas2008 [39]	Redundant routing combined with filters.	Addresses only routing issue.
Sit2002 [27]	Iterative routing.	Iterative routing has a significant overhead.
Ganesh2005 [38]	Proof of existence of a node within a certain id range.	Requires offline certification authority.

Table 3: Defences against routing and storage attacks.

2.1.4 Summary

The security issues of P2P overlay networks have been intensively studied during the last ten years. The proposed mechanisms are able to significantly improve the security in the overlay network if they are implemented completely. Many

researches focus only on subset of security thread, leaving other issues open or making different assumption regarding the attack model. Combining of these strategies to make “secure” P2P network remains unresolved.

As we mentioned in the section 2.1.1, the Sybil attack has no reliable solution which means that the routing and storage attacks are also possible. Therefore building the fully secure P2P overlay network in the open and anonymous environment is nearly impossible. Nevertheless, described security techniques should be implemented to filter out individual or unsophisticated malicious nodes. Larger groups of cooperated malicious nodes still represent the significant threat to the current P2P networks.

DHTs have been used in numerous popular peer-to-peer systems such as KAD network [40], BitTorent or Limwire. All these implementations are based on Kademlia which provides a relative security due to a build-in replication and iterative routing. However, it is still vulnerable to Sybil attack and arbitrarily generated identities. None of current DHT deployments are specially designed to tolerate malicious nodes. Despite the fact that several security mechanisms have been proposed in last years, their implementation into current P2P systems is complex and requires trade-off between security and performance. Additionally, there are still open problems. The most challenging issue for securing P2P decentralized systems is a robust and secure assignment of node identifiers. This is crucial to guarantee that malicious nodes occupy only a small fraction of identifier space and cannot place themselves into strategic positions in the network.

2.2 Attacks on the application level

Apart from the attacks exploiting application bugs, the biggest problem of the P2P networks is the lack of cooperation. For P2P networks to be effective, nodes participating in the network must work together. However, when human nature is allowed to intervene, this does not always happen. In the case of P2P networks cooperation is very difficult to enforce. The users behind the applications follow their own interests which do not have to correspond with interests of others.

2.2.1 Pollution in file-sharing networks

In the most extensively used P2P file-sharing networks the primary problem is pollution [41], [42]. There are several forms of pollution studied in real P2P networks. The first form, called data pollution, consists in the sharing deliberately corrupted files. These files can contain undesirable or even harmful content. Many viruses use file-sharing networks to replicate themselves. They copy themselves into shared folders under camouflaged names to lure download. Due to the fact that 10% of the most popular files create almost 90% of all traffic in the file-sharing network [43], the virus can speed up their dissemination by choosing popular names. Every

downloaded file should be strictly checked on viruses and spyware, but this can be done only after successful download. Even if the downloaded file is not infected, it can contain different data then was desired. The number of useless downloads results into wasting network capacity and annoying the users.

The second form of pollution, called index poisoning or meta-data pollution, consists in corrupting indexation mechanisms by introducing a large number of spurious files which are not shared by any peers. After the poisoning, the major portion of indexes contains invalid information. The client downloading the poisoned file always fails to establish connection with the other peers. Kong et al. [44] studied index poisoning in BitTorrent network and Locher et al. [45] proved that the index poisoning can also affect the KAD network by corrupting DHT entries, either by publishing fake records on the responsible peers or by inserting malicious nodes which are close to them.

The protection against pollution is far from trivial. The successful mechanisms have to assess the downloaded file before it is actually downloaded. The only clue can be the information from other peers which downloaded the file from the same peer in the past. These mechanisms are called reputation-based and are discussed in detail in section 3.

2.2.2 Free-riders in file-sharing networks

Other problem is represented by peers called Free-rides which consume more than their fair share of network resources [46]. They exploit the system only to download and do not offer anything in return. This behaviour breaks the basic idea of P2P networks and leads into overloading of honest peers. A number of studies have shown that free-riding is a problem of current P2P systems, with resulting into serious performance degradation. For instance, Hughes et al. [47] in their experiment in Gnutella network in 2005 find out that 85% of peers do not share any files. And Sirivianos et al. [48] present an experimental study on the behaviour of BitTorrent network when selfish peers attempt to maintain high download rates without uploading. Their modified free-rider client achieved better download rates than compliant client in most cases, but as the number of free-riders clients increases they incur substantial performance degradation for both free-rider and compliant clients.

Varying incentive schemes have been proposed to encourage peers to cooperate by sharing resources. BitTorent currently uses reciprocity based-scheme [49] which as shown Sirivianos et al. [48] is ineffective in discouraging free-riding. Kyuyong Shin [50] proposed a scheme called Treat-Before-Trick with a secret sharing algorithm. Files are divided and encrypted by the owner. The key is divided into n subkeys, any of t of which are sufficient to recover the original key and decrypt the file pieces. The owner distributes the file pieces and subkeys to a set of

requesting peers, called *leechers*. The leechers barter with each other by exchanging keys for file pieces. The peers are enforced to share their downloaded pieces to get the necessary number of keys to decrypt downloaded files. This scheme is shown to penalizing Free-riders. However, there is an added cost of requiring encrypt and decrypt file, and distribute keys.

According to Karakaya et al. [51] solutions to combat Free-riding can be categorized into three main groups: *monetary*-, *reciprocity*- and *reputation*-based approaches. Monetary-based approaches work on the basis of charging peers for the services they receive. Any monetary-based mechanism requires two key mechanisms: an accounting module to store virtual currency for each peer and settlement module to fairly exchange virtual currency for services. Most of the monetary-based solutions implement these modules on the central nodes and are not useful in the pure P2P networks. In reciprocity-based solutions, peers monitor other peers' behaviour and evaluate their contribution. But this information is supplied by the other peers themselves and Free-rides can easily supply false information. We have already mentioned reputation-based techniques as a defence against pollution and the same techniques can be used against free-riders as well. These techniques include numerous different approaches with different success rate and issues discussed in detail in section 3.

2.2.3 Summary

The file-sharing P2P networks are currently the most used P2P applications and attacks against them are well documented. But the same attack scenarios can be used in every P2P application. The general scheme of all P2P application consists in two basic functions: searching and utilizing the services. The peers connected in the networks offer their services, which can be files to download, storage, network or computational capacity to use. These services must be indexed first to allow other peers to discover them. The malicious peers can violate indexing procedures (index poisoning), provide corrupted services (data pollution) or they do not provide services at all (free-riding). The defence mechanisms are similar as well, in the most cases it can be used only as a slight modification of the previously described techniques.

3 Reputation-based trust management systems

In the traditional client-server model, servers represent trustful points in the network application. They are typically operated on the provider's infrastructure and there is an assumption that they operate correctly because it is in the provider's best interest. The malicious behaviour is expected only on the client side. Therefore, there are only two problems which need to be solved: (1) securing connection between clients and server to be sure that both are correctly identified, and (2) protection of the servers from being exploited by malicious clients.

The situation in the P2P network is much more complex. Every peer can operate as a client and a server simultaneously; therefore, the servers are no longer trustworthy by definition, but every member of the P2P network have to earn each other's trust. Currently, there are two major approaches for managing trust [52]: policy-based and reputation-based trust management. Policy-based trust relies on objective "strong security" mechanisms such as signed certificates and trusted certification authorities. The access decision is usually based on rules with well defined semantics providing strong verification and analysis support. The system makes decision based on "non-subjective" attributes such as a requester's age, an address or a credit card number which should be certified by trusted certification authorities. As we can see, such systems require trusted certification authorities with extended verification capability which are very difficult to implement in the open and anonymous environment.

Reputation-based trust management is mainly based on the notion of reputation. In general, reputation is the opinion of the public towards a person, a group of people, an organization or a resource. In the context of collaborative application such as P2P systems, the most suitable is the definition provided by Abdul-Rehman and Hailes [53]: "The reputation is an expectation about agent's behaviour based on information about or observation of its past behaviour." Reputation allows parties to build trust, the degree to which one party has a confidence in another.

Like in a human community, the peers are exchanging information on their previous collaborations with others. This information includes the degree of satisfaction with the services provided by the remote peer and can be used by other peers which are considering cooperation with the same remote peer. Obviously, there is a higher probability that the peers with a large number of positive recommendations will provide correct services. The reputation-based systems help participants decide whom to trust, encourage trustworthy and fair behaviour. Therefore, they can be used as a defence against pollution and free-riding [54].

However, there are also several issues. For instance, accepting

recommendations from unknown peers blindly is tricky because these recommendations can be deliberately misleading. The group of malicious peers can cooperate and provide each other false positive recommendations. The detection of such false recommendations is very difficult and makes the whole system more complex. More about attacks on the reputation-based trust management systems is discussed in section 4.

Due to the difficult implementation of policy-based trust management, the reputation-based trust management became the only option in the pure and even in some centralized P2P networks. For instance, the eBay [55], [56] uses reputation-based trust system which operates on central servers but completely relies on information from the clients (sellers and bidders). In the rest of this thesis, we will use the notion trust management system (TMS) instead of a more correct reputation-based trust management system, because the policy-based trust management systems are no longer concern us.

3.1 Taxonomy of trust management systems

All TMSs have to solve three basic problems: (1) how to distribute recommendations among peers, (2) how to verify recommendations, and (3) how to compute trust based on these recommendations. There are plenty of different approaches solving these problems. Several researches presented taxonomy to organize existing ideas and facilitate system design. One of the first taxonomy was proposed by Marci and Garcia-Molina [57]. They identify three basic components of reputation system: Information gathering, Scoring and Ranking, and Response. Each of these components has to solve unique problems and the authors discussed possible design choices proposed by the research community.

Broader survey was proposed by Hofman et al. [58]. They developed an analytical framework by which reputation system can be decomposed, analysed and compared using a common set of metrics. They identified three dimensions fundamental to any reputation system: formulation, calculation and dissemination. Formulation is the abstract mathematical specification of how the available information should be transformed into a usable metric. Calculation is the part of the reputation system that receives input information and produces the reputation values. And dissemination part of the reputation system is responsible for delivering calculated values to interested parties. Although the calculation and dissemination parts often influence each other, the authors separate them for analytical purposes.

In the next part, they defined an attacker model and classified known and potential attacks on reputation system within this model. But these sections skip several attacks scenarios and mix attacks on reputation systems with the attacks on P2P layer. The defences against these attacks are typically built outside the reputation systems as was discussed in section 2.1. And finally, they used their analytical

framework to analyse several well-known TMSs.

We consider their analytical framework as the main contribution of this work. This framework is a valuable insight into implication of design choices. We used some ideas in our work [59] that analyse requirements for a reliable TMS in insecure environment. Unlike the previous work, we divided TMS into three-layered architecture: (1) secure P2P layer, (2) information handling layer, and (3) formulation and calculation layer. We define the functions on each layer and establishe several simple criteria to each component. We also analysed several published systems according these criteria, none of them met all criteria completely.

For the purpose of this work, we use simplified classification which was already used in [60]. It is composed from four characteristic: (1) type of rating, (2) feedback aggregation, (3) feedback verification and (4) calculation. In contrast to [58], its ambition is not to embrace all possible design choices or provide the analytical framework, but organize the major approaches for building TMSs.

In our classification, we distinguish two basic types of rating: global and personalized. The core of the global rating is to define a single global trust value for each peer in the network. The calculation is typically done only once and the calculated value is distributed to all peers in the network. Therefore, the global reputation system is dominant in the centralized networks. Nevertheless, the personalized trust rating is more common in distributed P2P networks. In personalized reputation system each peer has self maintain reputation values to other peers.

The feedback aggregation scheme can be full or selective. The full-aggregation TMS calculates the reputation value of a peer considering the opinions from all other peers who have interacted with this peer. The full-aggregation scheme can be very accurate but it is connected with a high load for underlying network. The selective approach involves a trade-off between the accuracy and load, the reputation is derived from a subset of all existing opinions in the network.

We also distinguish four types of feedback verifications: none, a good provider, a personal experience, a global experience. The TMSs without feedback verifications are vulnerable to peers providing false feedbacks. The elementary protection is based on the idea that a good provider is a good recommender as well, but some malicious peers do not have to fulfil this premise. Better protection is provided by the personal experience verification. Each feedback source is assessed according to the usefulness of its previous feedbacks for the local peer. The last type, global experience, extends personal experience over the usefulness of previous feedbacks for other peers in the network.

The last part, calculation, is a component of the TMS that receives input information and produces the reputation values. Each TMS has a unique calculation

scheme based on the mathematical specifications, for example: weighted average, fuzzy logic, Bayesian approaches, etc.

The rest of this section contains examples of TMSs in chronological order. This overview demonstrates the variety of different approaches proposed by the research community in the last decades. We focus primary on TMSs operated in a fully distributed environment, so they do not rely on any kind of central authority. Systems with a central authority have its task much easier but the existence of such central authority breaks the basic idea of P2P network and presents the single point of failure and additional security threat.

3.2 EigenTrust

EigenTrust [61] is one of the oldest and the most cited trust model for P2P networks. EigenTrust calculates a global trust value which is based on the idea of transitive trust.

Basic algorithm is simple. After peer *i* interacts with peer *j*, it can compute its normalized local trust value $c_{i,j}$ based on direct observations. If there are no direct interactions, peer *i* can calculate the reputation metric for another identity *k* by asking other peers for their opinions of peer *k*. The calculated reputation from peer *i* to peer *k* is: $t_{i,k}=\sum_j c_{i,j}\cdot c_{j,k}$. We can write this in matrix notation: If we define *C* to be the matrix $[c_{i,j}]$, than $\vec{t}_i=C^T\cdot\vec{c}_i$ is a vector of trust values from peer *i* towards all other peers in the network which reflects the local experiences and the opinion of its friends. We can extend it by asking its friends' friends $\vec{t}_i=(C^T)^2\cdot\vec{c}_i$ and so on. After *n* interactions we get $\vec{t}_i=(C^T)^n\cdot\vec{c}_i$. For *n* large enough, $\vec{t}_i$ converges to the left principal eigenvector of matrix *C*. Therefore, all peers have the same $\vec{t}_i$ which contains global trust values for all peers in the network.

EigenTrust works in an iterative way; in each step *k*, vector *t* converges to the left principal eigenvector $t^{(\vec{k+1})}=C^T\cdot\vec{t^k}$. There are two principal issues with this approach. First, the malicious peers providing false feedback jeopardize the convergence to the correct trust vector. The authors deal with this problem by introducing pre-trusted peers. These peers are known on all peers in the network and are included into computation in each step. The new formula is $t^{(\vec{k+1})}=(1-a)C^T\cdot\vec{t^k}+a\,\vec{p}$ where *a* is some constant less than 1 and $\vec{p}$ is a vector of pre-trusted peers. This also makes the matrix *C* irreducible and aperiodic, which guarantees the convergence.

Second issue is that there is not any central authority; therefore, the calculation has to be distributed among peers. Fortunately, the calculation is mainly based on the matrix multiplication which can be easily implemented in a distributed manner. The naive implementation assumes that peer *i* calculate *i*'th item of vector

$\vec{t}$,which is its own trust value. Malicious peers would be able to easily report false trust value to hide their maliciousness. The secure implementation uses the so-called score manager. Each peer has assigned several score managers responsible for the calculation of its trust value. If any of the peers needs the trust value of a remote peer, it contacts all its score managers to query the actual trust value. The majority vote is used to filter out the values from malicious or broken managers.

EigenTrust is an example of TMS using global trust values, which is not common in a distributed TMS. It also uses the full aggregation scheme which tries to use all available information. This and the existence of several score managers increase the network load connected with functions of TMS. The feedbacks are verified on the principle of a good provider is a good evaluator and the calculation is based on transitive trust. The disadvantage is the necessity of pre-trusted peers; without them the system is not able to deal even with simple malicious techniques.

3.3 PeerTrust

PeerTrust [62] defines the general trust metric as a combination of five factors: (1) feedbacks from other peers, (2) the credibility of feedback sources, (3) the total number of transactions, (4) the transaction context factor, and (5) the community context factor. The first three factors are common in many TMSs; the transaction context factor can be used for discriminating mission critical transactions from less or non-critical ones. It can be seen as a simplified version of risk management. And the community context factor is used for addressing community-related characteristic and vulnerabilities.

Let *I(u)* denote the total number of transactions performed by peer *u* with other peers in a recent time window, *p(u,i)* denote the other participating peer in peer *u*'s *i*th transaction, *S(u,i)* denote the normalized amount of satisfaction peer *u* receives from *p(u,i)* in *i*ts ith transaction and *Cr(v)* denote the credibility of the feedback submitted by *v*. Then the basic form of the general metric without the transaction context factor and the community context factor is calculated using formula (1).

$$T(u)=\sum_{i=1}^{I(u)} S(u,i)\cdot Cr(p(u,i)) \tag{1}$$

The most important is the function *Cr(v)*. The authors propose two different approaches to credibility measurement. The first one, called trust value metric (T_{TVM}), uses trust values recursively as peers' credibility. The function *Cr(v)* for T_{TVM} is defined in formula (2).

$$Cr(p(u,i)) = \frac{T(p(u,i))}{\sum_{j=1}^{I(u)} T(p(u,j))} \tag{2}$$

The second one, called personalized similarity metric (T_{PSM}), uses the similarity of two feedback vectors. For peer *v* the recommendation from peer *w* is as trustworthy as similar were *w*' and *v*' recommendations in past. Unlike the T_{TVM}, this metric reflects a peer *w*'s subjective point of view. Peer *v* trusts peers which created similar recommendations and hopes that this similarity will continue. Let *I(u,v)* denote the total number of transactions performed by *u* with peer *v* and *IJS(v,w)* denote the set of peers that have interacted with both peer *v* and *w*. Than the credibility for T_{PSM} metric is defined in formulas (3) and (4).

$$Cr(p(u,i),w) = \frac{\mathrm{Sim}(p(u,i),w)}{\sum_{j=1}^{I(u)} \mathrm{Sim}(p(u,j),w)} \tag{3}$$

where

$$\mathrm{Sim}(v,w) = 1 - \sqrt{\frac{\sum_{x \in IJS(v,w)} \left(\frac{\sum_{i=1}^{I(x,v)} S(x,i)}{I(x,v)} - \frac{\sum_{i=1}^{I(x,w)} S(x,i)}{I(x,w)} \right)^2}{|IJS(v,w)|}} \tag{4}$$

PeerTrust also offers two implementations strategies: dynamic (DTC) and approximate (ATC) computation. The dynamic computation uses fresh data and is very expensive since a peer needs to retrieve actual trust data of all peers in the network. The approximate computation provides a more cost-effective algorithm by using a trust cache. PeerTrust is actually a collection of four techniques as each of two credibility measurements has two different implementations.

According to our classification, the TVM metric uses the global type of rating while PSM uses the personalized rating. The feedback verification type is the good provider in case of TVM and the personal experience in PSM. Both metrics try to implement the full aggregation scheme, although in case of ATC some data are older than others. And the calculation is based on the generalized trust metric.

PeerTrust introduces a general metric for calculation trust in a distributed environment and offers several different approaches of how to use it. The presented comparative analysis of all proposed approaches is one of the significant contributions of this work. The authors perform several simulations to demonstrate the feasibility, effectiveness and benefits of each approach. Unfortunately, the effective, distributed and secure manner of implementing it in a real P2P application was not described.

3.4 Lee2005

The interesting technique was proposed by Lee et al. in [63]. It was designed into file-sharing P2P networks and unlike other techniques it distinguishes a reputation for peers and for files. The separated reputation for peers and for files has several advantages. First, it prevents a malicious peer from sharing recognized bogus files even if it changes its identity. Second, it allows newcomers to share an honest file even if they do not earn trust yet.

Each file in the system has its own file reputation manager which keeps information about all versions of the file, the owner and the file reputation. The reputation is composed from a number of positive and negative recommendations. According to these values the file is put into one of three categories: trustworthy, untrustworthy and unknown. Only trustworthy and unknown files are taken into account for download.

After successful download, the downloading peer should evaluate the file by sending its opinion to the file reputation manager. The opinions are treated differently depending on the peer reputation of the source. The peer reputation is kept on a peer reputation manager. Similarly to the file reputation, the peer reputation is composed from a number of positive and negative recommendations which are simply summations of values of the files offered by the given peer. The procedure of categorizing the peer reputation into trustworthy, untrustworthy and unknown is performed as in case of file reputation and only recommendations from trustworthy peers are taken into account fully.

This work presents an attractive idea to combine a reputation of the provided service and a reputation of the service provider. This approach naturally solves several problems connected with traditional methods. Unfortunately, its implementation is limited only on the data-sharing network in which there is an assumption that one instance of the service exists on more peers. There are also several open problems which the authors do not address, for instance, the malicious peers can lie about their files and provide a bogus file instead of indexed and verified one. There is no defence against malicious reputation managers and so on.

In our taxonomy, the system uses global rating and performs the full aggregation scheme. As the feedback verification the good provider is used and calculation is based on summation of negative and positive feedbacks from trustworthy peers.

3.5 PET

PET is a personalized trust model designed by Liang and Shi [64]. The trust model distinguishes a reputation as accumulative assessment of the long-term behaviour and a risk as the opinion of the short-term behaviour. The trustworthiness

is directly derived from these two parts, as shown in figure 1. W_{Re} a W_{Ri} are weights of reputation and risk respectively.

The reputation is also calculated from two parts: a recommendation and interaction-derived information. The recommendation (E_r) is the average value of feedbacks from other peers and the interaction-derived information (I_r) reflects the local experience. W_{Er} and W_{Ir} are the corresponding weights. The calculation of E_r does not reflect different trustworthiness of feedback sources and all feedbacks have an equal weight. The authors avoid the concept of transitive trust for two reasons. First, they do not believe that an honest service provider must be an honest feedback source as well. Second, they pointed out the increased load of the system caused by examining transitive trust.

The risk is derived only from a local experience and it is normalized to the worse case, i.e., the sum of all bad services received from this peer divided by the sum of the worst possible results of all services received from this peer during the last time interval. Its purpose is to perceive sudden changes in peer behaviour.

The risk represents a novel approach in TMSs. It should reflect the suspicious patterns in peers' behaviour and warn that the calculated reputation can be manipulated. Unfortunately, denoting the risk only as a short-term opinion cannot react on all attempts to manipulate the reputation.

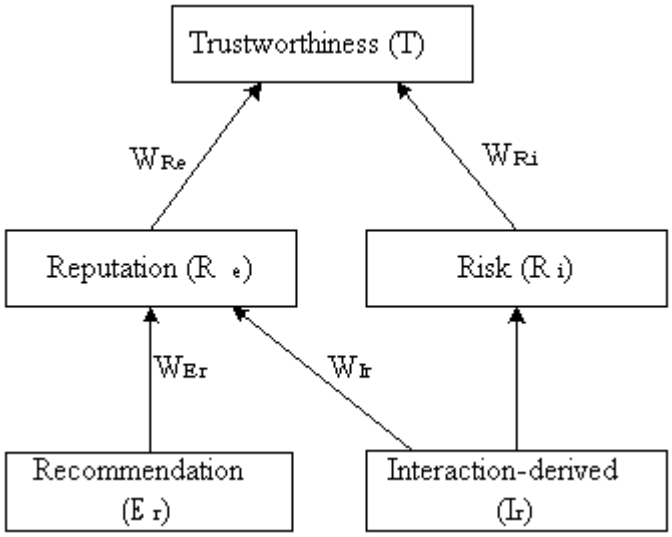


Fig. 1: Derivation of trustworthiness in PET

3.6 Scrivener

Scrivener [65] is based on the idea of pairwise exchange content between overlay participants. A Scrivener node maintains a relationship with each of its overlay neighbours. These relationships keep a history of interactions between

involved peers expressed by two values on both sides: credit and confidence. The credit is the difference between the amount of data sent to the amount of data received; the negative value is called debt. The request for downloading data is granted only if the requester's debt is below a certain limit. The confidence value reflects the reliability of the remote peer in providing services and transferring request. The unreliable peers can be expelled from neighbours list in an overlay network and replaced by another peer. The credit limit for each peer is derived from its confidence value.

In order to allow peers to join into the network, each newcomer has a small initial credit from that peers which choose it as a neighbour. However, it does not obtain any credit from peers that it chooses as neighbours. This prevents malicious peers from constantly selecting new neighbours and abusing the initial credit limit.

The basic protocol allows the transaction only between overlay neighbours. But it is most unlikely that the requested data can be found on one of them, especially in the network with a large content set. Scrivener introduces a strategy called transitive trade, which identifies a credit path from a source peer to the peer that is the owner of the requested data. After the credit path is identified, the credit can be rearranged so that the payment from the source peer arrives to the data provider.

Let the credit path be composed from peers *A* to *Z*, where *A* has a relationship with *B*, *B* with *C* and so on. The first message called *path discovery* is sent along the path from *A*. Each node has to pay for this message to the next peer in the path and at the same time decrease its confidence in it. After the message arrives to *Z*, the confirmation message is transmitted directly to *A*. *A* now can route request messages for each chunk of the data along the credit path and receives the data directly from *Z*. The same payment policy as in case of the *path discover* message is applied. A final message, announcing that *A* successfully downloaded requested data, is routed along the credit path again and causes each peer to increase the confidence of its successor to compensate the reduction in the first step and gain an additional confidence as a result of a successful transaction.

In this protocol, each participating peer has an incentive to cooperate. *Z* wants to be credited for transmitting all chunks and intermediate peers do not want to lose confidence of their predecessors in the credit path. Each peer also has to pay for each request even if it is not satisfied. This discourages flooding requests into the system.

Scrivener is primary targeted to free-riders and thus it only partially addresses other malicious behaviours. Unlike other TMSs, it is closely attached to overlay layer and encourages the message passing as well. The credit path is established through the reliable peers. The authors suppose that the credit and confidence value is hold in persistent storage and the unreliable peers can be excluded from the network

definitively.

Scrivener does not fully fit into our taxonomy. The system uses the personalized type of rating. There is no feedback aggregation because each peer uses only local experience with its overlay neighbours. Neither the feedback verification is needed. The calculation is based on the amount of sent/received data and the success rate of the requests sent through the peer.

3.7 TrustGuard

TrustGuard [66] is a framework for building distributed TMSs presented by Srivatsa, Xiong and Liu. The system supposes the existence of decentralized feedback storage (e.g. DHT based protocol). If a peer wants to transact with another peer, the following sequence of actions is performed: (1) feedback collection to collect all feedback towards queried peer, (2) dishonest feedback filter to filter out untrustworthy feedbacks, (3) feedback aggregation to compute trust value from obtained feedbacks, (4) strategic oscillation guard to deal with strategic behaviour of malicious peer, (5) trust-based peer selection to use the trust value for decision which peer is suitable for transaction, (6) transaction proof exchange, (7) transaction execution and (8) feedback submission. After the feedback is submitted, the peer responsible for storing feedback is able to verify that the feedback originates from a real transaction between two peers. The authors identified three critical components and proposed solutions for each of them. These components are: strategic oscillation guard, fake transaction detection and dishonest feedback filter.

The aim of the strategic oscillation guard is to combat malicious oscillation behaviour. For instance, a malicious peer may behave non-maliciously until it achieves a good reputation, then behaves maliciously, after losing its good reputation returns to its non-maliciousness and so on. The strategies oscillation guard takes into account the reputation history and tries to detect and suppress such behaviour.

The next critical component is the fake transaction detection. In a typical transaction-based feedback system, both participants have an opportunity to submit feedbacks about each other after each transaction. But malicious peers may flood numerous fictitious feedbacks about other peers without realizing transaction with them. The purpose of these false feedbacks is to damage the reputation of honest peers. In TrustGuard each feedback is bind to transaction through transaction proofs. In other words, the malicious peers cannot evaluate fictitious transactions and feedbacks between peers are stored if and only if these peers indeed transacted with each other.

The fake transaction detection component uses a public key cryptography based scheme. Let that every peer n has an associated public-private key pair, namely, $<PK_n,SK_n>$. TrustGuard assumes the existence of a trusted authority which binds key pairs to peers. Every peer is able to generate a transaction proof which

contains transaction description and time stamp signed by its private key. These transaction proofs are exchanged between peers before the actual transaction takes place. If the exchange fails, an honest peer would continue in the transaction. Nevertheless, if the exchange succeeds and one of the peers refuses to participate in the transaction, both peers still can evaluate transaction that never actually happens.

The transaction proofs have to be exchanged atomically; that is, either both participants have a transaction proof from other or none of the proofs is exchanged. The Optimistic Fair-Exchange Protocol was proposed to achieve this aim. This protocol guarantees fair-exchange of two electronic items between two mutually distrusting parties by utilizing trusted third parties. However, these parties are involved to only such exchanges that results into conflict. Assuming that most of the peers are honest, the trusted parties are hopefully involved infrequently.

The last component, the dishonest feedback filter, introduces the credibility factor which is computed in the same way as in PeerTrust described in the section 3.3. Therefore, two methods are offered: trust-value based credibility measure (TVM) and personalized similarity measure (PSM).

We consider strategic oscillation guard as the main contribution of this work. The fake transaction detection component requires trusted third parties, complicated protocol for fair-exchange and does not eliminate fake transactions completely. The dishonest feedback filter component does not come up with any new ideas and uses PeerTrust model. On the other hand, TrustGuard is the first TMS dealing explicitly with strategic malicious behaviour and proposes the ideas which were later used by other researches.

3.8 P2PRep

P2PRep protocol [67] has been proposed for unstructured P2P network. The basic sequence of operation is simple. When a peer wants to use some network resource, it (1) broadcasts query for resource location and receives a list of possible resource providers, (2) polls the network about the reputation of the providers, (3) verifies the received votes and (4) aggregates them into a reputation value. This value is synthesized with the local reputation representing direct interactions with the remote peer.

The security of this protocol is guaranteed by two mechanisms. First, the poll query contains a public key and all replies have to be encrypted by this key. Only an initiator of the query has the corresponding private key. This protects the identity of the responder and the data integrity. Second, the vote verification process randomly chooses some of the votes and sends a vote verification message to the address associated with these votes. This ensures that the vote truly originated from the corresponding address. However, it does not prevent malicious peers from creating

fake feedbacks towards honest peers or lying about the results of their transactions. Hence there is no feedback verification in a sense of checking the feedback quality. This makes the P2PRep vulnerable to malicious collectives.

3.9 NICE

NICE is a cooperative framework for implementing distributed application over the Internet developed on the University of Maryland. Sherwood et al. [68] proposed a protocol in context of the NICE system for efficiently storing peers' opinions in a completely decentralized manner and identifying non-cooperative peers.

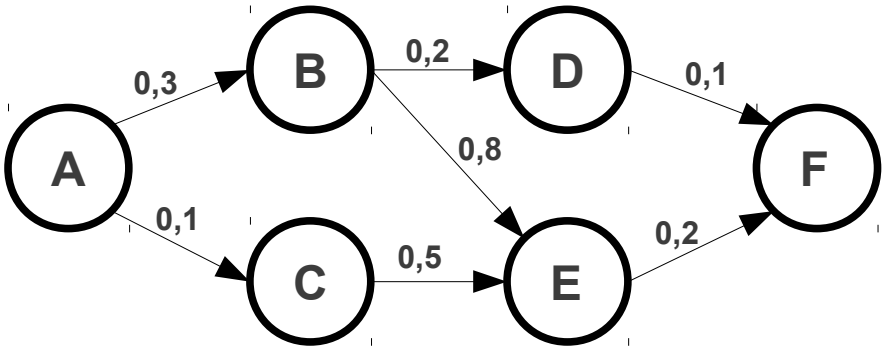


Fig. 2: Example of trust graph.

For each transaction, both involved peers produce a signed statement (called cookie) about the quality of the transaction. The peers send these cookies to each other and store them in the permanent storage. Later, the peer can use these cookies to prove its trustworthiness. The protocol uses a weighted directed graph called the trust graph. The vertices in the trust graph correspond with the peers in the network and there is an edge directed from peer *A* to peer *B* if and only if peer *B* hold a cookie from peer *A*. The weight of the edge denotes the quality of the past transactions between peer *B* and *A* included in the cookie. The example of the trust graph is shown in Figure 2.

Let peer *A* wants to communicate with peer *B*, these peers had prior transactions, hence the cookie exists and its value can be used as a *B*'s reputation. A more interesting case is when peer *A* wants to communicate with peer *F*. Because there are no prior transactions, F's reputation is calculated using the trust path. A trust path is an oriented path in the trust graph. There can be more trust paths. For instance, in Figure 2 the possible trust paths from A to F are: *A-B-D-F*, *A-B-E-F* or *A-C-E-F*. The quality of the path is called "strength". The authors propose two methods of calculating the strength of the path: (1) as a minimum valued edge along the path, and (2) the product of all edges along the path. And they also propose two methods of calculating the reputation according the strength of the trust paths: (1) it is the

strength of the strongest path or (2) weighted sum of strongest disjoint path.

To complete the protocol, we need a procedure how to find trust paths between given peers. One possible solution is flooding query through the network, but it is extremely inefficient. The authors propose modified flood-based algorithm. Whenever a peer receives a cookie from other peer, it receives a digest of all other cookies at the remote peer. These digests is used to optimize forwarding queries.

Other issue is that low-valued transactions are potentially not recorded in the system. The cookie evaluating peer *B* is stored on peer *B*, and this peer has no motivation to store and distribute cookies containing negative recommendation to it. The authors propose storing negative cookies on the issuer. If the peer *B* wants to interact with peer *A*, it can initiate a search for *B*'s negative cookies. This search follows high trusted edges from *A*. If this procedure finds a sufficient number of negative cookies, peer *B* is considered untrustworthy.

This framework provides a low overhead information storage and search algorithm usable in unstructured P2P network. The main idea that every peer holds only data beneficial to them hence they are motivated to participate on the TMS. The disadvantage is a separate algorithm for dealing with negative recommendations which makes the system susceptible to malicious collectives.

3.10 Credence

Credence [69] is a TMS developed for the Gnutella file-sharing network and its primary goal is the defence against file pollution. Unlike in other TMSs, reputation is connected with objects (files) shared in the network. In Gnutella style file-sharing networks the peers want to download a file send a search query containing required file's attributes. As a result it receives a list of files matching its query with hashes of file contents and meta-data describing the files. The user picks up one of the file to download. Credence should be able to guarantee that a file with a given hash has desired attributes.

After a file is downloaded, a user can manually enter positive or negative vote indicating the authenticity of a downloaded file. Each vote is a signed tuple containing a file content hash, an evaluation and a time-stamp. A peer evaluating file's authenticity actively queries the network to find and collect a sample of relevant votes. The final reputation is formulated by taking a weighted average of obtained votes. The weight assigned to each vote depends on the statistical correlation between other votes originated from the same remote peer and votes created by a local peer. The authors assume that each peer keeps a *vote database* containing received and emitted votes. This database is used for satisfying vote queries from other peers and for a calculation of correlation. The peers also hold a *correlation database* which is periodically recalculated according to *vote database*.

The system guarantees that a file with a given hash has desired attributes. But it does not help to choose peers which share this file reliably. The malicious peers can provide bogus files instead of files announced in the search result. Although such bogus files are quickly recognized (hashes do not match) this can be done only after the downloading is complete.

3.11 Multilevel Reputation System

The authors of the TMS described in [70] introduced the concept of reputation levels. The reputation levels are attached to peers and classify them into different classes. The system enforces the access control rule: a peer can use the resources only from peers on the same or lower reputation level. Obviously, each peer will prefer the most reputable sources which are the peers on the same level. It means that the peer is motivated to acquire high reputation not only for distributing its resources but also for an opportunity to download from more trustworthy peers.

The system requires a central component called Central Computation and Enforcement Agent (CCEA), which computes the peers' reputation levels and enforce the access control rule. It means that all search requests must be processed by CCEA and after each transaction, both participants must report the outcome of the transactions to CCEA. Authors propose two schemes for reputation level calculation, which are Level Up Reputation Computation (LURC) and Level Keeping Reputation Computation (LKRC). First scheme LURC decides whether the reputation level of the peer should be increased. This scheme takes into account only the peer's contribution to peers with higher reputation level. If the level is not increased, the LKRC scheme is used to check whether the peer can keep its current reputation level. This depends on its contribution to the peer with the same reputation level.

At the very beginning, all peers are in the lowest reputation level. The previously described schemes do not allow any peer to increase its reputation level and the system would stay in this state forever. Nevertheless, random selections are performed and some of the peers are promoted. This procedure is run periodically until the number of peers in each level is up to threshold. The system works best if peers are equally distributed among all levels.

The proposed system elegantly solves the load balancing problem, described in section 4.1.1. The access rules in the multilevel reputation system guarantee that the requests are spread more uniformly across the network.

The main disadvantage is the necessity of the central component. Moreover, this component is heavily loaded by running computation for each peer. This central component makes many things easier than in a full decentralized TMS but creates a single point of failure and degrades the distributed system.

3.12 WTR

The system WTR [71] described by Bonnaire and Rosas uses DHT as a storage for peers' feedbacks. The DHT allows any peer to store its feedback and retrieve all feedbacks towards a given peer. The standard DHT algorithms have to be extended for replication to resists DDOS attacks and peer failures. The authors propose to use their previous recursive replication scheme [72].

The reputation for peer *A* at time *t* is calculated using formula (5).

$$R_t(A)=\frac{\sum_{i=0}^{m-1}\log(m-i+1)\times F_i^K(A)\times C_t(K)}{\sum_{i=0}^{m-1}\log(m-i+1)\times C_t(K)} \tag{5}$$

where $F_i^K(A)$ is the feedback of index *i* towards peer *A* emitted by peer *K*. To compute a reputation of peer *A*, the algorithm uses *m* more recent recommendations for node *A*. The expression *log(m-i+1)* is used as a sliding factor to give more weight to more recent feedbacks. And $C_t(K)$ represents the credibility of peer *K* as a recommendation source. The credibility of peer *K* is a discrete exponential function of $R_t(K)$ giving peers with a reputation higher than 0.5 much more credibility than the other ones.

Besides the reputation, WRT introduces the risk factor. The risk of peer *A* reflects the probability of how much the reputation corresponds with real peer behaviour. The high risk means that the reputation may be calculated inaccurately due to a low number of recommendations or fluctuation in peer behaviour. The computation of the risk $J_t(A)$ is composed from two factors T_1 and T_2.

$$T_1(A)=\alpha\left(1-\frac{r}{m}\right) \tag{6}$$

$$T_2(A)=4(1-\alpha)\frac{\sum_{i=0}^{k}(F_i(A)-\overline{F_i(A)})^2}{r} \tag{7}$$

T_1 is used to evaluate a number of recommendations and T_2 reflects the variance of the recommendations emitted by other peers. In formula (6) and (7), *r* is the number of available recommendations in a window of size *m* for node *A* and α is a dynamic parameter that allows to give more weight to T_1 or T_2. The final risk is summation of T_1 and T_2. The risk factor should help an application to eliminate peers with short history or suspicious changes in their behaviour.

The proposed system does not require any kind of centralized authority and it

is designed to structured P2P network. The reputation computation is quite simple using weighted average weighted by credibility and sliding factor. The risk factor reflects fluctuation in peers' behaviour similarly to strategic oscillation guard in TrustGuard [66]. Moreover, TrustGuard has much more complex mechanisms for that. WTR is inspired by previously published TMSs and implements some time-proven methods.

3.13 H-Trust

H-Trust [73] is inspired by H-Index [74] aggregation approach. H-Trust scheme is implemented in five phases: trust recording phase, local trust evaluation phase, trust query phase, spatial-temporal update phase and group reputation evaluation phase. In the trust query phase the whole network is queried the feedbacks to the particular peers, but only peers with a high credibility are taken into account in the reputation evaluation phase. In H-Trust these peers are called qualified peers.

H-Trust uses a simple algorithm to compute the recommender's credibility. If the recommendation proves to be truthful, the recommender's credibility increases and vice versa. There is only one way to prove recommendation credibility - using a resource from the recommended peer. Therefore, after each transaction, not only the reputation towards the target peer is changed, but also the credibility of the peers which recommended it is revised. The credibility of all recommenders is changing, not only of the qualified ones. Due to this fact, the non-qualified peers can prove their honesty.

The H-Trust aggregation scheme is described by the following statement: A peer *i* has trust rating $T_{i,j} = H$ towards peer *j* if *H* of the qualified *N* peers have at least trust rating score *H* towards peer *j*, and the other *(N-H)* peers have at most trust rating score *H* towards the peer *j*. If there is no exact H-point, the approximate rank value is used.

The system introduces a new calculation approach used previously to quantify scientific researcher papers. This approach seems to be effective in a P2P environment too. It reflects the reputations of the recommender and the strenght of the recommendation. The peers' credibility is not directly used in H-Index computation, it serves only for distinguishing between qualified and non-qualified peers.

3.14 Summary

Many papers attempting to solve building secure P2P networks have been published in recent years. In the previous sections, we tried to describe the most known of them. We focus on systems which introduced some new ideas and push knowledge forward, hence it should not be considered as a complete list of all published TMSs.

We summarized all described system in table 4. This table give us a basic idea of different aspects of each TMS but cannot decide which system is better. It is difficult to compare individual methods because there are not fixed criteria which can be used to measure the efficiency of the reputation management. We have only a vague notion of 'trusted P2P network'. The TMSs have been built under the premise that this notion is well understood. Previous researchers use the simulations to prove the potential benefit of their proposal. These simulations are often oriented only towards the ratio of successful and failed transactions; in some cases the communication overhead and response time are also taken into account. The individual simulation models are different, simulate different communication patterns and malicious behaviour. And only a small number of competitive systems is included in the simulation, if any. As a result of this, the output of these simulations cannot be used for comparison with other systems.

Other possibility is to carefully analyse individual methods against described attacks and try to formally determine their resistance against these attacks. Such approach has been used in [58], [75] or [76] and has some limitations. For instance, we can decide whether the system is vulnerable against a certain kind of attack because authors do not implement any protection mechanism; however, we cannot compare the efficiency of two systems with the different protection mechanisms. Additionally, each application environment is different and different threats are possible. Therefore, the efficiency of TMS should be considered in the context of application.

System	Type of rating	Feedback aggregation	Feedback verification	Calculation
EigenTrust [61]	G	F	GP	Weighted average.
$PeerTrust_{TVM}$ [62]	G	F	GP	General trust metric.
$PeerTrust_{PSM}$ [62]	P	F	PE	General trust metric.
Lee2005 [63]	G	F	GP	Sum of negative and positive feedbacks from trustworthy peers.
PET	P	N/A	N	Average of feedbacks and risk calculated from local experience.
Scrivener [65]	P	N/A	N	The amount of sent/received data and the success rate of the requests.
TrustGuard [66]	G/P	N/A	GP/ PE	Strategic oscillation guard incorporating peer's history (Integral component) fluctuation (Derivative component) and credibility.
P2PRep [67]	P	S	N	Ordered weighted average.
NICE [68]	P	S	GP	The strong of the strongest trusted path or weighted sum of strongest disjoint trusted path.
Credence [69]	P	S	PE	Weighted average of votes.
Multilevel RS [70]	G	F	N	Reputation level is increased if there is enough transactions with a higher reputable peers or decreased if there is not enough transactions with the equally reputable peers.
WTR [71]	G	F	GP	Using two values: reputation and risk. The reputation is a weighted average and the risk reflects the number of feedbacks and fluctuation in peer behaviour.
H-Trust [73]	P	S	PE	H-Index [74]

Table 4: Basic classification of trust management systems. Type of rating: G - Global, P - Personalized, Feedback aggregation: F - Full, S - Selective, Feedback verification: N - None, GP - Good provider, PE - Personal experience, GE - Global experience.

4 Attacks against TMS itself

In section 2.2 we described attacks targeted at the application layer and presented the defence mechanisms based on reputation management in section 3. Unfortunately, these mechanisms have their own weaknesses which can be exploited by treacherous peers. These peers can use sophisticated strategies to circumvent TMS and maximize their malicious impact. In this section, we analyse these weaknesses and describe malicious strategies exploiting them.

The weaknesses can be classified into three categories: (1) unwanted side effects, (2) individual strategies, and (3) collective strategies. This chapter describes the known malicious strategies and suggests three new malicious tactics targeted to the current TMSs.

4.1 Unwanted side effects

The issues related to this category can be seen in the network without malicious peers. They are caused by the function of the TMS itself. The literature describe two issues which are part of this category: (1) load balancing problem, and (2) cold start problem. Generally, both problems can be relatively easily solved, but it is necessary to pay attention to them when a new TMS is designed.

4.1.1 Load balancing problem

We have already mentioned the load balancing problem in DHT in section 2.1, but the use of wrongly designed TMSs can make it worse. Suppose that the TMS uses the continuous value for the reputation and the global rating. It is very likely that one of the providers has a higher reputation than others and all other peers prefer cooperating with it. This results into a situation when one top-rated peer is overloaded and other reliable peers with a slightly lower reputation are idle. Similar situation can occur in TMSs with personalized rating. All requests are targeted to the group of most rated peers while there is a number of equally reliable providers.

The TMSs have to be carefully designed to avoid this problem. One possible solution supposes that peers do not automatically choose the most rated partner, but randomly choose one above a defined threshold. Unfortunately, this rule is difficult to enforce, some peers can ignore it and still prefer the highest-rated peers for selfish reason - maximizing its downloading success. Other solution is to forbid this situation in the TMS itself. For instance, TMSs can use discrete reputation value instead of continuous one, or service providers can limit access to their services according to a requester reputation, like in [77].

4.1.2 Cold start

The problem of newly connected peers is its initial reputation. It should be zero, because the system does not have any previous experience with them. But the peers with a zero-reputation have only a little chance to be chosen for cooperation and cannot prove its trustworthiness. For such peers the process of building the reputation is very slow. This is called a cold start problem. In the worst case scenario, the newcomers can stack in a zero-reputation state forever.

In order to prevent this problem, TMSs should provide an initial reputation to all newcomers. But this can be misused in attacks called whitewashing in which peers leave the network after gaining bad reputation and connect again with a new identity with a fresh initial reputation. Whitewashing is discussed in detail in section 4.2.1. The most TMSs use only small initial reputation as a trade-off between cold start problem and making the system attractive for whitewashers.

Other possibility is to allow newcomers to gain a reputation in another way, for instance, on the basis of the social relationships between users. The user already connected to the network can send invitations to his friends and provides them an initial reputation [78].

4.2 Individual strategies

Let us suppose that each malicious peer works alone without cooperation with others. There are several strategies which it can utilize to circumvent TMSs. Most of the TMSs can detect these attempts easily and such peers have only a little chance to succeed. The individual strategies pose a danger if they are used together with other strategy. Malicious peers can use them as a part of sophisticated collaborative strategies described in section 4.3.

4.2.1 Whitewashing

This strategy consists in periodical joining and leaving the network. The malicious peer after gaining bad reputation leaves the network and joins again with a new identity. All negative recommendations bound with its previous identity become useless and the peer can start over. This problem has been discussed by many researchers [79], [58] or [80].

The only possibility to fight against whitewashers on the trust management layer is to provide a very small initial reputation to newcomers and deal with the cold start problem (4.1.2). However, the general solution should guarantee that the peers keep the same identifier during their lifetime. It should be difficult for peers to generate a new identity easily. The same problem is solved in the defence techniques against Sybil attack discussed in section 2.1.1. Although these techniques are not completely reliable, they can significantly improve the security.

4.2.2 False meta-data

The fundamental principle of many P2P applications is sharing resources. The peers publish resources together with meta-data describing them. Typical P2P application is equipped with a mechanism which allows to search in meta-data and identifies peers which offer resources connected with them. However, there is no guarantee that the meta-data describe the resources truthfully. Malicious peers can insert false attractive information into the meta-data describing their bogus resources to increase the demands for them. For instance, in the P2P music file sharing network, the attacker most likely names its infected file like some very popular song. Due to the fact that 10% of the most popular files create 90% of all transactions in file sharing P2P networks [43], the attacker dramatically increases the probability that the file will be downloaded. This is a basic, simple and very effective malicious strategy. Some of the TMSs are specifically designed to fight against this issue (see section 3.4 or 3.10) but they are limited to file-sharing P2P networks.

4.2.3 Camouflage

The malicious peers that are aware of the presence of the TMS can provide a few honest resources. These resources allow them to maintain higher reputation even if they provide also some bogus resources. Their reputation will be probably smaller than the reputation of honest peers but it can be enough to trick part of the honest peers. However, if the honest peer has a choice between several providers of the same resource, it probably chooses the most trustful one. Therefore, the camouflaged malicious peers have to offer some unique resource to be chosen despite the lower reputation. The combination with false meta-data strategy is suggested.

The basic scheme of this strategy is depicted in figure 3a. The honest peers evaluate the camouflage peer alternately as honest and malicious. There can be many variants of this strategy, differing in the ratio of honest and bogus services or the period between changing behaviour. In some literature, the variant of this strategy is called Traitors [57], [58] or [81].

4.3 Collective strategies

Malicious peers have a significantly higher chance to succeed if they work in a cooperative manner and coordinate their effort to trick the TMS. Many papers designated malicious collusions as the biggest treat for P2P applications [82], [58] or [75]. However, only two collective strategies are considered in the most papers. Their names are not stabilized, so we use a designation (1) full collusion and (2) spies for them. In the full collusion the malicious peers are providing false positive recommendations to other malicious peers and the spies is a strategy when a part of malicious peers behave honestly and recommend the second part of peers which

perform malicious activity.

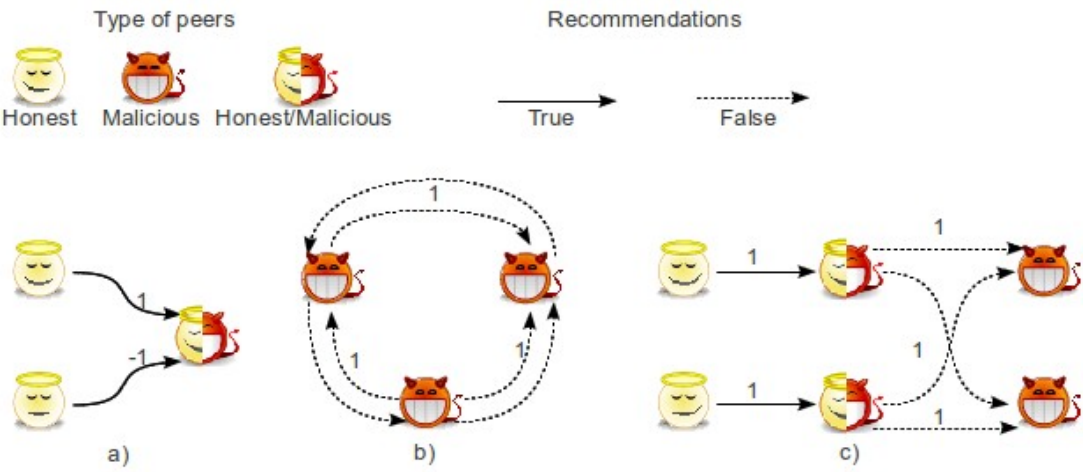


Fig. 3: Basic malicious strategies. a) camouflage, b) full collusion, c) spies.

4.3.1 Full collusion

It is the basic collective strategy. All members of a malicious collective provide bogus resources and create false positive recommendations to all other members of the collective. The purpose of these recommendations is to artificially increase the reputation of other malicious peers. Figure 3b shows full collusion with three malicious peers.

This strategy is effective only in TMSs which do not verify the credibility of feedbacks. The feedbacks are generated by malicious peers and most TMSs should consider them as untrustworthy. On the other hand, these false recommendations can be very easily generated; therefore, it does not pose additional load for malicious peers. Malicious peers are able to generate a great number of such false recommendations which can jeopardize the proper function of TMSs.

4.3.2 Spies

In order to increase trustworthiness of their false recommendations, malicious peers can use several techniques dependant on algorithms used by TMSs. If the TMS assesses the credibility of the feedback source according to its reputation as a service provider, malicious peers can use strategy called Spies. The malicious collective is divided into two groups: spies and malicious. The spies provide honest services to earn a high reputation and simultaneously provide false positive recommendations to the malicious part of the collective. The recommendations between peers in the network with spies are shown in figure 3c.

The success of this strategy depends on the number of spies in the collective.

More spies are able to outvote the peers with negative experiences with malicious part of the collective. But even there this method is not faultless. The malicious part of the collective would have a large number of contradictory recommendations which should be considered suspicious and TMS can designate these peers as oscillated and discard them.

The honest transactions performed by spies implied extra load for malicious collective. The spies have to provide some resources which are beneficial for others. On the other hand, the spies assume that they are not punished for their false recommendations and can provide only as many honest services to keep its provider reputation.

4.4 Newly proposed malicious strategies

We analysed published TMSs and known malicious tactics carefully and suggest three new collective malicious strategies. Each strategy is designed into a particular type of TMS and tries to exploit its specific weakness.

4.4.1 Evaluator collusion

If the TMS assesses credibility of the feedback source according to the truthfulness of its previous feedbacks, malicious peers can try to trick the TMS by using the services from peers outside the collective and evaluating them correctly. These true feedbacks increase the credibility of malicious peers as recommenders and give more weight to their feedbacks towards other member of collective.

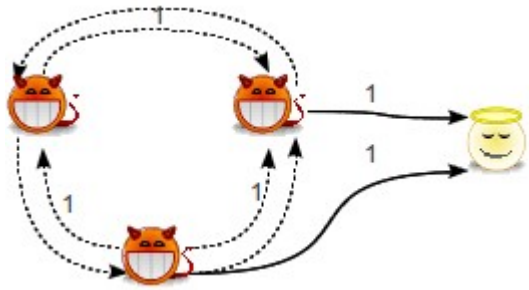


Fig. 4: Schema of evaluator collusion

Figure 4 shows the schema of this strategy. It is similar to full collusion except the truthful recommendations towards peers outside the collective. The effectiveness of this strategy depends on the ratio of truthful and faked recommendations. But the truthful recommendations are very disadvantageous for malicious peers for two reasons. First, they must be preceded by complete transactions outside the collective, which significantly increase the load of malicious peers. Second, the malicious peers have to evaluate these transactions honestly. This

can be a difficult task in some systems, except the objective criteria like bandwidth, file integrity or time to complete a request, the final evaluation is subjective and it is often left to the user behind the application.

The idea that malicious peers implement some robots which automatically evaluate all services in the network seems like a beneficial approach. But it can open a way to censorship when a collective of peers decide which service is good and which is bad.

4.4.2 Evaluator spies

This strategy is a combination of evaluator collusion and spies. The spies implement three techniques to maintain a high credibility as a feedback source: (1) they provide an honest service, (2) they use resources outside the collective and evaluate them correctly, and (3) they create positive recommendations towards other spies. The scheme depicted in figure 5 is similar to regular spies. Additionally, the evaluator spies create truthful recommendation outside the collective and between each other.

This strategy is designed into TMSs which assess credibility of the feedback source according the quality of the previous feedback, similarly to Evaluator collusion. Unlike the Evaluator collusion, it tries to minimize the number of truthful recommendations outside the collective because of the disadvantages discussed above. Some of them are replaced by providing honest services, like in the spies strategy, and creating false recommendations between spies which can be easily generated.

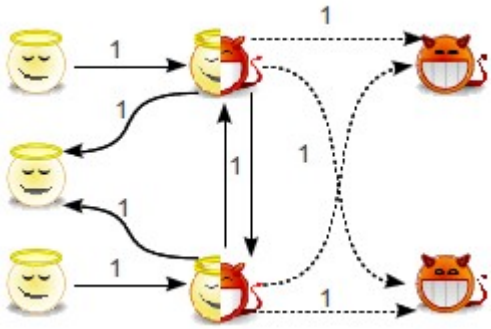


Fig. 5: Scheme of evaluator spies

4.4.3 Malicious spies

The malicious spies strategy is a slight modification of the previous strategy. It is based on the idea that spies do not require a high reputation as resource providers. They can provide bogus resources and generate negative recommendations between each other. These recommendations are still truthful and should increase their credibility as a feedback source. In this way, the attacker can eliminate the need

of providing honest services to trick the TMS.

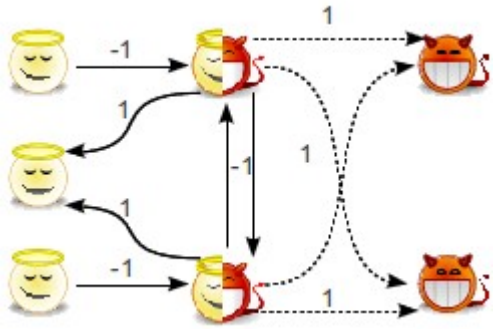


Fig. 6: Scheme of malicious spies

The scheme of this strategy is shown in figure 6. There is only one modification compared to evaluator spies. The spies provide bogus services; therefore, they have negative recommendations from honest peers and from each other.

This strategy is not suitable to all TMSs. It assumes that the TMS has no correlation between provider and evaluator rating. Otherwise, it is counterproductive.

4.5 Summary

A bigger number of malicious peers using collective strategy causes a disaster for each P2P network. The authors of the most previously published TMSs expected only simple collective strategies and tried to make its system immune to them. But the attacker with detail knowledge of the internal functions of TMS can develop more sophisticated strategies targeted to the specific TMS. The strategies described in this chapter exploited some general principles used by several TMSs. Therefore, they are more dangerous than common collective strategies. Additionally, other modifications or combinations of these strategies are possible.

5 BubbleTrust

The previously described TMSs suffer several security deficiencies which prevent the use of full decentralized P2P architecture in more types of applications. Currently, the P2P architecture is mainly used in non secure sensitive applications or in applications in which the administrator has the full control over the end nodes. In the both scenarios the previously described TMSs are perfectly usable. In the first case, the attacker has not a motivation to deploy sophisticated malicious strategies because the potential benefit does not outweigh the cost. And in the second case, the probability of the peer's maliciousness is very low.

Imagine a full decentralized auction application similar to eBay. Such application cannot exist without proper security mechanisms because the vision of financial profit is very attractive for potential attackers. Even if the application operates on centralized servers (like eBay), the ensuring trust between sellers and buyers is questionable [56]. Our ultimate goal is to allow the deployment of P2P architecture in more secure sensitive applications. To accomplish this task, we decided to develop a new TMS which should be resistant even against sophisticated malicious strategies and at the same time usable in large P2P networks.

In our previous work [83], we presented a novel trust management system called BubbleTrust which uses some new approaches. The basic idea behind the BubbleTrust is the separation of a peer role into that of a resource provider and of a transaction evaluator. The peer is evaluated for both roles separately, hence the system is able to distinguish peers that provide honest resources but do not participate correctly in the TMS. This tactic is often used by members of malicious collectives as described in section 4.3. Using the BubbleTrust makes it less effective. Other important concept is a data management which ensures that malicious peers cannot create fake feedbacks towards honest peers or suppress unflattering feedbacks towards themselves or allied malicious peers.

The BubbleTrust is based on trust graph, which was first introduced in NICE [68]. In contrast, it does not try to find a trust path between a consumer and a provider, but it tries to involve as many relations as possible into the decision making process. It takes into account opinions of a great number of peers which have the strongest relations towards the queried peer. These relations create a trust bubble around the queried peer. BubbleTrust would not be possible to use without limiting the size of this bubble or using caches which significantly reduce the complexity of the algorithms.

5.1 Basic concept

Let us assume that the P2P network can be decomposed into a set of two-party transactions. In each transaction one party is designated as a consumer and

other as a provider. The provider is a peer which owns some resources and offers them to the public. The consumer is a peer which uses these resources. Each peer in a P2P network can act as both a provider and a consumer.

After each transaction, the consumer can express its satisfaction with the quality of the acquired resource and transaction parameters. On the basis of all transactions with a given peer the consumer can create an opinion about peer's reliability as a provider, this opinion is called provider rating.

The consumer publishes all opinions on the network. In other words, the consumer evaluates every provider, which it cooperates with, and makes this evaluation public accessible. From the TMS point of view the better notion for consumer is an evaluator. In the following text we use notion evaluator interchangeably with the notion consumer if we want to stress its evaluation function. Other peers download the provider rating from the evaluator and use it for its own calculation of the provider rating. But the foreign provider rating is not as trustworthy as locally created ratings. We use a notion recommendation for the provider rating acquired from other peers.

In the BubbleTrust, every peer has two ratings. First, it is the provider rating, which we defined above. The higher provider rating means that the peer is more likely to be a reliable resource provider. This rating corresponds to the notion of the reputation commonly used in the contemporary TMSs. Second, the evaluator rating is connected with the evaluation function of the peers. The opinions from the peers with higher evaluator ratings are more trustworthy than opinions from the peers with lower evaluator ratings; this rating is often called credibility.

Every peer creates both ratings locally towards each peer which has required resources. The primary purpose of these ratings is to help peers to make a decision whether a given peer is reliable for cooperation. If the peer acts as a consumer, it is looking for a provider with a higher provider rating to ensure that the transaction will be successful. If the peer acts as a provider, it prefers the consumer with a higher evaluator rating to ensure that the transaction will be correctly evaluated and this evaluation will be trustworthy for other peers in the network.

The provider rating originates from direct transactions with evaluated peers or is calculated from the recommendations acquired from other peers. The evaluator rating is calculated by comparing network experience with recommendations from evaluated peers. The calculations of both ratings influence each other. The calculation of the provider rating requires the evaluator ratings of all peers which evaluated the given peer. Analogously, the calculation of the evaluator rating requires the provider ratings of all peers which were evaluated by the given peer. These two observations give us a brief outline of the calculation algorithm which will be explained below. The system creates a bubble around the unknown peers which

contains the peers having references to them.

Most of the previously published trust managements used only one rating and this rating supplies the function of the both ratings in our system. The authors assumed that a quality provider should be a quality evaluator too and vice-versa. But this is not generally true. Especially the peers which are members of malicious collectives can break this assumption in an effort to advantage some other members of the collective. The separation of ratings facilitates the detection of such behaviour.

5.2 Calculation

Before each transaction, the consumer needs the provider ratings of the all possible providers; on the basis of these ratings the consumer chooses the most reliable partner for cooperation. There are two possibilities of how the consumer can get provider ratings. First, both peers have already cooperated in the past and the consumer has the rating created by itself. Second, the consumer has never cooperated with the remote peer and has to ask other peers for recommendations. These recommendations are used to compute the required rating.

Similar situation occurs on the provider's side. It needs the evaluator rating of the consumer which asks for its resource. The evaluator ratings are always calculated from the recommendations originated from the given evaluator. In this section, we explain the calculation of both ratings. We start with the provider ratings; the calculation of the evaluator ratings is analogous. At first, we give several definitions:

Definition: Provider rating (V_P) is a real value in a range [-1,1] which expresses an opinion about the provider reliability. The positive value expresses the satisfaction and negative value dissatisfaction. The absolute value represents the strength of this opinion.

Definition: Evaluator rating (V_E) is a real value in the range [0,1] which express an opinion about the quality of ratings offered by the evaluator. The higher value means more trustworthy rating.

We choose different ranges to stress the different interpretation of both values. The provider rating expresses two states: satisfaction and dissatisfaction and certainty about belonging into these states, whereas the evaluator rating expresses the quality of the opinion. The recommendation from an evaluator towards a provider is stored as a relation.

Definition: The relation is a 5-tuple $r = <E, P, v, w, t>$ where E is a transaction evaluator, P is a transaction provider, v is a provider rating, w is a transactions weight and t is a time of the last modification. We use a notation $r.E$, $r.P$, $r.v$, $r.w$ and $r.t$ for

elements in the relation *r*.

The relation can originate or be altered only after the transaction takes place between involved peers. The storing and seeking relations in the network is described in section 5.4. Meanwhile, we assume that all relations created in the network are available for every peer.

Definition: The transactions weight expresses the consumer's opinion about the importance of the transactions between the consumer and the provider. This opinion is a real value in a range [0,1].

The transactions weight gives the evaluator the opportunity to express the importance of the transactions, for instance on the basis of the size or character of the data. The importance of the whole relation is calculated from the transaction weight and the time of the last modification.

Definition: Weight function (W) determines the weight of the each relation and is defined by the formula:

$$W(r) = f(r.t) * r.w \tag{8}$$

where f (x) : N → [0, 1] is a time function which maps the age of transaction into a range [0,1]. The design of time function will be discussed later.

The next question is how the evaluator rating influences the opinion originated from the evaluator. We define a two dimensional function, called provider function, which expresses this dependency.

Definition: The provider function accepts two arguments, the provider rating originated from one evaluator and the evaluator rating of this evaluator. The result is altered provider rating which takes into account the evaluator trustworthiness. The function has a form: $pv(x_1, x_2) : [-1, 1] \times [0, 1] \rightarrow [-1, 1]$ Where x_1 is a provider rating originated from the remote evaluator and x_2 is an evaluator rating of the remote evaluator. This function can have several interpretations, we analyse function requirements and provide possible interpretation in section 5.5.

Now we are able to give a formula to compute the provider rating. The basic idea is simple: The peer is a good provider if a majority of good evaluators agrees on it. The formula (9) takes into account the altered provider ratings and weights of all relations where the given peer acts as a provider. Let R is a set of relation such $r \in R$

and $r.P = A$, the provider rating of peer A, $V_P(A)$, is calculated according the formula:

$$V_P(A) = \frac{\sum_{r \in R} pv(r.v, V_E(r.E)) \cdot W(r)}{\sum_{r \in R} W(r)} \quad (9)$$

The formula for computation of the evaluator rating is very similar. First, we need the evaluator function.

Definition: The evaluator function accepts two arguments, the provider rating originated from a given evaluator and the reference provider rating. The reference provider rating reflects the majority opinion of other peers or local provider ratings if it is available. The result is an evaluator rating of the given evaluator which takes into account the difference between both ratings. The function has a form: $ev(x_1, x_2)$: $[-1, 1] \times [-1, 1] \rightarrow [0, 1]$. Where x_1 is a provider rating originated from the remote evaluator and x_2 is a reference provider rating.

Similarly to provider function, this function can have several interpretations. We analyse function requirements and provide possible interpretation in section 5.5.
The idea behind the evaluator rating: the peer is a good evaluator if a majority of its ratings correctly evaluates the providers. The formula (10) takes into account the calculated evaluator ratings and weights of all relations where the given peer acts as an evaluator.

Let R is a set of relation such $r \in R$ and $r.E = A$, the evaluator rating of peer A, $V_E(A)$, is calculated according the formula:

$$V_E(A) = \frac{\sum_{r \in R} ev(r.v, V_P(r.E)) \cdot W(r)}{\sum_{r \in R} W(r)} \quad (10)$$

The aim of the trust management is to calculate the provider rating and the evaluator rating for the given peers. In the proposed system those values are computed locally on each peer. The calculated values are used only for decision on the local peer and are not exported to other peers. It means that each peer has a unique view on the network and the trust values towards one peer can be different on the different peers, the similar approach is used for example in Fuzzy [67] or Core [84]. The opposite approach represents systems like EigenTrust [61] where the trust values are global: All peers share the same opinion towards others.

5.3 Basic algorithm

In this section, we describe the algorithms which implement the formulae (9) and (10). We demonstrate the algorithm which computes the provider rating; the calculation of the evaluator ratings is analogous. Both computations influence each other. We need to calculate evaluator ratings of all evaluators for a given peer if we want to calculate the provider rating for the given peer.

The basic algorithm works recursively; in each level it computes either provider ratings or evaluator ratings of all peers in the input set. The first level computes provider ratings; the second level computes evaluator ratings; the third provider ratings and so on. The sequence of computation is illustrated in figure 7.

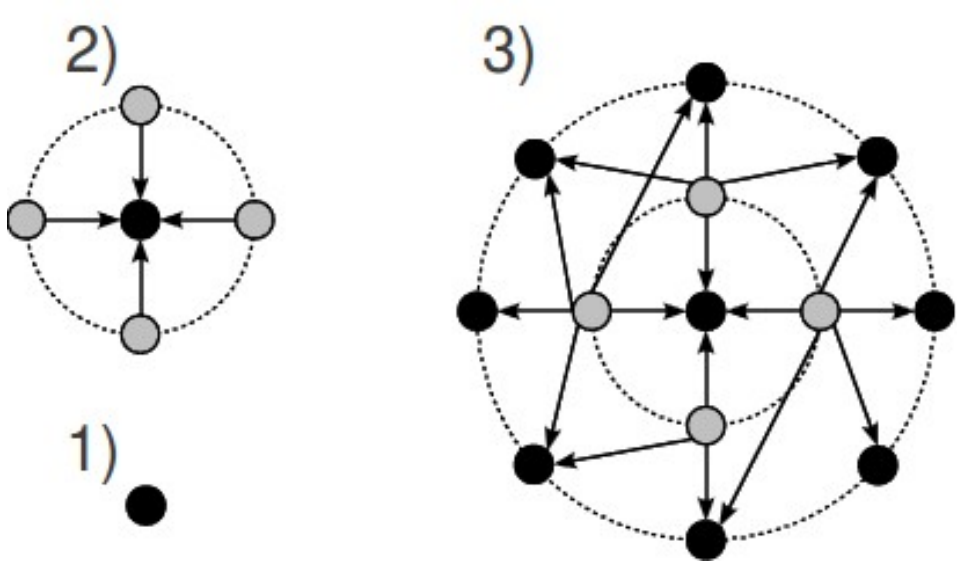


Fig. 7: Incrementally growing trust bubble. Black dots are peers in the provider role, grey dots are peers in the evaluator role and arrows represent feedbacks.

In each level the algorithm performs the following steps:

- Find evaluators (or providers) for all peers in the input set.
- Recursively computes the evaluator (or provider) ratings for new peers.
- Compute provider (or evaluator) ratings according the formulae (9) or (10).

```
node {
        VP = unknown|processing|[-1,1]
        VE = unknown|processing|[0,1]
        dividendP = 0;
        dividendE = 0;
        dividerP = 0;
        dividerE = 0;
}
```

Fig. 8: Data structure for one peer in BubbleTrust

The information about one peer is held in a data structure depicted in figure 8. The items V_P and V_E correspond to the provider rating and the evaluator rating. They can take the value *unknown*, *processing* or a number in the proper range. The newly discovered peers have this value set to *unknown* and the value *processing* means that the calculation is in progress. The data structure for the local peer has V_P and V_E set to 1, because every peer always trusts itself. The variables *dividendP, dividendE, dividerP* and *dividerE* are auxiliary variables.

The function calculating provider rating for group of peers is described in figure 9. This function calls three auxiliary functions: *get_relations(role*, *A) get_peers(role, S)* and *get_optimization(role, S)*. The *role* is either "provider" or "evaluator", A is a set of peers and S is a set of relations. The function *get_relations(role, A)* returns all relations *r* from the network where $r.E \in A$ if *role* is "evaluator" or $r.P \in A$ if *role* is "provider". This function queries the network to get required information and is described in section 5.4. The function *get_peers(role, S)* returns all peers *p* for which exists some relation *r* belongs to *S* where $r.E = p$ if *role* is "evaluator" or $r.P = p$ if *role* is "provider". The implementation of this function is trivial. And the last function *optimization(role, S)* implements optimization mechanisms which will be described further in the text. Meanwhile, let this function is empty.

The algorithm sets the provider ratings for all peers in the input set at once. Every peer is able to calculate ratings for a larger number of peers in one algorithm run. This is a typical situation when the peer has several possible providers and needs to know the ratings of all. The function *basic_evaluator_ratings(A)* is analogous, swaps the words evaluator and provider and uses the evaluator function instead of the provider function (line 15).

The algorithm visits each peer twice at the most, when it calculates a provider rating and when it calculates an evaluator rating. At the beginning all peers have the both ratings set to unknown. The function *basic_provider_ratings(A)* accepts only peers with unknown provider ratings (line 2) and as the first step their ratings are set to processing (line 3). This ensures that any recursive call of this function does not deal with these peers again. At the end of the function all peers which had unknown

provider ratings are set to numeric value (line 20 or 22). The algorithm also ignores the relations between peers in the input set (line 11) because these relations do not provide new information. The rules above ensure that the basic algorithm finishes after visiting all nodes in the network or if there are no relations from the visited nodes to the rest of the network.

```
Input: Input - Set of nodes.
Output: Set items VP for all nodes from the input set.

function basic_provider_ratings(Input) {
 1:  foreach p from Input {
 2:    if p.VP == unknown then {
 3:       p.VP = processing;
 4:       P = P + {p};
 5:    }
 6:  }
 7:  if empty(P) then return;
 8:  S = get_relations(provider,P);
 9:  optimization(evaluator,S);
10:  E = get_peers(evaluator,S);
11:  E = E \ P;
12:  basic_evaluator_ratings(E);
13:  foreach s from S {
14:    if s.evaluator.VE == processing then continue;
15:    s.provider.dividendP += pv(s.val,s.eval.VE) * W(s);
16:    s.provider.dividerP += W(s);
17:  }
18:  foreach p from P {
19:    if (p.dividerP != 0) then {
20:      p.VP = p.dividendP/p.dividerP;
21:    } else {
22:      p.VP = default_VP;
23:    }
24:  }
25: }
```

Fig. 9: Basic algorithm for calculation of provider rating.

In the practical application we cannot let the algorithm explore the whole network due to limited network performance and time requirements. Further in the text, we introduce several methods of reducing a number of visited nodes without a significant degradation of the results.

5.4 Data management

All TMS need to store a large amount of data containing a history of peers' behaviour, or more precisely, the opinion of other peers about behaviour of their transaction partners. Typically, only a recent history is held due to the storage limitation. The parameter *history_period* determines how long the network remembers the information about peers' behaviour.

In BubbleTrust, the history is held in the form of relations as defined in section 5.2. These relations are stored somewhere in the network and accessible through the function *get_relations* for each peer. The data management layer discussed in this section has to provide a secure way of creating, storing and looking up relations. First, we formulate four requirements on secure data management:

1. For each relation r, it is verifiable that the values in a relation ($r.P$, $r.v$, $r.w$ and $r.t$) are created by peer $r.E$. In other words, the relation can be created or modified only by the evaluator stated in the relation.
2. For each relation r, it is verifiable that peers $r.P$ and $r.E$ agreed on cooperation and this agreement precede $r.t$. In other words, the evaluator cannot create a relation towards a remote peer without its knowledge.
3. Only one relation can exist between evaluator and provider. This relation expresses the cumulative values from all previous transactions.
4. The function *get_relations* returns all relations matching the criteria and created or modified in the last *history_period*.

Requirements 1 and 2 need that every peer that takes part in the system has a unique unforgeable identifier. We assume that this identifier is the hash value of the peer's public key. It is convenient to use this identifier at P2P layer as well. We have already discussed issues connected with this approach in section 2.1.1.

Let (S_E, P_E) is the private/public key pair of peer E (evaluator) and (S_P,P_P) is the same for peer P (provider). The public keys are freely distributed in the network. The following protocol describes the creation of the relation between them.

1. E -> P: Req(E,P,T_R)
2. P -> E: Ack(E,P,#(E,P,T_R)$_{SP}$)
3. E <-> P: Transaction
4. E: Create r=(A,B,R,W,T,#(A,B,T_R)$_{SP}$,#(r)$_{SE}$)

In the first step, the evaluator sends a request to the provider, this request contains the identification of both peers and an actual time-stamp T_R. It can also contain other transaction specific information which is not shown here. If the provider is willing to accept the request, it replies with an acknowledgement

containing the digital signature of the items in the request. The provider should refuse the request if the time-stamp in the request is significantly different from its local time or lower than in the previous transaction with the same peer. After the evaluator receives the acknowledgement and verifies the signature, the transaction can start. In the last step, the evaluator has a possibility to create a secured relation and express its satisfaction with transaction. The secured relation contains two extra items: (1) the digital signature of the request confirming that the provider agreed with transaction and (2) the digital signature of the whole relation confirming that the evaluator creates the relations. These two items allow verifying requirements 1 and 2.

The protocol above should ensure that each relation is preceded by the transaction between involved peers. Clearly, this is not true in a case when both peers are malicious. The group of malicious peers can create relations among them without limitations. We will analyse situations when either the provider or the evaluator is malicious.

If the provider is malicious, there are two possibilities of malicious activity. First, the provider does not send the correct acknowledgement in step 2. The evaluator should mainly verify the signature and whether it corresponds with the request. If the acknowledgement does not pass, the evaluator does not perform the transaction. Second, the bogus transaction is provided in step 3. In this case, the evaluator is able to create a relation with negative rating. The provider has no means of stopping it.

The situation with malicious evaluator is more complicated. The evaluator can receive the acknowledgement in step 2 but does not perform the transaction and skips to step 4. The information provided in such relation is completely fabricated and can damage the reputation of honest providers. The defence against this behaviour consists in checking of the evaluator rating before the acknowledgement is generated. The providers should not agree with transactions with poor evaluators for two reasons. First, the successful transaction does not increase its provider rating and, second, there is a probability that it does not even get a chance to prove its trustworthiness. Other vulnerability is that the malicious evaluator can use the acknowledgement from previous transaction and change the relation once created. The storage procedure described bellow should deal with such behaviour.

The next issue is how the relations are stored in the network. It is not suitable to store the relations on peers which are involved in them because the malicious peers can suppress some relations to improve their reputation. Instead, we use a storage based on the distributed hash table (DHT). In order to make the relations searchable by both participants, the basic implementation suppose that each relation is stored in the network twice under different keys, once under a key derived from the evaluator identifier and once under a key based on the provider identifier. Any

usable implementation has to be extended by the replication to prevent data loss in case of peer failure or maliciousness. This problem is more serious due to the fact that all relations from one evaluator or towards one provider are stored together on one peer. If this peer fails, the complete history of one remote peer is lost. There are several replication algorithms proposed in literature (e.g. [72], [85], [86] or [87]) which can be used. In the following text, we suppose only the basic implementation without replication.

After the relation is created in step 4, the evaluator stores it into the DHT. The peer in the DHT responsible for storing the relations should perform the following series of operations before the relation is stored in its local database:

1. If the relation already exists, it tests whether a new relation has a newer time-stamp T and a newer time-stamp T_R in the acknowledgement than relations already stored in a local database. The relation is dropped otherwise.
2. It tests whether time-stamp T and time-stamp T_R are not too old and whether $T > T_R$.
3. It verifies the evaluator signature to eliminate forged relations.
4. It verifies the provider signature in the acknowledgement to eliminate not approved relations.
5. The relation is stored in local database. The older relation is replaced if it exists.

This ensures that only the newest and genuine relations are stored and that the malicious evaluator cannot use the acknowledgement from the previous transaction to update its previously created relation.

The function *get_relations* sends a DHT query for each peer from the input set together with a specification whether provider or evaluator key is required. The important property is that each relation has to be stored simultaneously under an evaluator and a provider key. The relation r has to be included in the results of both function *get_relations(provider,{r.P})* and *get_relations(evaluator,{r.E})* or none of them. Otherwise the malicious peer can exploit it. For instance, the malicious evaluator can store its false relation only under a provider key, which means that this relation will be used in the calculation of the provider rating but not used in the calculation of the evaluator rating. To prevent this vulnerability, the peer responsible for storing the relation under a provider key notifies its counterpart for an evaluator key before it stores the relation into its local database and vice versa. Therefore, the evaluator can send store message only once. Additionally, if we modify this procedure to do this notification periodically, we have a simple replication algorithm.

The fundamental question is how the transactions are evaluated. The previous researchers used either continuous values in a limited range or a discrete value to

mitigate the subjectivity of the evaluation. Besides, the reputation value included in the relation is a cumulative value which contains the evaluation of all previous transaction towards the remote peer. More precisely, it is the evaluation of the remote peer based on its past transactions. We intentionally do not provide a mechanism of evaluating the transaction or cumulating evaluations of transactions into an evaluation of a peer. The evaluation of transactions in a file-sharing network can be completely different from the mechanisms used in a distributed storage or a distributed computation. In each application the peers have different means to verify the result of the transaction and different demands on the transaction parameters. So, these mechanisms are related with the application layer, and BubbleTrust is not bind with any specific application layer.

5.5 Provider and evaluator functions

The next task is to design the evaluator and the provider functions defined in section 5.2. Those functions have a crucial impact on the algorithm result. The provider function determines how the evaluator rating influences its recommendations. Analogously, the evaluator function determines how the accuracy of the recommendations influences the evaluator rating of its originator. In this section, we discuss the requirements on both functions and propose their formulations. We start with the provider function which has a form:

$$pv(x_1, x_2):[-1,1]\times[0,1]\rightarrow[-1,1]$$

The first argument is a provider rating originated from an evaluator (recommendation) and the second argument is the evaluator rating of this evaluator. The function result is the altered provider rating according the evaluator trustfulness. This implies four natural conditions:

1. If $x_2 = 1$ then $pv = x_1$ (Let the recommendation unchanged.)
2. If $x_2 = 0$ then $pv = 0$. (Ignore the recommendation.)
3. The pv is an increasing function in x_2 for $x_1 > 0$.
4. The pv is a decreasing function in x_2 for $x_1 < 0$.

We introduce a fifth condition which allows us to parametrize the function by the parameter T_P. This parameter determines the degree of toleration and is in the range (0,1].

5. If $x_2 = 0.5$ then $pv = x_1 \cdot T_P$.

In other words, if the evaluator trustfulness decreases to the mid-value, the recommendation is decreased by the parameter T_P. This parameter is called provider toleration. We designed the simplest function which meets all five conditions:

$$pv(x_1, x_2) = x_2^{\log_{0.5}(T_P)} \cdot x_1 \tag{11}$$

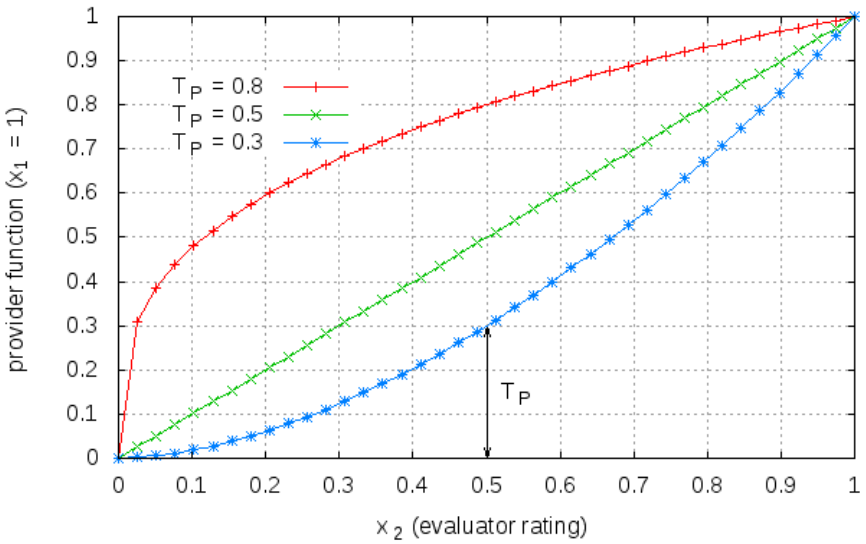


Fig. 10: Provider function with fixed T_P and variable x_1.

Figure 10 demonstrates the meaning of the parameter T_P. The bigger T_P implies that the *pv* decrease slowly, hence the system is more tolerant to the peers with lower evaluator ratings. In this figure, the function value in the point $x_2 = 0.5$ is equal to the value of T_P. And figure 11 illustrates the provider function for different x_1 value. The figure shows only positive values, the graph of negative values is symmetrical on axis x_2.

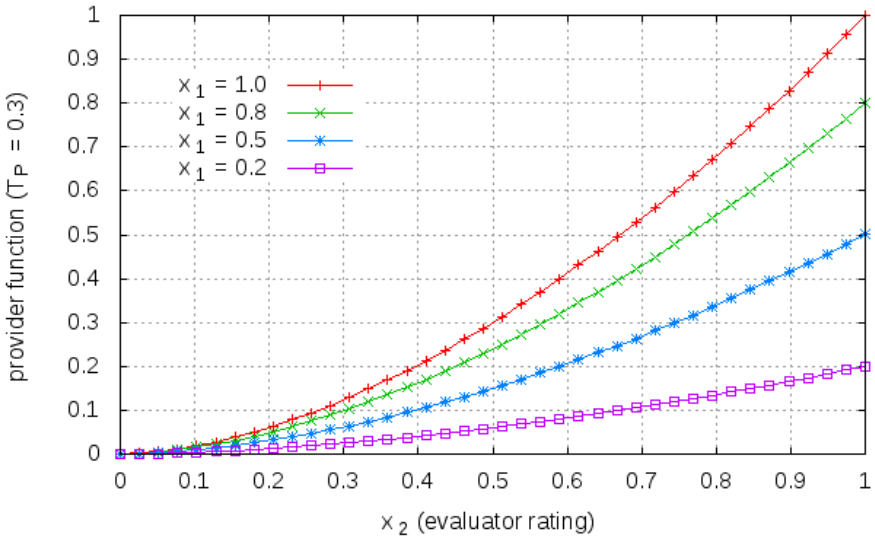


Fig. 11: Provider function with fixed x_1 and variable T_P.

The evaluator function is a little bit more complicated. It has a form:

$$ev(x_1, x_2):[-1,1]\times[-1,1]\rightarrow[0,1]$$

The first argument is a provider rating originated from a given evaluator and second argument is the reference provider rating. On the basis of those values, the function determines the evaluator rating of the given evaluator. This implies three natural conditions:

1. If $x_1 = x_2$ then ev = 1 (Accurate recommendation)
2. If difference between x_1 and x_2 increases, then *ev* decreases.
3. The decreasing rate depends on the absolute value of x_2. The smaller $|x_2|$ implies lower decrease rate.

We also introduce the next condition which allows us to parametrize the function by the parameter T_E. Similarly to the evaluator function this parameter can be interpreted as a degree of toleration.

4. If $x_2 = \mp 1$ and $x_1 = x_2 \cdot (1 - TE)$ than ev = 0.5

In other words, if the known peer is completely trustful or completely distrustful and the recommendation differs by the parameter (1 - T_E), then the given evaluator has an evaluator rating 0.5 (the mid-value). This condition is analogous to condition 5 for the evaluator function.

The following function meets all conditions:

$$ev(x_1, x_2) = 0.5^{\left(\frac{x_1 - x_2}{(1-T_E)\cdot|x_1|-1}\right)^2} \qquad (12)$$

Figure 12 shows the evaluator function for fixed T_E and variable x_2. The maximum is in the points where $x_1 = x_2$ (accurate recommendation). The parameter T_E determines how quickly the function decreases from its maximum.

The last function is the time function which maps the age of transaction into a range [0,1]. There is only one simple condition: the older relation weights less than newer one. The simplest implementation is the linear function:

$$f(t) = 1 - \frac{\Delta T}{\text{history\_period}} \quad \text{for } \Delta T < \textit{history\_period}, \text{ 0 otherwise.}$$

However, in our system we use the exponential function which reflects dependency between time and the relation weight better. We use the following

exponential function:

$$f(t)=e^{-(\Delta T\cdot k)^2} \qquad k=\frac{\sqrt{-\ln(\text{min\_weight})}}{\text{history\_period}}$$ for $\Delta T <$ *history_period*, 0 otherwise.

where parameter *min_weight* is the minimum acceptable value. The function reaches this value if ΔT is close to *history_period*. The function is shown in figure 13.

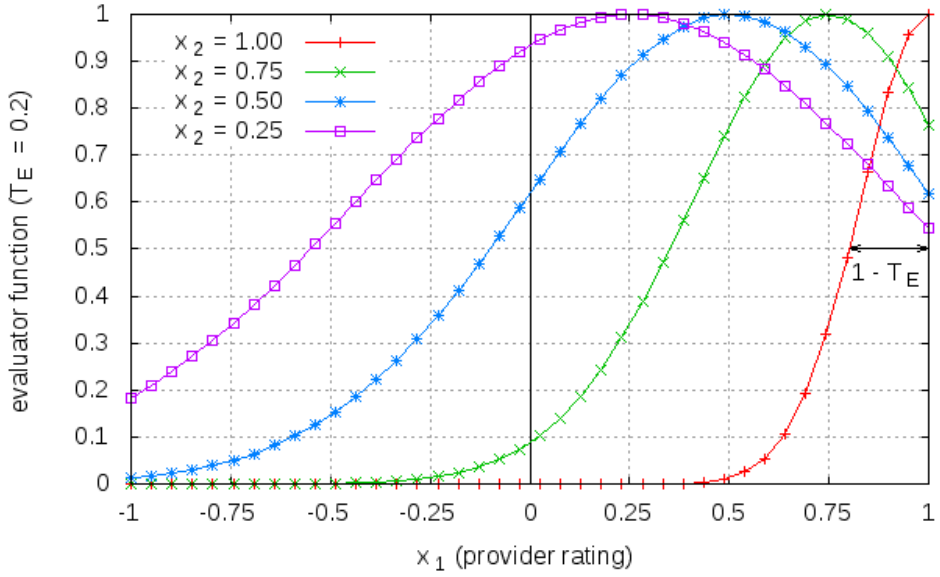


Fig. 12: Evaluator function with fixed T_E and variable x_2.

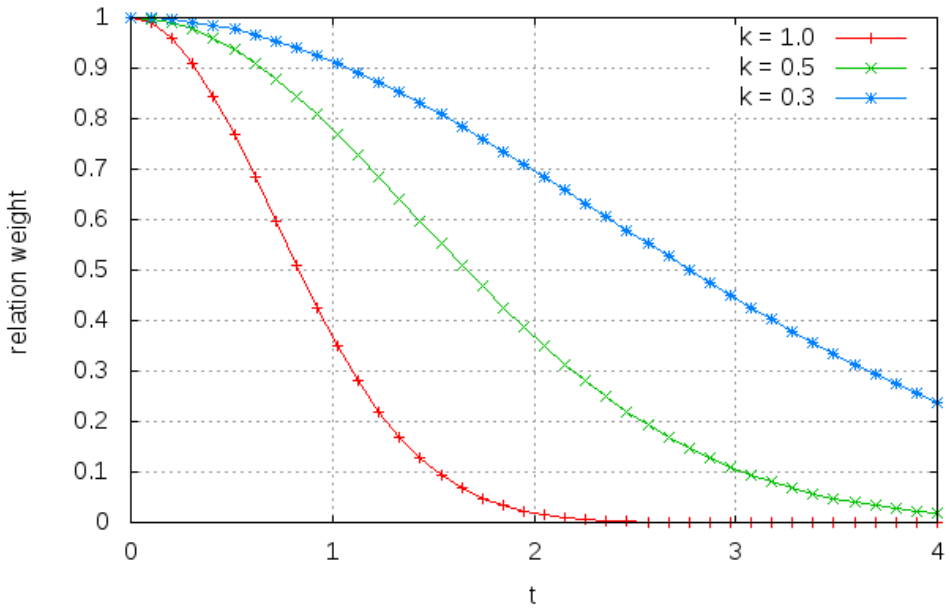


Fig. 13: Time function for different parameter *k*.

5.6 Optimized algorithms

The basic algorithm introduced in section 5.3 tries to reach all peers in the network. The load related with the basic algorithm can be unacceptable for real usage. In the following text we introduce several improvements which reduce the algorithm complexity without significant degradation of its efficiency.

5.6.1 Cutting off

The first improvement is based on the idea that in the formula (9) or (10) it is not necessary to involve the relations which have small weight in comparison to other. These relations have only limited influence on the result and can be neglected. The optimized algorithm ignores the peers whose relations have a low contribution to the calculated trust value and there is no point in dealing with them. We also implement the maximal number of nodes added in each level. The algorithm ensures that less important peers are removed first. The function in figure 14 realizes the described restrictions and can implement the function *optimizations(evaluator,S)* in the basic algorithm.

```
function optimization(evaluator,S) {
  W = 0;
  foreach s from S {
    e = s.get_evaluator();
    e.weight += W(s);
    W += W(s);
    E.add(e);
  }
  foreach e from E { e.weight = e.weight/W; }
  E = sort_weight(E);
  removedWeights = 0;
  foreach e from E {
    removedWeights += e.weight;
    if ((removedWeights <= limit) || (S.size() > maxNodes)) then
      E.remove(e);
  }
  foreach s from S {
    if (!E.contains(s.get_evaluator()) then S.remove(s);
  }
  return S;
}
```

Fig. 14: Cutting off relations with the smallest weight.

The algorithm calculates cumulated weight for each evaluator, because this evaluator can have relations towards more peers in the input set. The algorithm

removes the evaluators which cumulative weights towards all peers in the input set are smallest.

5.6.2 Limiting depth

The basic algorithm finishes after visiting all nodes in the network or if there are no more relations from visited nodes to the rest of the network. The restrictions on number of nodes in each level result into increasing the depth of recursion, so it is necessary to limit the depth as well. The peers in the last level have assigned the default trust values. We choose the default provider rating equal to 0 and default evaluator rating to 0.5, both represent a mid-value. In simulations presented further in the text we recommend an appropriate number of levels.

5.6.3 Using values from previous runs

One algorithm run computes trust values for a large number of peers. Using these values in next runs seems to be a good idea. There are two related issues. First, not all trust values are equally reliable. For instance, the trust value which was calculated in level 6 is less reliable than the value from level 2. On level 6 there are less remaining levels to maximal depth and calculation is more limited. Second, the trust values obsolete in time because new transactions take place and relations are changed.

Nevertheless, using the cached values can significantly decrease the network load. We introduce three simple rules for manipulating with the trust cache which should minimize the impact on the accuracy.

1. The level where the trust value was calculated is stored into the cache along with the trust value.
2. The cache record is used in calculation only if the level of the record is less or equal to the current level in the calculation algorithms.
3. Each level has its time to live (TTL). After this time the level of the record is increased. If the record is on the maximal level, the record is deleted.

These rules ensure that the calculation is not less accurate than without using the cache. In fact, using cached values from the lower level than the current level increases accuracy, because the values on the lower levels are more precise.

5.7 Evaluation and data analysis

To evaluate performance of our method, we implemented the simulation model. We generate a network with a given number of nodes and relations between them. The relations are generated according to both zip's law and random distribution; each relation has a different size and time.

We focused on performance issues which can be measured by the average

number of visited nodes for each query. The basic algorithm reaches as many peers as possible; if there are some relations from the visited peers to the rest of the network, the basic algorithm follows them. Hence the number of visited nodes depends on the density of relations in the network. The practical simulations showed that the average number of visited nodes for the basic algorithm and zipf's distribution is $1.4 \cdot N$, where N is the number of the peers in the network. Do not forget that each peer can be visited twice.

The optimized algorithm with cutting off and limiting depth reduces the maximal number of visited peers to $(\text{max\_nodes} \cdot \text{max\_levels})$. The parameter *max_level* limits the depth of the recursion and *max_nodes* limit the number of peers on each level. Obviously, the peers which are farther from the original peer have smaller influence on the calculated rating. Our goal is to set these parameters as low as possible with minimal impact to the result.

We performed several simulations with different network sizes and number of relations. The figure 15 displays how the parameters *max_nodes* and *max_levels* change the reputation value in the network with 2000 nodes and 40000 relations.

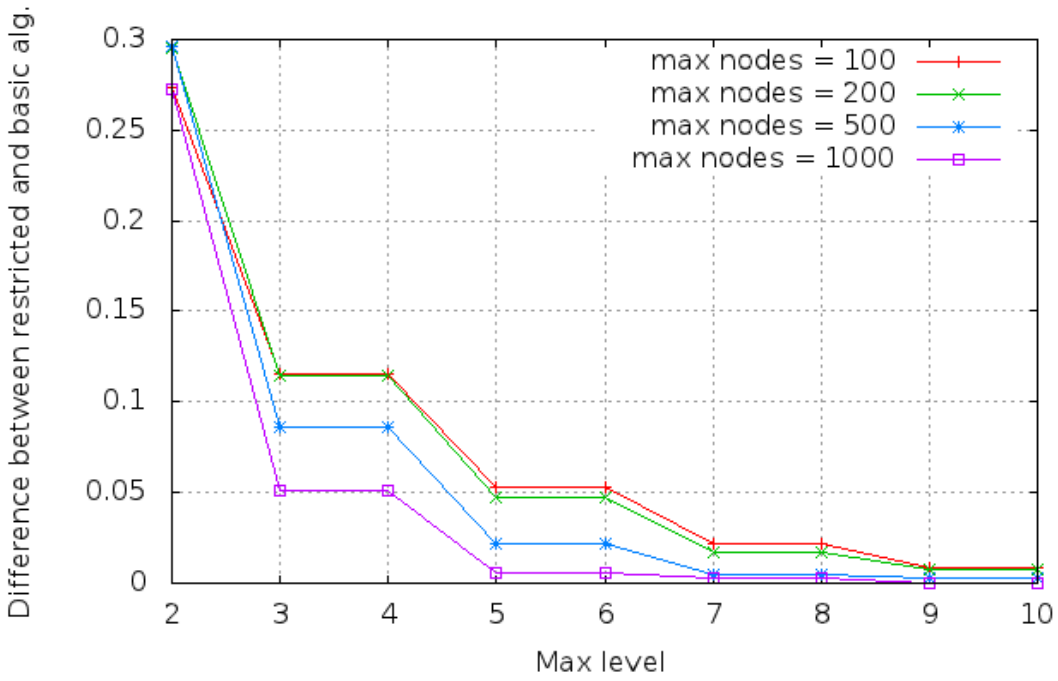


Fig. 15: The dependency between the restriction parameters and calculation result.

The graph shows the difference between outputs of the basic algorithm and restricted algorithm with the given parameters. An important property of the BubbleTrust is that it does not depend on the size of the network. As a result of these simulations, we recommend using the parameters *max_nodes* = 200 and *max_levels* = 5 where the difference is less than 0.05. However, the algorithm can still visit 1000

nodes, so the next optimization is necessary.

The cached version has a great potential to reduce the number of visited peers. We performed the simulation of the network with 1000 peers and 20000 relations per day with a zipf's distribution. The efficiency of this optimization depends on the peer activity. We suppose that the tested peers ask to the some others peer ratings every 10 minutes. The figure 16 shows the simulation after two days. This simulation does not implement any other optimization.

On recommended level 5, the number of visited peers falls to the one quarter. The rules for using cached values ensure that this optimization does not deteriorate the result values.

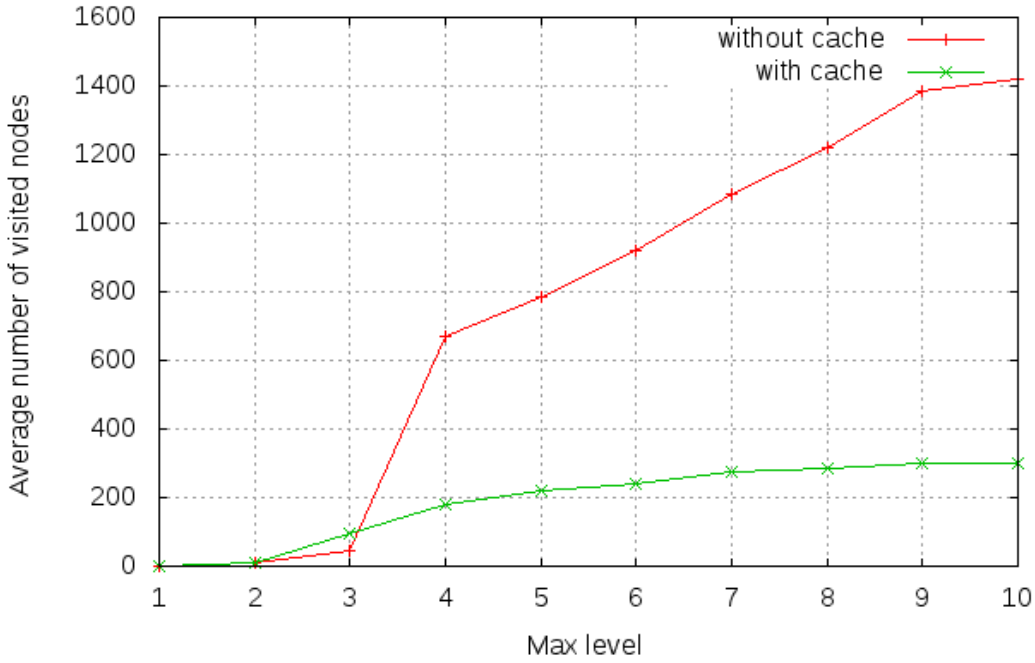


Fig. 16: Efficiency of using cache in the network with 1000 nodes and 20000 relations per day.

Of course, further simulations are necessary to prove the algorithm efficiency against previously proposed malicious activities compared with previously published methods. This section focuses on the network load caused by the algorithms itself and effectiveness of the proposed optimization methods.

5.8 Summary and future work

BubbleTrust differs from the previously published TMSs in several ways. The data management layer ensures that the malicious peers cannot create fake relations towards honest peers or suppress unflattering relations towards them. All peers have to acquire an acknowledgement from their transaction partner before the relation is created. Unfortunately, this does not stop malicious peers to create fake relations towards allied malicious peers.

The evaluator rating is calculated using experience of other peers in the network. TMSs discussed in section 3 use either only personal experiences or rely on the premise that good provider is a good evaluator as well. As far as we know, BubbleTrust is the first TMS using global experience as feedback verification (see table 4). In BubbleTrust we formalized the relation between evaluator and provider rating and vice versa. The provider function and the evaluator function allow to parametrize these relations and establish the level of toleration. The evaluator rating has also a new use in BubbleTrust. The providers check the evaluator ratings of all consumers requesting their resource.

Unlike the most other TMSs, BubbleTrust does not try to find the chain of the trust from the consumer to the provider. This chain can be easily compromised if only one malicious peers succeed in attaching itself into this chain. Moreover, this chain can be relatively long, especially in a large P2P networks and the process of finding such chain can be very expensive. BubbleTrust uses the concept of trust bubbles. In the centre of the bubble is a potential provider and on each level are peers which have the strongest relations with peers on the lower level. The size of the bubble is not determined by the size of the network but only by the capacity of computing peers. The peers which initiate the computation may or may not be a part of the bubble. Thanks to this mechanism, the peer's decision is based on the opinion from the largest group of peers with strongest relations towards the target peer. Each peer in BubbleTrust can choose the size of the trust bubble and adjust the trade-off between computation accuracy and the load of computation process. All this improvements should significantly help to fight with malicious peers. The simulation and comparison with previously published TMSs are in section 7.

In our future works we want to focus on the next reduction of the complexity of the algorithm. For instance, at the moment, every peer makes a decision on the basis of information served by data management layer and performs all calculations by its own. However, the peers which trust each other at most can share information stored in the cache and reduce the network load.

To complete the system, it is also necessary to develop a reliable method for replication information stored in the DHT. The primary purpose of replication is to prevent data loss caused by the node failure or its maliciousness. But it can also significantly speed up the lookups in DHT and reduce the network load.

6 Simulation framework

In order to facilitate comparison of different TMSs and their behaviour under different malicious strategies, we created a simulation framework [88] called P2PTrustSim. We used FreePastry [89], a modular, open-source implementation of the Pastry [3], P2P structured overlay network. Moreover, FreePastry includes a network simulator which can be used as a base for P2PTrustSim. Above the FreePastry, we created the peer simulation layer which implements various peers' behaviour. The framework is highly configurable and allows implementing various simulation scenarios, trust managements and malicious strategies.

6.1 Architecture of P2PTrustSim

The framework is written in Java and available via svn [90]. The main class accepts two parameters: the configuration file and the output directory. The configuration file is a standard Java properties file containing entire simulation settings. All logs and statistics are stored into the output directory.

The basic properties located in the configuration file are *simulation_class*, *trustmanagement_class* and *malicious_nodes_implementation*. These properties determine which class will be used for these tasks. Except the few properties related to log facility, all other properties depend on the settings of these three basic properties.

6.1.1 Simulation class

The simulation class creates and controls the simulation. Its task is to prepare a simulation scenario, which includes initializing a required number of honest and malicious nodes and initial distributing resources among them. This class is also responsible for starting the simulation, reading and aggregating statistical data from all nodes. It implements UserSimulatorInterface which is quite simple. This interface contains only two methods: addPeer and run.

So far we have created only one implementation of the simulation class called BasicUserSimulator. This class initializes malicious nodes according the properties *malicous_nodes* and *malicious_nodes_implementation*. To reflect the typical resource distribution in a P2P network better, each resource has a parameter called popularity. More popular resources are most often used in transactions; they are shared on one peer longer and have a higher probability that the downloading peer becomes a new provider. These rules imply that resources have no stable providers but travel in the network; new resources can be created, old ones can vanish and more popular resources are provided by more peers. The resource popularity is distributed according the zipf-law. The resources are distributed according the zipf-law which reflects the typical resource distribution in the most used file-sharing P2P networks

[43]. The parameters of zipf-law distribution are determined by the properties *zipflaw_exponent* and *zipflaw_ranks*.

In a regular interval determined by the property *one_cycle_min* the simulation dumps the information about the simulation progress and the memory usage on the standard output. The property *one_cycle_min* is defined in minutes of simulation time. And every *dump_every_nth_cycle* the statistical information from all nodes are stored into the file *./stats/global.log* in the output directory. All other actions are driven by the user implementations.

6.1.2 User implementations

There can be many different user behaviours. The peers can be either honest or malicious and malicious peers can implement many malicious strategies. In P2PTrustSim the user behaviour is simulated in class AbstractUserImplementation. This abstract class contains implementation of functions performing transactions, sharing resources and communicating with trust management. The specific user behaviour is implemented in classes inherited from AbstractUserImplementation. All subclasses must implement these three methods: run, serveTransaction and evaluateTransaction. The first method is called periodically by the simulation framework and should perform the actions initiated by the user itself, like downloading a resource from a remote peer or creating fake transactions. Other two methods are called when the framework requires serve or evaluate transactions. P2PTrustSim contains eleven classes implementing different user behaviours.

The first class is called UserHonest and simulates behaviour of honest peers. In regular intervals, the honest peer picks a random resource, finds all providers for this resource, chooses the best provider, downloads the resource from this provider and evaluates the transaction. Optionally, the peer can be a new provider for this resource and share it with other peers for some time. If some remote peer requires the resource located on the honest peer, the honest peer always provides the resource in a full quality. The honest peer also always evaluates all transactions truthfully.

All other classes implement some type of malicious behaviour described in chapter 4 or a combination of them. The simplest malicious strategy is implemented in UserIndividualSimple. All resources shared by the malicious peers are malicious. Malicious peers do not use any strategy to help spread their resources nor download any resource from other peers. Slightly more clever is a strategy implemented in UserIndividual which uses also false meta-data strategy to make all their resources most popular ones. The current TMSs should easily deal with both these strategies. The last individual malicious strategy is implemented in the class UserCamouflage. The camouflaged malicious peers provide both malicious and honest resources. The ratio between malicious and honest transactions is determined by the property

malicious_trans_ratio.

The next five classes correspond with names of collective malicious strategies described in chapters 4.3 and 4.4. In all cases the combination with false mata-data strategy is used. The remaining two classes UserMaliciousNewcomer and UserMaliciousChange is used for special scenarios with oscilation behaviour explained in chapter 7. The number of faked or other auxilary transactions for each malicious strategy can be set through the configuration file.

6.1.3 Trust managements implementation

The implementations of trust management systems are separated into the independent project. Our goal was to create an implementation which will be usable together with FreePastry to develop any P2P application. There are not any dependencies on P2PTrustSim. The project is accessible via svn [91].

Currently, the project contains basic implementations of seven TMSs. These implementations are usable together with P2PTrustSim and can be used by application developers as well. But so far we have implemented only the features necessary for our simulations; therefore, some critical functions are still missing. For instance, any implementation does not provide replication. The implementation of BubbleTrust does not use cryptography in the data management layer. Each peer in EigenTrust is a score manager for itself. All implementations should be extended before they can be used in the production deployment.

All TMSs must implement interface called TrustManagementInterface. This interface exports all TMS functions. The main functions are: getProviderRatings, getEvaluatorRatings and evalTransaction. But some implemented TMSs either do not use an evaluator rating or use a vector value for a provider rating. In this case we mapped the results into a range [-1,1] for provider rating and a range [0,1] for evaluator rating. In case that the TMS does not support evaluator rating, the function getEvaluatorRating returns always 1. Therefore, the decision algorithm in P2PTrustSim does not have to be modified for each TMS.

P2PTrustSim is built above the structured P2P network which limits the number of TMSs which can be implemented. We can afford it because the structured P2P networks became prevalent in recent years and replaced the less efficient unstructured P2P networks in many applications. We also implemented only the basic model using the reputation connected with peers, not with resources. The resource reputation model has special requirements on the way of distributing resources. We do not assume anything about P2P application above TMSs.

We implemented seven TMSs demonstrating different approaches, namely: DummyTrust, SimpleTrust, EigenTrust, PeerTrust, H-Trust, WTR and BubbleTrust. First implementation, DummyTrust, represents a network without trust management. A random provider is chosen regardless of its reputation. SimpleTrust is a very

simple TMS which uses only local experiences. Recommendations from remote peers are ignored. These two implementations serve as a base for a comparison. The remaining five implementations correspond with TMSs described in the sections 3.

There are some implementation details which should be mentioned. The EigenTrust is not able to work correctly without pre-trusted peers, so we had to set 10% honest peers as pre-trusted. Therefore, EigenTrust has a significant advantage over other TMSs. In PeerTrust we implemented only PSM variant because TVM is conceptually similar to EigenTrust and ATC implementation is used due to DTC excessive overhead. The system WTR uses the risk factor besides the reputation, in our model we cumulated the risk and the reputation into a single value which is used for the decision process. This calculation follows the procedure described in the WTR [71].

6.1.4 Communication in P2PTrustSim

The communication between the user implementation and the trust management is provided by the P2P layer. This layer implements a FreePastry application node, realizes the transactions between peers and communicates with TMS. This layer simulates the functions of a P2P application while the user implementation represents physical users sitting behind this application.

Note that malicious behaviour is implemented only on the user level. There is only one implementation of the P2P layer. The attacker controlling this layer would have the capability to perform other attacks like whitewashing or attacks against the overlay network described in 2.1. However, as we have already announced, we focused primary on the attacks on the application level.

Figure 17 illustrates the communication between all components of the system during one transaction. On one side is a consumer requesting a resource and on the other side is a provider serving this resource. The P2P layer is responsible for communication with the TMS and implement a decision algorithm. The framework is prepared for the data management as proposed in BubbleTrust, although currently only BubbleTrust uses it.

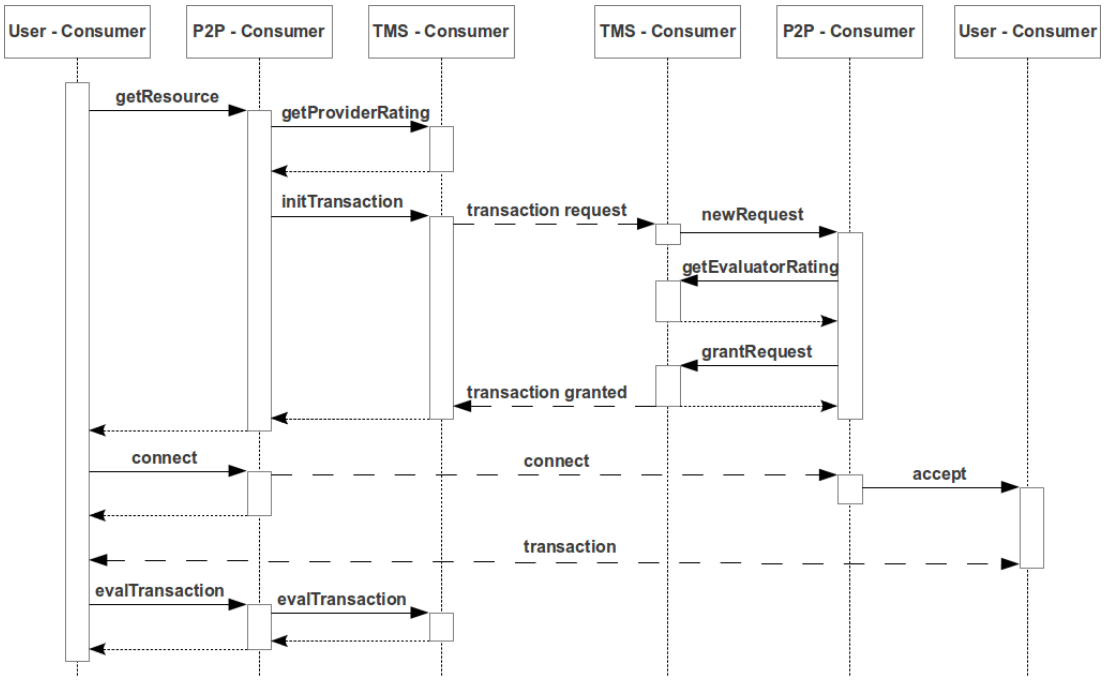


Fig. 17: Sequence diagram of communication in P2PTrustSim

6.2 Evaluation criteria

It is difficult to compare individual TMSs because there are no fixed criteria which are able to measure the efficiency of the reputation managements or their resistance against malicious strategies. We have only a vague notion of "trusted P2P network". Most of the TMSs have been built under the premise that this notion is well understood. For the comparison of different TMSs, first we need to determine the criteria of success.

Our framework provides statistical data which can be used as a substrate for these criteria. Each transaction is categorized and counted on both sides (provider and consumer). The categories distinguish the type of the peer (honest or malicious); on which side of the transaction the peer was (provider or consumer); and the result of the transaction. Figure 18 shows all nine types of transactions. The honest and bogus transactions correspond with their names. The ulterior transactions represent honest transactions which malicious peers have to perform to fix their reputation. These transactions can be on the provider side to fix a provider reputation or on the consumer side to fix an evaluator reputation. The faked transactions are between malicious peers and their purpose is to transfer reputation from one malicious peer (e.g. spy) to other malicious peers. If none provider is sufficiently trustful, the transaction is refused and counted as ConsumeRefused. The originated peer typically tries to pick different service and repeat the transaction.

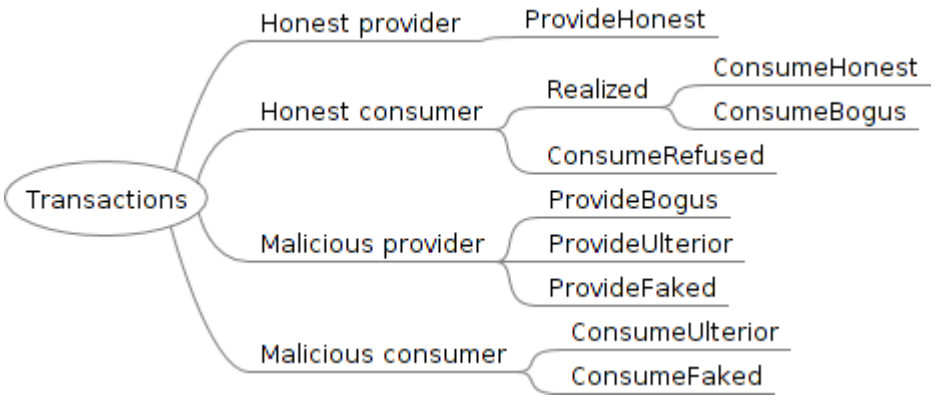


Fig. 18: Categorization of transactions in the simulation framework.

Let us suppose that all the malicious peers cooperate within only one malicious collective and the transactions from honest peers are always honest. The following four invariants are always valid:

1. $\sum ProvideFaked = \sum ConsumeFaked = TotalFaked$
2. $\sum ProvideBogus = \sum ConsumeBogus = TotalBogus$
3. $\sum ProvideUlterior + \sum ConsumeUlterior = TotalUlterior$
4. $\sum ProvideHonest + \sum ProvideUlterior = \sum ConsumeHonest + \sum ConsumeUlterior = TotalHonest$

The sums are over all peers in the network. Our primary goal is to evaluate the success of each malicious strategy in a different TMS. Therefore, we defined four criteria: MaliciousSuccessRatio, BogusRatio, MaliciousCost, and MaliciousBenefit.

MaliciousSuccessRatio is a ratio between bogus transactions provided by malicious peers in the network with TMS and in the network without TMS (DummyTrust). It reflects the contribution of the given TMS and it is defined by the following formula:

$$MaliciousSuccessRatio = \frac{TotalBogus_{withTMS}}{TotalBogus_{withoutTMS}}$$

The result should be smaller than 1, otherwise the TMS is not useful. We will require values smaller than 0.5 to consider the TMS to be resistant against the given

malicious strategy.

BogusRatio is a ratio between bogus and all services consumed by the honest peers. It tells us the percentage of bogus services in the network. It is defined by the following formula:

$$BogusRatio = \frac{TotalBogus}{\sum ConsumeHonest + TotalBogus}$$

We accept values smaller than 50%, otherwise there is more bogus than honest services and the participation in such P2P network is not useful for any honest peer.

MaliciousCost studies TMS from the malicious peers' point of view. It monitors the load associated with a malicious strategy. And it is defined as a ratio between extra transactions performed by the malicious peers to trick the TMS and the bogus transactions in the network. These extra transactions include faked and ulterior transactions and represent additional overhead for malicious peers which they try to minimize. We defined it by the following formula:

$$MaliciousCost = \frac{TotalUlterior + TotalFaked/2}{TotalBogus}$$

The formula takes into account the fact that the overhead connected with faked transactions is smaller than overhead connected with ulterior transactions. One side of the ulterior transaction is an honest peer; therefore, the transaction has to be completed to produce the recommendation. On the other hand, the faked transactions are solely between malicious peers who can produce recommendations without the transaction really happening.

This metric gives us an idea of how much computation power and network utilization is required for a given malicious strategy. It should be as big as possible to make the strategy useless for the malicious peers. It is necessary to know how strong the peer's motivation for its maliciousness is to determine the limit value for this metric.

The last criterion is a MaliciousBenefit. It represents how much beneficial transactions the malicious peers have to perform to pass one malicious service. It is defined by the following formula:

$$MaliciousBenefit = \frac{TotalUlterio}{TotalBogus}$$

The value above 1 means that there is benefit from the malicious collective

which is bigger than the damage caused by the collective. This criterion is helpful in a situation when we want to maximize benefit for peers in the network and do not want to bother with some bogus transactions.

Apart from these four criteria, the next important property of each TMS is how quickly the sudden changes in peer's behaviour are recognized. We considered two situations: malicious newcomer and treasons. The malicious newcomers are the peers which join the network and start harm immediately. It is convenient to provide a small initial reputation to each newcomer in order to give them a possibility to reveal their intentions. Without these initial reputations, the newcomers would have only a little chance to be chosen for cooperation and their trustworthiness could not be verified. But this initial reputation can be also exploited by malicious peers to push their bogus services.

In treason, the peers which are already connected in the network and which behaved correctly in the past suddenly start to provide bogus services. This situation is more dangerous than first one, because such peers have already built a good reputation and they are usually able to push more bogus services until they are detected. In our simulation, we have tested both these scenarios and measured the time necessary to detect such peers and suppress their malicious influence.

6.3 Common simulation settings

We tried to set the similar parameters for all TMSs. The most important parameter is the *history_period* which determines how long the network remembers the information about the last transactions. We set this parameter to 5 hours (in order to have a history window appropriate to the total simulation time) in all TMSs. Other parameters used in implemented TMSs are chosen either according the authors proposals or according the best results of our preliminary simulations. The complete lists of parameters are given in table 5.

The numbers of ulterior and faked transactions are the same for all malicious strategies which use them. The camouflage strategy has the ratio between bogus and honest transactions 0.5. In both cases we choose the values which present trade-off between success of the malicious strategy and its cost.

EigenTrust	
Parameter	**Value**
Ratio of pre-trusted peers	10%
Weight of pre-trusted peers (*a* in original paper)	0.2
PeerTrust	
Parameter	**Value**
Default trust	0.2
Default similarity	0.2
Cache limit	60m
H-Trust	
Parameter	**Value**
H-Index multiplicator	10
Query threshold	7
WTR	
Parameter	**Value**
Window size	10 transactions
Alpha	0.5
BubbleTrust	
Parameter	**Value**
Max levels	5
Max nodes	20
T_P	0.3
T_E	0.5
TTL_0	30m
TTL_1	30m
TTL_2	60m
TTL_3	60m
TTL_4	120m

Table 5: Parameters for TMS implementations

7 Simulation results

In the simulations, we focused on two problems. First, the effectiveness of the different malicious strategies measured by the four criteria presented in previous section. Second, we measured how quickly the TMSs are able to react on changes in peers behaviours.

Our basic simulation network contains 200 peers and 80 peers are malicious. Hence 40% of nodes in the network are malicious, which represents a very dangerous environment. The honest peers wake up every 10 minutes and use one service from the network. The malicious peers also wake up every 10 minutes and perform a given number of faked or ulterior transactions. The whole simulation takes 24 hours. The size of the network was designed with regards to simulation possibilities of the FreePastry and the heavy load produced by our simulation. We have also run other series of test with the different settings and analysed their impact on result in chapter 7.3.

7.1 Efficiency criterion

Figure 19 shows MaliciousSuccessRatio in the network with BubbleTrust. We can see that the most successful strategy is evaluator collusion with almost 25% of realized malicious transactions. In other words, BubbleTrust decreases the number of malicious transactions to 25% compared to the network without TMSs. This is the worst case scenario. Other malicious strategies are not so successful. The malicious transactions are nearly completely suppressed in individual strategies. In these strategies, we can observe a phenomenon called starvation. The malicious peers are quickly recognized and nobody wants to cooperate with them. Hence, new recommendations are not created. After expiring the old recommendations, all information about peer's maliciousness are lost and they can start over. This causes the spikes in the graph repeated ones every two *history_period*.

Figure 20 shows the same graph in network using PeerTrust. The results are completely different. The most successful strategies are still the evaluator collusion and the evaluator spies but they reach success ratios near 1. This means that the number of malicious transactions is almost the same as when we do not use any TMSs. Therefore, the peers are completely vulnerable against these strategies. We analysed the reason for this failure and found out that the credibility of malicious peers is very high due to their correct evaluations. The credibility in PeerTrust is calculated from mutual recommendations towards same peers. The set of these recommendations is relatively small and malicious peers in collusion can easily manipulate with it.

MaliciousSuccessRatios of other TMSs are shown in table 6. The numbers in

the table are average values for last 10 hours of simulations. The values above the threshold 0.5 are displayed in bold; these values indicate that TMSs failed against this malicious strategy. We can see that only BubbleTrust is resistant against all tested malicious strategies. There is at least one effective malicious strategy against all other TMSs.

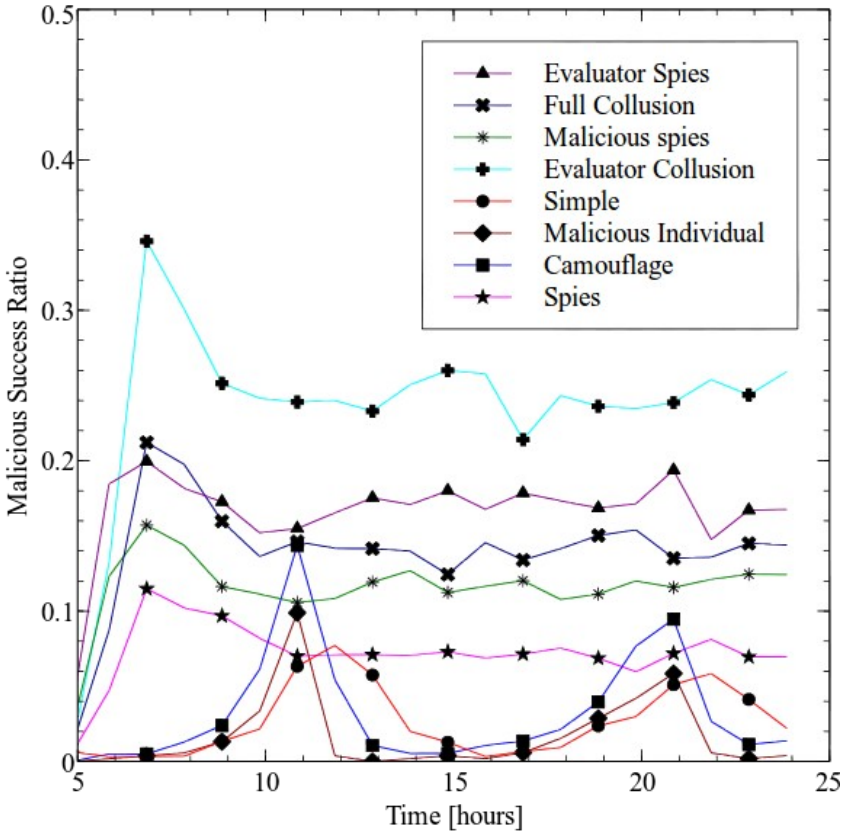


Fig. 19: MaliciousSuccessRatio for different malicious strategies in BubbleTrust

SimpleTrust is ineffective against all malicious strategies since the history period is too short. Without cooperation with other peers, the information about peer's maliciousness is lost after 5 hours and the delay between two transactions towards the same peer can be longer. Any real implementation of SimpleTrust should use much longer history period since all experiences are stored only locally and there is not an additional cost of storing them in the network.

EigenTrust, despite its advantage, is vulnerable to spies. The spies are even able to perform more bogus transactions than it would be possible in a network without TMS. The collusion tactics are completely useless because they are not designed into this type of TMS. It is notable that the individual strategies are relatively successful. The reason for this is that the trust matrix converges slowly. The convergence speed is investigated further in the text.

H-Trust is fully resistant against Simple, Individual, Full collusion and Spies;

all bogus transactions are suppressed. However, it is vulnerable against Evaluator collusion or Evaluator spies. WTR copes very well with individual strategies; especially the camouflage is ineffective due to the risk factor. But the collective strategies can easily circumvent it.

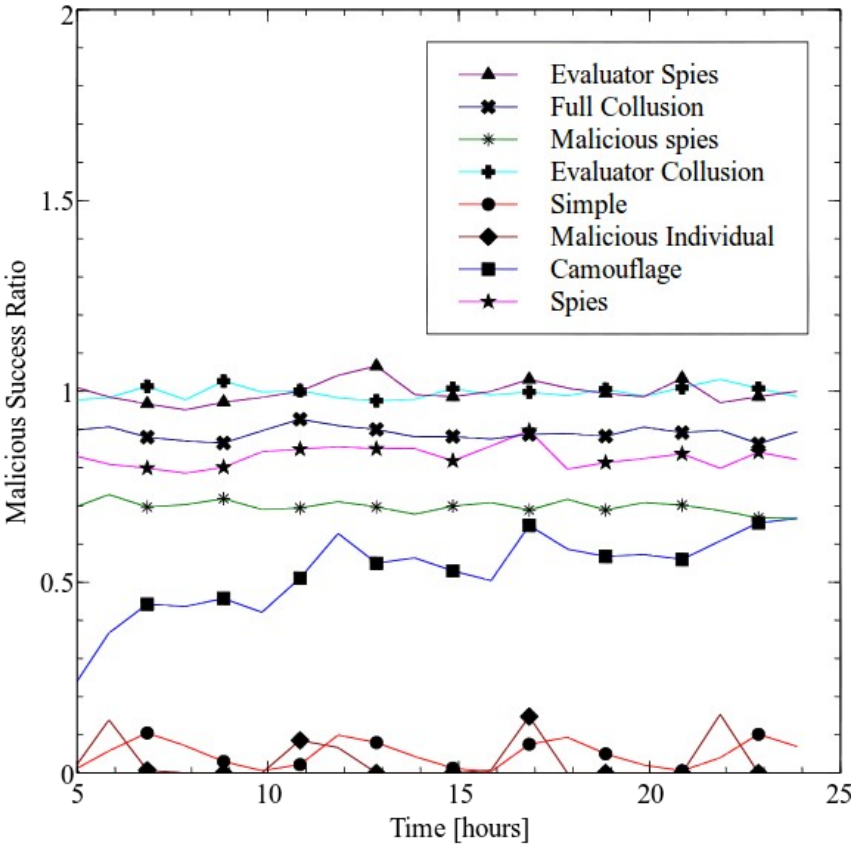


Fig. 20: MaliciousSuccessRatio for different malicious strategies in PeerTrust

The next criterion BogusRatio is shown in table 7. The results are very similar to the MaliciousSuccessRatio, it is only a different point of view. In the worst case scenario, only 28% of all transactions in the P2P network with the BubbleTrust can be bogus. Other TMSs tolerate 60% (EigenTrust), 70% (H-Trust), 72% (PeerTrust) and 73% (WTR) bogus transactions

Strategy	Simple Trust	Eigen Trust	H Trust	Peer Trust	WTR	Bubble Trust
SIM	**0.90**	0.28	0.00	0.05	0.00	0.04
IND	**0.90**	0.20	0.00	0.03	0.00	0.02
CAM	**0.93**	0.26	0.14	**0.60**	0.00	0.04
FCOL	**0.89**	0.00	0.00	**0.87**	**0.98**	0.13
ECOL	**0.88**	0.00	**0.94**	**0.99**	**0.99**	0.24
SPS	**0.81**	**1.06**	0.00	**0.81**	**0.81**	0.07
ESPS	**0.83**	**0.99**	**0.99**	**1.00**	**0.99**	0.17
MSPS	**0.88**	0.15	**0.68**	**0.68**	**0.62**	0.11

Table 6: MaliciousSuccessRatio for different malicious strategies and TMSs. Malicious strategies: SIM - Simple, IND - Malicious Individual, CAM - Camouflage, FCOL - Full Collusion, ECOL - Evaluator Collusion, SPS - Spies, ESPS - Evaluator Spies, MSPS - Malicious Spies.

Strategy	Simple Trust	Eigen Trust	H Trust	Peer Trust	WTR	Bubble Trust
SIM	41%	16%	0%	3%	0%	3%
IND	**67%**	20%	0%	4%	0%	3%
CAM	47%	18%	10%	36%	0%	4%
FCOL	**67%**	0%	0%	**67%**	**73%**	18%
ECOL	**67%**	0%	**70%**	**72%**	**71%**	28%
SPS	43%	**54%**	0%	43%	**50%**	5%
ESPS	43%	**60%**	**50%**	**50%**	**50%**	12%
MSPS	**66%**	27%	**56%**	**56%**	**57%**	15%

Table 7: BogusRatio for different malicious strategies and TMSs. Malicious strategies: SIM - Simple, IND - Malicious Individual, CAM - Camouflage, FCOL - Full Collusion, ECOL - Evaluator Collusion, SPS - Spies, ESPS - Evaluator Spies, MSPS - Malicious Spies.

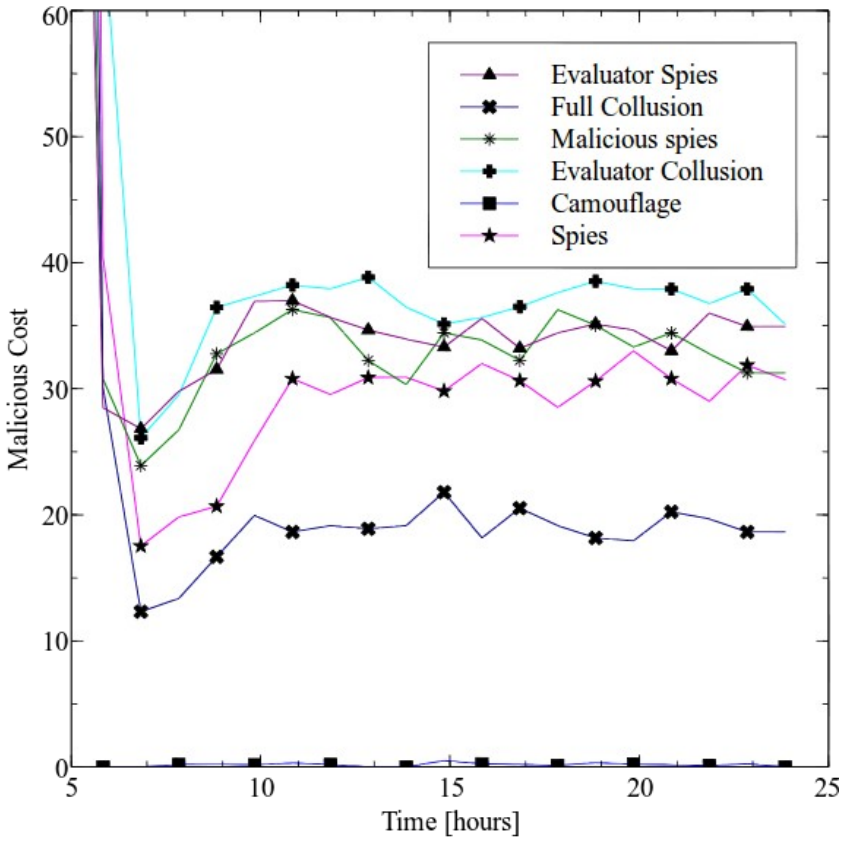


Fig. 21: MaliciousCost for different malicious strategies in BubbleTrust

A more interesting parameter is MaliciousCost which represents the overhead of malicious peers for one malicious transaction. Figure 21 shows MaliciousCost of malicious strategies which use ulterior or faked transactions in BubbleTrust.

The cheapest strategy is the camouflage with approximately 0.16 honest transactions to one bogus. But this strategy has also a negligible MaliciousSuccessRatio. From the collective strategies, the cheapest is the full collusion with a little less than 20 additional transactions to one bogus transaction which is able to reach MaliciousSuccessRatio 0.13. If we compare all malicious strategies using the ratio of cost to success, full collusion seems to be the most advantageous strategy. The creator of a malicious collective should always consider the cost associated with considered malicious strategies and a potential benefit from realized malicious transactions. The values for other TMSs are depicted in table 8.

The attacker most likely uses a strategy which has the best cost/success ratio. For instance, in the PeerTrust the most successful strategy is Evaluator collusion but it is very expensive (above 9), better choice is Full collusion with success ratio 0.87 and cost only 3.12. The Camouflage strategy is relatively efficient in EigenTrust, H-Trust and PeerTrust; although it has a low success ratio, it is compensated by its very

low price. In the BubbleTrust, all strategies have cost above 20 (except Camouflage) and the most expensive strategy (Evaluator collusion) has almost 38. This is significantly higher value than have other TMSs.

The last criterion MaliciousBenefit offers a different point of view. The question is whether malicious strategies emitting ulterior transactions can be beneficial for other peers in the network. Table 9 shows MaliciousBenefit for different malicious strategies and TMSs. The strategies like evaluator collusion, evaluator spies and malicious spies have always more beneficial transactions than bogus ones. Strictly speaking, the designation malicious collective is no longer suitable.

Strategy	Simple Trust	Eigen Trust	H Trust	Peer Trust	WTR	Bubble Trust
CAM	0.16	0.19	0.11	0.12	N/A	0.16
FCOL	3.04	N/A	N/A	3.12	2.73	20.29
ECOL	10.30	N/A	9.65	9.23	9.17	37.97
SPS	2.55	1.96	N/A	2.55	2.10	29.01
ESPS	6.97	5.96	5.76	5.74	5.83	33.74
MSPS	4.42	N/A	5.70	5.71	5.43	35.00

Table 8: MaliciousCost for different malicious strategies and TMSs.

The attackers, whose primary goal is to destroy the network functionality for other peers, probably do not choose malicious strategy with a high MaliciousBenefit. But attackers who desire to spread their malicious services at any cost do not bother with MaliciousBenefit.

Strategy	Simple Trust	Eigen Trust	H Trust	Peer Trust	WTR	Bubble Trust
CAM	0.16	0.19	0.11	0.12	N/A	0.16
ECOL	7.24	N/A	6.79	6.48	6.44	26.68
SPS	0.12	0.10	N/A	0.13	0.09	1.83
ESPS	3.37	2.88	2.77	2.76	2.80	16.53
MSPS	2.09	N/A	2.70	2.70	2.56	16.56

Table 9: MaliciousBenefit for different malicious strategies and TMSs.

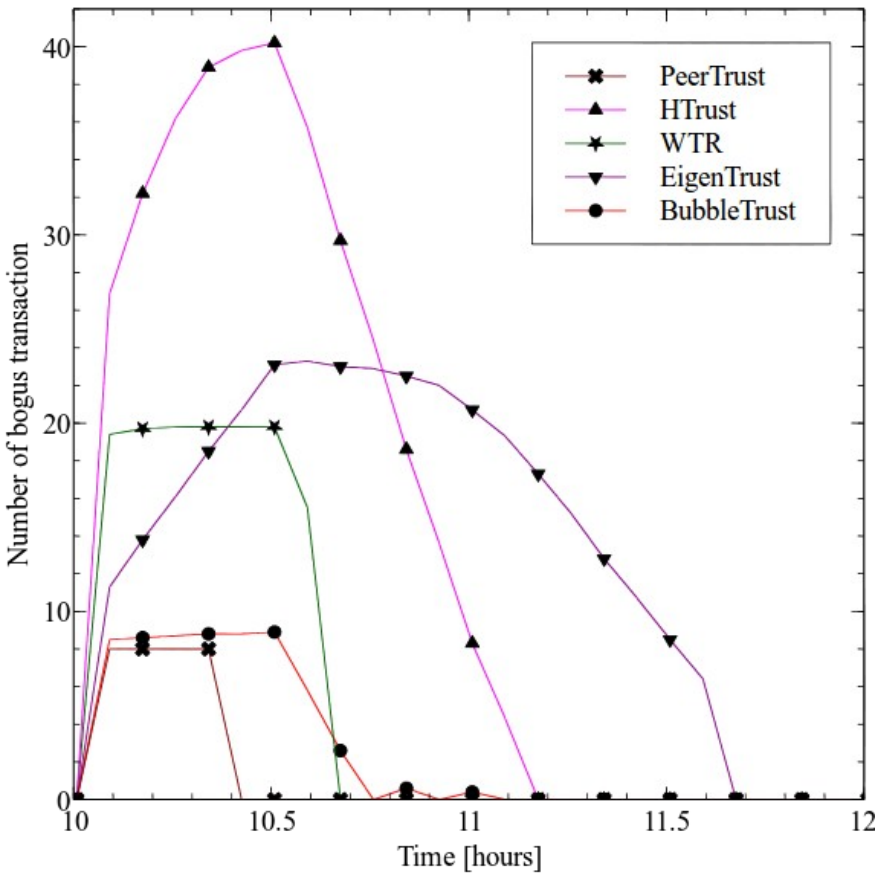


Fig. 22: Number of bogus transactions after joining new malicious peers.

7.2 Dynamic criterion

In the following simulations, we focus on convergence speed; therefore, on the time which the TMSs need to react to newcomers or sudden changes in peer behaviour. To test a newcomer scenario, we used similar settings like in previous simulations. In this simulation, the 40% of malicious peers join the network after ten hours. The number of bogus transactions increases as expected and after some time drops to zero. Figure 22 shows the progress in a number of bogus transactions in this scenario. The PeerTrust, BubbleTrust and WTR are able to recognize all malicious newcomers in less than one hour. The best results have been measured with PeerTrust which deals with all newcomers in a half hour. The slowest reaction has been measured in EigenTrust. However, these results are not sufficiently informative because they depend mainly on initial trust. Each TMS gives newcomers a small initial reputation in order to allow them to prove their trustworthiness. This value differs in each TMS and influences the results in this simulation.

Nevertheless, the results of this simulation can help to determine resistance of

individual TMSs against whitewashing attacks. The longer time to recognize a malicious newcomer the more attractive the system is for whitewashers.

In the next simulation scenario, the 40% of honest peers suddenly change behaviour and start provide bogus resources. These peers have already the highest possible reputation, hence they can cause more damage until detected. Figure 23 shows progress in number of bogus transactions in scenario with traitors.

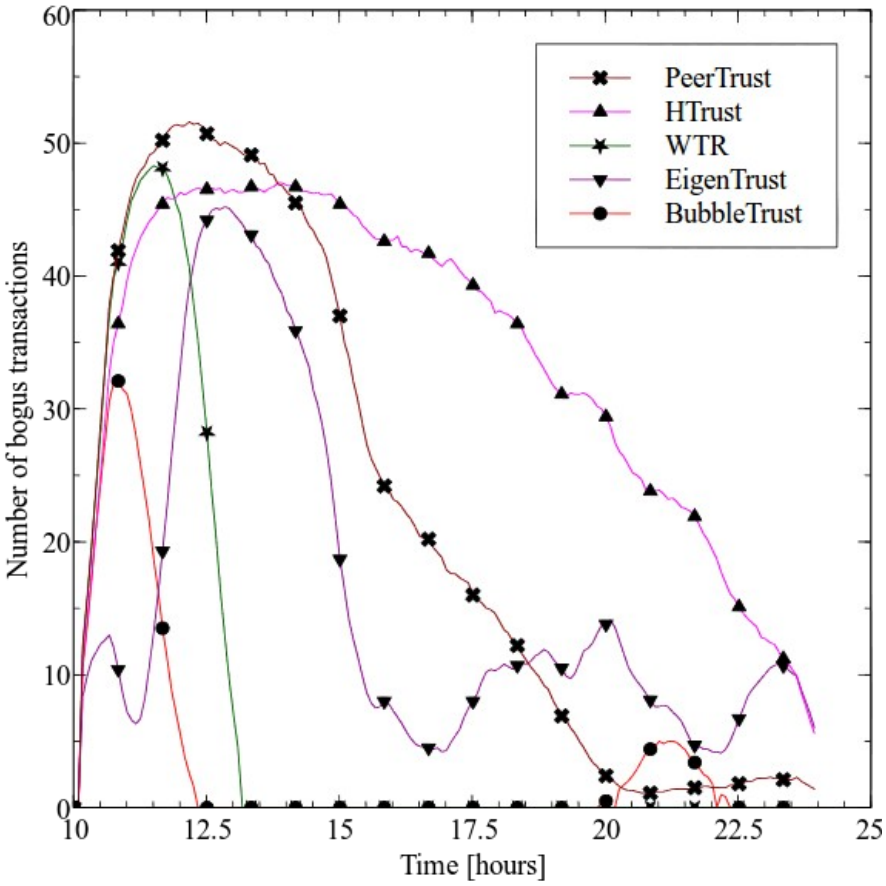


Fig. 23: Number of bogus transactions after betrayal.

As we expected, in all cases the time to discover malicious peers is longer than in the previous scenario. The quickest TMS is BubbleTrust followed by WTR with times about 2.5 hours. On the other hand, the trust in traitors in H-Trust is lost after more than 13 hours. EigenTrust does not suppress all bogus transactions even in simple malicious strategy, see table 6. We have to measure the time for which the number of bogus transactions falls to its normal level, in this case it is 6 hour. Similarly for PeerTrust, the effect of traitors diminishes after 10 hours.

Traitors are very dangerous for each TMS. The success of this strategy mainly depends on the speed of how the changes in peer behaviour are recognized. There are two time intervals which need to be taken into account. First, it is the time to reveal a

traitor. It is the interval between the moment when a high-reputable peer starts to be malicious and the moment when all peers in the network are aware of his maliciousness. This has been investigated in the previous simulation. Second interval is the time to rehabilitate a malicious peer. In other words, it is the time between the moment when a malicious peer starts to provide honest resources and the moment when all peers in the network trust them again. Some TMSs do not allow malicious peers to regain its reputation after treason at all. But this requires that history is remembered indefinitely which is not case in most TMSs.

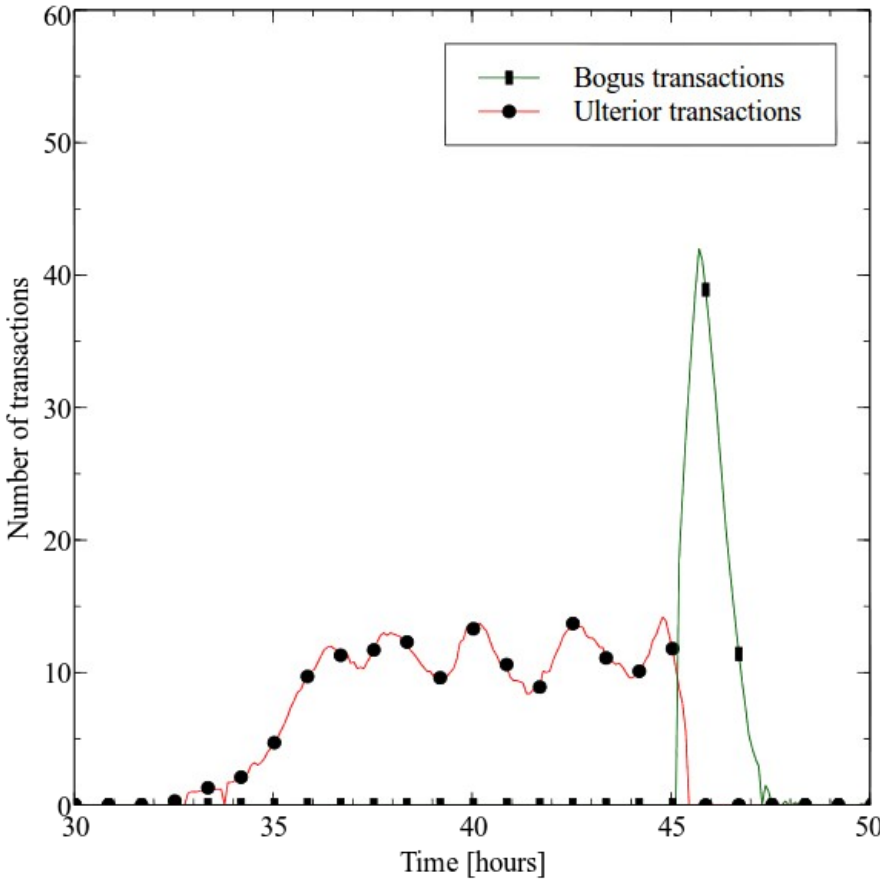


Fig. 24: Rehabilitation after treason in BubbleTrust.

In the last simulation, we investigated the time to rehabilitate malicious peers. This simulation takes 60 hours and malicious peers change their behaviour every 15 hours. Figure 24 shows the time interval from 30 to 50 hours. Therefore, it starts in a moment when a recognized malicious peers return to honest behaviours. The increasing number of ulterior transactions expresses the process of regaining trust. After approx. 6 hours the number of ulterior transactions reaches its normal level which indicates that all malicious peers are trustful again. Table 10 shows times of all other TMSs.

The quality TMSs should have the shortest treason detection time and the longest rehabilitation time. The parameter *history_period* has an appreciable impact on these results, in all our simulation it is set to 5 hours. This parameter determines how long the feedbacks are stored in the network. The individual peers can have its own database according the algorithm of used TMS.

7.3 Influence of different simulation settings

We have tried different simulation settings. The basic simulation network contains 200 nodes and 40% of them are malicious. At first we have adjusted the number of nodes in the network with preserving the ratio of malicious nodes. We have made the following observation: an increase in number of nodes does not affect the MaliciousSuccessRatio. The reason for this fact is that each TMS can handle only a limited number of nodes in the calculation of ratings. For instance, the size of the trust bubble in BubbleTrust is limited to 100 nodes. A similar limitation can be found in all TMSs. The information from nodes which are very distant in a trust chain is negligible. On the other hand, the results change if we decrease the number of nodes in the network. This change can be in both directions; it depends on the TMS and the malicious strategy. In this case the TMS has to rely on information from a smaller number of nodes than it expects. It works in some kind of degrades modes. Therefore, we chose the network with 200 nodes as optimal for simulation a real application.

Next we have altered the ratio of malicious nodes. Figure 25 shows the results for BubbleTrust. As we can see, the malicious success increases with the ratio of malicious nodes in the network. BubbleTrust resists relatively well even in the network with more than 50% of malicious nodes. In our tests we stayed at 40% because it is very unlikely that the overlay network beneath the P2P application can handle with the situation where half of the peers are malicious. The defence techniques described in 2.1 assume that only a small fraction of nodes is malicious. In fact, 40% already cause big problems.

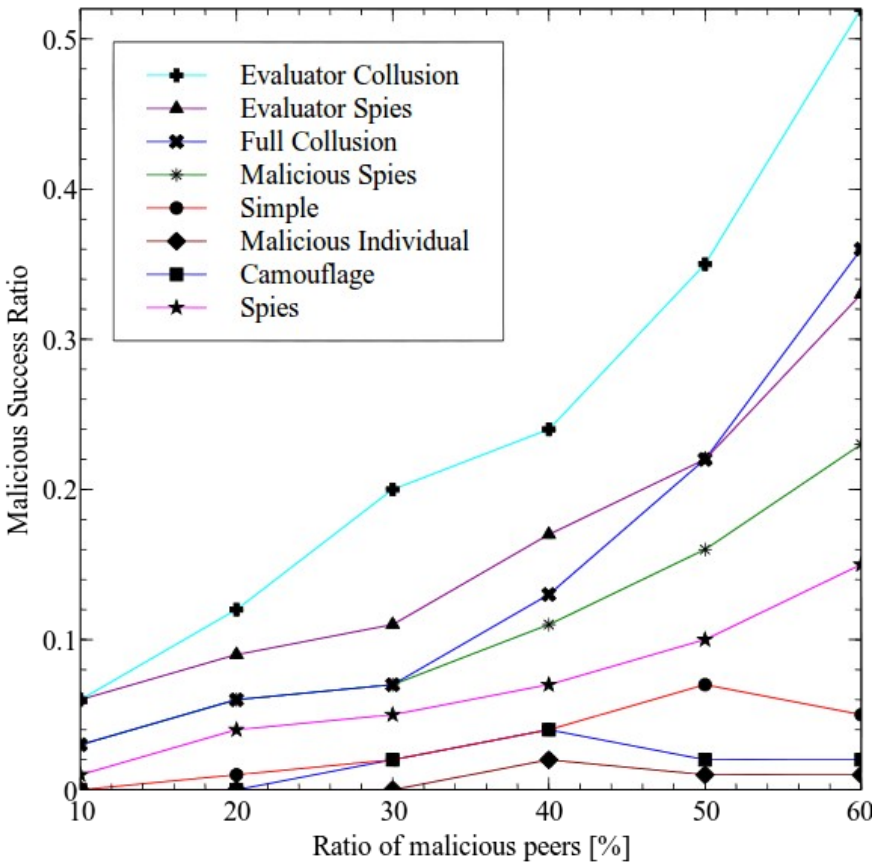


Fig. 25: Effect of ratio of malicious peers on Malicious Success Ratio in BubbleTrust.

7.4 Summary

Table 10 shows that the best TMS in our comparison is BubbleTrust. It has the shortest treason detection time, the longest rehabilitation time and allows only 28% of bogus transaction under the most successful malicious strategy. As far as we know, it is the only one TMS using global experience as feedback verification.

H-index calculation used in H-Trust proved to be vulnerable to traitors. It takes too long to detect traitors and malicious peers are rehabilitated too quickly. The system WTR permits the highest number of bogus transactions from all tested TMSs, but it is followed closely by PeerTrust and WTR. EigenTrust has better results than HTrust, WTR and PeerTrust but it has a significant advantage in the form of pre-trusted peers as well.

Our tests proved that it is very difficult to resist against the sophisticated malicious techniques. Especially the calculation of the evaluator rating is susceptible to rigging. The previously published TMSs do not pay as much attention to the evaluator rating as they pay to the provider rating. This must change if the TMS

should be resistant against the evaluator collusion or the evaluator spies.

Trust management system	Rehabilitation time [hours]	Treason detection time [hours]	Type of rating	Feedback aggregation	Feedback verification	Best malicious strategy	Maximal BogusRatio
EigenTrust	4	6	G	F	GP	SPS	60%
HTrust	1.3	14	P	S	PE	ESPS	70%
PeerTrust	5	10	P	F	PE	ECOL	72%
WTR	2.5	3.1	G	F	GP	ECOL	73%
BubbleTrust	6.5	2.4	P	S	GE	ECOL	28%

Table 10: Summary of all tested TMSs.

All TMSs have been tested in a very dangerous environment and against the sophisticated malicious strategies. If we run similar tests in less dangerous environment, the results of all TMSs have been significantly better and similar to each other. Additionally, the use of such sophisticated malicious strategies in a real P2P application has not been recorded yet. Currently, there are not P2P applications widespread enough and handling with such attractive resources to be worth it. These tests should be understood as stress tests realizing the worst case scenario. They should verify whether P2P networks can be used in more security sensitive applications.

8 Conclusion

In this thesis, we focused on the security aspects of the distributed applications built over P2P networks. This is a very extensive topic covering the common network attacks, the attacks against the overlay network and last but not least the application attacks exploiting the natural trust among the peers. We summarized the state of the art in this field and assessed the application layer as the most challenging area. The reputation-based trust management has been proposed in the the last years as a novel way of dealing with the security deficiencies inherent to P2P networks. Many trust management systems have been developed in last years. We described the most known of them and organized them into a simple taxonomy which gives us a basic idea about usable approaches. Many published trust management systems are conceptually similar. In this thesis, we focused only on systems which introduced new ideas or push knowledge forward, hence it should not be considered as a complete list of all published TMSs.

One of the crucial parts of this thesis is an analysis of possible malicious strategies. The collective of malicious peers which cooperate with each other can develop sophisticated malicious strategies which present a big challenge for each TMS. The most previously published TMSs have been designed considering only simple collective strategies, if any, but the attacker with detailed knowledge of the internal function of the TMS can adapt its strategy to be more efficient under used TMS. We propose several modifications of the known malicious strategies targeted towards the general principles used by some TMSs. These strategies are more dangerous than common collective strategies. Additionally, other modifications or combinations of these strategies are possible with even bigger efficiency.

It is expected that malicious peers work in a collective try to use the most effective strategy. Therefore, the quality of TMSs has to be assessed according to the most successful malicious strategy. Additionally, other properties have to be taken into account too; e.g. the cost connected with the malicious strategy can exceed the potential benefit for malicious peers.

We organized the existing models and ideas of trust management systems and confronted them to the potential strategies used by attackers associated in malicious collectives. We also analysed the existing malicious strategies and proposed new strategies specifically designed according the weaknesses of the current trust management systems.

On the basis of this analysis, we developed a novel trust management system called BubbleTrust. This system implements several new approaches compared to the previously published systems. The data management layer uses asymmetric cryptography to ensure that malicious peers cannot create faked feedback towards

honest peers or deny unflattering feedbacks towards other malicious peers. BubbleTrust also uses a unique way of calculating an evaluator rating. Most of the other TMSs calculate evaluator rating either using only local experience with evaluator's previous feedbacks or using the premise that a good provider must be a good evaluator as well. In BubbleTrust, the peers calculate the evaluator rating similarly to provider rating; in cooperation with other peers. This approach provides more precise results and better resistance against collaborative malicious strategy using false feedbacks.

Unlike the most other TMSs, BubbleTrust does not try to find the chain of the trust from the consumer to the provider. BubbleTrust uses the concept of trust bubbles. In the centre of the bubble is a potential provider and on each level are peers which have the strongest relations with peers on the lower level. The size of the bubble is not determined by the size of the network but only by the capacity of computing peers. The peers which initiate the computation may or may not be a part of the bubble. Each peer in BubbleTrust can choose the size of the trust bubble and adjust the trade-off between computation accuracy and the load of the computation process.

All this improvements should significantly help to fight with malicious peers. To verify this assumption, we created a simulation framework called P2PTrustSim. Using this framework we can compare trust management systems against different malicious strategies. We also proposed several efficiency criteria which can be evaluated using this framework. In this thesis, since we cannot test all trust managements due to their appreciable numbers, we chose the most representative systems for each type according to the presented taxonomy. Although, the simulation framework is easily extensible with any TMS suited for structured P2P networks. We tested five TMSs including BubbleTrust against eight malicious strategies. The results indicate that only BubbleTrust is resistant against all considered malicious strategies; it is, therefore, the best choice for deployment in the secured P2P networks.

As a future work, we plan to extend the simulation possibilities of P2PTrustSim to test malicious strategies directed to overlay layer and implement more simulation scenarios. We would also like to implement more TMSs to emphasize our results.

As for BubbleTrust, our next task is the addition reduction of the complexity of the algorithm. For instance, at the moment, every peer makes a decision on the basis of information served by data management layer and performs all calculations by its own. However, the peers which trust each other at most can share information stored in the cache and reduce the network load.

For the purposes of the simulations we created five TMS implementations which can be directly used together with FreePastry for deployment of any P2P

application. Nevertheless, the implemented functionality covers only requirements of tested simulation scenarios. For instance, the cryptography in data management layer in BubbleTrust is not implemented or the peers in EigenTrust calculated their own trust values. We plan to improve the BubbleTrust implementation by adding next security measures. We would like to provide a fully usable security solution for P2P networks based on BubbleTrust which reacts on all threats in the application and the network layer.

Our ultimate aim is to make the P2P architecture suitable for implementation in more security sensitive applications in which traditional client-server model still dominates.

Bibliography

[1] Ratnasamy, S.; Francis, P.; Handley, M.; Karp, R. & Shenker, S., "*A scalable content-addressable network*", SIGCOMM Comput. Commun. Rev., 2001, vol. 31 pp. 161-172.

[2] Stoica, I.; Morris, R.; Karger, D.; Kaashoek, M. F. & Balakrishnan, H., "*Chord: A scalable peer-to-peer lookup service for internet applications*", In: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 149-160, 2001

[3] Rowstron, A. & Druschel, P., "*Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems*", Lecture Notes in Computer Science: Middleware 2001, 2001, vol. 2218, pp. 329-350, 2001

[4] Maymounkov, P. & Mazières, D., "*Kademlia: A Peer-to-Peer Information System Based on the XOR Metric*", Lecture Notes in Computer Science: Peer-to-Peer Systems, 2002, vol. 2429, pp. 53-65, 2002

[5] Aberer, K.; Cudré-Mauroux, P.; Datta, A.; Despotovic, Z.; Hauswirth, M.; Punceva, M. & Schmidt, R., "*P-Grid: A Self-organizing Structured P2P System*", SIGMOD Rec., 2003, vol. 32 pp. 29-33.

[6] Stutzbach, D. & Rejaie, R., "*Understanding churn in peer-to-peer networks*", In: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement, pp. 189-202, 2006

[7] Rhea, S.; Geels, D.; Roscoe, T. & Kubiatowicz, J., "*Handling churn in a DHT*", In: Proceedings of the annual conference on USENIX Annual Technical Conference, pp. 10-10, 2004

[8] Godfrey, P. B.; Shenker, S. & Stoica, I., "*Minimizing churn in distributed systems*", In: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 147-158, 2006

[9] Ou, Z.; Harjula, E.; Kassinen, O. & Ylianttila, M., "*Performance evaluation of a Kademlia-based communication-oriented P2P system under churn*", Comput. Netw., 2010, vol. 54 pp. 689-705.

[10] Rao, A.; Lakshminarayanan, K.; Surana, S.; Karp, R. & Stoica, I., "*Load Balancing in Structured P2P Systems*", Lecture Notes in Computer Science: Peer-to-Peer Systems II, 2003, vol. 2735, pp. 68-79, 2003

[11] Byers, J.; Considine, J. & Mitzenmacher, M., "*Simple Load Balancing for Distributed Hash Tables*", Lecture Notes in Computer Science: Peer-to-Peer Systems II, 2003, vol. 2735, pp. 80-87, 2003

[12] Karger, D. R. & Ruhl, M., "*Simple efficient load balancing algorithms for peer-to-peer systems*", In: Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures, pp. 36-43, 2004

[13] Godfrey, B.; Lakshminarayanan, K.; Surana, S.; Karp, R. & Stoica, I., "*Load balancing in dynamic structured P2P systems*", In: INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies, pp. 2253 - 2262 vol.4, 2004

[14] Zhu, Y. & Hu, Y., "*Efficient, proximity-aware load balancing for DHT-based P2P systems*", Parallel and Distributed Systems, IEEE Transactions on, 2005, vol. 16 pp. 349 - 361.

[15] Urdaneta, G.; Pierre, G. & Steen, M. V., "*A survey of DHT security techniques*", ACM Comput. Surv., 2011, vol. 43 pp. 8:1-8:49.

[16] Dahan, .. S. & Sato, .. M., "*Survey of Six Myths and Oversights about Distributed Hash Tables Security*", In: ICDCSW 07: Proceedings of the 27th International Conference on Distributed Computing Systems Workshops, pp. 26, 2007

[17] Douceur, .. J. R., "*The Sybil Attack*", In: IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems, pp. 251-260, 2002

[18] Dinger, J. & Hartenstein, H., "*Defending the Sybil Attack in P2P Networks: Taxonomy, Challenges, and a Proposal for Self-Registration*", In: Proceedings of the First International Conference on Availability, Reliability and Security, pp. 756-763, 2006

[19] Castro, .. M.; Druschel, .. P.; Ganesh, .. A.; Rowstron, .. A. & Wallach, .. D. S., "*Secure routing for structured peer-to-peer overlay networks*", SIGOPS Oper. Syst. Rev., 2002, vol. 36 pp. 299-314.

[20] Wang, H.; Zhu, Y. & Hu, Y., "*An Efficient and Secure Peer-to-Peer Overlay Network*", In: Proceedings of the The IEEE Conference on Local Computer Networks 30th Anniversary, pp. 764-771, 2005

[21] Bazzi, R. A. & Konjevod, G., "*On the establishment of distinct identities in overlay networks*", In: Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing, pp. 312-320, 2005

[22] Bazzi, R.; Choi, Y.-r. & Gouda, M., "*Hop Chains: Secure Routing and the Establishment of Distinct Identities*", Lecture Notes in Computer Science: Principles of Distributed Systems, 2006, vol. 4305, pp. 365-379, 2006

[23] Borisov, N., "*Computational Puzzles as Sybil Defenses*", In: Peer-to-Peer Computing, 2006. P2P 2006. Sixth IEEE International Conference on, pp. 171 -176, 2006

[24] Rowaihy, H.; Enck, W.; McDaniel, P. & La Porta, T., "*Limiting Sybil Attacks in Structured P2P Networks*", In: INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE, pp. 2596 -2600, 2007

[25] Yu, H.; Kaminsky, M.; Gibbons, P. B. & Flaxman, A., "*SybilGuard: defending against sybil attacks via social networks*", In: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 267-278, 2006

[26] Yu, H.; Gibbons, P.; Kaminsky, M. & Xiao, F., "*SybilLimit: A Near-Optimal Social Network Defense Against Sybil Attacks*", Networking, IEEE/ACM Transactions on, 2010, vol. 18 pp. 885 -898.

[27] Sit, E. & Morris, R., "*Security Considerations for Peer-to-Peer Distributed Hash Tables*", In: Revised Papers from the First International Workshop on Peer-to-Peer Systems, pp. 261-269, 2002

[28] ll Namgung, J.; Shin, S.-Y.; Park, S.-H.; Lee, L.-S. & Jeong, D., "*Self-organizing P2P overlay network applying dynamic landmark mechanism for contents delivery network*", In: Proc. Third ACIS Int Software Engineering Research, Management and Applications Conf, pp. 317-324, 2005

[29] Hildrum, K. & Kubiatowicz, J., "*Asymptotically Efficient Approaches to Fault-Tolerance in Peer-to-Peer Networks*", Lecture Notes in Computer Science: Distributed Computing, 2003, vol. 2848, pp. 321-336, 2003

[30] Condie, T.; Kacholia, V.; Sankararaman, S.; Hellerstein, J. M. & Maniatis, P., "*Induced Churn as Shelter from Routing-Table Poisoning*", In: In Proc. 13th Annual Network and Distributed System Security Symposium (NDSS, , 2006

[31] Singh, A.; Ngan, T.-W.; Druschel, P. & Wallach, D. S., "*Eclipse Attacks on Overlay Networks: Threats and Defenses*", In: INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings, pp. 1 -12, 2006

[32] Awerbuch, B. & Scheideler, C., "*Towards a scalable and robust DHT*", In: Proceedings of the eighteenth annual ACM symposium on Parallelism in algorithms and architectures, pp. 318-327, 2006

[33] Awerbuch, B. & Scheideler, C., "*Towards scalable and robust overlay networks*", In: Proc 6th Int Workshop on PeerToPeer Systems IPTPS, pp. 1-7, 2007

[34] Fiat, A.; Saia, J. & Young, M., "*Making Chord Robust to Byzantine Attacks*", Lecture Notes in Computer Science: Algorithms – ESA 2005, 2005, vol. 3669, pp. 803-814, 2005

[35] Harvesf, C. & Blough, D., "*The Effect of Replica Placement on Routing Robustness in Distributed Hash Tables*", In: Peer-to-Peer Computing, 2006. P2P 2006. Sixth IEEE International Conference on, pp. 57 -6, 2006

[36] Artigas, M.; Lopez, P. & Skarmeta, A., "*A novel methodology for constructing secure multipath overlays*", Internet Computing, IEEE, 2005, vol. 9 pp. 50 - 57.

[37] Xiang, X., "*Providing Efficient Secure DHTs Routing*", In: Computational Intelligence and Security, 2009. CIS '09. International Conference on, pp. 510 -514, 2009

[38] Ganesh, L. & Zhao, B., "*Identity theft protection in structured overlays*", In: Secure Network Protocols, 2005. (NPSec). 1st IEEE ICNP Workshop on, pp. 49 - 54, 2005

[39] Sanchez-Artigas, M.; Lopez, P. & Skarmeta, A., "*Bypass: Providing secure DHT routing through bypassing malicious peers*", In: Computers and Communications, 2008. ISCC 2008. IEEE Symposium on, pp. 934 -941, 2008

[40] Steiner, M.; En-Najjary, T. & Biersack, E. W., "*A global view of kad*", In: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement, pp. 117-122, 2007

[41] Liang, J.; Kumar, R.; Xi, Y. & Ross, K., "*Pollution in P2P file sharing systems*", In: INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE, pp. 1174 - 1185 vol. 2, 2005

[42] Montassier, G.; Cholez, T.; Doyen, G.; Khatoun, R.; Chrisment, I. & Festor, O., "*Content pollution quantification in large P2P networks : A measurement study on KAD*", In: Peer-to-Peer Computing (P2P), 2011 IEEE International Conference on, pp. 30 -33, 2011
[43] Gummadi, K. P.; Dunn, R. J.; Saroiu, S.; Gribble, S. D.; Levy, H. M. & Zahorjan, J., "*Measurement, modeling, and analysis of a peer-to-peer file-sharing workload*", SIGOPS Oper. Syst. Rev., 2003, vol. 37 pp. 314-329.
[44] Kong, J.; Cai, W. & Wang, L., "*The Evaluation of Index Poisoning in BitTorrent*", In: Communication Software and Networks, 2010. ICCSN '10. Second International Conference on, pp. 382 -386, 2010
[45] Locher, T.; Mysicka, D.; Schmid, S. & Wattenhofer, R., "*Poisoning the Kad network*", In: Proceedings of the 11th international conference on Distributed computing and networking, pp. 195-206, 2010
[46] Obele, B.; Ukaegbu, A. & Kang, M., "*On tackling free-riders in P2P networks*", In: Advanced Communication Technology, 2009. ICACT 2009. 11th International Conference on, pp. 2084 -2089, 2009
[47] Hughes, D.; Coulson, G. & Walkerdine, J., "*Free Riding on Gnutella Revisited: The Bell Tolls?*", IEEE Distributed Systems Online, 2005, vol. 6 pp. 1-.
[48] Sirivianos, M.; Han, J.; Rex, P. & Yang, C. X., "*Free-riding in BitTorrent Networks with the Large View Exploit*", In: In IPTPS '07, , 2007
[49] Jun, S. & Ahamad, M., "*Incentives in BitTorrent induce free riding*", In: Proceedings of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems, pp. 116-121, 2005
[50] Shin, K.; Reeves, D. S. & Rhee, I., "*Treat-before-trick: Free-riding prevention for BitTorrent-like peer-to-peer networks*", In: Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing, pp. 1-12, 2009
[51] Karakaya, M.; Korpeoglu, I. & Ulusoy, O., "*Free Riding in Peer-to-Peer Networks*", Internet Computing, IEEE, 2009, vol. 13 pp. 92 -98.
[52] Bonatti, P.; Duma, C.; Olmedilla, D. & N., S., "*An integration of reputation-based and policy-based trust management*", In: Proceedings of the Semantic Web Policy Workshop, , 2005
[53] Abdul-Rahman, A. & Hailes, S., "*Supporting trust in virtual communities*", In: System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on, pp. 9 pp. vol.1, 2000
[54] Novotny, M. & Zavoral, F., "*Reputation Based Methods for Building Secure P2P Networks*", In: Proceedings of ICADIWT 2008 - First IEEE International Conference on the Applications of Digital Information and Web Technologies, pp. 403-408, 2008
[55] Houser, D., "*Reputation in Auctions: Theory, and Evidence from eBay*", Journal of Economics &amp Management Strategy, 2006, vol. 15 pp. 353.
[56] Resnick, P.; Zeckhauser, R.; Swanson, J. & Lockwood, K., "*The value of reputation on eBay: A controlled experiment*", Experimental Economics, 2006, vol. 9 pp. 79-101.
[57] Marti, S. & Garcia-Molina, H., "*Taxonomy of trust: Categorizing P2P reputation systems*", Computer Networks, 2006, vol. 50 pp. 472-484.
[58] Hoffman, K.; Zage, D. & Nita-Rotaru, C., "*A survey of attack and defense techniques for reputation systems*", ACM Comput. Surv., 2009, vol. 42 pp. 1:1-1:31.
[59] Novotny, M. & Zavoral, F., "*Towards Reliable Trust Management in Insecure P2P Environments*", In: 3rd International Symposium on Intelligent Distributed Computing, pp. 283-288, 2009
[60] Novotny, M. & Zavoral, F., "*Trust Management Systems in P2P Networks and their Resistance against Malicious Collectives*", Agent-oriented Computing for Distributed Systems and Networks, Special Issue of the Journal of Networks and Computer Applications, 2012, .
[61] Kamvar, S. D.; Schlosser, M. T. & Garcia-Molina, H., "*The Eigentrust algorithm for reputation management in P2P networks*", In: WWW '03: Proceedings of the 12th international conference on World Wide Web, pp. 640-651, 2003
[62] Xiong, L. & Liu, L., "*PeerTrust: supporting reputation-based trust for peer-to-peer electronic communities*", Knowledge and Data Engineering, IEEE Transactions on, 2004, vol. 16 pp. 843 -857.
[63] Lee, S. Y.; Kwon, O.-H.; Kim, J. & Hong, S. J., "*A reputation management system in structured peer-to-peer networks*", In: , pp. 362-367, 2005
[64] Liang, .. Z. & Shi, .. W., "*PET: A PErsonalized Trust Model with Reputation and Risk Evaluation for P2P Resource Sharing*", In: HICSS '05: Proceedings of the Proceedings of the

38th Annual Hawaii International Conference on System Sciences, pp. 201-2, 2005
[65] Nandi, A.; Tsuen; Singh, A.; Druschel, P. & Wallach, D. S., "*Scrivener: Providing Incentives in Cooperative Content Distribution Systems*", In: ACM/IFIP/USENIX 6th International Middleware Conference (Middleware 2005), , 2005
[66] Srivatsa, M.; Xiong, L. & Liu, L., "*TrustGuard: countering vulnerabilities in reputation management for decentralized overlay networks*", In: WWW 05: Proceedings of the 14th international conference on World Wide Web, pp. 422-431, 2005
[67] Aringhieri, R.; Damiani, E.; Sabine; Paraboschi, S. & Samarati, P., "*Fuzzy techniques for trust and reputation management in anonymous peer-to-peer systems*", Journal of the American Society for Information Science and Technology, 2006, vol. 57 pp. 528-537.
[68] Sherwood, .. R.; Lee, .. S. & Bhattacharjee, .. B., "*Cooperative peer groups in NICE*", Comput. Netw., 2006, vol. 50 pp. 523-544.
[69] Walsh, K. & Sirer, E. G., "*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, pp. 1, 2006
[70] Xu, .. Z.; He, .. Y. & Deng, .. L., "*A Multilevel Reputation System for Peer-to-Peer Networks*", In: GCC '07: Proceedings of the Sixth International Conference on Grid and Cooperative Computing, pp. 67-74, 2007
[71] Bonnaire, X. & Rosas, E., "*WTR: A Reputation Metric for Distributed Hash Tables Based on a Risk and Credibility Factor*", Journal of Computer Science and Technology, 2009, vol. 24 pp. 844-854.
[72] Bonnaire, X. & Marin, O., "*Recursive Replication: A Survival Solution for Structured P2P Information Systems to Denial of Service Attacks*", Lecture Notes in Computer Science: On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops, 2007, vol. 4806, pp. 931-940, 2007
[73] Zhao, H. & Li, X., "*H-Trust: A Group Trust Management System for Peer-to-Peer Desktop Grid*", Journal of Computer Science and Technology, 2009, vol. 24 pp. 833-843.
[74] Hirsch, J. E., "*An index to quantify an individual's scientific research output*", Proceedings of the National Academy of Sciences of the United States of America, 2005, vol. 102 pp. 16569-16572.
[75] Bonnaire, X. & Rosas, E., "*A critical analysis of latest advances in building trusted P2P networks using reputation systems*", In: Proceedings of the 2007 international conference on Web information systems engineering, pp. 130-141, 2007
[76] Novotny, M. & Zavoral, F., "*Matrix model of trust management in P2P networks*", In: Proc. Third Int. Conf. Research Challenges in Information Science RCIS 2009, pp. 459-468, 2009
[77] Liu, X., "*hiREP: Hierarchical Reputation Management for Peer-to-Peer Systems*", In: 2006 International Conference on Parallel Processing (ICPP 06), pp. 289, 2006
[78] Wang, Y.; Hori, Y. & Sakurai, K., "*Characterizing economic and social properties of trust and reputation systems in P2P environment*", Journal Of Computer Science And Technology, 2008, vol. 23 pp. 129-140.
[79] Feldman, M.; Papadimitriou, C.; Chuang, J. & Stoica, I., "*Free-riding and whitewashing in peer-to-peer systems*", Selected Areas in Communications, IEEE Journal on, 2006, vol. 24 pp. 1010 - 1019.
[80] Mekouar, L.; Iraqi, Y. & Boutaba, R., "*Reputation-Based Trust Management in Peer-to-Peer Systems: Taxonomy and Anatomy*", Handbook of Peer-to-Peer Networking, 2010, pp. 689-732, 2010
[81] Selvaraj, C. & Anand, S., "*Peer profile based trust model for P2P systems using genetic algorithm*", Peer-to-Peer Networking and Applications, 2011, vol. 1 pp. 1-12.
[82] Suryanarayana, G. & Taylor, R. N., "*A Survey of Trust Management and Resource Discovery Technologies in Peer-to-Peer Applications*", , 2004, .
[83] Novotny, M. & Zavoral, F., "*BubbleTrust: A Reliable Trust Management for Large P2P Networks*", Communications in Computer and Information Science: Recent Trends in Network Security and Applications, 2010, vol. 89, pp. 359-373, 2010
[84] Michiardi, P. & Molva, R., "*Core: a collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks*", In: Proceedings of the IFIP TC6/TC11 Sixth Joint Working Conference on Communications and Multimedia Security, pp. 107-121, 2002
[85] Cates, J., "*Robust and efficient data management for a distributed hash table*", , 2003, .

[86] Chun, B.; Dabek, F.; Haeberlen, A.; Sit, E.; Weatherspoon, H.; Kaashoek, M.; Kubiatowicz, J. & Morris, R., "*Efficient replica maintenance for distributed storage systems*", USENIX Association Proceedings of the 3rd Symposium on Networked Systems Design & Implementation (NSDI 06), 2006, pp. 45-58.
[87] Harvesf, C. & Blough, D., "*Replica Placement for Route Diversity in Tree-Based Routing Distributed Hash Tables*", #IEEE_J_DSC#, 2010, .
[88] Novotny, M. & Zavoral, F., "*Resistance against Malicious Collectives in BubbleTrust*", In: Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2011 12th International Conference on, pp. 56 -61, 2011
[89] "*FreePastry*" , http://www.freepastry.org/freepastry
[90] "*P2PTrustSim - Simulation framework*" , https://data.ksi.ms.mff.cuni.cz/svn/P2P_Trust/P2PImplementation/
[91] "*P2PTrustSim - Trust management implementations*" , https://data.ksi.ms.mff.cuni.cz/svn/P2P_Trust/TrustManagement/

Appendix A

List of Tables

Appendix B

List of Figures

Appendix C

Context of attached CD

The attached CD contains the following directories:

Documentations - contains the javadoc documentation of all related packages: FreePastry, P2PTrustSim and TrustManagement.

Results - contains complete simulation results presented in this thesis.

Simulations - contains compiled version of P2PTrustSim and various scripts for run simulations.

Sources - Source code of P2PTrustSim and TrustManagements.

Thesis - PDF version of this thesis and used figures.