UNIVERSITÀ DEGLI STUDI DI PARMA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Dottorato di ricerca in Tecnologie dell'Informazione

XX Ciclo

Natalya Fedotova

SECURITY IN DHT-BASED PEER-TO-PEER NETWORKS

DISSERTAZIONE PRESENTATA PER IL CONSEGUIMENTO

DEL TITOLO DI DOTTORE DI RICERCA

Gennaio 2008

To my Mother

Моей Маме

Table of Contents

1 Introduction 9

2 DHT-based P2P Networks 13

2.1 Overlay Networks 13

2.2 Distributed Hash Tables 14

2.2.1 Chord 18

2.2.2 CAN 20

2.2.3 Kademlia 23

2.2.4 Pastry 26

2.2.5 Tapestry 29

2.3 References 31

3 Vulnerabilities and Security Threats in DHT-based P2P Environment 33

3.1 Specific attacks 33

3.2 Countermeasures 37

3.3 Summary 39

3.4 References 39

4 Trust and reputation management in DHT-based P2P Environment 41

4.1 Reputation in P2P 42

4.2 Reputation management techniques for DHT-based P2P networks 44

4.3 Applicability analysis of some reputation management techniques 47

4.3.1 Supporting Trust in Virtual Communities and Fuzzy Model for Context- dependent Reputation 47

4.3.2 PeerTrust 48

4.3.3 Personalized Trust Model (PET) 48

4.3.4 Poblano 49

4.3.5 NICE 50

4.3.6 XREP 51

4.3.7 Sporas and Histos 51

4.3.8 Beta Reputation System 51

4.3.9 Debit-Credit Reputation Computation (DCRC) and Credit-Only Reputation Computation (CORC) 52

4.3.10 Summary 53

4.4 Combination of different reputation mechanisms for the trust management in DHT-based P2P environment 55

4.4.1 Preliminary remarks 55

4.4.2 Realization details of the proposed solution 56

4.4.3 Findings 63

4.5 Byzantine Agreement for Reputation Management in DHT-based Peer-to-Peer Networks 64

4.5.1 Byzantine Generals Problem in distributed computer systems: history of the problem 64

4.5.2 Applicability of classical solutions for BGP to P2P environment 67

4.5.3 Algorithms and techniques involved in the proposed solutions 73

4.5.4 Reputation evaluation algorithm with use of Byzantine Agreement protocol... 75

4.5.5 Complexity evaluations of the modified and the previous algorithms 79

4.5.6 Summary... 85

4.6 References 86

5 Distributed Hash Tables in collaborative environments..................... 91

5.1 Enterprise networks... 91

5.2 General description of the proposed approach 93

5.3 Kademlia for data storage and retrieval in enterprise networks . 95

5.3.1 Why Kademlia? 95

5.3.2 Proposed scenario of the network organization 96

5.3.3 Assignment of node identifiers 97

5.3.4 Key assignment and data storage procedures.................... 99

5.3.5 Data publication process 102

5.3.6 Modification and update of stored data........................... 104

5.3.7 Security mechanisms and countermeasures involved in the presented solution.. 107

5.4 Findings and future work 109

5.5 References 110

6 Conclusions 113

7 Acknowledgments 117

1 Introduction

Today, peer-to-peer technology has reached the peak of its popularity. Currently, P2P file sharing represents the dominant usage component of Internet bandwidth. Moreover, P2P networks enable sharing of other different computer resources and services, including distributed (collaborative) computing, processing cycles, instant messaging, CPU and storage resources, etc.

Peer-to-peer (P2P) is a communication model in which multiple autonomous devices interact as equals. In a pure P2P network each node implements functions of both client and server, and either peer can initiate a communication session at any moment.

In terms of Internet users, P2P is a sort of “transient” network that allows a group of computers with the same networking software (P2P client) to directly access files from one another's hard drives via Internet connection – simple to join, simple to use. Nevertheless, P2P network is a quite complex system that represents a synthesis of several technological components, and one of them is overlay network.

Nowadays, overlay networks based on Distributed Hash Tables (DHT) are a building block of many peer-to-peer applications. DHT mechanisms provide guaranteed data retrieval, moderate lookup times, automatic load balancing and self-organizing data storage and lookup system [1].

However, DHT-based peer-to-peer networks represent a particular environment susceptible to some specific threats and attacks due to their completely distributed nature without any centralized control. Generally, these attacks are caused by malicious behaviour of some nodes of the network and aimed at routing and lookup processes.

This thesis is meant to find solutions for some specific security problems of DHT-based P2P environment. At the same time, it is also proposed to exploit the advantages of DHT mechanisms applying them to systems where they can be effective against some particular problems. In other words, the research was conducted in two directions:

- DHT as an object to specific attacks;
- DHT as a security improving tool.

Regarding the first direction, we propose some solutions based on the use of trust and reputation evaluation mechanisms to cope with some types of specific attacks of DHT environment. The character of interactions between peers and the presence of “misbehaving” and “honest” peers indicate an analogy between P2P environment and human communities. So, it makes sense to apply reputation evaluation techniques to avoid further contacts with nodes that have already demonstrated malevolent behaviour in order to resolve the problem of polluting routing tables by malicious contacts.

In this work a detailed analysis of applicability of several existent reputation evaluation techniques as protection from some types of attacks in DHT-based P2P networks is presented. Possibilities of incorporation of some reputation mechanisms in DHT routing and lookup processes are analysed. Then we propose a solution that combines different reputation management instruments involved by some analyzed techniques in order to provide a single peer with necessary individual instruments to analyze and independently evaluate reputation and trustworthiness of other peers.

Following this direction, we also apply Byzantine Agreement (BA) concept and some existing solutions for Byzantine failure to cope with some types of malicious activity in P2P networks. It is motivated by analogy between Byzantine failure adversary model and some specific attacks in DHT-based environments. We propose to integrate algorithms for Byzantine Agreement proposed by Lamport, Shostak and Pease for distributed computer systems and some reputation mechanisms designed for DHT-based P2P networks. The goal is to obtain a simpler and efficient reputation management algorithm for the completely distributed P2P environment.

The second direction concerns possibilities of application of DHT mechanisms for data storage and retrieval to systems with a hierarchical organization (such as enterprise networks) instead of use a client-server model. Now, many corporations are looking at the advantages of using P2P as a way for employees to share files without expenses caused by maintaining a centralized server, and as a way for businesses to exchange information with each other directly. For example, many companies of healthcare industry, along with the scientific research and development sectors, use distributed information infrastructure offered by P2P technology to exchange and retrieve important data.

It is proposed to apply DHT principles to enterprise networks in order to avoid some typical problems of centralized environments regarding information security, data retrieval efficiency and reliability. We introduce a distributed peer-to-peer (P2P) data organization system into the enterprise environment in order to create a system that exploits hardware and memory resources of all terminals of the network, provides a reliable data storage system and possibilities of effective collaboration between geographically distant users. The presented solution is based on application of Kademlia DHTs to an enterprise data sharing system.

2 DHT-based P2P Networks

2.1 Overlay Networks

Overlay network is an important functional component of most peer-to-peer applications. This is a virtual network where the nodes are connected with each other by logic or virtual links, and each of these links corresponds to a path that consists of multiple physical links of an exploited transport network (Fig. 2.1). For example, P2P networks are overlay networks in relation to Internet, while Internet via dial-up connection is an overlay for a telephone network.

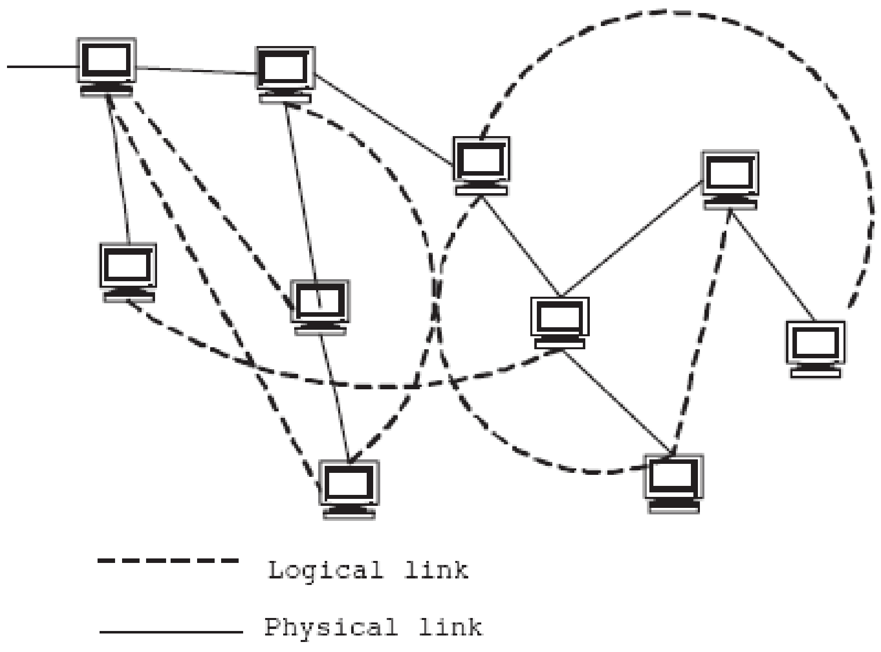


Fig. 2.1Overaly Network

Regarding P2P environment, overlay networks create a structured virtual topology above the basic transport protocol level for implementing lookup processes and some supplementary services. Overlay networks enable routing of messages between peers and search of resources (i.e. IP addresses of nodes that host them) according to predefined lookup protocol.

This work considers structured overlay networks based on Distributed Hash Tables (DHT). Recently, a great number of P2P platforms have adopted DHT lookup mechanisms: eDonkey (Kademlia), BitTorrent (Kademlia), CFS (Chord), OceanStore (Tapestry), etc.

In DHT-based systems a group of distributed hosts collectively manages a mapping from keys to data values without any fixed hierarchy and with very little human assistance. It is realized in accordance with some predefined lookup algorithm, e.g. CAN, Chord, Pastry, Tapestry, Kademlia. DHT-based overlay networks provide guaranteed data retrieval, moderate lookup times, automatic load balancing and self-organizing data storage and lookup system [1]. Let's consider how do DHT mechanisms work.

2.2 Distributed Hash Tables

The base of a typical DHT-based network is a routing table-based lookup service, which maps a given key to a node that is responsible for the key using a hash function. In such system each node is analogous to an array slot in a hash table.

Responsibility for maintaining the mapping from names to values is distributed among all nodes, in such a way that a change in a set of participants causes a minimal probability of disruption. This allows DHTs to scale to extremely large numbers of participants and to handle continual joins, leaves, and failures of nodes.

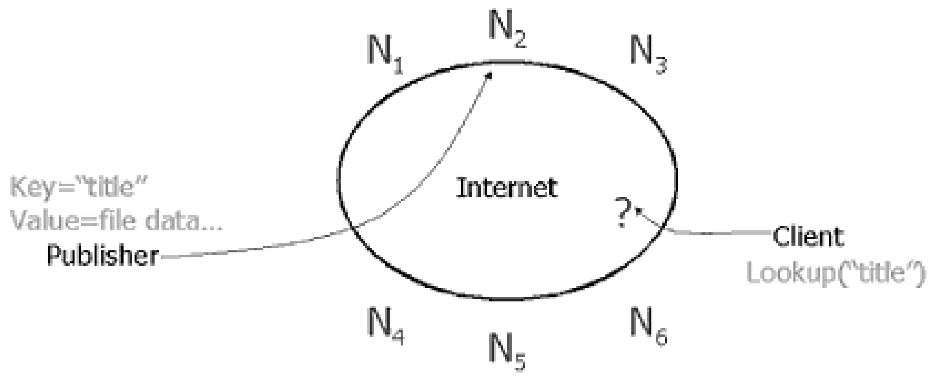


Fig. 2.2 Data storage and retrieval in DHT-based networks

For example, to publish a resource with some predefined name, a user should convert its name to a numeric key using a hash function. Then the publisher invokes a "lookup (key)" operation and sends a file with corresponding metadata to a node with an identifier coinciding with the key (Fig.2.2). The latter should store the file. So, another node, that needs to get this file, should only convert its name into the key, invoke a "lookup (key)" and request a resulting node for a copy of the required file [2]. Hence, the lookup process in such type of networks consists in defining the closest node to a key corresponding to some desired resource. It is important to note, that the concept of "closeness" in DHT-based systems depends on the type of lookup scheme used.

For instance, in Chord the closeness is defined by a numeric difference between two IDs; in Pastry and Tapestry it depends on a number of equal bits in prefixes of two IDs; in Kademlia it's calculated by XOR function applied to a pair of IDs. Anyway, the concept of closeness in this case has nothing in common with geographical distance and concerns only key-space.

Depending on the mode of organization of the identifier space DHT-based lookup algorithms can implement routing in one dimension (Chord, Pastry, Tapestry, Kademlia) and multiple dimensions (CAN).

The data structure of routing tables maintained by existing DHT lookup protocols can present:

- skip-list (Chord);
- tree-like data structure (Pastry, Tapestry, Kademlia);
- rectangles (CAN).

The lookup process can be realized in *iterative* or in *recursive* mode. In the case of iterative lookup (Fig.2.3), a search query is sent to a node that is considered by a requestor to be the “closest” to a desired key amongst all contacts maintained in its routing table. If that node is not responsible for the key, it replies with an ID of the next hop of the lookup. Then the querying peer redirects its request according to this reply. Iterative routing can be performed concurrently, with multiple outstanding requests to decrease latency and reduce the impact of timeouts [3].

When we deal with a recursive lookup (Fig.2.4), the first contacted node forwards the query to a node it regards “closer” to the key than itself without any reply to the lookup initiator. This process continues until the key is found and the query is satisfied.

Despite the particular differences in data structure and routing implementation, all DHT protocols for data storage and retrieval are based on the idea of consistent hashing and they share the following fundamental principle: route a message to a node responsible for an identifier in $O(\log_b N)$ steps using a certain routing metric where N is the number of nodes in the system and b is the base of the logarithm with values (2, 4, 16…) [4].

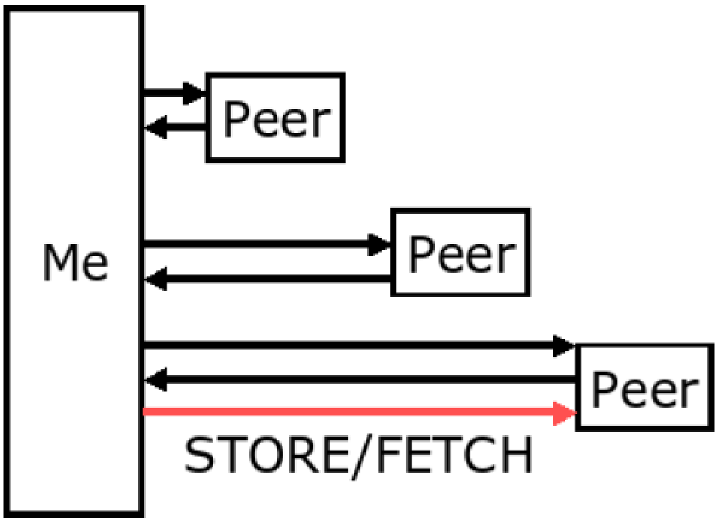


Fig. 2.3Iterative lookup [3]

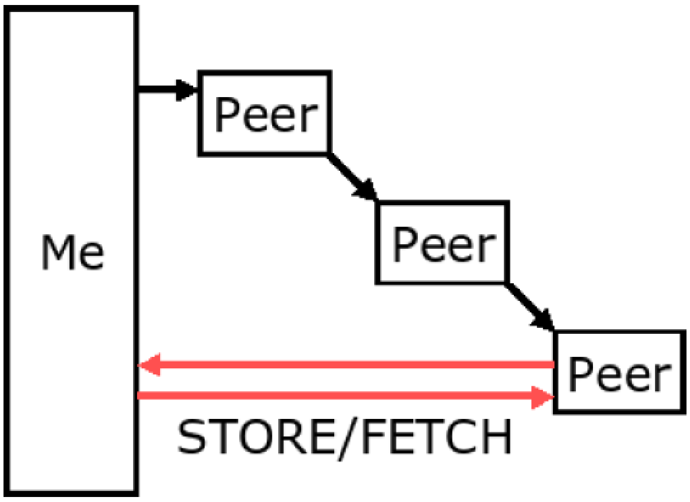


Fig.2.4 Recursive lookup [3]

In the following sections we briefly describe the original DHT protocols mentioned above.

2.2.1 Chord

Chord is a lookup protocol based on consistent hashing that provides fast distributed computation of a hash function mapping keys to nodes responsible for them. This mechanism assigns each node and key (resource) a unique m-bit identifier using a base hash function such as SHA-1. A node's identifier is a result of hashing the node's IP address, while a key identifier is produced by hashing the key.

Chord views the identifier space as a circle formed by no more than 2^m nodes (where m = 160) with identifiers/keys ranging from 0 to $2^m - 1$ [5].

Each node of a Chord network maintains two data structures:

- successor list;
- finger table.

The first is a list of peers immediately succeeding the key in the identifier circle in a clockwise direction. So, a node with the smallest ID that is greater than or equal to *i* represents the successor of a key (or node identifier) *i.* Chord defines a key's successor as a node responsible for the key.

Figure 2.5 shows a simple example of a Chord identifier circle represented by three nodes with identifiers 0, 1 and 3 that are successors of keys 6, 1 and 2 respectively.

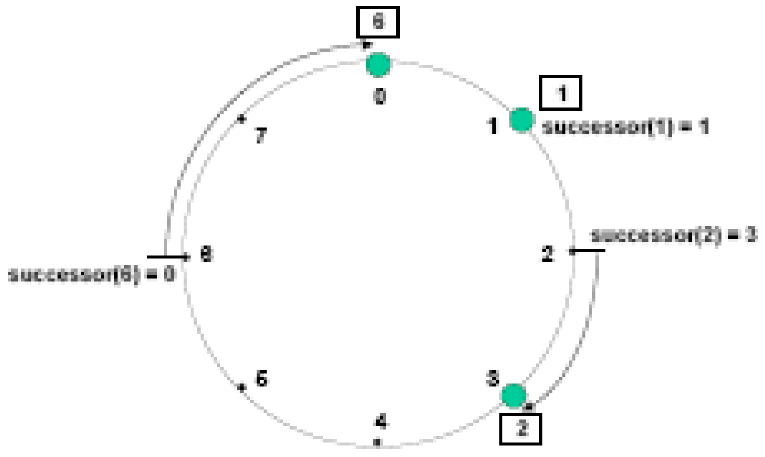


Fig.2.5 Identifier space organization in Chord [5]

This hashing scheme lets nodes join and leave a network with minimal disruption. When a node *n* leaves a network, all the keys it is responsible for should be reassigned to its successor. In the case when a node *n* joins a network, certain keys previously assigned to *n*'s successor pass to *n.* To join a Chord network, a node contacts any peer in the network and requests for an ID to be assigned to the "newcomer". Once the ID is assigned, the node occupies an appropriate position in the identifier circle, and the predecessors of the newly joined peer update their successor lists.

A finger table is a routing table which contains IP addresses of peers halfway around the ID space from the node, a quarter-of-the-way, an eighth-of-the-way and so forth in a data structure that resembles a skip-list (Fig. 2.6). The size of a Chord routing table is $\log_2 N$, where *N* is the number of nodes in the network. If a node is looking for a resource with a key *k*, it forwards the query to a node in its finger table with the highest ID not exceeding *k*. Due to the skip-list structure a desired key can be reached in $O(\log_2 N)$ steps.

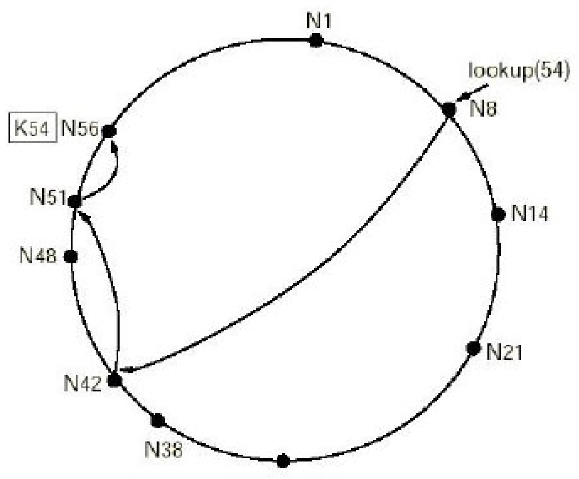


Fig.2.6 Skip-list -like data structure of a routing table in Chord [5]

Hence, we can conclude that a successor list is required for maintaining the correct organization of the identifier space and data structure, while a finger table is meant to speedup lookup processes [3]. The lookup in Chord can be implemented in both iterative and recursive modes, but the requests should be forwarded sequentially.

2.2.2 CAN

Content-Addressable Network (CAN) is a distributed infrastructure that provides "hash table-like functionality on Internet-like scales" [6]. CAN is a scalable, fault tolerant and completely self-organizing system. To organize the identifier space CAN uses a virtual d-dimensional Cartesian coordinate space. A hush function is applied to deterministically map keys (file names) into points in a logical coordinate space. This coordinate space is partitioned dynamically among the peers of the network such that each peer covers a certain region (zone) within the overall space (Fig. 2.7).

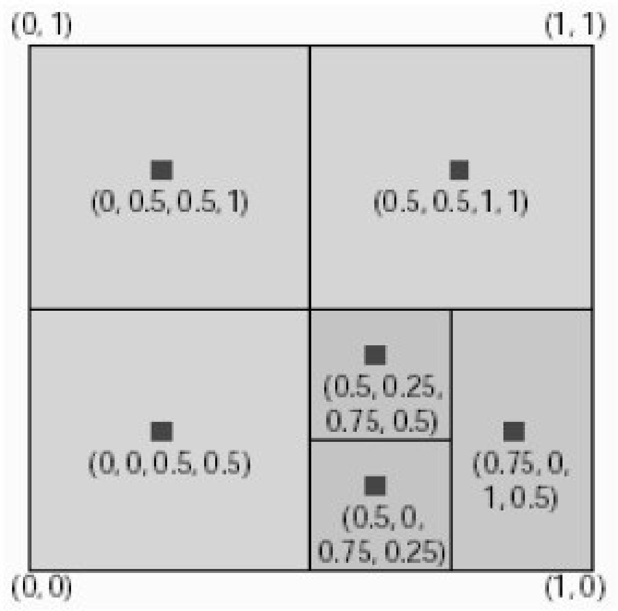


Fig. 2.7 Bidemensional CAN identifier space with 6 nodes [6]

A peer is responsible for storing (key, value) pairs for those keys that are hashed into a point which is located within a zone it covers. Each peer maintains a routing table that contains IP addresses of all neighbour nodes whose virtual coordinate zones are contiguous to its own zone. In a d-dimensional coordinate space, two nodes are neighbours if their coordinate spans overlap along *d -1* dimensions and abut along one dimension.

A lookup operation consists in routing a query towards its destination along a path that approximates a straight between a querying node and a point with the destination coordinates (Fig.2.8). It is implemented by simple greedy forwarding to the neighbor peer closest to the destination.

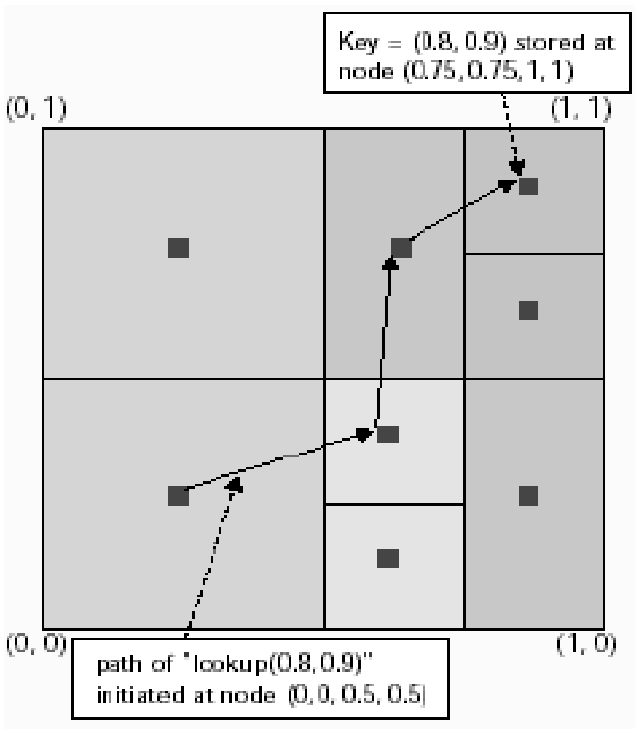

Key = (0.8, 0.9) stored at node (0.75,0.75,1,1)
(0, 1)
(1, 1)
(0, 0)
(1, 0)
path of "lookup(0.8,0.9)" initiated at node (0,0,0.5,0.5)

Fig. 2.8 Lookup in CAN [6]

To join the network a peer chooses a casual point *P* in the coordinate space. Then, the peer contacts a node already in the network and initiates a lookup for a node *n* whose zone contains *P*. Once the node *n* is found, its zone should be split in half and one half should be assigned to the new node.

To update routing tables all node should send an update message followed by periodic refreshes, with their currently assigned zone to all their neighbours. A too long absence of an update message from a peer indicates its failure.

2.2.3 Kademlia

Kademlia [7] is a DHT-based peer-to-peer system based on the XOR metric. So, the distance between to identifiers is defined as: *d (x,y) = x XOR y*.

All nodes and resources in this system have 160-bit identifiers (keys). The data are replicated by finding *k* (the recommended value for *k* is 20) closest nodes to a key and storing the key/value pair on them. As it was noted above Kademlia has a tree-like data structure. So, Kademlia considers network nodes as leafs of a binary tree (Fig.2.9).

Routing processes are implemented in prefix-matching mode. The routing table size is $\log_2 N$ (*N* is a number of nodes in the network) [4].

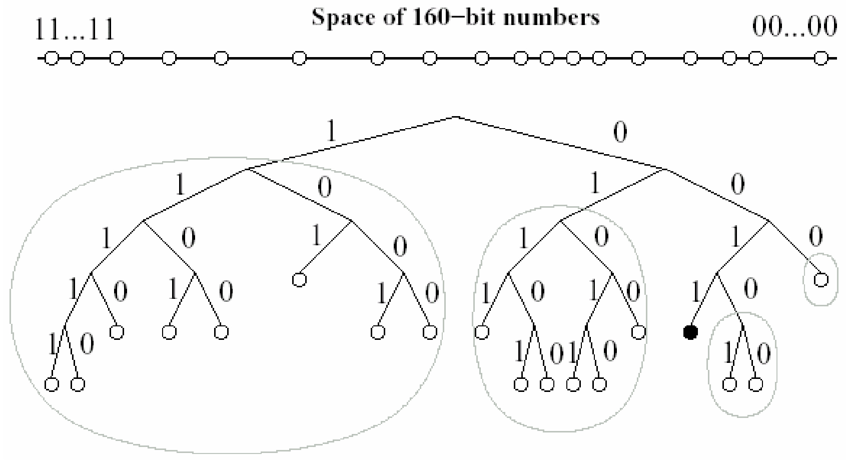


Fig. 2.9 Kademlia binary tree: node 0011... and sub-trees where it has contacts [7]

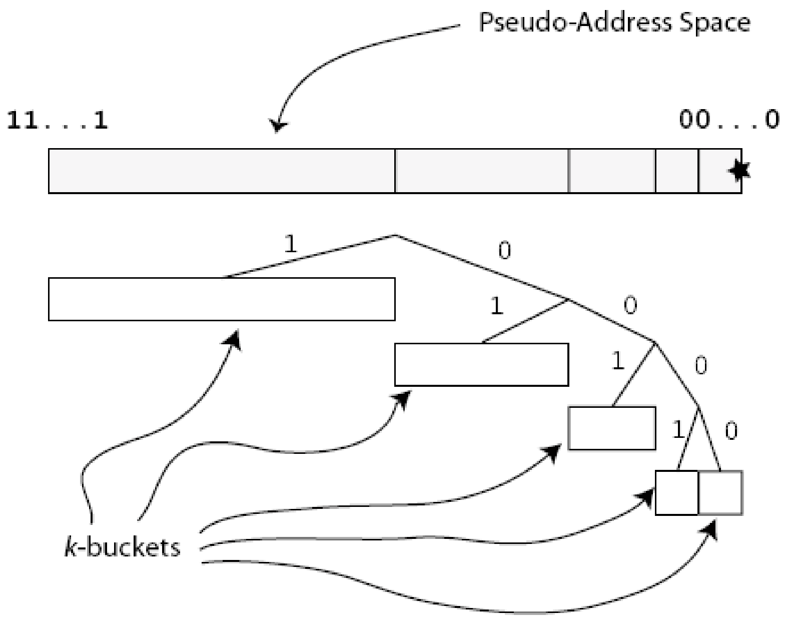


Fig.2.10 Routing table data structure in Kademlia [7]

Each Kademlia node stores information about IP address, UDP port and node ID for nodes from the interval: $d \in \left[2^i; 2^{i+1}\right)$.

Nodes from this interval form a group called *k-bucket* (Fig. 2.10). So, a Kademlia network can be presented as a bucket table. Due to the mechanism of k-bucket a Kademlia node has at least one contact in each sub-tree. This facilitates and makes faster lookup and routing processes: the lookup speed can be increased by considering *b* bits (instead of one bit) at each step, reaching a desired resource in less time.

The symmetry of XOR-metric provides peers with a possibility to learn and update routing information from queries they receive during a lookup process. So, in Kademlia updates of routing tables are implemented by nodes automatically, as a “secondary effect” of ordinary lookups and interactions with other nodes.

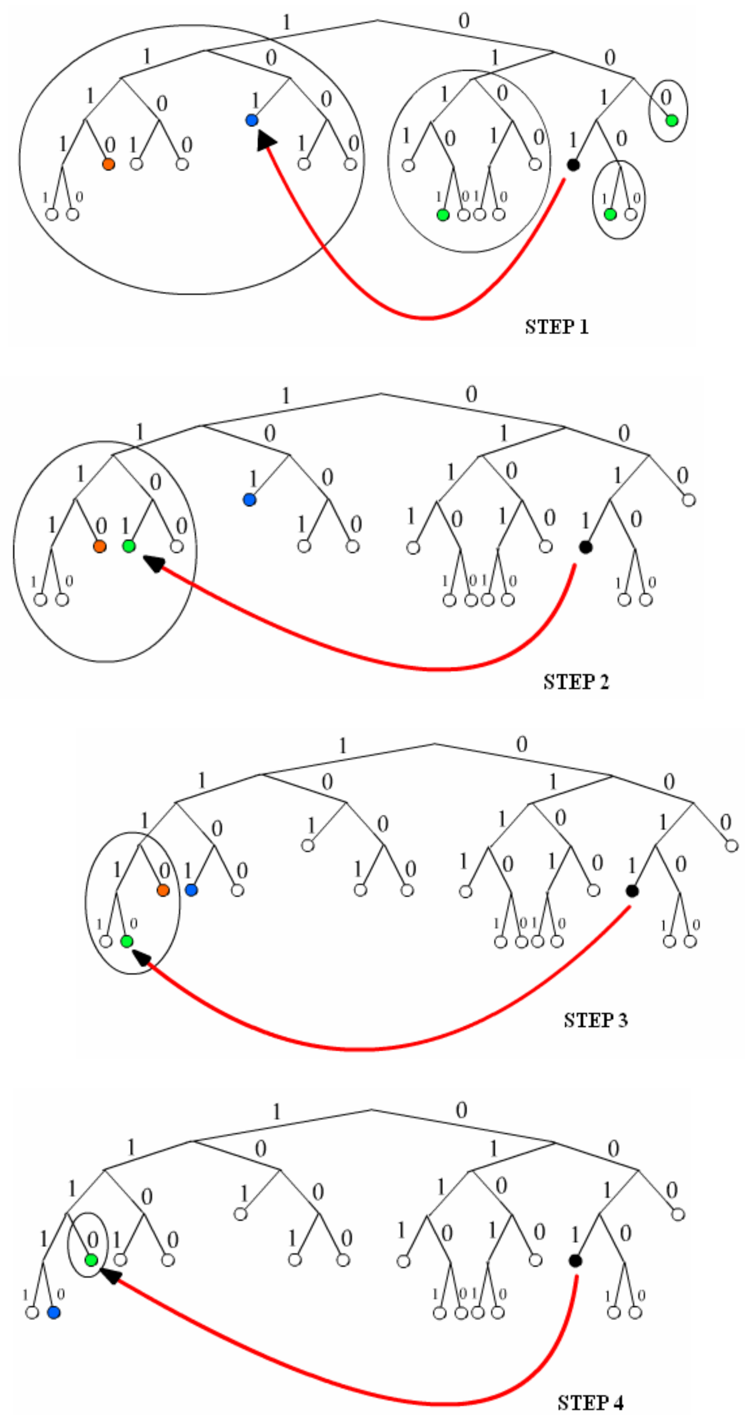


Fig. 2.11 Kademlia lookup [8]

Kademlia uses iterative lookup that is performed in parallel mode [4]: a host contacts peers with progressively smaller XOR distances to the target ID in turn.

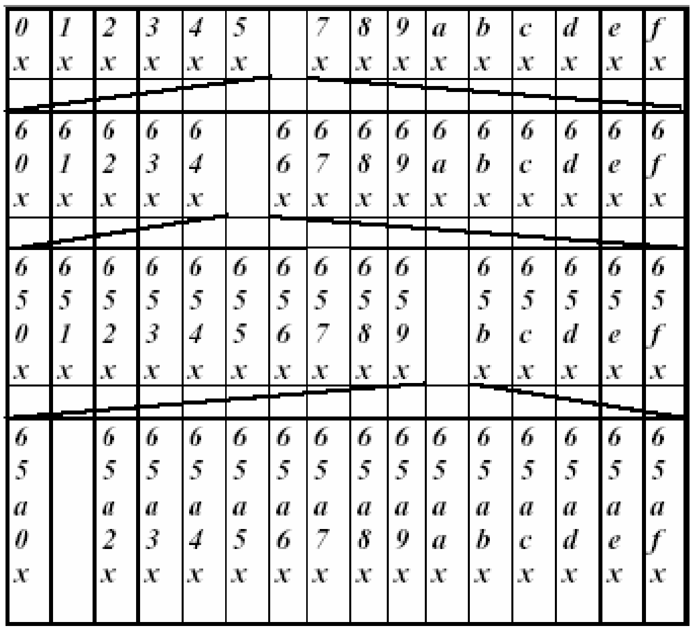
As shown in the figure 2.11, the node with prefix 0011… initiates the process of look-up for some resource, sending FIND_VALUE RPC to a node residing in another sub-tree that is considered as closer to the target resource. The contacted node returns a triple <IP address, UDP port, Node ID> for node 1101… that is closer to the target than itself, and so on. As we can see, every step of the lookup process narrows a ”search area” until the target node is localized.

2.2.4 Pastry

Pastry assigns to each node a unique numeric identifier consisting of 128 bit. Like in Chord, all node identifiers in Pastry can be logically positioned in a circular identifier space [9]. However, routing tables has a tree-like structure and lookup processes are performed by prefix-matching.

Each Pastry node maintains a routing table, a neighbour set and a leaf set.

A peer's routing table is organized into $\log_{2^b} N$ rows with $2^b - 1$ entries each (Fig. 2.12). So, a routing table contains IP addresses of peers with no prefix match, with *b* bits prefix match, *2b* bits prefix match and so on, where *b* is a configuration parameter (typically *b=4*).



0x	1x	2x	3x	4x	5x		7x	8x	9x	ax	bx	cx	dx	ex	fx
60x	61x	62x	63x	64x		66x	67x	68x	69x	6ax	6bx	6cx	6dx	6ex	6fx
650x	651x	652x	653x	654x	655x	656x	657x	658x	659x		65bx	65cx	65dx	65ex	65fx
65a0x		65a2x	65a3x	65a4x	65a5x	65a6x	65a7x	65a8x	65a9x	65aax	65abx	65acx	65adx	65aex	65afx

Fig. 2.12 Routing table of a Pastry node with node ID 65a1x, b=4 [9]

The leaf-set *L* of a node *n* contains information about |*L*|/*2* closest nodes with identifiers that are numerically smaller than that of the node *n* and |*L*|/*2* closest nodes with identifiers that are numerically larger than that of the node *n* (|*L*| is a configuration parameter with a typical value of 16 or 32). The leaf-set in Pastry is conceptually similar to the Chord's successor list.

Routing in Pastry is recursive: each host forwards a lookup message along a chain of nodes to a destination. At each step of a routing process a contacted peer tries to route a lookup message to a node whose ID contains a longer sequence of bits coinciding with those of a sought key than its own ID (Fig. 2.13).

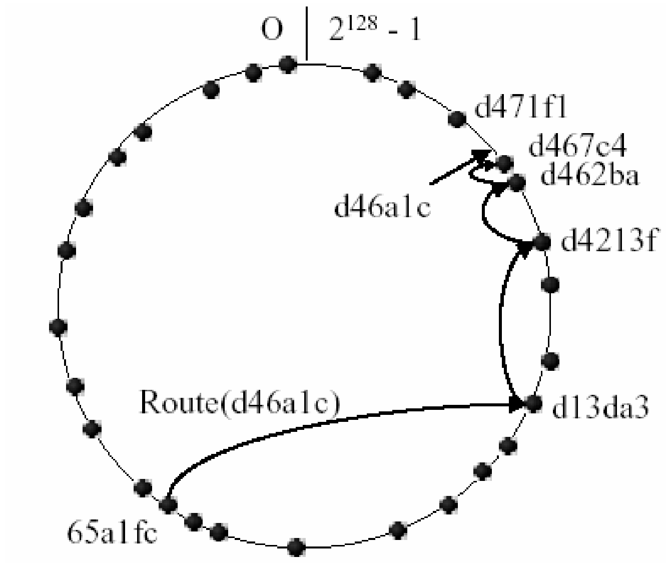


Fig. 2.13 Routing of a lookup message from node 65a1fc to key d46a1c in Pastry identifier space. Black points represent currently active peers [9]

To join a Pastry network a node should contact any active node in the network that implements bootstrap functions for the newcomer. It is realized in the following mode: a new node with identifier X tries to get an active status sending to some currently active node a special message and using X as a key. The message is forwarded by hop-by-hop routing to a node Z whose ID is the closest to X. Then X obtains from Z information about the neighbourhood in order to build its own leaf set. To create its routing table X uses routing data obtained along the path from the bootstrap node to Z. After that, the newcomer announces its “alive” status to the neighbourhood. So, the neighbour nodes should update their routing tables and leaf sets taking in consideration the presence of the new node.

In the case of leaving the network by some node, only leaf sets of neighbours are immediately updated; routing tables data are corrected on demand, only when some node tries to contact a node that is currently is not available.

2.2.5 Tapestry

Tapestry [10] DHT structure is very similar to Pastry system. However, there are some differences regarding key mapping and management of data replicas.

Tapestry is an extensible infrastructure that provides decentralized object location and routing focusing on efficiency and minimizing message latency. This is achieved since Tapestry constructs only locally optimal routing tables and maintains them in order to reduce routing stretch (Fig.2.14). Furthermore, Tapestry allows flexible data (objects) distribution according to particular needs of a given application.

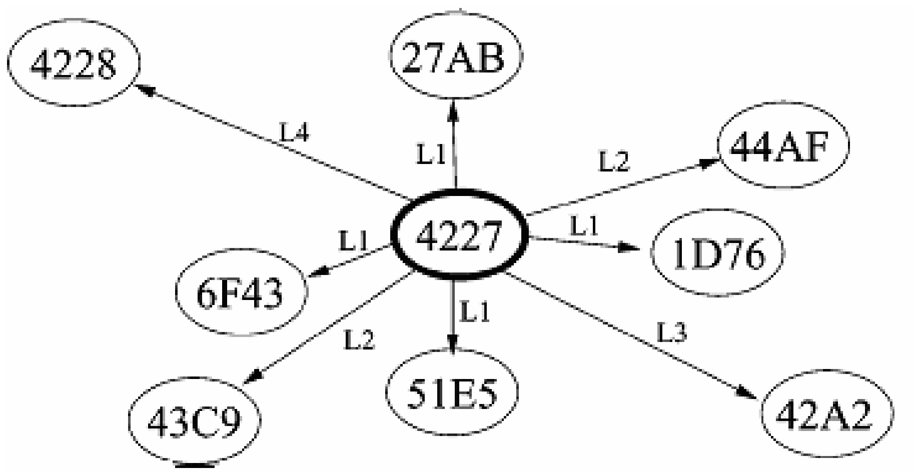


Fig. 2.14 Tapestry routing mesh from the point of view of a single node. Outgoing links point to nodes with a prefix match. Higher level entries match more digits. Together these links represent a local routing table [10]

Each node is assigned a unique nodeID. All node identifiers are uniformly distributed in a large identifier space. Tapestry uses SHA-1 to produce a 160-bit identifier space represented by a 40 digit hex key. Pastry defines specific endpoints GUID's that are similarly assigned unique identifiers. NodeID's and GUID's are roughly and evenly distributed in the identifier space.

Tapestry implements so called “surrogate routing” (Fig.2.15). At each hop of a routing process a message is progressively routed closer to a targeted key by incremental suffix routing. Each routing table has multiple levels, and each level contains links to nodes with IDs matching up to a certain digit position. It means that level 1 has links to nodes with IDs that have nothing in common, contacts of level 2 have the first digit in common, etc. When a certain digit of an ID cannot be matched, a lookup is redirected to some “close” digit. So, each nonexistent ID is mapped to some live node with a similar ID. The number of hops in a routing process in Tapestry is defined as $log_2^b N$ (N is a number of nodes in the network).

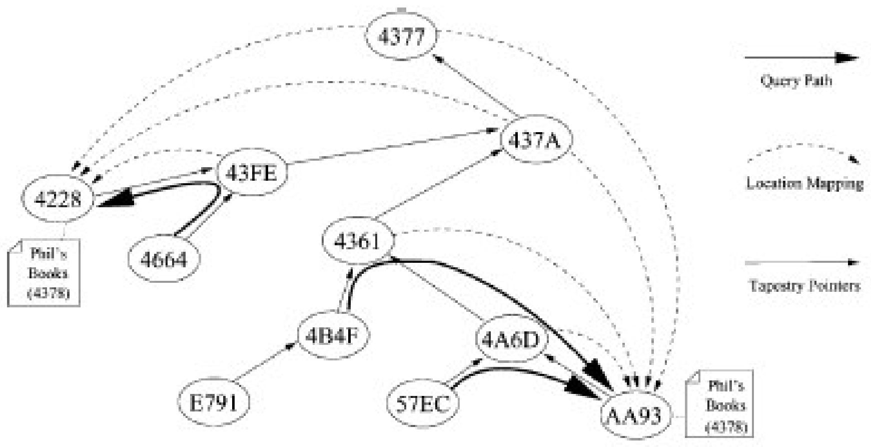


Fig. 2.15 Lookup routing in Tapestry [10]

To join the network a node sends a multicast message to all active nodes with the same prefix (i.e. to those of them that share with the newcomer the longest sequence of digits of the ID). These nodes should add the new contact to their routing tables. Then, the nodes contact the new node to provide a temporary neighborhood list. After that, the new node performs an iterative search for the nearest neighbor contacts to fill all levels of its routing table.

Leaving the network a node informs other nodes about its intention and communicates IDs of replacing nodes for each level of the routing table. Resources and data stored at the leaving node are redistributed or provided from redundant copies.

2.3 References

[1] D. Doval, D. O'Mahony. Overlay Networks: A Scalable Alternative for P2P. IEEE Journal on Internet Computing, Vol.7, No.4, pp. 79-82, August 2003.

[2] H. Balakrishnan, M. F. Kaashoek et al. Looking Up Data in P2P Systems. Communications of the ACM, Vol. 46, No. 2, pp.43-48, Feb. 2003.

[3] S.A. Crosby, D.S. Wallach. An Analysis of BitTorrent's Two Kademlia-based DHTs. Technical Report TR-07-04, Department of Computer Science, Rice University, June 2007

[4] S. Baset, H. Schulzrinne, E. Shim. Common Protocol for DHT Algorithms. Internet-Draft at: http://www.ietf.org/internet-drafts/draft-baset-sipping-p2pcommon-00.pdf

[5] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In Proceedings of SIGCOMM-2001, San Diego, California, USA, August 2001.

[6] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker. A Scalable Content-Addressable Network. In Proceedings of SIGCOMM-2001, San Diego, California, USA, August 2001.

[7] P. Maymounkov, D. Mazières, “Kademlia: A Peer-to-peer Information System Based on the XOR Metric”, in Proceedings of the 1st International Workshop on Peer-to-peer Systems, MIT, March 2002.

[8] http://ntrg.cs.tcd.ie/undergrad/4ba2.02-03/p9.html#Kademlia

[9] A. Rowstron, P. Druschel,. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001), March 2002.

[10] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, J. D. Kubiatowicz. Tapestry: A Resilient Global-Scale Overlay for Service Deployment. IEEE J-SAC, 22(1), January 2003.

3 Vulnerabilities and Security Threats in DHT-based P2P Environment

3.1 Specific attacks

In a DHT-based P2P network adversaries are represented by participants of its own distributed hash lookup system that do not follow the protocol correctly [1]. Thus, this environment is vulnerable to some specific threats and attacks that are generally caused by malevolent behaviour of some nodes of the network and aimed at routing and lookup processes.

Routing attacks that take place in P2P systems using DHT-mechanisms usually consist in *incorrect lookup routing, incorrect routing updates* and *partitions*. Let's consider them in details.

Incorrect lookup routing takes place when some malicious peer tries to forward lookup process to an incorrect or non-existent node (Fig.3.1). As we know, the lookup process in such type of networks consists in defining the "closest" node to a key corresponding to some desired resource. So, each step of the lookup is supposed to get closer to the node responsible for the key. But a malicious peer can confuse the process of routing claiming that some random node is the closest to a sought key. Hence, lookup process can be directed incorrectly and this can prevent a pair key/value from being found. The figure below represents incorrect lookup routing in CAN. The initiator (the node with coordinates [0.0, 0.5, 0.5]) starts the lookup for a key stored at node [0.75, 0.75, 0.1]. At the third step of the lookup the malicious node

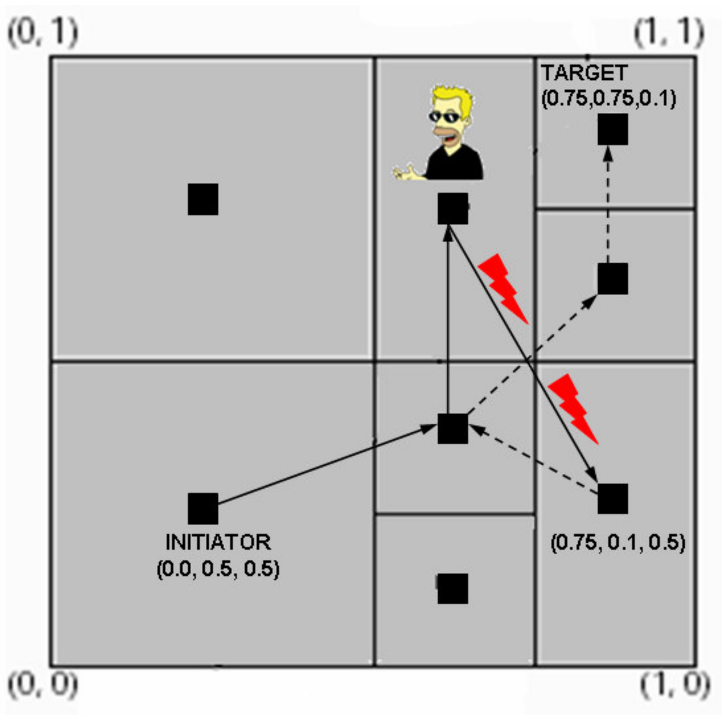


Fig. 3.1Incorrect lookup routing in CAN

incorrectly forwards the query to node [0.75, 0.1, 0.5]. So, this step annuls the search progress that has been reached previously. Anyway, the lookup process can be “saved” by backtracking to the precedent correct step and asking for an alternative hop that maybe offers less progress but directs the lookup in a correct manner.

Incorrect routing updates take place when a malicious peer corrupts routing tables of other peers by sending them incorrect updates. It is possible because in P2P networks using mechanisms of DHT peers create their routing tables by consulting each other. As result, “well-behaving” peers direct their queries to inappropriate or non-existent nodes, as in the case with incorrect lookup routing.

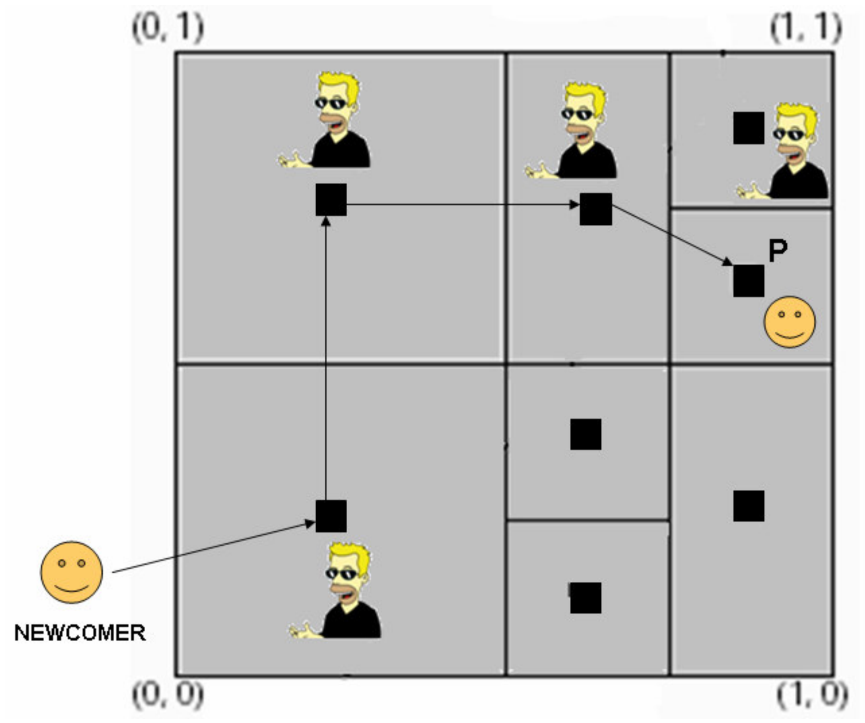


Fig.3.2Partitions in CAN

The problem of *partition* appears while bootstrapping a new peer, i.e. when a new node contacts some already active peer to join the network. So, if some malicious peer has been chosen as a bootstrap node, the newcomer can be partitioned into an incorrect (parallel) network created by a set of malevolent nodes. Fig.3.2 illustrates partitions in CAN network: the newcomer has chosen the point P in the coordinate space to join the network. Unfortunately, the node contacted for bootstrapping is malicious and the point P belongs to a zone controlled by the malicious peer [0.75, 0.75, 0.1]. So, our newcomer is partitioned into the network segment controlled by malevolent users.

The same happens when one of the malicious nodes is cross-registered in the "right" network. So it is able to make new nodes to be connected to the parallel network even if firstly a legitimate node has been contacted to effectuate the bootstrapping process [1].

Rapid joins and leaves represent another type of malicious activity that causes rebalancing process on the network and, as a result, an unjustified excess of data transfers.

Inconsistent behaviour of some node is manifested in its correct behaviour in respect of certain nodes (for instance, its “neighbours” in the identifier space) and misbehaving in regard to others. So, neighbour nodes don't remove such malicious contacts from their routing tables giving them the possibility to participate in routing processes and to continue confusing “less lucky” peers.

Some malicious nodes can follow rules of a lookup protocol correctly, but deny the existence of resources they are responsible for or refuse to provide interested users with these resources. In this case we deal with *storage and retrieval attacks.*

Sybil attack consists in forging multiple identities by a malicious entity in order to obtain the possibility to act as a number of peers with different identities. This type of malicious activity exploits the mechanism of identifier-to-key (ID-to-key) mapping that represents a basic element of DHT-based P2Psystems [2]. As we know, a DHT-based overlay network uses a virtual addressing scheme based on logical identifiers obtained through consistent hashing. Such scheme provides for each entity of the underlay network a corresponding unique identity in the overlay network, i.e. forms a “ID-to-key” mapping pair for each entry [3]. In the case of the Sybil attack malevolent nodes break “one entity-one identity” relation spoofing multiple identities.

In the next section we describe several countermeasures and protective mechanisms provided by DHT-based lookup algorithms to cope with some effects of the above attacks.

3.2 Countermeasures

The self-organizing nature of DHTs enables some countermeasures against several effects of different types of malicious activity described in the previous section.

Incorrect lookup routing can be detected by checking the progress of lookup at each step. In the case of absence of any progress (blatantly incorrect query forwarding), lookup process is backtracked to the previous “right” step and then proceeds with looking for an alternative direction of the search that maybe offers less progress but leads to a desired target. This checking procedure makes the routing and lookup processes slower, but helps to prevent a lookup from failure.

In the lookup algorithms with iterative character (Kademlia, Chord) incorrect lookup rooting is easy to detect due to the possibility of lookup progress control at each step after a corresponding <key; value> pair has been returned by a contacted peer.

In the case of recursive lookup it is problematic to apply verifying mechanisms at each step, as a query is forwarded without interacting with the requestor. So, this countermeasure is not applicable to recursive lookup.

Incorrect routing updates can be prevented by setting certain requirements for correct routing updates that should be verified. For example, in Pastry routing updates are considered as correct if each table entry has a correct prefix. So, blatantly incorrect updates can be easily identified and annulled. Hence, it is important to verify whether a newly updated contact is reachable (existent) before introducing it into a routing table [1].

In Kademlia the problem of incorrect routing updates is solved due to the particular mechanism of updates used in this system: every routing table update is implemented by a Kademlia node automatically, as a “secondary effect” of ordinary lookups and interactions with other nodes. In this mode each update inserted into a routing table is verified by the previous experience of the node.

The problem of inconsistent behaviour may be resolved by implementing routing by short hops only through close (“locally good”) nodes. In this case each participant of the lookup has to demonstrate a good behaviour interacting only with its neighbours. As we know, misbehaving is not convenient for malicious nodes in this case because it causes removing malevolent contacts from routing tables and, as a result, impossibility to participate in further routing processes. However, almost all routing systems use hops toward distant points in the identifier space to reach a desired key in less time [1].

Storage and retrieval attacks can be prevented by replication of files using multiple hash functions. In such way we avoid the responsibility of a single node for replication or facilitating access to the replicas. So, if some node refuses to provide a sought resource, the last can be obtained from another responsible peer.

To resolve the problem of partitions it is proposed to implement bootstrapping through some trusted nodes. The trusted nodes can be represented by some predefined authority or by some nodes that have been previously discovered or successfully used as bootstrap by a node rejoining the network.

Sybil attacks cannot be excluded in a distributed computing environment, but a lookup efficiency can be improved by parallel routing (issuing α lookup requests at a time) [4]. The solution that is frequently used to resolve this problem is establishing a trusted certificate authority that can guarantee a one-to-one correspondence between entity and identity [5].

3.3 Summary

The countermeasures provided by the nature of DHT-based lookup algorithms have a “short-term” character: they help to cope only with instantaneous effects of malicious activity and usually don’t resolve a problem of detection and elimination of malevolent contacts from routing tables. Moreover, as we have seen, for some specific security problems of DHT-based environment opportune countermeasures don’t exist (e.g. incorrect routing updates in the case of recursive lookups, inconsistent behaviour, Sybil attacks).

As mentioned above, some solutions require use of centralization elements that contradict the completely distributed nature of P2P networks. Moreover, it means introducing into the system a central critical point and significant increase of maintenance costs. Thus, such solutions involve some typical problems of centralized systems.

In the next chapter some ways to resolve the above problems are proposed.

3.4 References

[1] E. Sit, R. Morris. Security considerations for Peer-to-Peer Distributed Hash Tables. In Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS’02), Cambridge, Massachusetts, March 2002

[2] M. Srivatsa, L. Liu. Vulnerabilities and Security Threats in Structured Overlay Networks: A Quantitative Analysis. In Proceedings of ACSAC-2004

[3] L. Wang. Attacks Against Peer-to-Peer Networks and

Countermeasures. Seminar on Network Security, Helsinki University of Technology, December 2006

[4] D. Stutzbach, R. Rejaie. Improving Lookup Performance over a Widely-Deployed DHT. In Proceedings of the 25th IEEE International Conference on Computer Communications, INFOCOM'06, April 2006

[5] J.R. Douceur. The Sybil Attack. In Proceedings of the 2nd Annual IPTPS Workshop, 2002

4 Trust and reputation management in DHT-based P2P Environment

"...Ideas that have great results are always simple ones.
My whole idea is that if vicious people are united and constitute a power,
then honest folk must do the same. Now that's simple enough..."
Lev Tolstoy, "War and Peace", 1869

To resolve some security problems considered before there is need to apply mechanisms based on analysis of the activity of peers and the acquired reputation in order to "clear" routing tables from contacts that have evinced malicious or inconsistent behaviour to avoid them in the future.

Applying opportune mechanisms for verifying lookup progress, the querying node can make a conclusion about "honesty" of the nodes participating in the lookup process, assigning to them the corresponding reputation values. Analogically, a node that honestly shares its resources with other nodes gets reputation "points", and a node denying the existence of data it is responsible for (storage and retrieval attacks), loses them.

In the case of recursive lookups to avoid forwarding queries to malicious peers (incorrect lookup routing), all participating nodes (not only a lookup initiator but also each intermediate node) should control all stored reputation values and choose among possible hops the most reliable one.

Reputation techniques based on exchange and analysis of opinions of different nodes can be used in the case of inconsistent behaviour. These techniques can be applied to make nodes realize that some "locally good" nodes are malicious in respect to distant peers.

Thus, to cope with some types of malicious activity the collaboration between "honest" network nodes is indispensable.

In this chapter we analyze possibility of application of existent reputation techniques to DHT-based P2P systems.

We also propose integration of reputation mechanisms with other instruments used in distributed computing environment in order to improve resilience of such systems to destructive actions of malevolent or faulty components. The goal of this integration is to obtain a more efficient, less expensive (in terms of data transferred, computational resources involved and time spent) and possibly simple solution to cope with the specific problems of DHT-based environment described in Chapter 3.

4.1 Reputation in P2P

In P2P networks, like in any human community, nodes (users) interact, create new contacts, and progressively gain their own experience and reputation. These two factors help them to evaluate trustworthiness of other nodes and to understand what kind of behaviour can be expected from a certain node. Hence, the entities that enjoy a high reputation are considered as trusted.

According to Abdul-Rehman and Hailes [1]*"reputation is an expectation about an individual's behaviour based on information about or observations of its past behaviour"*. So, we can see that the reputation and the experience are particularly interrelated factors. To evaluate the reputation of some individual it is possible to use an own direct experience, recommendations and experiences of other persons, or all these factors.

The reputation is an integral part of the trust concept and it is very important for the establishment of trust relationships between two entities. Grandison and Sloman [2] define trust as *"the firm belief in the competence of an entity to act dependably, securely and reliably within a specified context"*. So, while the reputation concept considers only real facts regarding the behaviour and the activity of some entity in the past to evaluate a level of trustworthiness of the entity, the trust often can be based on such subjective factors as recommendations of some friends, intuition or banal sympathy.

Recently, a number of trust and reputation management techniques for P2P networks has been proposed by different researchers.

All existing decentralized trust management techniques for P2P communities can be divided in two groups depending on the approach used to establish and evaluate trust relationships between peers [3]:

- credential and policy based
- reputation based.

In credential and policy based trust management systems peers use a set of credentials and policies to determine whether a certain peer can be trusted or not. This approach is typically used for authorization and access control in open systems, and it is meant for systems with strong protection requirements. Obviously, in this case the presence of some sort of certification authority is required. Such techniques often require a central server for storing and distributing reputation information. Therefore, credential and policy based mechanisms are to be applied in centralized systems with a hierarchical structure.

The backbone of each reputation-based technique is a trust computational model that provides mechanisms to evaluate the level of trust toward both a resource and its possessor. In this case the reputation management is based only on the recommendations and direct experiences of the users. Normally, the data (opinions regarding reputation of other users) exchanged by the peers is not signed by certification authorities, but it can be self-signed by the source of the information [4].

4.2 Reputation management techniques for DHT-based P2P networks

In the case of completely distributed DHT-based P2P networks, we need the techniques providing mechanisms that can be realized in decentralized systems with instruments applicable to the overlay network environment. In such networks a central server, that assigns a univocal reputation value to each peer, is absent. So, each single peer should be provided with all necessary means to analyze and independently evaluate the reputation and the trustworthiness level of other peers.

Thus, taking in consideration the particularities of deploying reputation mechanisms in a DHT setting, we introduce the following applicability criteria for reputation mechanisms:

1. technical realizability in overlay networks;
2. availability of individual reputation evaluation instruments.

The first criterion is related to a category of reputation evaluation technique: pure reputation or with credential and policy elements.

The second criterion is represented by some different parameters, such as:

- possibility to provide recommendations;
- possibility to “weigh” recommendations, i.e. recommendations from different peers have different levels of trustworthiness;

- responsibility for the behaviour of recommended entities;
- evaluation of the community context (the average level of vulnerability of the network environment and the level of cooperation between peers);
- incentives for feedback compilation.

A reputation technique that satisfies the above criteria can provide a single node with all necessary instruments for independent evaluation of trustworthiness levels of other nodes. Only techniques based on pure reputation mechanisms without credential and policy elements can offer this possibility.

Reputation techniques specifically designed for DHT-based networks are heavily based only on evaluation mechanisms of successful and unsuccessful downloads. These techniques regard only file sharing P2P applications [5,6]. However, P2P technology also supports instant messaging, collaborative applications, distributed computing, etc.

In the last few years P2P systems have been successfully used for sharing computation under various distributed computing projects like FightAIDS@Home, Genome@Home, Seti@Home, United Devices Cancer Research Project and others [7]. These projects represent a public-resource computing that relies on personal computers with excess capacity, such as idle CPU time, to resolve some complex research problems. Public-resource computing is an aspect of the peer-to-peer paradigm, even if it uses a grid technology to realize its tasks. Currently, such systems approach a DHT nature. Some steps of computational processes become completely independent from central servers: calculations results of some node are stored in the network and retrieved by a successor that use them for its own part of the task; if a peer leaves a network while processing a work unit, the work unit is eventually sent to another peer that becomes responsible for it (like in

DHT data storage systems when a node becomes responsible for resources of some failed node if their identifiers are considered as the closest to each other).

Another type of systems that use DHT principles are collaborative applications for data storage and editing by several geographically distant work groups. Such systems should provide a rapid and secure data exchange between different system units and possibility of team-work in real-time and transparent mode.

The systems described above represent active distributed collaborative environments, where every interaction between peers is important and, as L. Lamport said, *"the failure of a computer you didn't even know existed can render your own computer unusable".* So, in these systems a number of successful downloads cannot be a sufficient instrument for reputation management. In collaborative environments it is also very important to consider possible risks and various parameters regarding the community context (number of lookup requests without response, number of join and leaves for a node, off-line status time), because the cost of a mistake, caused by a malicious activity in such systems is incomparably higher than in file-sharing networks. Just some unreliable peers that have not been discarded form routing tables in time can interrupt a long chain of calculations.

In the next section we present a detailed analysis of applicability of several existent reputation evaluation techniques as protection from some types of attacks in P2P networks based on DHTs. The analyzed techniques are not designed for DHT-based environment, so none of them represents a universal solution for such systems. At the same time, different reputation management instruments used by these techniques could be a quite effective in some particular cases in a DHT environment. These techniques give the possibility to evaluate the community context parameters mentioned above.

4.3 Applicability analysis of some reputation management techniques

Here we briefly describe some distributed trust and reputation management techniques, underlining some specific characteristics and mechanisms. Then, the possibility of application of these techniques to the DHT-based P2P environment is analyzed.

4.3.1 Supporting Trust in Virtual Communities and Fuzzy Model for Context- dependent Reputation

These two models propose relatively simple solutions regarding the reputation data management. Both of them have a possibility to provide recommendations.

In the first case [1] the trustworthiness of recommendations is defined by the "semantic distance" between the recommendation provided by some entity and other entity's own perception of the recommender's trustworthiness. So, the "semantic distance" is a value applied to a recommendation (that may be subjective or lying) to obtain possibly realistic information based on one's own opinion of a recommender.

In the second case [9] we don't have a mechanism for evaluation of the trustworthiness of recommendations, but they can be expressed with different "levels of certainty". A recommender can be absolutely sure of the future behaviour of a recommended entity or can have some doubts about it, and this technique gives him the possibility to express it. Moreover, in this model the behaviour of a recommended entity affects (in a balanced and perfectly symmetric mode) the reputation of the recommender. So, the network presents a particular community, where each entity is responsible in some degree for events that take place here.

4.3.2 PeerTrust

PeerTrust [10] presents the most complete model among all the analyzed techniques. It takes in consideration a lot of parameters of great importance to calculate a reputation value: feedbacks and the trustworthiness of entities realizing them; transactions and conditions in which they are executed; community and environment context. The “weight” of each of these parameters in evaluation of the reputation of a single node can be modified depending on the situation, and this possibility render this model more flexible than others. The solutions regarding the algorithm for calculation of the trustworthiness level propose mechanisms of defence against malicious behaviour of some nodes who:

- provide good services, but compile feedbacks incorrectly to confuse other peers;
- due to collaboration with other malicious peers gain a high reputation value according to feedbacks provided by the malicious “allies”.

This technique propose to peers an incentive to make them compile feedbacks correctly, assigning to “good” nodes a corresponding recompense. There is another very important feature of PeerTrust that helps to cope with concomitant problems of dynamic character of P2P environment: the peers should keep in consideration feedbacks obtained during some predefined time interval. Then, comparing the information received in different intervals of time, it is possible to find out the peers with inconsistent behaviour.

4.3.3 Personalized Trust Model (PET)

PET [11] has two main particularities in respect of other techniques analyzed:

1. recommendations play a very modest role in calculation of the trust

value, and all of them have the same level of importance and trustworthiness. It is explained by the fact that an entity, considered as trusted by some peer, is not automatically considered as trusted by another peer;

2. the highest priority is assigned to direct experiences of the peers, that undoubtedly provide the highest level of certainty.

The incidence of these components in the analysis concerning trustworthiness of a single peer is modifiable, but anyway, the incidence of the recommendations shouldn't exceed 20%.

This technique also involves mechanisms to resolve the problems caused by the dynamic character of peers. Here, like in PeerTrust, it is proposed to analyze information received from other peers (feedbacks, recommendations) at stated intervals. This model can be quite efficient in environments, where peers' status is particularly dynamic, and the great part of them is unreliable.

4.3.4 Poblano

Poblano [12] is a distributed trust model created by JXTA developers, that proposes solutions, which are completely diverse from those we have just described. Here, the analysis is focused on the trust based on interests of different peer groups. Discussing the precedent techniques, we always considered such aspects as "honesty" and reliability of single peers, trustworthiness of their recommendations, but we didn't take in consideration the quality of available resources. In this case, the trust relationships are established on the base of quality evaluation of data (resources) provided by users (peers). It is important to note, that each peer evaluates the data representing its sphere of interests, associated with some specific Codat. Codat is defined as a unit of information (that can present either code or data) shared and exchanged within a single peer group. So, in Poblano the evaluation of the reputation of some single node is implemented on the base of quality of resources provided by this node.

An algorithm of calculation of the trust value proposed in this technique, takes in consideration not only the original quality of a resource provided by a certain node, but it regards also the quality of the path the resource has gone through before being read by a requestor node.

In the context of data authentication, this model involves some principles of “Web of Trust” conception and use of certificates, both those self-signed and signed by Certification Authority.

4.3.5 NICE

The characteristic that makes NICE [13] unique in respect of other techniques consists in the following: at the end of an interaction each peer creates a cookie, where it registers a level of its satisfaction of the transaction’s results, signs it, and then the signed cookies are exchanged by the interaction’s participants. But the cookies can contain either positive or negative estimations. In the case of the positive estimation the cookies are exchanged by interacting peers as noted above, and each of them saves these cookies as a proof of its high reputation. In the case of negative estimation they are retained by the peers that create it.

So, when a peer requests for a certain resource from another peer, that it has interacted with before, it presents the provider with a cookie signed by the provider itself. The provider peer verifies its own signature and accepts the cookie as a proof of the requestor’s trustworthiness. If the peers have never interacted before, the requestor should find a “path of trust” between itself and the provider, and presents this path instead of a “direct cookie”. Another very important characteristic of NICE is the establishment of a solid cooperation between “good” peers in order to isolate malicious peers.

4.3.6 XREP

XREP [14] proposes a model, where each peer has its experience repository, that contains information about the quality of the resources and the trustworthiness of other peers, that it has directly interacted with before. Once a needed resource has been individuated, a peer initiates a “vote process” involving all peers, that have had a direct experience regarding a certain resource and can present the corresponding information registered in their repositories. In this way, the requestor has the possibility to compare opinions of all these peers and then to make its own conclusion about the trustworthiness level of this peer.

The main disadvantage of this technique is a great number of messages the peers exchange with each other during the vote process.

4.3.7 Sporas and Histos

These are very similar models [15], based on a quite simple mechanism. Anyway, both of them offer to each peer the possibility to analyze data received at stated time intervals and even to personalize these intervals. Moreover, peers can differentially evaluate opinions about behaviour of other peers, taking in consideration the reputation of the authors of these opinions. In some degree it is possible to consider Histos as evolution of an algorithm defined in Sporas. Histos exploits relations between peers, that already have been evaluated, to provide an estimation mostly personalized.

4.3.8 Beta Reputation System

This technique [16] is entirely based on the statistical theory and uses probability density function beta as a key instrument. Originally this model was created for a system with a more centralized character, but it is possible to adapt it in decentralized systems too, but it is easier to realize this model in decentralized environment with some elements of centralization. Anyways, this technique consists of different blocks, that can be utilized and combined according to certain requirements.

Beta Reputation System supports such mechanisms as: management and analysis of feedbacks from multiple sources; differential evaluation of opinions about peers and their resources provided by other peers; data analysis at stated personalized time intervals.

4.3.9 Debit-Credit Reputation Computation (DCRC) and Credit-Only Reputation Computation (CORC)

These techniques [17] propose a partially distributed solution, that involves the presence of special peers called "reputation computation agents" (RCA). These agents periodically calculate and validate "points" assigned to different peers during their interaction with other peers. These points represent so called "reputation score". So, RCAs determine and "formalize" a reputation value of each peer. The points is derived from credits assigned to peers when they implement activities useful for the community: elaboration and forwarding of queries, providing resources and remaining on-line for a long time.

The debits (in DCRC) reduce the points and it takes place when peers download resources provided by other peers of the network. Each peer generates a couple of keys (private and public) and registers it at the RCA host. The digest of public keys is used by the RCAs to identify a peer.

4.3.10 Summary

The summary table below contains data regarding all the basic characteristics of the analyzed reputation models.

Concerning the first applicability criterion, almost all the analyzed techniques can be subsumed under the reputation based category. NICE and DCRC/CORC realize some credential and policy elements: digital signatures of cookies in NICE, peer identification by "reputation computation agents" (RCAs) using public key in DCPC/CORC. Poblano and XREP also involve some mechanisms with a centralized nature. This fact represents some difficulties for application of these techniques to completely distributed DHT-based networks.

As to the second criterion, all these techniques have different completeness degrees. PeerTrust and Fuzzy Model represent the most complete techniques, as they realize almost all possible mechanisms for evaluation of a peer's trustworthiness.

We can conclude that none of these techniques in its pure form represents a suitable solution for DHT-based P2P networks. At the same time, different reputation management instruments used by these techniques could be a quite effective in some particular cases in DHT environments. So, in the next section we propose a way to exploit advantages of single mechanisms provided by different reputation management models.

Parameter Model	Reputation Value Scale	Providing recommen-dations	Possibility to "weigh" recommen-dations	Responsibility for the behaviour of recommended entities	Transaction consideration	Community context	Incentive to feedback compilations	Credential and policy elements
Supporting Trust in Virtual Communities	4 possible levels	Yes	Yes	No	No	No	No	No
Peer Trust	Normalized from 0 to 1	Yes[1]	Yes[2]	No	Yes	Yes	Yes	No
Personalized Trust model	Normalized from 0 to 1	Yes	No	No	No	No	No	No
Fuzzy Model	Normalized from 0 to 1	Yes	Yes	Yes	Yes[3]	Yes	No	No
Poblano	6 possible levels [-1; 4]	No	No	No	No	No	No	No
NICE	Normalized from 0 to 1	Yes[4]	No	No	No	No	No	No
XREP	Binary [5]	Yes[6]	No	No	No	No	No	Yes
Sporas and Histos	From 0 to 3000 exclusive	Yes[1]	Yes[2]	No	No	No	No	No
Beta	Normalized from 0 to 1	Yes[1]	Yes[2]	No	No	No	No	No
DCRC/CORC	Non- negative	No	No	No	Yes	No	No	Yes

1 – feedbacks compiled by other users
2 – regards an entity compiling a feedback
3 – only the events' number is considered
4 – recommendations regarding some determined peer
5 - a binary value scale is not obligatory
6 – in the form of a vote

4.4 Combination of different reputation mechanisms for the trust management in DHT-based P2P environment

The solution we propose is a combination of different instruments for reputation management offered by the analyzed techniques: each of these instruments should be applied when it is considered as the most efficient one for a certain situation. In this section we present a possible scenario of application of the proposed solution to a P2P network based on Kademlia DHTs, extracting some reputation evaluation instruments from the models analyzed before and adapting them to particularities of DHT-based environment.

4.4.1 Preliminary remarks

As an environment for our scenario we chose a network based on Kademlia DHT protocol that is widely used by a number of P2P platforms (eDonkey, BitTorrent, etc). Taking in consideration that in Kademlia lookups are implemented in iterative mode, we can describe the following model of integration of reputation mechanisms and lookup processes:

- each node of the network after every contact with another node assigns a new reputation value to the contacted peer depending on the interaction results;
- all the assigned reputation values are to be stored by the querying node and should be consulted before contacting corresponding nodes again;

- these reputation values are used as recommendations that each node exchanges with others, sending them with the corresponding IP addresses that indicate the next step of iterative lookup.

The proposed combination of the reputation instruments includes:

- risk evaluation method provided by PET model [11];
- resources and servent repositories from XREP model [14];
- debit-credit based reputation computation model (DCRC) [17].

The proposed scenario represents a situation in which a peer joins a network the first time and initiates a lookup process for a data file with a certain ID. Since it is a new node for this network, it has no idea about trustworthiness of other nodes.

4.4.2 Realization details of the proposed solution

As reputation is an accumulative value, it is not possible for a newcomer to evaluate someone's reputation after the first contact. However, it is possible to define a level of vulnerability of the network environment on the base of results of the first experiences using the appropriate instrument offered by PET.

PET model derives the trustworthiness value T from two components: reputation factor R_e and risk factor R_i with different weights (incidence) W_{R_e} and W_{R_i} respectively. The trustworthiness value is defined as follows:

$$T = \begin{cases} (\alpha, 1-\alpha) \times (R_e, (1-R_i))^T, 0 \le \alpha \le 1 \\ R_e, if R_i = 0 \\ R_i, if R_e = 0 \end{cases} \tag{4.1}$$

where $W_{R_e} = \alpha$ and $W_{R_i} = 1 - \alpha$, and the values of *T*, R_e and R_i are all from 0 to 1.

So, in our scenario $T = R_i$, as the node doesn't have sufficient data to evaluate the reputation factor, but it is necessary for it to define a level of vulnerability of the network it has joined. The risk value in PET model is calculated by the formula:

$$R_i = \frac{\sum_{i=B,N,L}(N_i \times h(i))}{h(B) \times \sum_{j=G,B,N.L} N_j} \qquad (4.2)$$

where *G, B, N, L* are four levels of quality (*Q*) of services provided by a peer (it is applicable to any type of interaction between peers - elaboration and forwarding of queries during a lookup process, providing resources, etc.):

G – Good,

L – Low Grade,

N – No Response,

B – Byzantine Behaviour;

N_i is the number of services (interactions) provided with quality *i*; h is a map function from *Q* to a score for one interaction between nodes, i.e. it shows how many reputation points a node has gained or lost at the end of one interaction:

$$h(Q) = \begin{cases} S_1, Q = G, S_1 > 0 \\ S_2, Q = L, S_2 < 0, and\ |S_2| > S_1 \\ S_3, Q = N, S_3 < S_2 \\ S_4, Q = B, S_4 < S_3 \end{cases} \qquad (4.3)$$

So, we can see that misbehaving reduces a reputation value faster than a good behaviour increases it. *No Response* is considered as a bad behaviour here. It helps to avoid too frequent and long leaving in P2P networks: it is not convenient for a peer to be off-line for a long period, because all requests sent to the node during its absence will rest without response, significantly reducing its reputation. It is explained by a high importance of the cooperation component for collaborative environments: a node that doesn't participate in routing and lookup mechanisms by processing and forwarding requests from other nodes cannot be positively evaluated by the community.

The risk value is normalized to the worst case, that represents a situation when all services received by a peer in a certain time interval are Byzantine services. In our example we assign to S1, S2, S3 and S4 the values 1, -2, -3 and -4 respectively.

In Kademlia, identifiers of nodes consist of 160 bit. Here we present a simplified model of Kademlia's binary tree with a little number of sub-trees and leaves.

As shown in figure 4.1, the node with prefix 0011 initiates the process of look-up for some resource, sending FIND_VALUE RPC to two nodes residing in two different sub-trees with prefixes 0111 and 1011.

The first node doesn't respond (the one-directional dashed arrow). The second node returns a triple <IP address, UDP port, Node ID> for the node 1101. Applying the look-up progress verification mechanism, the requestor concludes that the Node ID sent by the second node is really closer to the key.

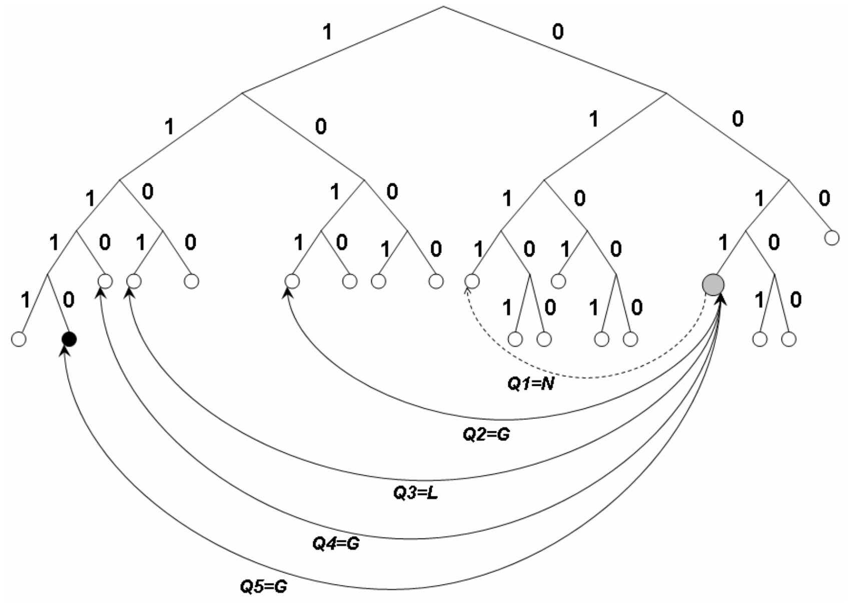


Fig.4.1. Look-up process based on Kademlia algorithm in terms of PET model

According to PET model our requestor assigns to this contact the quality value G, and to the first contact – the value N. Then, it sends FIND_VALUE RPC to node 1101, that returns a triple containing Node ID 1110, but with a little delay. In its turn, 1110 replies with Node ID 11110, that stores the desired data. 11110 returns the stored value. Having controlled at each step the look-up progress, the requestor assigns to nodes 1101, 1110 and 11110 the quality values L, G and G respectively.

The risk value in our case is:

$$R_i = \frac{N_L \times S_2 + N_N \times S_3}{S_4 \times (N_G + N_L + N_N)} = \frac{1\times(-2)+1\times(-3)}{-4\times(3+1+1)} = 0{,}25 \cdot$$

Information, regarding the quality of resources and the reliability of the peers obtained during this look-up process, should be stored in the resource and servent repositories [14].

The repositories represent two tables with the following data structures:

- resource_ id, value;
- servent_id, num_plus, num_minus.

XREP's authors don't precise a type of data representing quality values, and leave for us a liberty of interpretation: it may be a numeric value, or it may be simply defined as good or bad.

The data of a servent repository contain IDs of contacted peers and corresponding numbers of successful and unsuccessful transactions effectuated with these peers. As in our case we use DCRC model, it makes sense to substitute the num_plus and num_minus with data regarding the total number of uploaded and downloaded megabytes of content for a certain peer, e.g. mb_up and mb_down.

To calculate the QRC parameter, for each contacted node the total number of queries addressed to a node and a number of queries processed and forwarded by a node should be stored. Adapting the data structure of the servent repository to our case, we have:

<*servent_id, mb_up, mb_down, num_query, num_reply*>.

Since the information of the resource repository is not used in further calculations, in our case it is optional. Possibility of supporting both repositories depends on memory resources a peer has at disposal. Having downloaded the desired resource, the node 0011 stores the following data in its servent repository:

servent_id	*mb_up*	*mb_down*	*num_query*	*num_reply*
0111	0	0	1	0
1011	0	0	1	1
1101	0	0	1	1
1110	0	0	1	1
11110	20	0	1	1

In terms of technical realization this mechanism represents a quite simple solution: a simple counter seems to be sufficient.

According to DCRC model [19], a reputation value of a peer is defined by credits it gains or loses during a certain period interacting with other peers. The total number of reputation points is calculated by the following formula:

$$N = a \times QR + \sum_{l} b \times UC_l - \sum_{m} c \times DD_l + d \times SC \qquad (4.4)$$

where, QR is the number of points gained by a node for each processed query; a is the total number of queries processed by a peer; b is the number of uploads facilitated by a peer; c is the number of downloads performed by a peer; d is the predefined time factor (in hours) that determines a time interval during which the described interactions have been performed; UC_l and DD_m are the upload credit and download debit for files l and m respectively.

In its turn the *UC* is defined as follows:

$$UC = \frac{s}{f} \times \frac{bw}{b} \qquad (4.5)$$

where: *s* is the size of an uploaded file (in megabytes); *bw* is the bandwidth available (in megabytes); *f* is the file size factor that determines how many megabytes of data transfer increases the reputation score by a unit; *b* is the bandwidth factor that classifies peers on the base of bandwidths they have at disposal.

The DD for a download of a resource of size s is given by:

$$DD = \frac{s}{f} \times \frac{bw_i}{b} \tag{4.6}$$

where bw_i is the bandwidth of a peer *i* from which a download is performed.

The Sharing Credit (*SC*) for a peer that shares *n* files during some predefined time interval is given by:

$$SC = \sum_{j}^{n} \frac{s_j}{f} \tag{4.7}$$

where, s_j is the size of *jth* file.

Let's calculate the reputation value of the node 11110 using the data stored by 0011 at the repository after the considered look-up process. Let the size *s* of the file downloaded by the node 0011 from 11110 be 20 MB, the file size factor *f* =2MB, the available bandwidth *bw* = 6MB, and the bandwidth factor *b* = 2MB. Then, the *UC* of 11110 after this interaction is:

$$UC = \frac{20}{2} \times \frac{6}{2} = 30$$

For simplicity, let the number of points gained by a peer for each query processed QR=1. Let's suppose that the total size of the resources shared by 11110 node is 500 MB and the predefined time interval is 1 hour. Hence, *SC*=250.

Then, at *a=1* and *DD=0*, the total number of reputation points gained by the peer 11110 after the interaction in question is:

$$R = QRC_{tot} + UC_{tot} - DD_{tot} + SC_{tot} = 281.$$

4.4.3 Findings

We can conclude that, the risk calculation method used in PET model helps to define a level of vulnerability of an unfamiliar environment, while DCRC technique represents a an objective method of reputation evaluation based on points gained by a peer due to its collaboration with the community. In its turn, the repository mechanism is a simple and efficient solution for systematization of the data necessary for reputation points calculation.

All of these instruments can by easily adapted to DHT-based environment. This example demonstrates that the mechanisms used in our scenario successfully complement each other, even if they have been "extracted" from three different reputation evaluation models.

As is clear from the described scenario, the proposed solution represents an individual mechanism of reputation management for a single peer based on its own experience. However, as it has been discussed before, the reputation evaluation cannot be defined as complete and objective without taking in consideration opinions of other members of the community. For example, in the case of inconsistent behaviour of malevolent peers, it is very difficult to reveal the presence of malicious activity for some peers without opportune warning messages from "already attacked" peers.

So, it is necessary for peers to interact with each other and to share their personal opinions in order to define the trustworthiness level of a certain member of the community.

The communication between peers in DHT-based P2P networks should be based on some opportune interaction algorithm that :

- doesn't require introduction of centralization elements into the system;
- doesn't cause the exceeding increase of data traffic;

- uses as few computational resources as possible.

In this work we propose to integrate the classical algorithm for reaching Byzantine Agreement (BA) [18] and some reputation mechanisms designed for P2P networks based on DHTs in order to obtain a simpler and efficient reputation management algorithm for the completely distributed P2P environment that respects the above requirements. BA algorithms are currently used in distributed computer systems to cope with Byzantine Generals Problem (Interactive Consistency Problem) that provokes Byzantine failure.

Particularities of the proposed integration and the obtained algorithm are provided in the next section.

4.5 Byzantine Agreement for Reputation Management in DHT-based Peer-to-Peer Networks

4.5.1 Byzantine Generals Problem in distributed computer systems: history of the problem

Byzantine Generals Problem [18] (also known as Interactive Consistency Problem) takes place in distributed computer systems in the presence of malfunctioning components that give conflicting information to other parts of the system. It causes a Byzantine failure, that may consist in:

- failure to pass to the next step in the algorithm;
- system's inability to correctly implement the actual algorithm;
- arbitrary execution of a step different from one predicated by the

algorithm (incorrect hops).

Originally, BGP is a problem about an agreement between generals of the Byzantine army, that must take a common decision: to attack or not to attack an enemy army? The generals are geographically distant from one another and they have to communicate with each other only through messengers. This situation is complicated by the presence of traitors among the generals. The traitors try to confuse loyal generals, sending them a false information about decisions of other generals, and moreover, they may do it in an arbitrary manner, i.e. different loyal generals receive different false information.

This problem was formulated by L. Lamport, R. Shostak, and M. Pease in 1982, who were the first to apply the concept of BGP to distributed computer systems.

They presented several solutions and algorithms for reaching agreement between system units in the presence of failed components. It has been proved that, assuming the possibility of sending messages directly to each other by all generals, there must exist some round-by-round algorithm of information exchange, equal for all generals, so, that:

- all generals make the final decisions;
- all loyal generals decide upon the same plan of action;
- this final plan of loyal generals must coincide with final decision of one loyal general at least.

Oral message-based *(OM(m))* and signed message-based *(SM(m))* algorithms have been proposed. The former allows loyal generals (system components) to reach agreement in the presence of t traitors only if the total number of generals is $3t+1$ at least. The latter works for any number of generals and possible traitors.

The algorithms for BA presented by Lamport, Shostak and Pease are quite simple but expensive in both the amount of time and the number of messages sent by participants.

Later, a number of “faster” and “lighter” algorithms for reaching agreement have been proposed at different times by different researchers. Developers of these algorithms used diverse approaches including deterministic, randomized and quantum principles, applying them to synchronous and asynchronous computational models and different types of adversaries (fail-stop, Byzantine, etc.).

Typical quality measures for a BA protocol are the total number of processors, the number of communication rounds, and its communication complexity, alternatively given by the maximal message size, or the total number of transmitted bits. The known lower bounds are, respectively, $n > 3t$, $t+1$, 1, and $n\ (t+1)/4$ [19].

Algorithm presented by M.Rabin and M. Ben-Or in 1983 [20,21] use principles of randomness and probability, and significantly reduce the number of necessary rounds in respect of the solution by Lamport et al.

Later, Lewis and Saia presented a new scalable protocol for BA [22] which is very similar to Rabin's one, but they proposed to use random sampling: in each round each processor takes input from a small random sample of all the other processors, in distinction from Rabin's protocol, in which a processor takes input from all other processors in each round. This algorithm was created for those types of distributed computer systems where the direct communication network components with each other is not realizable because of their great quantity. This solution reduces the number of messages exchanged by nodes. The authors say that their protocol can tolerate any fraction of faulty processors which is strictly less than 1/6. The protocol is correct only with high probability and involves simultaneously all the components of the network.

In 2005 M. Ben-Or and A. Hassidim proposed a fast quantum Byzantine agreement protocol [23], that reaches agreement in *O(1)* expected communication rounds against a strong full information, dynamic adversary in the presence of up to $t < n/3$ faulty participants in synchronous setting, and up to $t < n/4$ faulty participants for asynchronous setting. This solution consists in postponing coin flips and using quantum superposition.

In 1998 J. Garay and Y. Moses presented a fully polynomial deterministic protocol for reaching Byzantine agreement in $t + 1$ rounds for $n > 3t$ processors (where *t* is an a priori upper bound on the number of faulty processors possible) [24]. For the moment it is the shortest deterministic algorithm executable with the minimal number of processors and in the minimal number of rounds.

Currently, Byzantine Agreement is a central problem in fault-tolerant distributed computing and secure multi-party computation, and it abstracts out a variety of situations where conflicting parties must coordinate their decisions and cooperate towards achieving a common goal.

4.5.2 Applicability of classical solutions for BGP to P2P environment

In DHT-based P2P networks malevolent peers exploit routing mechanisms to trouble reliable communication between nodes and to break down consistent operation of the network. So, the analogy between the behaviour of malicious nodes in P2P networks and actions of traitorous components in the case of Byzantine failure is obvious. Byzantine behaviour in P2P environment manifests in different types of attacks, such as incorrect lookup routing, incorrect routing updates, etc.

So, it is possible to apply some classical solutions developed for distributed computer systems to cope with some types of malicious activity in DHT-based P2P networks.

Let's consider an example of application of classical solutions for BGP to the case of incorrect routing updates. In this example we present a situation with a small number of participating nodes. A malicious node in the considered case demonstrates poor behaviour regarding all other nodes (e.g. doesn't behave in an inconsistent manner).

To update information about keys of resources and the identifiers of appropriate responsible nodes in routing tables, peers should exchange information they have with each other. So, each node sends to others a list of keys of resources it is responsible for. The loyal peers send the real information to other nodes, and the traitors may send different information to each node.

Let's say, the first node (peer) sends to others the list of keys in the form of vector N1 consisting of binary numbers of a fixed length that present resource identifiers. The second node sends vector N2; the third sends three different vectors X, Y and Z to node 1, node 2 and node 3 respectively, the fourth node sends vector N4 to all of them. Then, they exchange the received information with each other in accordance with oral message-based algorithm *OM(m)* for BA. The traitor sends arbitrary values to all other peers again.

Hence, after this data exchange, each peer can form its matrix on the base of the information received. The element n_{ij} of each matrix represents a vector that node *i* received from node *j*. So, in our case we have the following matrixes:

$$P1 = \begin{vmatrix} N1 & N2 & X & N4 \\ N1 & N2 & Y & N4 \\ A & B & C & D \\ N1 & N2 & Z & N4 \end{vmatrix} \quad P2 = \begin{vmatrix} N1 & N2 & X & N4 \\ N1 & N2 & Y & N4 \\ E & F & G & H \\ N1 & N2 & Z & N4 \end{vmatrix}$$

$$P3 = \begin{vmatrix} N1 & N2 & X & N4 \\ N1 & N2 & Y & N4 \\ N1 & N2 & N3 & N4 \\ N1 & N2 & Z & N4 \end{vmatrix} \quad P4 = \begin{vmatrix} N1 & N2 & X & N4 \\ N1 & N2 & Y & N4 \\ I & J & K & L \\ N1 & N2 & Z & N4 \end{vmatrix}$$

As we can see, in each matrix there are one row and one column containing values different from other cells, and these values are sent by the third peer. So, the peers update their routing tables with the values received from the majority of peers. In this case the application of the *OM(m)* algorithm is quite efficient.

However, in a real P2P network the application of the classical solutions for BGP in their pure form is complicated by some specific aspects regarding the nature of P2P. These systems represent an environment with a great number of participants, where nodes join and leave the network continuously. So, a P2P network cannot be considered as a static system where each component can contact directly all others at any moment. But in accordance with interactive consistency conditions [18], each participant of the algorithm should have the possibility to directly contact all others.

Moreover, as already mentioned above, the solutions presented by Lamport et al are expensive from the point of view of time spent and the number of messages sent by nodes. In fact, to reach an agreement the number of rounds executed must be linear in the total number of participating components. The algorithms (*OM(m)* and *SM(m)*) require message paths of length up to $t + 1$ (where t is a number of traitors). Thus, the total number of messages that nodes send to each other to reach an agreement is:

$$(n - 1)(n - 2) \ldots (n - t - 1) = (n - 1)! / (n - t - 2)! \qquad (4.8)$$

i.e., there is a factorial-like dependence between the number of messages and the number of nodes. In P2P environment it implies sending an exceedingly massive amount of messages and, as a result, the overload of the network.

So, to apply the solutions for Byzantine failure to P2P systems we should find a way to reduce the data traffic load on the network during execution of the BA algorithm.

Taking in consideration that the algorithms by Lamport et al are effective for small groups of participants, it makes sense to present a P2P network as a number of peer groups. In this case each node launches the algorithm only within a group it belongs to, and the total number of messages sent by network nodes is:

$$N_m = ((n_i - 1)(n_i - 2) \ldots (n_i - t_i - 1)) \times k \qquad (4.9)$$

where n_i - the average number of nodes in one group; t_i - the average number of traitors in one group; k - the number of groups in the network.

The diagrams (Fig.1 and Fig.2) show that the number N_m of messages decreases with increase of the number of groups k, at various network sizes (number of nodes in the network $N = 100; 1000; 10000; 100000; 1000000$), for the cases when the number of traitors on the network is $t = 1/5N$ and $t = 1/3N$ (the worst case).

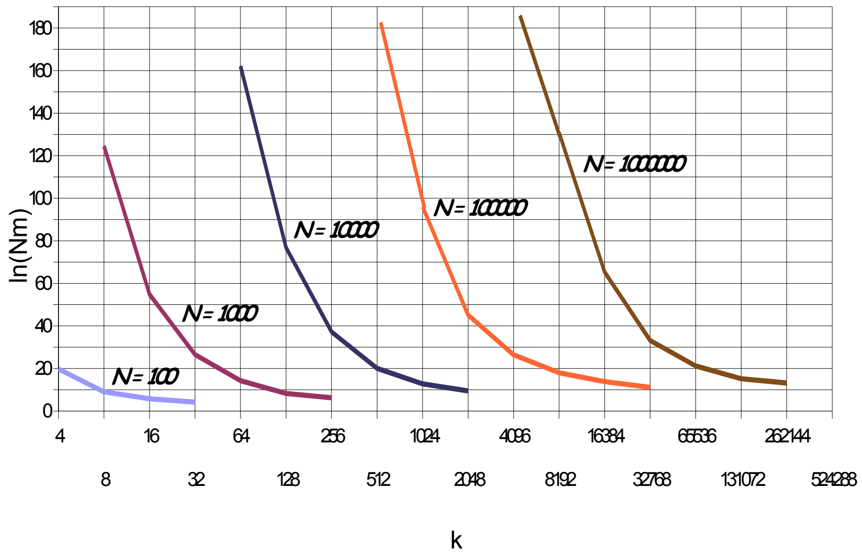


Fig.4.2 Diagrams of Nm change at k increased for t = 1/5N

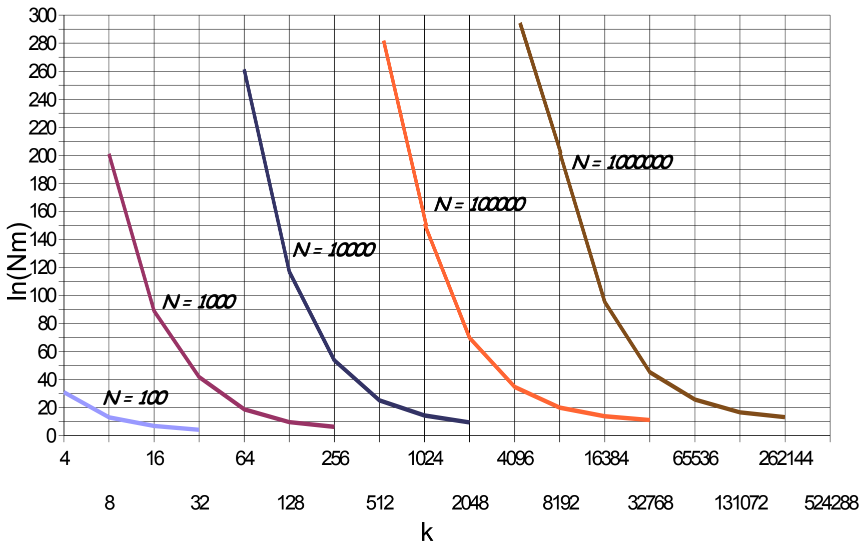


Fig. 4.3 Diagrams of Nm change at k increased for t = 1/3N

So, the problem of overload can be resolved applying the solutions for BGP to small groups of peers. Here emerges the following question: how can peer groups be created and individuated?

The network may be divided into subgroups in many senses, such as: a group of the most frequent contacts, a group of neighbour nodes with "close" identifiers according to the XOR-metric, groups of peers with common interests, etc. However, the listed principles of “peers clustering” are hardly realizable in P2P networks because of particularly dynamical nature of this environment explained by continuous joins and leaves. These principles suppose creating groups that are stable in terms of time and membership, but it is not possible in P2P. For instance, in DHT-based networks the identifier of a node that has just left the network can be assigned to a node that will join it in the next moment.

It is also important to note, that in our example we deal with the malicious peer who demonstrates poor behaviour toward all other nodes. In the case of inconsistent behaviour it is not so easy to detect the presence of malicious activity. When a malicious peer is more “cunning” and sends the real values to one part of the nodes, and to other part of them sends some arbitrary values, it becomes possible to consider one of the right values as false. Thus, in the case of inconsistent behaviour application of reputation evaluation mechanisms integrated with the above solutions makes sense.

Thus, the goal of our research is to find a solution that exploits the simplicity and the effectiveness of BA protocol and at the same time can be easily adapted to the dynamic and distributed nature of P2P, in order to cope with some effects of specific attacks of DHT-based environment.

In the next section we describe some mechanisms and techniques that we propose to integrate in order to achieve this goal.

4.5.3 Algorithms and techniques involved in the proposed solutions

As a starting point we take EigenTrust algorithm [25] and modify it replacing some mechanisms with the techniques listed below, but keeping some original features unchanged. In particular the algorithm we propose uses the following instruments:

- “score managers” from EigenTrust;
- concept of responsible nodes (primary tier) from OceanStore distributed data storage system [26];
- algorithm for reaching agreement based on oral messages *OM(m)* by Lamport et al [18].

Let’s consider each of these instruments in details.

We start from EigenTrust [25] algorithm for reputation management that is designed specifically for DHT-based P2P environment.

EigenTrust assigns each peer a unique global trust value based on a peer’s history of uploads. The global trust value assigned to some peer *i* reflects the experiences of all peers in the network regarding peer *i.* This algorithm is based on the following statements:

- self-policing of the system;
- anonymity of peers;
- no profit to newcomers;
- minimal overhead in terms of computation, infrastructure, storage and message complexity;
- resilience to alliances of malicious peers.

The trust value of a peer in this system is computed by a responsible group of M peers called score managers. Score managers for a peer are located by applying a set of one-way secure hash functions to a unique ID of the peer (e.g. IP address) and searching for nodes that can be responsible for theobtained key (those with the IDs closest to the key in the identifier space).

Since each node in the network acts as a score manager, it is assigned a set of daughters D_i represented by indexes of peers whose trust values are computed by this peer.

When a score manager leaves the network it passes all stored trust values to its neighbour peer in accordance with DHT data distribution principles.

Another mechanism we use for our solution is the concept of the "primary tier" nodes proposed by authors of OceanStore distributed data storage system [26]. In OceanStore privileged nodes of the "primary tier" cooperate with one another through a BA protocol to take a decision about a final commit order for an update generated by some node of the "secondary tier". The BA algorithm is launched on the base of an update request received from the initiator of the update. Once the agreement is reached primary nodes send the result to all interested secondary nodes.

In this case the classical solutions for BGP are efficient due to the small number of the algorithm's participants. However, here we deal with a hierarchical model where nodes of the primary tier represent a sort of central authority. The static set of privileged nodes contradicts the completely distributed nature of DHT-based P2P environment.

So, we propose to exploit the mechanism of score managers used by EigenTrust algorithm to define a group of responsible nodes. In this case, responsible nodes are assigned randomly (by a set of M hash-functions) and they are changed in course of time.

At the same time, we modify secure EigenTrust algorithm introducing the use of a BA protocol by score managers. Executing a BA algorithm score managers take a common decision about what trust value is to be reported when some peer requests information about a trustworthiness of another peer they are responsible for. As a BA algorithm in our solution we apply the classical oral message-based *OM(m)* algorithm by Lamport et al. This choice is made to avoid complications inherent to signed message-based techniques.

4.5.4 Reputation evaluation algorithm with use of Byzantine Agreement protocol

Preliminary remarks

Before describing the modified algorithm, it is necessary to set the following assumptions.

- Let $k \in N$ be a fixed value that defines the maximum number of trust values that a peer can report to its score managers. So, a local trust vector sent by peer *d* to its score manager *i* $c_d^i \in [0;1]^k$.
- The oral message-based algorithm *OM(m)* for reaching agreement is applied instead of the iterative calculation of trust values used in Eigen Trust.
- Normalized trust values will be used as in EigenTrust.
- Each peer has *M* score managers, whose DHT coordinates are defined by applying a set of hash functions $h_0, h_1, ..., h_{|M|-1}$, to a peer's unique identifier; pos_i represents coordinates of peer *i* in the hash space.

- Since each peer also acts as a score manager it is associated with a set of daughter peers D_i, comprising IDs of peers it is responsible for. Moreover, a peer stores each daughter peer's opinion vector.
- Furthermore, for each daughter peer $d \in D_i$, i learns A_d^i representing a set of peers which have downloaded files from its daughter peer.
- Finally, for each $d \in D_i$, a peer learns B_d^i representing a set of peers from which its daughter peer d has downloaded files.

Description of the proposed algorithm

As in EigenTrust, in the proposed solution each peer acts simultaneously as an ordinary user and as a score manager for some other peers in the network.

Each peer i (as a user) provides score managers with its local trust vector $\vec{c_i}$ that contains all local trust values c_{ij} computed by peer i regarding other peers j. Acting as score managers, peers process M vectors $\vec{c_i}$ received before through BA algorithm. After the algorithm has been performed, each score manager stores the resulting trust value. A score manager execute this procedure for each peer it is responsible for.

A node that needs information about the trustworthiness of some peer i asks corresponding score managers at positions $h_0(pos_i),...,h_{|M|-1}(pos_i)$ for trust values they have obtained through the described algorithm. Then, it receives M trust values and computes the median value of the vector containing these values in ascending order.

Below we provide the original secure EigenTrust Algorithm (Algorithm 1) and the modified reputation evaluation algorithm (Algorithm 2).

```
foreach peer $i$ do
    submit local trust values $\overrightarrow{c_i}$ to all score managers at positions
    $h_m(pos_i)$, $m = 1...M-1$;
    collect local trust values $\overrightarrow{c_d}$ and sets of acquaintances $B_d^i$
    of daughter peers $d \in D_i$;
    submit daughter $d$'s local trust values $c_{dj}$ to
    score managers $h_m(pos_d)$, $m = 1...M-1, \forall j \in B_d^i$
    collect acquaintances $A_d^i$ of daughter peers;
    foreach daughter peer $d \in D_i$ do
        query all peers $j \in A_d^i$ for $c_{jd} p_j$;
        repeat
            compute $t_d^{(k+1)} = (1-a)(c_{1d}t_1^{(k)} + c_{2d}t_2^{(k)} + ... + c_{nd}t_n^{(k)}) + ap_d$
            send $c_{dj}t_d^{(k+1)}$ to each score manager of each peer $j \in B_d^i$;
            wait for each score manager of each peer $j \in A_d^i$ to return $c_{jd}t_j^{(k+1)}$;
        until $|t_d^{(k+1)} - t_d^{(k)}| < \varepsilon$;
    end
End
```

Algorithm 1. Secure EigenTrust Algorithm

```
foreach peer $i$ do:
    send $\vec{c_i}$ to $M$ score managers at positions $h_0(pos_i),...,h_{|M|-1}(pos_i)$;
    for $\forall\ d \in D_i$ perform a BA protocol with other score managers
    in order to form a resulting vector $\vec{c_d}^{BA}$ form $M$ values of $\vec{c_i}$
    foreach $d \in D_i$ do:
        $\forall l \in B_d^i$ ask $d$ for a local trust value of peer $l$ $c_{dl}$
        send value $c_{dl}$ to each score manager of peer $l$
        $\forall j \in A_d^i$ wait for each score manager of peer $j$ to return $c_{jd}$
        choose by a majority vote $k$ values of $\overline{c_{jd}}$
        compute the mean value and get $d$'s trust value $t_d$
    end
end
```

Algorithm 2. Reputation Evaluation using Byzantine Agreement protocol

The modified algorithm appears computationally simpler than EigenTrust, since it does not involve iterative calculation of trust values $t_d^{(k)}$ by score managers. However, it is necessary to estimate complexity of both EigenTrust and the modified algorithms to confirm our expectations. In the next section we provide the complexity evaluation of both algorithms in terms of average number of sent messages per node.

4.5.5 Complexity evaluations of the modified and the previous algorithms

At first, the following parameters should be introduced:

- n is the total number of nodes in the network;
- k is the maximum number of evaluations a node can report to its score managers;
- M is the number of score managers per node;
- c is the number of necessary iterations needed by EigenTrust to converge (typically 10).

According to the previous definition of k and M, it can be shown that, upon the average, for each peer i and for each daughter peer d, $|D_i|=M$ (if robust hash functions are used) and $|A_d^i|=|B_d^i|=k$. Moreover, every message is assumed to have computational cost 1, regardless of differences between scalars and vectors.

It is important to note that in the presented solutions the algorithm complexity does not depend on the total number n of nodes in the network. As a consequence, the number of sent messages per node is also independent from the number of connected users.

Both the algorithms described above for a single node i should be performed by each node in the network. So, each step of the algorithms is repeated n times at every round. However, while computing the average number of sent messages per node, the total number is divided by n, i.e. n is cancelled out of the calculation.

Complexity evaluation of EigenTrust

Let's consider how many messages are to be sent at each step of EigenTrust.

At the first step, sending a vector $\overrightarrow{c_i}$ by peer i to its score managers requires M messages to be generated.

Then, each score manager of peer i sends $M - 1$ messages to other score managers. It means that $(M - 1)M$ messages are sent in total to execute this step of the algorithm.

At the third step, for each $d \in D_i$ peer i:

- asks each peer $j \in A_d^i$ for a trust assessment of its daughter peer;
- performs c iterations consisting of computing the trust value $t_d^{(k)}$ and sending $c_{dj} t_d^{(k)}$ to each score manager of all $j \in B_d^i$. Hence, kM messages are sent.

Thus, the total number of messages generated at this step is

$$M(k + Mkc) = M^2(kc) + kM.$$

So, in total, the "cost" of EigenTrust in terms of sent messages per node is:

$$M + ((M - 1)M) + (M^2(kc) + kM) =$$

$$= M^2 + M^2(kc) + kM = M^2(kc + 1) + kM. \qquad (4.10)$$

Complexity evaluation of the modified algorithm

Firstly, the number of messages exchanged by nodes during the execution of the BA algorithm should be calculated. It is performed at the beginning of the algorithm, after peer *i* has reported vector $\overrightarrow{c_i}$ to its score managers.

As we know, the number of messages exchanged in *OM(t)* algorithm with *t* traitors and *3t + 1*generals involved is

$$(n-1)(n-2)\ \ldots(n-t-1).$$

This step of the algorithm represents a situation comparable to the case with a commander (peer *i*) sending an order (vector $\overrightarrow{c_i}$) to his *M* lieutenants (score managers). In this case the total number of participants of the BA algorithm is *M + 1*. So, the number of sent messages is defined as:

$$M(M-1)\ldots(M-t).$$

To cope with *t* traitors, the minimum value of *M* is defined so, that:

$$M+1 \geq 3t+1.$$

Thus, $M = 3t$ and we have:

$$(M-t-1) = \left(M - \frac{M}{3} - 1\right) = \left(\frac{2M-3}{3}\right).$$

Then, it follows that:

$$M(M-1)\ldots(M-t) = \ldots = \left(\frac{M!}{\left(\frac{2M-3}{3}\right)!}\right)$$

Let’s set *b* as a number of messages sent by score managers to each other at this step of the algorithm in order to achieve an agreement regarding local trust values $\vec{c_i}$ provided by each peer for its responsible nodes:

$$b = \left(\frac{M!}{\left(\frac{2M-3}{3}\right)!} \right).$$

Then, each *i*’s score manager asks *M* score managers of each peer $l \in A_d^i$ for a value c_{li}. Since $| A_d^i | \approx k$, the average number of sent messages at this step is *kM.*

Hence, the complexity of the proposed algorithm in terms of messages sent is

$$b + kM^2 = \left(\frac{M!}{\left(\frac{2M-3}{3}\right)!} \right) + kM^2 \tag{4.11}$$

Figures 4.4 and 4.5 show how the number of sent messages per node changes with increase of the number of score managers at *k = 100* and *k =10000* respectively. As is clear from the graphs and the equations we have provided before, EigenTrust has a quadratic dependence between the number of messages sent and the number of score managers involved, while the modified algorithm provides the factorial-like dependence. The latter will be more efficient when *M* is quite small or *k* is quite large. We can see that when the maximum number of values reported by a peer to its score managers is k∈ [100, 10000] and $M \leq 12$ (in the first case) or $M \leq 17$ (in the second case) the modified algorithm is always “cheaper” than EigenTrust in terms of messages required.

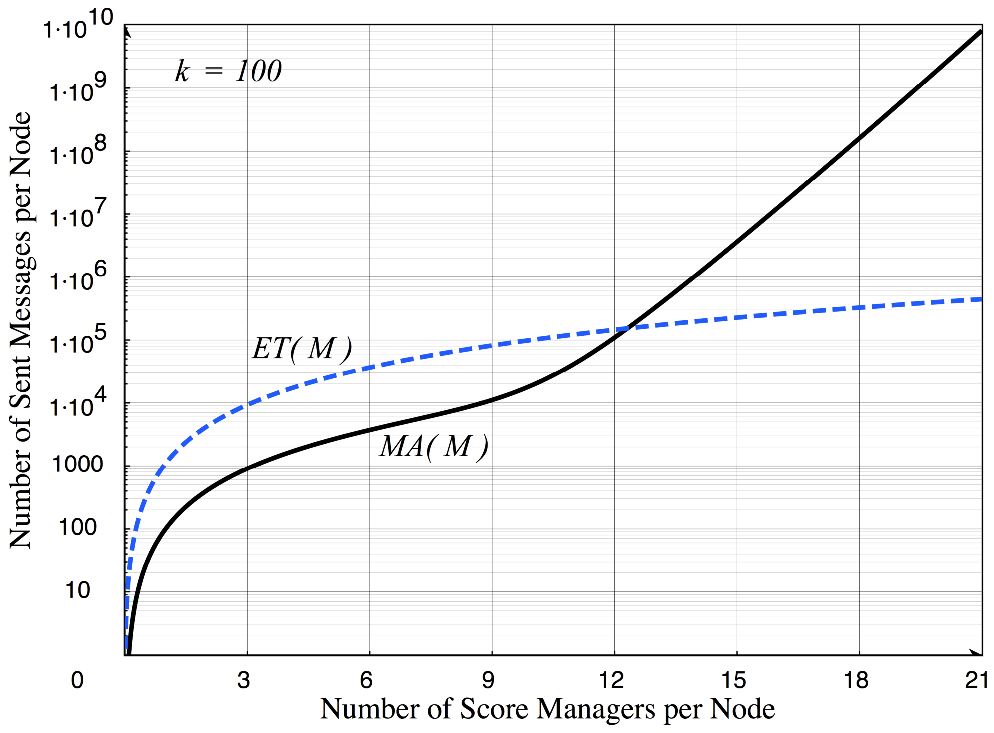


Fig.4.4 Complexity comparison between EigenTrust (ET) and the modified algorithm (MA) in terms of number of sent messages per node at k=100

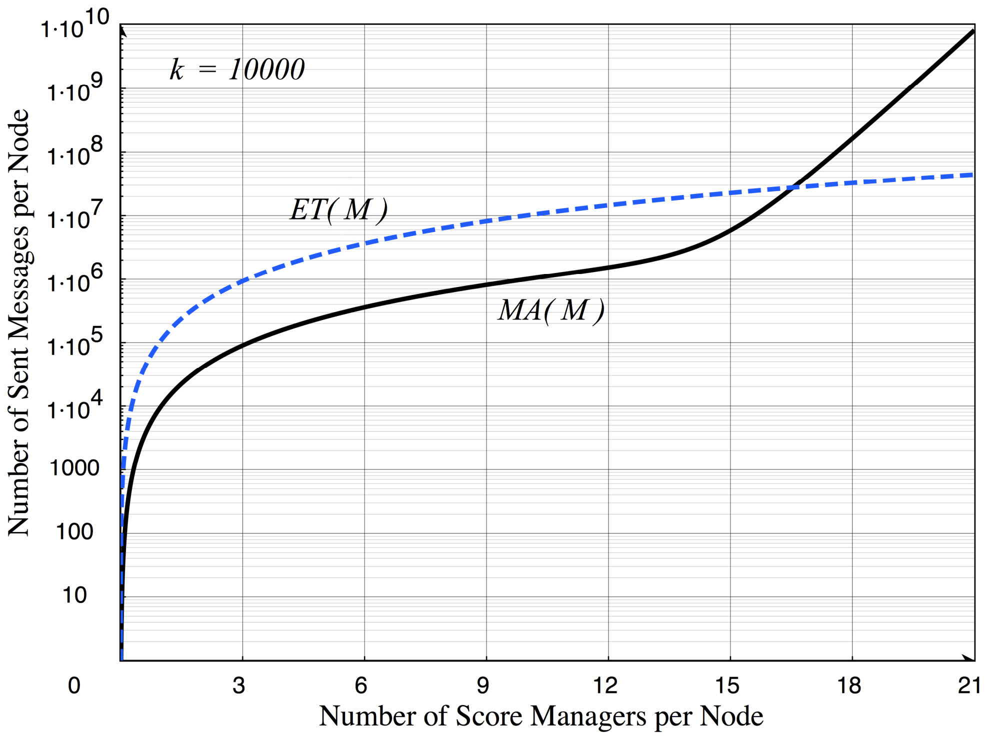


Fig.4.5 Complexity comparison between EigenTrust (ET) and the modified algorithm (MA) in terms of number of sent messages per node at k=10000

4.5.6 Summary

The proposed solution is less expensive in the number of messages sent per node in respect of the previously proposed EigenTrust algorithm. At the same time the use of Byzantine Agreement protocol instead of the iterative calculation of trust values significantly simplifies the algorithm saving computational and memory resources of the system.

The mechanism of assignment of score mangers using hash functions makes it impossible for malicious peers to effectively cooperate and to create alliances in order to confuse loyal peers: malicious peers can neither know which nodes they are responsible for, nor choose appropriate coordinates to become a score manager of a certain peer. Moreover, when most of peers are malicious, misbehaving towards honest peers is to be considered as regular, because in according to the self-policing principle [6] of the system the shared ethics of the user population are defined and enforced by the peers themselves.

To render the algorithm less expensive in the amount of time (rounds) required for execution it is possible to apply faster protocols for BA, such as that of Garay and Moses [24], instead of classical algorithms by Lamport et al.

Assessing the efficiency of the modified algorithm at small values of M in order to evaluate the possibility of further minimization of the total number of messages sent will be the next step of the optimization of the proposed solution.

4.6 References

[1] A. Abdul-Rahman, S. Hailes. Supporting Trust in Virtual Communities. In Proceedings of the 33rd Hawaii International Conference on System Sciences, Maui, Hawaii, January 2000.

[2] T. Grandison, M. Sloman. A survey of trust in Internet applications. In IEEE Communication surveys 3(4).

[3] G. Suryanarayana, R.N. Taylor. TREF: A Threat-centric Comparison Framework for Decentralized Reputation Models. In ISR Technical Report UCI-ISR-06-2, January 2006

[4] P.Bonatti, C.Duma, D.Olmedilla, N.Shahmehri. An integration of reputation-based and policy-based trust management. In Proceedings of Semantic Web Policy Workshop in conjunction with 4th International Semantic Web Conference, Galway, Ireland, November 2005

[5] S. D. Kamvar, M.T. Schlosser, H. Garsia-Molina. The Eigen Trust Algorithm for Reputation Management in P2P Networks. In Proceedings of the 12th International World Wide Web Conference, May 2003

[6] S. Y. Lee, O-H. Kwon, J. Kim, S. J. Hong. A Reputation Management System in Structured Peer-to-Peer Networks. In Proceedings of 14th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprise (WETICE'05)

[7] V. Martins, E. Pacitti, P. Valduriez. Survey of data replication in P2P systems. Research report, INRIA, December 2006

[8] N. Fedotova, M. Bertucci, L. Veltri, "Reputation Management Techniques in DHT-based Peer-to-Peer Networks", in Proceedings of ICIW'07, Mauritius, May 2007

[9] V. Grishchenko. A fuzzy model for context-dependent reputation.

Trust, Security and Reputation Workshop at ISWC, Hiroshima, Japan, 2004

[10] L. Xiong, L. Liu. A Reputation-Based Trust Model for Peer-to-Peer Ecommerce Communities. In Proceedings of IEEE Conference on E-Commerce (CEC '03), June 2003

[11] Z. Liang, W. Shi. PET: A Personalized Trust Model with Reputation and Risk Evaluation for P2P Resource Sharing. In Proceedings of the 38th Hawaii International Conference on System Sciences, 2005.

[12] Rita Chen and William Yeager. Poblano: A Distributed Trust Model for Peer-to-Peer Networks. Technical report, Sun Microsystems. http://www.jxta.org/docs/trust.pdf

[13] Lee, S., Sherwood, R., et al. Cooperative peer groups in NICE. In Proceedings of IEEE Infocom, San Francisco, USA, 2003

[14] Damiani, E., di Vimercati, S., et al. A Reputation-Based Approach for Choosing Reliable Resources in Peer-to-Peer Networks. In Proceedings of the 9th ACM Conference on Computer and Communications Security, Washington DC, 2002

[15] Zacharia, G. and Maes, P. Collaborative Reputation Mechanisms in Electronic Marketplaces. In Proceedings of the 32nd Hawaii International Conference on System Sciences, Hawaii, 1999

[16] A. Josang, R. Ismail. The Beta Reputation System. In Proceedings of the 15th Bled Electronic Commerce Conference, Bled, Slovenia, 2002

[17] M. Gupta, P. Judge, M. Ammar. A Reputation System for Peer-to-Peer Networks. In Proceedings of the 13th ACM International Workshop on Network and Operating Systems Support for Digital audio and Video. Monterey, California, 2003

[18] L.Lamport, R.Shostak, M.Pease. 'The Byzantine Generals Problem.

ACM Transactions on Programming Languages and Systems, Vol.4, No.3, pp. 382-401, July 1982

[19] http://cm.bell-labs.com/who/garay/#ba-hl

[20] M.O.Rabin. Randomized Byzantine generals. In Proceedings of FOCS'83, Los Alamitos, CA, USA, Nov. 1982, IEEE Computer Society Press

[21] M.Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In Proceedings of the Second Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, pages 27-30, Montreal, Quebec, Canada, 17-19 August 1983

[22] C.S.Lewis, J.Saia. Scalable Byzantine Agreement. In Proceedings of NIPS'03, Whistler, Canada, Dec. 2003

[23] M. Ben-Or, A. Hassidim. Fast Quantum Byzantine Agreement. In Proceedings of STOC’05, Baltimore, Maryland, USA, May 2005

[24] J.Garay, Y.Moses. Fully Polynomial Byzantine Agreement for n > 3t Processors in t+1 Rounds. SIAM Journal of Computing, 27(1), 1998

[25] S.D. Kamvar, M.T. Schlosser, H. Garsia-Molina. The EigenTrust Algorithm for Reputation Management in P2P Networks. In Proceedings of WWW-2003, Budapest, Hungary, May 2003

[26] J. Kubiatowicz et al. OceanStore: An Architecture for Global-Scale Persistent Storage”. In Proceedings of ASPLOS’2000, Cambridge, Massachusetts, USA, November, 2000

[27] N. Fedotova, L. Veltri. Reputation Management for DHT-based Collaborative Environments. Submitted to the 4th European Conference on Next Generation Internet Networks (NGI-2008), April 28-30, 2008, Krakow,

Poland.

[28] N. Fedotova, L. Veltri. Byzantine Generals Problem in the Light of P2P Computing. In Proceedings of the Third International Workshop on Ubiquitous Access Control, July 17, 2006, San Jose, California, USA

[29] N. Fedotova, G. Orzetti, L. Veltri, A. Zaccagnini. Byzantine Agreement for Reputation Management in DHT-based Peer-to-Peer Networks. Submitted to International Conference on Telecommunications (ICT-2008), June 16-19, 2008, St. Petersburg, Russia.

5 Distributed Hash Tables in collaborative environments

"The two offices of memory are collection and distribution"
Samuel Johnson (1709-1784)

5.1 Enterprise networks

Computer networks has become an integral part of the technical infrastructure of today enterprises and play an important role in management of their business activity and successful realization of any type of projects.

An enterprise network is defined as a geographically dispersed network for a large business enterprise, that typically comprises a number of local area networks (LANs), which have to interface with each other, as well as with a central database management system and many client workstations. Usually, such network has a hierarchical structure, and it is based on a client-server model and centralized architecture.

An enterprise network should provide effective mechanisms for:

- secure communication between network terminals;
- rapid and secure data exchange between different system units;
- reliable data storage system;
- secure network access;
- simple network administration.

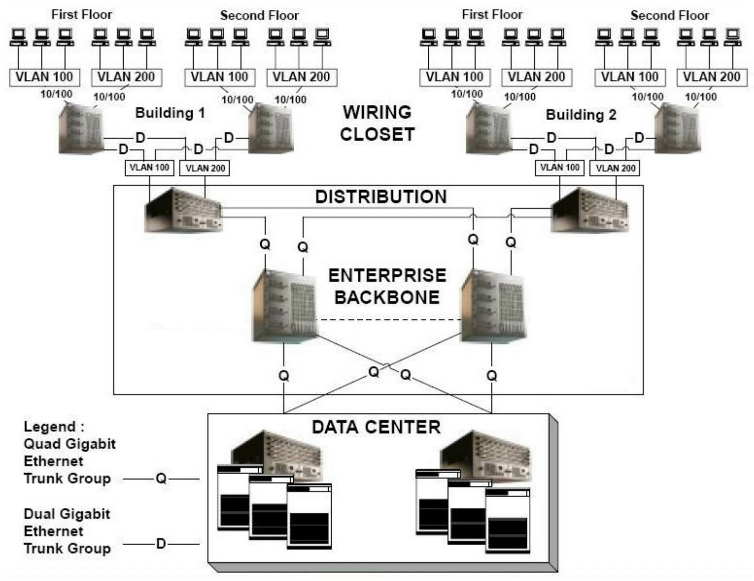


Fig. 5.1Enterprise network

It is also very important to make data, produced by some department maximally available for other interested entities. It should be realized in accordance with a predefined policy of attribution of different privileges, regarding data access, to users from various departments of the same organization.

Recently, the possibility of collaboration between different geographically distant organizational units in real-time and transparent mode is indispensable for many enterprises. Today there are numerous groupware applications, that provide the opportunity of synchronous collaboration among distant work groups engaged in a common task via computer networks (for example, Groove Virtual Office).

Currently, almost all enterprise networks as well as the groupware they utilize, have either a centralized architecture or a hybrid architecture based on the use of central servers that require high administration and maintenance costs. Moreover, centralized systems are subject to some problems regarding information security, data retrieval efficiency and reliability, such as:

- predisposition to “denial of service” and “packet filtering” attacks;
- single point of failure, represented by one (or more) server, that is a unique network component responsible for the storage of all data and resources;
- possible reduction of performance characteristics of network terminals in terms of bandwidth and time of access to resources, that takes place when a great number of clients simultaneously request for the access to different resources;
- low network scalability caused by growing number of terminals and, as supervention, necessary augmentation of storage capacity of servers, that, in its turn, causes additional costs;
- complexity of database organization;
- high administration and maintenance costs.

To cope with the above problems and to satisfy the requirements regarding effective data management and functionality of enterprise networks provided before it is advisable to pass from the centralized organization to a decentralized and distributed system. In particular, we propose to introduce a distributed peer-to-peer data organization system based on DHT to the enterprise environment.

5.2 General description of the proposed approach

The proposed approach consists in introducing a distributed peer-to-peer data organization system to the enterprise environment. This system exploits hardware and memory resources of all terminals of the network to provide a reliable data storage system and the possibility of effective collaboration between geographically distant users. In this case all the network terminals together form a huge distributed disk space. Moreover, the distributed structure gives the possibility of incremental growth of the network, delivering complementary capacity when and where needed. This feature is very appreciable in potentially extensible environment of enterprise networks.

The solution presented here is based on resource sharing and data storage principles inherent to peer-to-peer networks based on Distributed Hash Tables.

As already mentioned above, in a DHT-based P2P system a group of distributed hosts collectively manage a mapping from keys to data values according to some predefined algorithm (CAN, Chord, Pastry, Tapestry, Kademlia), without any fixed hierarchy and with a very little human assistance. Today, a great number of P2P platforms use DHT-based overlay networks, that create a structured virtual topology above the basic transport protocol level implementing effective self-organizing data storage and lookup mechanisms.

The concept of Virtual Enterprise Networks involves use of overlay mechanisms as well: Supernet layer with the appropriate address system is employed to protect data transmitted by the network layer [1]. The Supernet is based on communications tunnelling technique (mainly IP over IP), that is widely used to implement such services as multicasting, virtual private networks (VPN) and mobility support. Due to the Supernetworking users of a “virtual” enterprise network can securely communicate with each other and access to the database of the enterprise to get information they need from any Internet access point. However, the problem of a single point of failure and other problems concerning the centralized network architecture are still present here.

We propose application of Kademlia DHTs to organize data storage and retrieval in enterprise networks. Due to the particular nature of the enterprise environment, it is necessary to make some modifications in Kademlia protocol in order to better suit several specific security requirements.

Since most of enterprise networks exploit trustless public network infrastructures, it is important to provide users with appropriate data authenticity and access control instruments, that can guarantee secure communication and data exchange.

We introduce a system of different levels of privileges based on use of prefix identifiers (*prefix_IDs*) for both nodes and resources to handle read/write permissions in accordance with a certain enterprise hierarchical model. This is realizable due to Kademlia's tree-like structure of the identifier space.

5.3 Kademlia for data storage and retrieval in enterprise networks

5.3.1 Why Kademlia?

Kademlia DHT has been already described in Chapter 2. Here we provide only a brief summary on Kademlia system.

Kademlia is a peer-to-peer DHT based on the XOR metric. The distance between two identifiers is defined as: $d\ (x,y) = x\ XOR\ y$. All nodes and resources in this system have 160-bit identifiers (keys). The data are replicated by finding k (the recommended value for k is 20) nodes closest to a key and storing the key/value pair on them.

Kademlia has a tree-like data structure and the routing process is implemented in prefix-matching mode. The routing table size is $\log_2 N$.

Comparing the main characteristics of different DHT-based algorithms, we can say that Kademlia is the most appropriate system to be applied to enterprise networks due to the following advantages it offers:

- the binary tree-based structure of the identifier space and routing by prefix matching permit to manage the assignment of IDs to enterprise terminals and diverse privileges to single departments in simple and intuitive mode in accordance with a predefined hierarchy;
- easy implementation of eventual algorithm modifications in the case of enhancement of a network's dimensions;
- the symmetry of XOR-metric provides peers with a possibility to learn and update routing information from queries they receive during a lookup process;
- a Kademlia network can be presented as a bucket table. So, the lookup speed can be increased by considering *b* bits (instead of one bit) at each step, reaching a desired resource in less time.

So, Kademlia offers all necessary mechanisms to create a flexible and effective overlay "infrastructure" for enterprise networks.

5.3.2 Proposed scenario of the network organization

Before describing the processes of publishing and modifying data stored by network nodes, it is necessary to explain how the new identification system is organized.

Let's consider an enterprise network of some hypothetical company. We suppose that our company consists of many departments of different levels (A, B, C), which have different privileges regarding the possibility to access and modify data produced by the same department or by another one. According to this system of privileges, any department of level A can access and modify data produced only by an office of the same level. Any B department has more privileges than A departments: it can get and modify data produced by any B office and also by any office of level A. Accordingly, C departments are enabled to access and modify files created by departments of lower levels A and B, and so on. Finally there are nodes with a manager status (M), that can get, change, store and cancel files produced by nodes of any department.

Some types of data should be accessible and shared by all the departments, for example administrative circulars, recommendations, instructions, etc. So, this information should be stored at A nodes, that belong to the lowest level of the described system, to provide free access to these resources for all nodes of the network.

Since an enterprise network is a quite particular system with specific requirements regarding information security and access control mechanisms, it is necessary to make some appropriate modifications in Kademlia protocol to adapt this algorithm to such environment.

Let's begin from keys and node IDs assignment.

5.3.3 Assignment of node identifiers

Although each workstation of an enterprise network may have a static IP address, it is not convenient to assign to a node an identifier obtained as a hash function of its static IP address, because the node would get the same node ID every time it joins the network, and it would potentially be more vulnerable to masquerade attacks (a type of attack in which one system entity illegitimately poses as another entity to gain access to confidential data).

The solution we propose is a random attribution of node IDs by some trusted bootstrap terminal that should be contacted by nodes to join the network. In this case we avoid a situation when a malicious node can get and use a specific ID in order to possess certain keys related to confidential data.

To organize all work processes (storage, retrieval and exchange of data) and interactions between network terminals in accordance with hierarchical principles described above, we propose a solution explained below.

Each Kademlia node has a 160-bit node ID, that we divide in two strings of bits: the former is called *prefix_ID*, the latter is called *node_ID*. The prefix consists of β bits, and the remaining $160 - \beta$ bits represent the *node_ID*.

The length of the prefix and of the node ID depends on the network dimensions and on the corresponding structure, i.e. on the number of nodes in the network and the number of different departments that an enterprise consists of. The prefix defines a level that a node belongs to.

When some node contacts a bootstrap terminal, this one recognizes the level of privileges the node can enjoy examining its certificate, and assigns to the node the corresponding *prefix_ID* predefined for the departments of this level. The *node_ID* should be randomly generated by the bootstrap terminal, that also verifies that the matched pair <*prefix_ID; node_ID*> doesn't coincide with some node that is already online. The assigned ID expires immediately when a node leaves the network.

Since Kademlia system has a tree-like data structure it is quite simple to realize this ID assignment technique (Fig. 5.2).

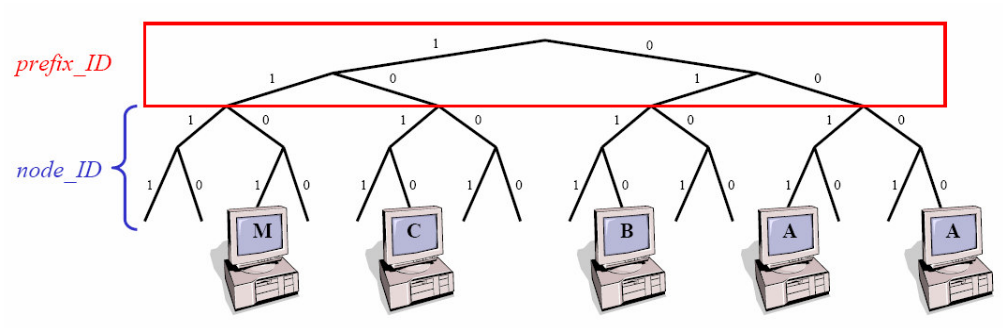


Fig. 5.2. Assignment of IDs to the network terminals

5.3.4 Key assignment and data storage procedures

Regarding the key assignment to files published on the network, the 160-bit key ID cannot be simply calculated as hash function of the file, as it happens in Kademlia. In fact, in this case, the prefix-based mechanism is also needed. Without involving the prefixes we can have the following situation: when a node A is going to publish some file, it first calculates the key applying a hash function to the file's content, then looks up for the nodes closest to the key. Since these ones may belong to departments of level B or C, the file could be stored at nodes of the higher level that is inaccessible for nodes of level A for the future modifications and use. So, in terms of files retrieval efficiency and according to the "competence principle" it's more useful to store data using the already introduced prefix-based approach.

In order to implement this principle, we use the same technique as for node IDs, dividing a key identifier in two strings of bits that represent the *prefix_ID* (the first β bits) and the *key_ID* (the remaining $160-\beta$ bits).

A *key_ID* can be obtained applying a hash function to the file content, as it is implemented in Kademlia. A *prefix_ID* of a resource usually coincides with a *prefix_ID* of the node that produced it. So, the prefix indicates a level of "confidentiality" of a file's content, i.e. if it is for common use or only for limited use of certain departments. However, if a node B, for example, intends to make a file it has created available for all departments, it should store the file on some node of level A. To do it the "publisher" should assign to the file a *prefix_ID* predefined by the bootstrap for nodes of level A.

Hence, according to Kademlia principles the file will be stored at some node of level A with the *node_ID* "closest" to the file's key_ID. Obviously, a node is not permitted to store data at nodes of the level that is higher than its own one.

Thus, for the efficient data retrieval, files should be stored in such mode that nodes, using those files frequently during a work process, could easily get them. This is possible only if all necessary files are hosted by nodes of the same or the lower level in respect of the level of a certain node.

To illustrate the mechanisms described above we provide a simple example. In our example (Fig.5.3) instead of 160-bit key-space we consider 4-bit space, where the number of bits assigned to the *prefix_ID* is *β = 2* and the other two bits represent *node_ID* or *key_ID*. All terminals represented on the figure are online.

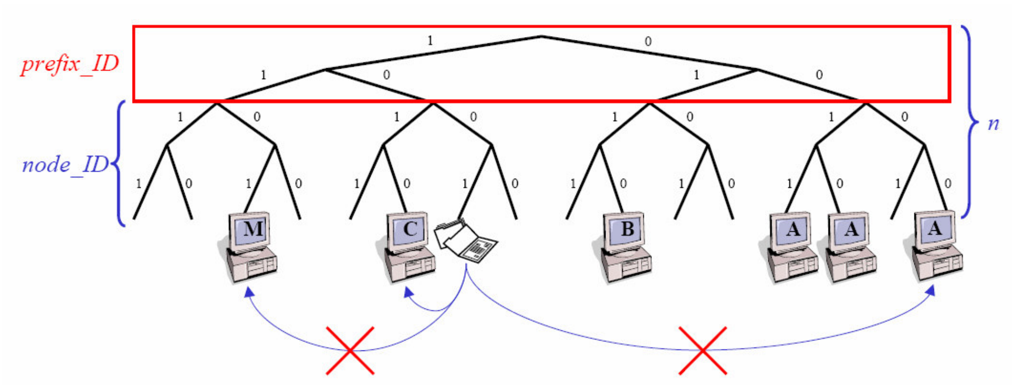


Fig 5.3. Example of data storage procedure

Let’s suppose that a node C needs to publish a file for internal use of department C with *key_ID* 01. In this case the node has to store the data at some node of level C. So the file’s ID (key) will be 1001. In this example there is only one node C online apart from the publisher.

Calculating the distances between the key of our resource and IDs of the active nodes according to XOR-metric, we obtain the following results:

dist(1001;1010)=0011=3

dist(1001;1101)=0100=4

dist(1001;0000)=1001=9

dist(1001;0011)=1010=10

dist(1001;0010)=1011=11

dist(1001;0110)=1111=15.

So, the closest nodes are: the node C with ID=1010, the Manager node 1101, and the node 0000 of level A.

To avoid the problem of data storage at inappropriate nodes, the publisher should verify the node IDs returned by the lookup procedure and choose for the storage the nodes with opportune prefix_IDs. In our case the opportune node is C with ID=1010 of the original sub-tree. Thus, the STORE RPC should be sent only to nodes with IDs that satisfy the condition:

$$dist(ID, key) < 2^{n-\beta},$$

where n is the total number of bits in the node ID, β is the number of bits in the prefix identifier.

This inequality imposes an upper bound to the distance between a key and a target node. If IDs of two nodes satisfy this condition, they reside at the same sub-tree of the identifier space and belong to the same department (Fig. 5.4).

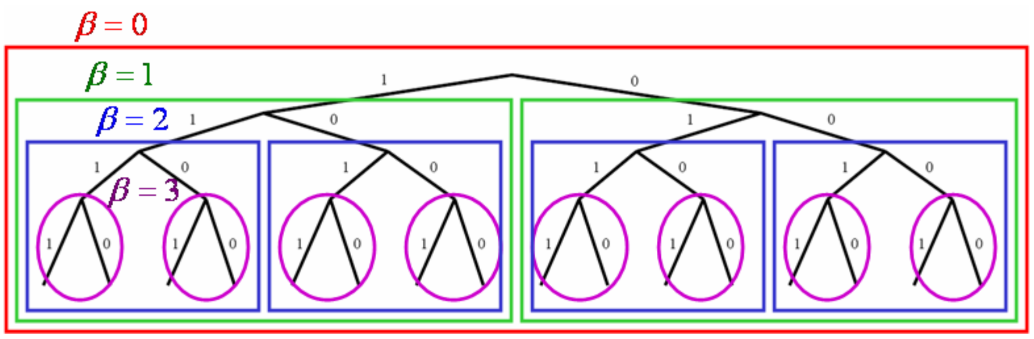


Fig. 5.4 Differentiation between peer groups in accordance with the proposed prefix-based identification technique at various number of bits in the prefix identifier β

5.3.5 Data publication process

Publication of data in Kademlia is implemented by storing of <*key, value*> pairs corresponding to a certain file at nodes with the IDs closest to the key.

The flexibility of DHT algorithms provides us with two possibilities: the "value" can represent information about an "address" (ID) of a node where a file can be found and the resource's description (metadata), as well as the file itself.

Since most of the files produced and exchanged by nodes in enterprise environment represent different types of documentation (text, diagrams, tables) that usually don't occupy a lot of disk space, the second way may be preferred. In this way we can use mechanisms of "integrated backup" to avoid situations when some node leaves the network and resources it possesses become unreachable for other terminals. It is realized in the following mode.

A publisher defines the *k* closest nodes for a resource to be stored according to Kademlia principles. In order to rationally distribute network memory resources and to avoid excessive traffic increase, the "value" representing a replica of the resource is stored at γ nodes from these k nodes ($\gamma < k$). The rest $k - \gamma$ nodes receive the STORE RPC regarding the same <key, value> pair, but with the "value" in the form of the file's metadata. So, on the network a certain number of the replicas will be presented, and leaving of some of the file's holders will not create any problem.

Besides the limit on the number of replicas γ, it also makes sense to introduce size limits for files to be replicated. The files that exceed this limit can be stored only at the nodes of origin, and the corresponding metadata should be hosted by the nodes defined by XOR metric.

To simplify the task of a publisher, we propose to apply a "tree-like data memorization model". Using this model a publisher doesn't need to send a <key, value> pair to all γ nodes. Instead of this, it sends the pair concerning a file to be stored, to two nodes that it considers the closest. In this case the <key, value> pair is complemented by a Time To Live (TTL) parameter such that:

$$\sum_{i=1}^{TTL} 2^{i} = \gamma$$

When the two closest nodes receive the STORE RPC, each of them store the received data and send to the node that has initialized the process a confirmation of the executed data storage. Then they verify that the TTL value is not null yet, as at each step of the process it is decremented by one. Hence, each of these nodes sends the new TTL value and the <key, value> pair to other two nodes it regards the closest to the key, and so on until TTL value has reached zero.

Figure 5.5 illustrates a tree-like data memorization process for $\gamma=6$.

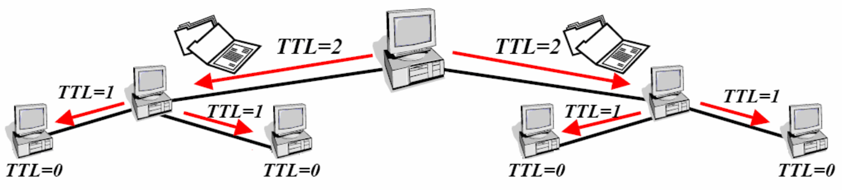


Fig.5.5 Tree-like data memorization model

This technique of data publication represents a quite fast and reliable mechanism. Such model provides a uniform distribution of a resource's replicas within the network. It guarantees that each node potentially interested in a certain file, keeps in its k-bucket at least one contact which possesses a replica of this file or an ID of its possessor.

5.3.6 Modification and update of stored data

Now, let's consider how a node can modify some file and then publish this modification, making it available for remote users working on the same file (editing of the same document, performance of some calculations based on results of the previous step). Realization of this mechanism provides a possibility of real-time collaboration between interested users.

A node that has modified some file should publish the updated version at a node from which the previous version of the file was downloaded. If the latter is not active at this moment, the file should be stored at some of the *k* nodes closest to the key that store the metadata or a copy of the original version of the file. In case of all these nodes leave the network, the updated file should be published in accordance with the data publication algorithm described above. As in the case of new data publication, the STORE RPCs are sent to the *k* closest nodes with appropriate <key, value> pairs.

The tree-like data memorization model is to be respected in the case of updates publication too. It means that once a node has published a modified version of some file at its node of origin, this last sends the STORE RPCs with replicas and metadata of the updated file to the appropriate γ and k nodes.

To effectuate the update procedure correctly, each <key, value> pair should be supplemented with such data as:

- identity data of an "author" of modifications confirmed by his digital certificate and the node ID of the used terminal;
- exact date and time of update;
- an original file's key and ID of a node that stores it (in the case of its off-line status).

When some user publishes successive modifications of the same file, the k nodes that store the file or the relative metadata simply remove his precedent updates every time a new update is performed.

Figure 5.6 illustrates data modification and update processes. In the presented example two nodes of level B (0011 and 0000) engaged in the same task download a file to be modified from two (amongst γ possible) diverse nodes (1000 and 1111) of level A responsible of this file (Step 1). After some necessary modifications have been made, the nodes 0011 and 0000 store the modified files at nodes from which the original versions have been downloaded (Step 2). Then, according to the tree-like data memorization model, replicas of the modified files are sent at γ responsible nodes by the nodes 1000 and 1111. As shown in the figure, the modifications of the node 0000 are effectuated later than those of 0011. So, the file with modifications made by the node 0000 will be the last version of this file.

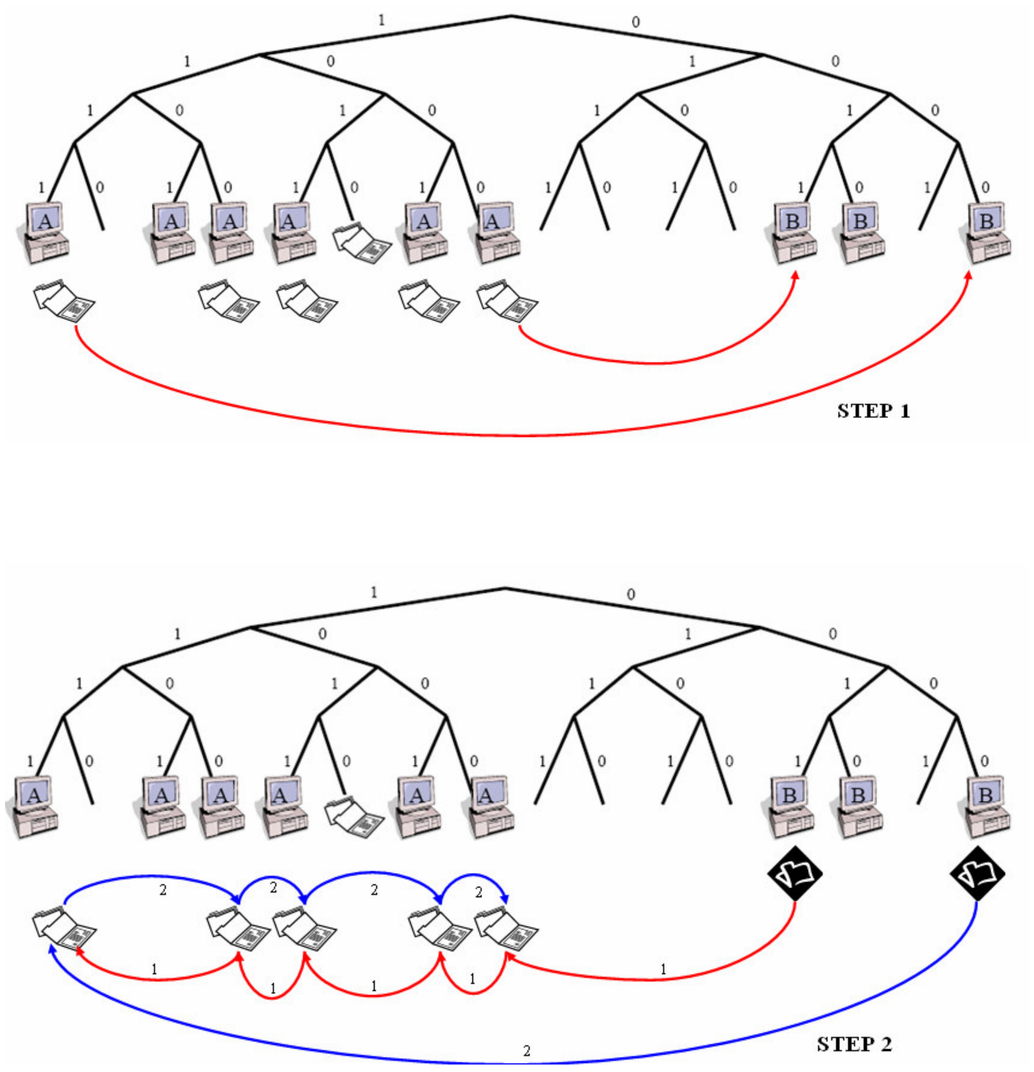


Fig. 5.6 Data modification and update

5.3.7 Security mechanisms and countermeasures involved in the presented solution

Secure assignment of node identifiers

In section 5.3.3 we have already described the procedure of secure assignment of node identifiers that consists in random attribution of IDs by trusted bootstrap terminals. As noted above, a node with ID, that is obtained as a hash function (SHA-1) of its static IP address, is potentially vulnerable to masquerade attacks. So, we try to avoid this undesired effect applying the described ID assignment mechanism.

Use of certificates

For effective handling of privileges regarding data access, it is necessary that each node, before downloading or modifying some file, is able to demonstrate its belonging to a department with the privileges equal or greater than those of the file's source node. To avoid some problems regarding identity falsification, the use of digital certificates makes sense.

A personal digital certificate associates a public key to some identity. Only the possessor of the certificate knows the corresponding private key, that permits him/her to create own digital signature and decrypt information encoded by the public key.

In our case, such certificate is attributed to a certain network terminal and attests its identity and level of privileges it possesses. So, regarding the described enterprise model, certificates of types A, B and C enable nodes of the corresponding levels to access, store and modify data within the same-name sub-trees and those of the lower levels (for B and C nodes). The certificates of type M are assigned to the nodes with a manager status and permit them to access, store, modify and cancel data produced and stored by any other node.

Countermeasures against specific attacks of DHT-based P2P environment

Regarding specific attacks of DHT-based environment, the described system proposes the following countermeasures and protective mechanisms.

Some effects of incorrect lookup rooting attacks can be avoided due to iterative character of Kademlia's lookup algorithm. In Kademlia such attacks can be detected by checking the progress of lookup at each step. In the case of absence of any progress (blatantly incorrect query forwarding), lookup process is backtracked to the previous "right" step and then proceeds with looking for an alternative direction of the search.

Incorrect routing update attack can be prevented, because in Kademlia the update of routing tables is implemented by a node automatically, as a "secondary effect" of ordinary lookups and interactions with other nodes.

Sybil attacks are not excluded, but lookup efficiency is improved by parallel routing (issuing α lookup requests at a time).

Partition attacks are prevented by involving trusted bootstrap terminals that should be contacted by nodes to join the network.

Mechanism of replication and storage of resources at the γ closest nodes prevents storage and retrieval attacks. So, even if one of the γ nodes maliciously denies the existence of data it is responsible for, there are other nodes enabled to provide the same resource. In this way we also eliminate a single point of failure represented by a unique node that stores a certain file. Moreover, the system can effectively cope with overload of targeted nodes with garbage packets, that represents a DHT analogue of Denial of Service attack. The tree-like data memorization model mitigates the impact of such attack due to uniform distribution of a file's replicas within the network. Since the replicas are stored in different sub-trees, even in the case of localized overload attacks to nodes of some selected part of the key-space, it is always possible to find at least one active node that stores a desired file's replica.

It is possible to additionally improve security of this solution applying the techniques for trust and reputation management presented in the previous chapter, i.e. using DHT mechanisms integrated with the proposed reputation evaluation algorithm.

5.4 Findings and future work

Kademlia DHT represents a secure, reliable and flexible infrastructure for data storage and retrieval system in enterprise networks.

Application of DHT principles to enterprise network environment is a novel approach, that allows to avoid such problems of centralized systems as denial of service, packet filtering attacks and low resistance to failure.

Taking in consideration specifics of deployment of P2P mechanisms in enterprise environment, it has been demonstrated that Kademlia is the most adaptable one to some particularities of hierarchical systems with clear distribution of different privileges and tasks between nodes. It is explained by the tree-like data structure of Kademlia DHTs.

Some novel approaches regarding the assignment of node identifiers and data storage in Kademlia are proposed: prefix identifiers are introduced to handle data access privileges, and the tree-like data memorization model is proposed to improve the storage mechanisms.

It is important to note that the presented system of data storage, retrieval and collective file editing can be adapted to any type of collaborative environment. It is not necessary to be applied to closed systems with specific hierarchical architectures.

So, while in the case of the hierarchical enterprise environment the prefixes are used to identify work groups with diverse privileges regarding data access, in the case of a network community consisting of nodes with equal "rights" the mechanism of prefixes can be used to distinguish groups with different interests, goals, tasks, etc.

We are currently working toward a software implementation of the described solution using available open-source implementations of Kademlia protocol.

5.5 References

[1] G. Caronni et al. Virtual Enterprise Networks: The Next Generation of Secure Enterprise Networking. In Proceedings of the 16th Annual Computer Security Applications Conference, ACSAC 2000

[2] P. Maymounkov, D. Mazières. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. In Proceedings of the 1st International Workshop on Peer-to-peer Systems, MIT, March 2002

[3] D. Stutzbach, R. Rejaie. Improving Lookup Performance over a Widely-Deployed DHT. In Proceedings of 25th IEEE International Conference on Computer Communications, INFOCOM'06, April 2006

[4] E. Sit, R. Morris. Security considerations for Peer-to-Peer Distributed Hash Tables. In Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS'02), Cambridge, March 2002

[5] N. Fedotova, S. Fanti, L. Veltri. Kademlia for Data Storage and

Retrieval in Enterprise Networks. In Proceedings of the Third International conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom-2007), November 12-15, 2007, New York, USA.

6 Conclusions

DHT-based peer-to-peer systems represent a quite young field of network technologies (the first four DHTs — CAN, Chord, Pastry, and Tapestry — were introduced about the same time in 2001). Overlay networks using DHT mechanisms offer many advantages, such as: efficient routing performance, high scalability, high exact-match accuracy of search, no single point of failure.

At the same time such systems represent a particular environment with specific security problems that are poorly studied yet. Even though DHT lookup algorithms provide some countermeasures against several effects of specific attacks, all these countermeasures have a "short-term" character: they help to cope only with instantaneous effects of malicious activity and don't resolve the problem of detection and elimination of malevolent contacts from routing tables. Moreover, for some specific security problems of DHT-based environment (e.g. incorrect routing updates in the case of recursive lookups, inconsistent behaviour, Sybil attacks) opportune countermeasures don't exist. Besides, some types of malicious activity in such kind of networks can cause the same problems as Byzantine failure in distributed computer systems.

This thesis has addressed the problem of integration of reputation management mechanisms and other instruments used in distributed computing environment with lookup processes in DHT-based peer-to-peer networks in order to improve resilience of such systems to destructive actions of malevolent or faulty components. The goal of this integration is to obtain a more efficient, less expensive (in terms of data transferred, computational resources involved and time spent) and possibly simple solution to cope with the specific problems of DHT-based environment.

A particular accent has been given to DHT-based environments with a collaborative nature. Unlike most of existing reputation management systems, we don't use the number of successful downloads as the main instrument to evaluate the trustworthiness of peers. It has been proposed to consider any type of interactions between nodes. It is explained by the fact that file sharing is not a unique service supported by DHT-based P2P systems. There are other application areas for P2P (e.g. collaborative and distributed computing) where it is important to consider such aspects as a risk factor, a "stay on-line" time or a number of requests without reply.

The solution that has been provided consists in application of a combination of different reputation mechanisms provided by some previously analyzed techniques for trust and reputation management in P2P. This solution represents an individual mechanism of reputation management for a single peer based on its own experience.

Then, it has been proposed to integrate the individual mechanism of reputation evaluation with an algorithm for trust and reputation management that takes in consideration the community context.

In particular, it has been proposed to introduce solutions for Byzantine Agreement used in distributed computing environment into a trust management algorithm designed for P2P networks based on DHTs, in order to obtain a simpler and efficient algorithm for reputation management in completely distributed environment. The complexity evaluation of the proposed solution compared to the previously proposed EigenTrust algorithm for reputation management in terms of number of messages sent per node shows that the algorithm presented in this thesis is less "expensive" in both number of messages sent and computational and memory resources involved.

Another issue considered in this work regards the application of DHT mechanisms to lookup and data retrieval processes in hierarchical collaborative environments, in particular, in enterprise networks.

Centralized organization of current enterprise networks doesn’t represent an ideal solution in terms of information security and reliability. Denial of service, packet filtering and low resistance to failure are frequent shortcomings of centralized systems. To avoid the above problems it has been proposed to introduce a distributed P2P data organization system to the enterprise environment. Kademlia-based Distributed Hash Tables has been applied to organize data storage and retrieval systems of enterprise networks. Due to the tree-like structure of the identifier space and the prefix matching lookup algorithm, Kademlia is easily adaptable to the context of enterprise hierarchy, e.g. it facilitates the assignment of different privileges regarding data access, to various system entities. The presented solution provides geographically separated users of enterprise networks with possibility to share, modify and publish data in parallel way.

Further optimization of the proposed algorithms and software implementations of the presented solutions are the future goals of the research activity presented in this thesis.

7 Acknowledgments

I would like to thank everybody who helped me in different ways in these three years of my PhD studies and collaborated with me on this research.

Special thanks go to:

my Ph.D. advisor, Luca Veltri, who gave me the possibility to work on this interesting topic and has made several direct contributions to this work. I really appreciate the freedom of research activity he let me have during these years and his precious suggestions that often helped me to resolve my doubts;

my husband Michele for being incredibly understanding, supportive, helpful, and most of all, patient with me all this time;

to all my "italian" family Tomaiuolo-Armillotta for their sincere encouragement and support that I feel from my first day in Italy. Thank you, my dears, for letting me enter not only in your family but also in your hearts!

to my closest friends Francesco and Susanna who are always ready to listen and to lend their shoulders to me when I need it, and to my good friends Luca and Marco for their always kind-hearted attitude towards me.

All my thanks will never be enough to express my gratitude to my Mother. Her infinite love and the support she gives me through all my life are the basis of everything I do. She always goes the extra mile for me, and she does it with incredible grace.