TrustedKad – Application of Trust Mechanisms to a Kademlia-Based Peer-to-Peer Network

DISSERTATION

To Obtain the Academic Degree
Doctor Rerum Naturalium (Dr. rer. nat.)
In Computer Science

Submitted to the
Faculty of Business Administration and Economics
Institute for Computer Science and Business Information Systems
University of Duisburg-Essen, Germany

By

Dipl.-Wirt.-Inf. Michael Kohnen
Born in Herten, Germany

Date of Disputation: 10 November 2014

Reviewers:
1. Prof. Dr.-Ing. Erwin P. Rathgeb
2. Prof. Dr. Bruno Müller-Clostermann

Abstract

Peer-to-peer networks (P2P) are distributed systems that consist of equal nodes ("peers"). In contrast to classic client/server systems, there is no hierarchy or central entity: All peers offer services and use them at the same time. In the past decade, a multitude of different P2P applications has been developed – filesharing applications such as BitTorrent and eMule and communication applications such as Skype are among the most popular of them.

Research has shown that P2P networks are vulnerable to different kinds of attacks. Known attacks include, e.g., the Sybil attack and the Eclipse attack. Traditional countermeasures against the attacks are replication and the usage of disjoint routing paths to reduce the probability of interacting with malicious nodes during a routing or storage operation.

More recently, trust mechanisms have been proposed and analyzed for applicability to P2P networks. The existing related work mostly targets unstructured P2P networks – however, in real-world environments, the structured networks prevail. Especially implementations of the Kademlia algorithm are widely spread, as it is used by BitTorrent and eMule. Nevertheless, none of the trust-based approaches that aim at structured networks specifically attempts to enhance Kademlia's security.

Due to Kademlia's prevalence, TrustedKad is particularly designed by the author to improve the security of the Kademlia algorithm. In this thesis, TrustedKad is introduced and its functioning is explained.

TrustedKad rates the behavior of nodes after routing and storage operations as either positive or negative. To do so, it defines the rules by which inoffensive and malicious behavior is identified in dependence of the functioning of the Kademlia algorithm. Based on the ratings, routing and storage trust values are calculated to identify inoffensive and malicious nodes. Every node uses thresholds for these trust values to decide which nodes it regards as trustworthy. Non-trustworthy nodes are avoided during a node's own operations. Furthermore, TrustedKad uses additional security features to further increase the security of the system. They are introduced in this thesis.

In order to evaluate TrustedKad, it is implemented and analyzed in a simulation environment. The results presented in this thesis show that TrustedKad is able to distinguish inoffensive and malicious nodes. It counters miscellaneous variations of known attacks and improves the security of Kademlia-based networks considerably.

Zusammenfassung

Peer-to-Peer-Netzwerke (P2P) sind verteilte Systeme, die aus gleichberechtigten Knoten („Peers“) bestehen. Im Gegensatz zu klassischen Client-Server-Systemen gibt es in P2P-Netzwerken keine hierarchischen Ebenen oder zentrale Kontrolleinheiten: Alle Peers bieten gleichzeitig Dienste an und nutzen sie. Im vergangenen Jahrzehnt ist eine Vielzahl verschiedener P2P-Anwendungen entwickelt worden – Filesharing-Anwendungen wie BitTorrent und eMule und Kommunikations-Anwendungen wie Skype gehören zu den bekanntesten von ihnen.

Forschungsarbeiten haben gezeigt, dass P2P-Netzwerke anfällig für verschiedene Arten von Angriffen sind. Bekannte Angriffe sind z.B. die Sybil- und die Eclipse-Attack. Die üblichen Gegenmaßnahmen gegen die Angriffe sind Replikation und das Verwenden von disjunkten Routing-Pfaden, um die Wahrscheinlichkeit zu reduzieren, während einer Routing- oder Storage-Operation mit bösartigen Knoten zu interagieren.

Seit einiger Zeit wird die Anwendung von Vertrauensmechanismen auf P2P-Netzwerke untersucht. Existierende Arbeiten betrachten meist unstrukturierte P2P-Netzwerke – in realen Umgebungen überwiegen jedoch die strukturierten Netzwerke. Insbesondere Implementierungen des Kademlia-Algorithmus‘ sind weit verbreitet, da er von BitTorrent und eMule genutzt wird. Dennoch versucht keiner der vertrauensbasierten Ansätze, die strukturierte Netzwerke behandeln, speziell die Sicherheit von Kademlia zu verbessern.

Aufgrund der Verbreitung von Kademlia wird TrustedKad vom Autor entwickelt, um die Sicherheit des Kademlia-Algorithmus‘ zu verbessern. In dieser Arbeit wird TrustedKad eingeführt und die Funktionsweise erläutert.

TrustedKad bewertet das Verhalten von Knoten nach Routing- oder Storage-Operationen als entweder positiv oder negativ. Dafür definiert TrustedKad unter Berücksichtigung der Funktionsweise von Kademlia die Regeln, nach denen gut- und bösartiges Verhalten identifiziert wird. Basierend auf diesen Bewertungen werden Vertrauenswerte für Routing und Storage berechnet, um gutartige und bösartige Knoten zu erkennen. Jeder Knoten nutzt Schwellwerte für diese Vertrauenswerte, um zu entscheiden, welche Knoten er als vertrauenswürdig ansieht. Nicht vertrauenswürdige Knoten werden während der eigenen Operationen eines Knotens vermieden. Darüber hinaus nutzt TrustedKad zusätzliche Sicherheitsfunktionen, um die Sicherheit des Systems weiter zu erhöhen. Diese werden im Verlauf dieser Arbeit vorgestellt.

Um TrustedKad zu evaluieren, wird es in einer Simulationsumgebung implementiert und analysiert. Die in dieser Arbeit präsentierten Ergebnisse zeigen, dass TrustedKad in der Lage ist, gutartige und bösartige Knoten zu unterscheiden. Es wehrt verschiedene Variationen von bekannten Angriffen ab und verbessert die Sicherheit von Kademlia-basierten Netzwerken deutlich.

Relevant Publications of the Author

The following publications of the author are related to this thesis:

Michael Kohnen, Mike Leske, and Erwin P. Rathgeb: "Conducting and Optimizing Eclipse Attacks in the Kad Peer-to-Peer Network". IFIP Networking Conference, May 2009. [1]

Michael Kohnen, Jan Gerbecks, and Erwin P. Rathgeb: "Applying Certificate-Based Routing to a Kademlia-Based Distributed Hash Table". IARIA Conference on Advances in Peer-to-Peer Systems (AP2PS), November 2011. [2]

Michael Kohnen: "Analysis and Optimization of Routing Trust Values in a Kademlia-Based Distributed Hash Table in a Malicious Environment". IEEE Baltic Congress on Future Internet Communications (BCFIC), April 2012. [3]

Michael Kohnen: "Applying Trust and Reputation Mechanisms to a Kademlia-Based Distributed Hash Table". IEEE International Conference on Communications (ICC), June 2012. [4]

The IEEE ICC and IEEE BCFIC publications contain parts of the TrustedKad concept and preliminary results. More details on the publications are given in Section 3.4.

List of Figures

List of Tables

List of Abbreviations

General

AIDS	Acquired Immune Deficiency Syndrome
API	Application Programming Interface
BCFIC	Baltic Congress on Future Internet Communications
CA	Certificate Authority
CDF	Cumulative Distribution Function
CPU	Central Processing Unit
DHT	Distributed Hash Table
DoS	Denial of Service
DTLS	Datagram Transport Layer Security
FTP	File Transfer Protocol
GiB	Gibibyte (2^{30} bytes)
IARIA	International Academy, Research, and Industry Association
ICC	International Conference on Communications
ID	Identifier
IEEE	Institute of Electrical and Electronics Engineers
IFIP	International Federation for Information Processing
IP	Internet Protocol
JXTA	Juxtapose
KBR	Key-Based Routing
MANET	Mobile Ad hoc Network
MD4	Message-Digest algorithm 4
MD5	Message-Digest algorithm 5
P2P	Peer-to-Peer
P2PTV	Peer-to-Peer Television
PNRP	Peer Name Resolution Protocol
QoS	Quality of Service
RAM	Random Access Memory
RPC	Remote Procedure Call
SETI@home	Search for ExtraTerrestrial Intelligence at home
SHA-1	Secure Hash Algorithm 1
SQL	Structured Query Language
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TTL	Time-To-Live

UDP	User Datagram Protocol
US	United States (of America)
VBA	Visual Basic for Applications
VoIP	Voice over IP
VPN	Virtual Private Network
XOR	eXclusive OR

Symbols and Parameters

α	Kademlia, Kad, TrustedKad: Number of parallel Lookup queries
C	TrustedKad: Colluding (malicious nodes)
Coll	TrustedKad: Colluding (malicious nodes)
g_R	TrustedKad: Number of grace routing ratings
g_S	TrustedKad: Number of grace storage ratings
k	Kademlia, Kad, TrustedKad: Number of nodes in one bucket of the Kademlia routing table
K_{pub}	TrustedKad: Public key
M	TrustedKad: Fraction of malicious nodes
NC	TrustedKad: Non-colluding (malicious nodes)
NoColl	TrustedKad: Non-colluding (malicious nodes)
numLookupResult	TrustedKad: Number of nodes a Lookup operation shall return when it ends
numReplica	TrustedKad: Number of replicas that shall be stored or retrieved during storage operations
$R_{R,i}^{+/-}$	TrustedKad: Number of positive (+) or negative (-) routing ratings of node *i*
$R_{S,g}^{+/-}$	TrustedKad: Number of positive (+) or negative (-) storage ratings of node group *g*
$R_{S,i}^{+/-}$	TrustedKad: Number of positive (+) or negative (-) storage ratings of node *i*
RT	TrustedKad: Routing trust value threshold
ST	TrustedKad: Storage trust value threshold
TNM	TrustedKad: Trusted nodes that are malicious
$T_{R,i}$	TrustedKad: Routing trust value of node *i*
$T_{S,g}$	TrustedKad: Storage trust value of node group *g*
$T_{S,i}$	TrustedKad: Storage trust value of node *i*
$AuthB_i$	Mekouar: Authentic behavior rating of node *i*
$Available_i$	Mekouar: Number of times node *i* was available for upload
$Average_{sup}$	Mekouar: Mean availability of all nodes connected to supernode *sup*
$ContrB_i$	Mekouar: Contribution behavior rating of node *i*
$CredB_i$	Mekouar: Credibility behavior rating of node *i*

$D_i^{+/-}$	Mekouar: Sum of the file sizes of the successful (+) and failed (-) downloads of node *i*
$Involvement_i$	Mekouar: Ratio between a peer’s up- and downloads
$NbrPeers_{sup}$	Mekouar: Number of peers that are connected to supernode *sup*
N_i	Mekouar: Total number of downloads of node *i*
N_i^*	Mekouar: Number of downloads of node *i* with unexpected rating
$U_i^{+/-}$	Mekouar: Sum of the file sizes of the successful (+) and failed (-) up-loads of node *i*
X	Mekouar: Weight for availability in contribution behavior
Y	Mekouar: Weight for involvement in contribution behavior
c_1	S/Kademlia: Number of zero bits at the beginning of the result of a crypto puzzle with the node ID as input
c_2	S/Kademlia: Number of zero bits at the beginning of the result of a crypto puzzle with a random number as input
s	S/Kademlia: Number of sibling nodes in the sibling table

Table of Contents

1 Introduction and Motivation

During the last decade, peer-to-peer (P2P) technology has emerged from only being used for mostly illegal filesharing activities to being used by diverse applications. Probably the most prominent example of a commercial P2P network is Skype [5], a Voice-over-IP (VoIP) system. Other applications for P2P technology include distributed storage systems such as Freenet [6] and the original Wuala[1] [7] and media streaming and P2P television (P2PTV) networks such as Zattoo [8]. There are also open frameworks and protocols available (e.g., JXTA – Juxtapose [9] and PNRP – Peer Name Resolution Protocol [10]) to create P2P applications in a variety of programming languages such as Java, C/C++ and C#. JXTA is used, e.g., by the collaboration and content sharing system Collanos Workplace [11], whereas PNRP is included in Microsoft Windows Vista and newer versions and is used, e.g., for Remote Desktop Assistance.

Large Internet companies such as Amazon, Facebook and Google are known to use P2P technology internally: Amazon Dynamo [12] is an internal data store for several of Amazon's core services that is similar to Chord [13], a structured P2P algorithm. Google's equivalent is called BigTable [14] and "*[…] is designed to scale to a very large size: petabytes of data across thousands of commodity servers*" [14]. Facebook has developed a key-value store system called Cassandra [15] that has been used for the inbox search feature until 2010. It has been published as an open-source project and is now maintained by the Apache Software Foundation.

Besides these more recent P2P applications, filesharing still has a considerable share of the total Internet traffic [16,17]. P2P technology has become widely known due to the filesharing application Napster[2] in 1999 [18], which was used mainly to exchange music files. The popularity of filesharing has led to the fact that, in the public, the terms "filesharing" and "peer-to-peer" are sometimes even used as synonyms.

Napster has been shut down in 2001 by a US court order [19]. As Napster has been a centralized P2P network that still relied partly on central servers, this shutdown has been

[1] Wuala has been acquired by the storage hardware manufacturer LaCie in 2009. LaCie has ceased the P2P service in October 2011, probably in order to promote fee-based centralized storage services offered under the same name.

[2] As with "Wuala", also the brand "Napster" is used for fee-based (music) services since 2003.

possible. The successful prosecution of Napster has probably a great share of the development of P2P networks that do not require any central infrastructure.

These examples show that P2P networks exist in various configurations: There are open networks such as the filesharing networks that can be joined without prior registration. Skype is an example for a network that requires registration and therefore also requires a central authority. However, Skype can still be considered an open network, as the registration is not limited to certain user groups, in contrast to the closed networks used internally by companies.

1.1 Types of Peer-to-Peer Networks

P2P networks can be divided into unstructured and structured networks [20]. The basic difference is that structured networks define a responsibility assignment between nodes and objects which does not exist in unstructured networks. That way, a definite set of nodes is responsible for a certain object, and there is an algorithm that describes how to reach the responsible nodes and thereby the object. This leads to the fact that every content item that exists can also be found, as only the responsible nodes need to be queried. This is not the case for unstructured networks: They typically use broadcasts to retrieve content items that terminate after a certain amount of hops. If the node storing the queried item cannot be reached this way, the content item is not found despite its existence.

Distributed Hash Tables (DHTs) are the most prominent form of structured P2P networks, so in most cases, the terms “structured P2P network” and “DHT” are used synonymously. DHTs, especially implementations of the Kademlia algorithm [21], are the state-of-the-art technique behind filesharing systems consisting of several million peers. They exist for several years now without (serious) complaints of the users about lacking security, so one might think that there is no reason to analyze these networks’ security, as it seems sufficient. However, a closer look reveals that there are indeed security issues which the developer community tries to cope with. Their security measures are rather simple, so, e.g., receiving another than the desired file (e.g., a meaningless home video instead of a blockbuster movie) is a daily occurrence. Nevertheless, no complaints become public, as the users want to stay anonymous and not be subject to legal prosecution.

Currently, the most prominent filesharing systems and applications are BitTorrent[3] [22] and eMule [23]. They are hybrid clients, as they use both unstructured and structured P2P networks for their operations. The unstructured networks are called “BitTorrent” and “eDonkey”, the structured ones “trackerless BitTorrent”[4] and “Kad”, respectively. Both structured networks are based on the Kademlia algorithm which uses a symmetric, XOR-based metric to route to the target nodes. Kademlia is explained in detail in Section 2.4.

1.2 Overlay Concept

P2P networks form an overlay network on top of an existing network, e.g., a TCP/IP network such as the Internet. The underlying network is called underlay network. Overlay networks introduce their own identifiers and routing concepts and use the underlay network for message transmission. Overlay networks do not take the underlay networks’

[3] There are several applications available that provide access to the BitTorrent and trackerless BitTorrent networks. For simplicity, they are summarized as “BitTorrent” in this thesis.

[4] It should be mentioned that there are two Kademlia implementations for trackerless BitTorrent operation that are not compatible to each other. As the differences are not relevant for this thesis, they are summarized as “trackerless BitTorrent”.

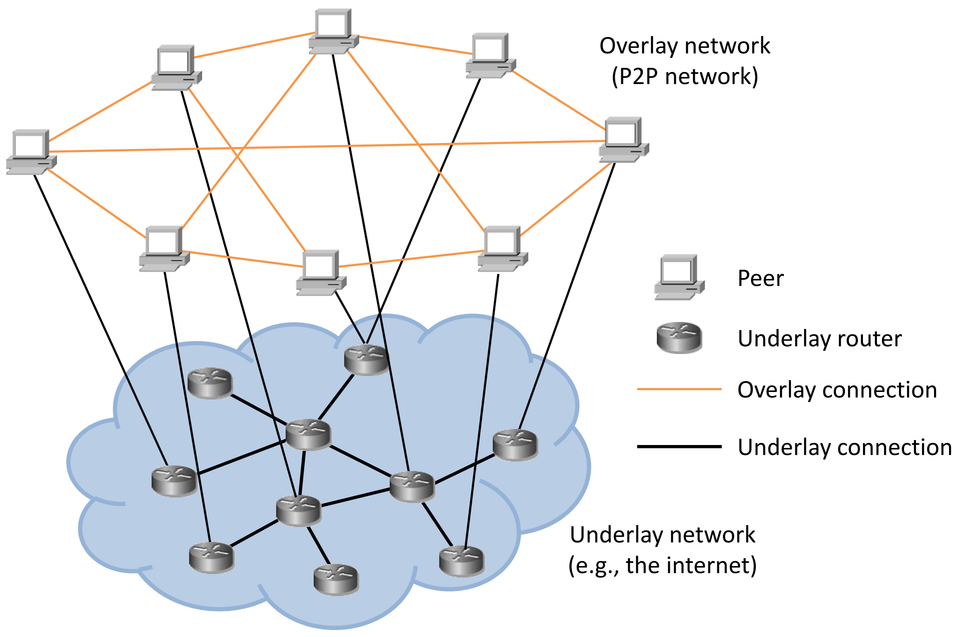


Figure 1: Relation between overlay and underlay network [24]

topology into consideration for their own overlay routing. The independent addressing schemes lead to the fact that logically close peers (overlay view) can easily be physically far away from each other (underlay view) and vice versa. Virtual Private Networks (VPNs) are another example for an overlay network: They introduce another addressing layer above the underlay network.

Figure 1 shows an overlay network with its underlay network: The orange lines show overlay connections; the black lines denote underlay connections. The leftmost peer and the rightmost peer are directly linked in the overlay. On the shortest underlay path, however, there are four hops between them. Contrary to this, the peers at 1 and 6 o'clock are connected to the same underlay router, but have at least two overlay hops between them.

1.3 Definition of Peer-to-Peer Systems

Steinmetz and Wehrle [25,26] and Oram [27] define peer-to-peer systems as follows: *"[a Peer-to-Peer system is] a self-organizing system of equal, autonomous entities (peers) [which] aims for the shared usage of distributed resources in a networked environment avoiding central services. In short, it is a system with completely decentralized self-organization and resource usage."*[5] P2P networks are not static, as peers can enter and leave the system as they wish. The rate by which nodes enter and leave the system is called "churn rate".

This definition clearly distinguishes peer-to-peer systems from client-server architectures in which a central server provides all resources: The clients only use these resources and do not provide resources themselves. P2P systems do not distinguish between different kinds of nodes (except for the hybrid systems that define two types of peers as explained in Section 2.2.3), because all peers are both providing and using resources. So in the following, the term "server" describes an entity that only provides services or resources.

[5] [25] Page 10, in turn referencing [26] and [27]. Reference [27] is not available to the author.

Analogously, a “client” only uses services or resources, but does not provide any. For the term “peer” as introduced above, also the terms “node” and (rarely) “servent”, a combination of “server” and “client”, are used [20]. The terms “peer” and “node” are used synonymously in the literature and also in this thesis.

The definition of a P2P system does not include services such as SETI@home: SETI@home is a distributed computing project of Berkeley University that aims to analyze radio signals from outer space to “*Search for ExtraTerrestrial Intelligence (SETI)*” [28]. Similar projects aim to find a cure for AIDS or perform other mostly scientific tasks requiring a huge amount of computing power. All of these projects have in common that they do not require the clients to communicate with other clients. They only receive tasks from the central server, execute them and return the results. There is no self-organization involved, as the whole process is controlled by the server, so they do not match the above definition of a P2P system.

1.4 Definition of Trust

Trust is an essential part of social life, and we rely on it every day: Every time before we interact with somebody, we evaluate if we regard the other one as trustworthy. If we do not trust another person, we avoid interacting with him. Trust is based on past experiences with the other and can be direct and indirect, i.e., it can be based on one’s own experiences or on the experiences of other persons. The sum of these past experiences is called “reputation”.

Gambetta defines trust as follows: “*[…] trust (or, symmetrically, distrust) is a particular level of the subjective probability with which an agent assesses that another agent or group of agents will perform a particular action, both* before *he can monitor such action (or independently of his capacity ever to be able to monitor it)* and *in a context in which it affects* his own *action […]. When we say we trust someone or that someone is trustworthy, we implicitly mean that the probability that he will perform an action that is beneficial or at least not detrimental to us is high enough for us to consider engaging in some form of cooperation with him.*” [29]

Trust in electronic systems can be expressed in a variety of ways: As a binary value, out of a set of discrete values or using continuous values, e.g., between -1 and 1. Common examples of electronic trust systems are rating systems of e-commerce platforms such as eBay and Amazon Marketplace. The eBay feedback system [30] asks every buyer to rate the seller after a transaction. These ratings are aggregated and formed into a percentage that denotes the fraction of positive ratings a user has received compared to his total number of ratings. In addition to that percentage, other users can also see the total number of ratings a user has received. This way, other users can inform themselves about sellers before they place their bid. eBay also asks sellers to rate their customers so that other sellers can deny a transaction if they regard the buyer’s rating as too low.

1.5 Security of Peer-to-Peer Systems

The research community has proven that P2P networks are vulnerable to a multitude of attacks. The commonly known ones are the Sybil attack [31], the Eclipse attack, and attacks on routing and storage processes (as explained in detail in Section 2.5) [32]. Additionally, the author has shown in [1] that the Kad filesharing network is still susceptible to such attacks in spite of the fact that it uses security measures that are designed to counter them.

Also botnets use P2P technology to spread their control information. The Alureon botnet does not create its own P2P network, but instead abuses the Kad filesharing network for its purposes [33]. Currently, the presence of Alureon peers in the Kad network does not disturb the network, as the peers seemingly do not behave maliciously. However, this can change anytime.

The successful attacks in both research and praxis and the abuse of existing P2P networks show that security in P2P networks is a topic that requires research. With TrustedKad, the author aims to improve the security of a Kademlia-based P2P network: Existing security measures are combined and adapted where necessary and new Kademlia-specific security features are added to create a consistent system.

1.6 Introduction and Scope of TrustedKad

In this thesis, TrustedKad is presented and evaluated. As the name suggests, TrustedKad applies trust mechanisms to a Kademlia-based DHT. Kademlia has been chosen as the base algorithm because it is the DHT algorithm that has had the largest number of users for the past years: BitTorrent and eMule both implement Kademlia and are in use for several years now by millions of users – BitTorrent has reached 150 million users in January, 2012 [34]. Unstructured filesharing networks such as Gnutella 0.4 [35] have collapsed before reaching such a large number of users, and the functioning of the Skype network is not publicly known, so both cannot be used as a basis. In addition to its proven scalability and functioning, Kademlia has fewer restrictions regarding the choice of the next hop as Chord [13] or Pastry [36], for example, which makes its routing process more robust.

Trust mechanisms have been applied to P2P systems before (more details can be found in Section 3.2), but none of the existing approaches specifically targets Kademlia and its properties. So TrustedKad is developed by the author as a system that applies trust mechanisms to a Kademlia-based DHT. TrustedKad is designed in due consideration of related work that indicates the key requirements for trust systems in P2P networks and takes Kademlia-specific properties into account, e.g., when defining inoffensive and malicious behavior of nodes.

The goal of trust mechanisms in the P2P context is to rate the behavior of a node and to avoid interaction with it if it does not behave correctly. This is also necessary in non-anonymous networks (e.g., with a central authority that issues certified identities) because certification and identification do not prevent nodes from acting maliciously. Furthermore, legitimate nodes can also be taken over by attackers so that their identities are abused. In addition, also a change in a node’s behavior from inoffensive to malicious can be detected by continuously rating a node. TrustedKad aims to enhance the security of open networks that do not possess a central authority for management purposes such as handling trust information. It is designed to be application-independent, so, for example, it does not rely on specific properties of filesharing networks.

Nodes are rated after routing processes and after content item retrievals. The ratings are binary, i.e., they are either positive or negative, and they are separately counted for routing and for storage processes. The counters are used to determine a node’s routing and storage trust values on a scale from -1 to 1. These trust values are used to identify inoffensive and malicious nodes: High trust values indicate an inoffensive node; low trust values a malicious one. Nodes use individual trust value thresholds to decide which nodes to interact with. If no or not enough trustworthy nodes are available for an operation, it shall be cancelled.

These individual views on the trustworthiness of nodes lead to the fact that every node must be able to inform other nodes about all existing nodes, i.e., also about nodes it does not trust itself. For its own operations, however, a node must only use trusted nodes. In this thesis, this is referred to as “clean routing process” approach. Other authors such as [37,38] propose to keep the routing table[6] free from malicious nodes, which is referred to as “clean routing table” approach. This approach filters the local routing table of a node and, as a result, a node cannot inform about all nodes that exist in the network, but only about a limited number of them.

The concept of structured P2P networks of assigning responsibility for content items to nodes according to an algorithm interferes with the individual views of the trustworthiness of a node: If a content item is successfully stored in the network and another node attempts to retrieve it, the item can only be found if it has been stored on nodes the querying node trusts. If all replicas are stored on nodes the querying node does not trust, it cannot find the content item despite its existence. A similar problem exists if the querying node cannot find the responsible nodes because there are not enough trusted nodes on the route to them. The analyses published by the author in [2] have revealed that the impact of these effects is rather low (more details can be found in Section 3.4).

1.7 Scope and Structure of This Dissertation

TrustedKad aims to improve the security of a Kademlia-based DHT using trust-based mechanisms. In this thesis, the components of TrustedKad are introduced and explained and it is shown that TrustedKad achieves its goal. In order to prove that TrustedKad works as intended, the presented simulation results focus on the trust values given to inoffensive and malicious nodes and on the success rates of the operations performed in the network. The optimization of TrustedKad’s performance, e.g., with regard to message size, number of messages or storage space, is out of the scope of this thesis; application-specific requirements are not analyzed in detail either.

The remainder of this thesis is structured as follows: In Chapter 2, the basics of unstructured and structured P2P networks are explained. The relevant functions of the Kademlia algorithm are explained in detail, followed by a description of general security challenges of structured P2P systems such as the Sybil attack and the Eclipse attack. The chapter ends with an explanation of the general attack types targeting trust systems and the key requirements of P2P trust systems that can be derived from these attacks.

In Chapter 3, the current state of research is presented. A non-trust-based security system that specifically targets Kademlia is shown in detail, followed by an explanation of a trust-based security approach that targets unstructured filesharing networks. In addition, other P2P trust systems are briefly presented, and trust-based approaches in mobile ad hoc networks are introduced in a short excursus. An overview of the previous work of the author concludes the chapter.

TrustedKad is introduced in Chapter 4. Based on the general attacks on P2P systems and the key requirements of P2P trust systems, the components of TrustedKad and their functioning are motivated and explained in detail. The chapter finishes with a brief discussion of the overhead that is introduced by TrustedKad and how it could be optimized.

[6] In order to communicate with other nodes, every node maintains a routing table in which it stores information about other nodes it knows. The structure of the routing table of Kademlia is explained in Section 2.4.3.

Based on the description of TrustedKad, the implementation used to evaluate it is described in Chapter 5. The simulation framework OverSim [39] that is based on OMNeT++ [40] is introduced, together with the changes required by TrustedKad. The simulation environment and the simplifications that have been made are described and the operations that take place in the network (such as routing, content storage and content retrieval) are explained in detail.

The results of the simulations are presented in Chapter 6. First, the parameters and the respective settings that have been used in the different configurations are explained, followed by the statistical indicators that are used to evaluate the differences between the configurations and the effectiveness of TrustedKad. The chapter is then divided into four more parts: In the first part, baseline configurations without any security measures are shown that are used as a reference for the other parts. The next part presents the results of configurations in which only routing operations are attacked. In the following, the effectiveness of TrustedKad in countering attacks only against storage operations is shown. In the last part of this chapter, simulation results of attacks against both routing and storage processes are presented. In this scenario, attackers can cause the most damage, so it is both a worst-case scenario and the most realistic one.

In Chapter 7, the results of this thesis are summarized and a brief outlook on possible next steps is given.

2 Basics

In this chapter, the difference between unstructured and structured P2P networks is explained first. Then, the basic layer structure of DHTs is shown and several properties of structured networks such as the different types of routing and storage procedures are introduced. Afterwards, the main functionalities of the Kademlia DHT algorithm are described in detail, as TrustedKad is based on Kademlia. The chapter ends with a brief introduction of the basic attack types against P2P and trust systems. If not noted otherwise, the explanations of the network types base on [20].

2.1 Client-Server Networks

The classic application type on the Internet is client-server-based: A server offers services that the clients use. There is a strict separation between servers and clients: Servers do not use any other services and the clients do not offer any services. The clients do not depend on other clients while using any service. Figure 2 shows the network topology of a client-server network: The server is shown in the middle, and the clients only connect to the server, but not to other clients.

The server is the bottleneck and single point of failure of this system: If it fails due to a failing connection or overload etc., the service is no longer available. If the number of clients increases, the server must be able to adapt to that which raises scalability problems.

In order to overcome these problems, P2P systems have been developed. In P2P systems,

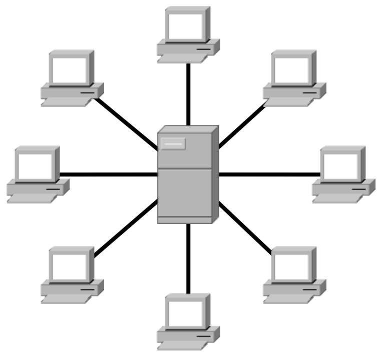


Figure 2: Topology of a client-server network [20]

the load is equally distributed across all participating nodes. So if the number of participants increases, the available resources (storage space, computing power, etc.) increase as well. P2P systems are divided into unstructured and structured networks, with the unstructured networks being divided again into centralized, pure and hybrid networks. These types are presented in the following sections.

2.2 Unstructured Peer-to-Peer Networks

In contrast to client-server networks, in P2P networks, all services are offered by peers – there is no distinction between clients and servers, as each peer offers and uses services at the same time. Sometimes, this is denoted by using the term “servent” instead of “peer” as a combination of the terms “server” and “client”. A central entity for management purposes may still exist, but the resource usage takes place only between the peers.

Unstructured P2P networks can be divided into three types of networks: Centralized, pure and hybrid. The three types are explained in the following. Because filesharing has propelled the development of the network types, filesharing examples will be given for each network type.

2.2.1 Centralized Peer-to-Peer Networks

The first widely known P2P network Napster has been a centralized network. In centralized networks, a central server exists for management purposes as shown in Figure 3. The nodes can join the network by contacting the server.

In the case of Napster, the server is responsible for managing information about the music files. All peers report the music files they store to the server. If the user looks for a certain music file, it asks the server for it. The server responds with a list of files matching the search criteria and the peer on which each of the files is stored. After a file has been chosen, the download takes place only between the peers without involvement of the server. This considerably reduces the amount of storage, bandwidth and computational resources required by the server.

As already mentioned before, Napster has been shut down due to a US court order after only two years of operation. Shutting down Napster was possible because the network is not operational without the central server. This applies to all centralized P2P networks. As a result, developers aimed to eliminate this single point of failure, which has brought forth the pure P2P networks.

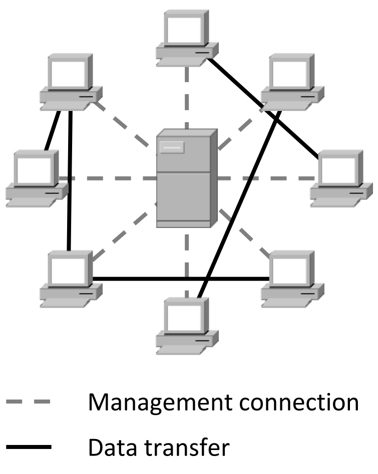


Figure 3: Topology of a centralized P2P network [20]

2.2.2 Pure Peer-to-Peer Networks

In order to eliminate the need for a central component as required by centralized networks, the pure P2P networks have been developed. Examples for such networks are Gnutella 0.4 [35] and Freenet [6]. As shown in Figure 4, they do not possess any central component anymore and rely on flooding for content retrieval (see below). Any arbitrary peer can be removed from the network without disturbing it. Each peer knows a certain amount of other nodes participating in the network; for Gnutella 0.4, the average number of known other peers is three. Only a peer itself knows about the content it stores; there is no entity storing information about all available content items.

For content retrieval, a peer sends the query to all peers it knows, which forward it to all peers they know (except for the one from which they have received the message), and so on. The forwarding terminates when the Time-To-Live counter (TTL, denoted in hops) of the message expires; the default TTL value of Gnutella 0.4 is seven. This retrieval technique is called flooding and creates a large messaging overhead, because the peers receive many messages concerning content items they do not store. If no node replies to the query, the query fails. Flooding is not able to cover the whole network, however, so it can occur that existing content items are not found by a peer if more hops lie between itself and the peer that stores the content item.

Response messages are routed back along the path the query has been transmitted. For transmission of a content item, a direct connection between the uploading and the downloading peer is established.

To join the network, another node that is already part of the network needs to be known, e.g., from the last session. A ping message is sent to it. The receiving node forwards it to all nodes it knows and replies with a pong message. This also repeats until the TTL limit is reached.

The flooding mechanism leads to an unacceptably large amount of routed query, response, ping and pong messages that congest the links of the peers. Thus, a too small share of a peer's bandwidth remains for the actual content exchange. This large amount of network discovery traffic has finally led to the breakdown of the Gnutella 0.4 network [41]. This in turn has resulted in the development of Gnutella 0.6 [42], a hybrid P2P network.

2.2.3 Hybrid Peer-to-Peer Networks

The pure P2P network Gnutella 0.4 has exposed the limitation of pure P2P networks: A too large amount of the traffic is generated by maintenance messages. Centralized P2P systems do not require maintenance messages to such an extent, as the server controls

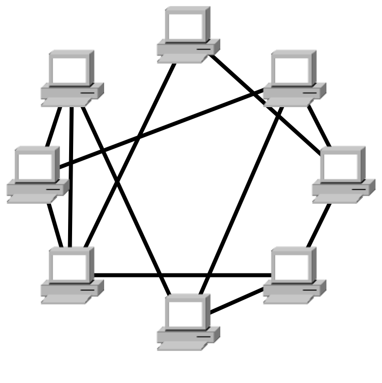


Figure 4: Topology of a pure P2P network [20]

the network and the clients do not need to perform routing operations.

Therefore, the hybrid P2P networks aim to combine the advantages of both network types while avoiding their disadvantages: They introduce a two-tier hierarchy of peers, dividing the peers into "superpeers" and "leafnodes". This means that hybrid P2P networks break with the definition of a P2P network in which all peers are equal, as they define two different types of peers.

The superpeers form a pure P2P network that replaces the central server of the centralized P2P networks. The leafnodes connect to a superpeer and direct all their queries to it. Figure 5 shows a hybrid network with six superpeers and three or four leafnodes per superpeer.

The superpeers use flooding in the superpeer network to forward queries to other superpeers. A hybrid P2P network is therefore a combination of one pure P2P network consisting of superpeers only and one centralized P2P network per superpeer with the superpeer acting as server for its leafnodes. This way, the number of nodes in the pure part of the network is significantly reduced. Hybrid networks overcome the centralized networks' problem of having a single point of failure and the pure systems' problem of congestion created by maintenance messages.

In the Gnutella 0.6 network, about 50 to 100 leafnodes connect to one superpeer. Just as with the centralized P2P networks, the leafnodes inform their superpeer about the files they store. Content queries are transmitted using flooding in the pure P2P part. Superpeers only forward content queries to leafnodes that store the queried content, so leafnodes do not receive queries for content items they do not store. Ping and pong message traffic is not forwarded to leafnodes.

When a new node joins the network by sending a ping message to a superpeer, the superpeer responds with information about other superpeers. This way, the leafnode is able to connect to another superpeer if its own superpeer leaves the network. Just as with the pure P2P networks, the superpeer that is contacted to join the network needs to be known from the previous session, for example. A superpeer regularly sends ping messages to the pure superpeer network to keep its list of other superpeers up to date.

In addition to Gnutella 0.6, the eDonkey network [43] is another example for a hybrid P2P network which is still in use today.

Regrettably, all unstructured P2P networks have in common that they cannot guarantee

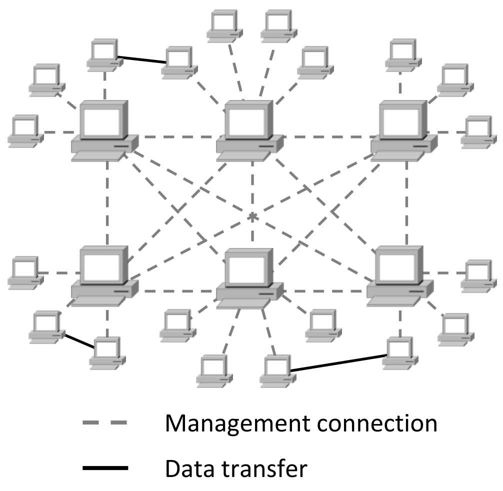


Figure 5: Topology of a hybrid P2P network [20]

that content that exists in the network can also be found by any arbitrary peer: The flooding mechanism that is limited by a TTL does not allow for that. Structured P2P networks overcome this limitation by assigning responsibility for a content item to algorithmically defined nodes.

2.3 Structured Peer-to-Peer Networks

Structured P2P networks have a similar topology as pure P2P networks: There is no central instance, and the network remains functional after the removal of arbitrary peers. However, there is another logic working on top of the topology: Pure P2P networks use simple broadcasts to retrieve content items. Instead, structured P2P networks introduce identifiers for peers. The identifiers enable a peer to address one particular other peer and route a message to it. This is sometimes referred to as "Key-Based Routing" [44]. The routing tables of the peers of structured P2P networks are considerably larger than the list of three to five other nodes that peers of pure P2P networks are connected to, which increases the performance of routing processes.

Most of the currently used structured P2P networks are networks for applications that store information using a Distributed Hash Table (DHT). Filesharing applications or telephony applications that store information about the current location of a user are typical examples. In the literature, the terms "structured P2P network" and "distributed hash table" are often used interchangeably.

2.3.1 Distributed Hash Table Algorithms

The most cited DHT algorithms are Chord [13], Pastry [36] and Kademlia [21]. Out of these three, Kademlia is the only algorithm that is known to be implemented in applications that are used by millions of concurrent users; the underlying algorithm of other popular P2P applications such as Skype cannot be determined because they are closed-source.

2.3.1.1 Hash Functions

Hash functions are used to derive a fixed-length hash value from an input value with arbitrary finite length. Common hash algorithms include MD5 [45] and SHA-1 [46], which use 128-bit and 160-bit hash values, respectively. The hash value can be used to identify or verify the input value – it is a "fingerprint" of the input value and shall be usable as if it were as unique as the input value [47].

Menezes et al. state that *"[t]o be of cryptographic use, a hash function* h *is typically chosen such that it is computationally infeasible to find two distinct inputs which hash to a common value (i.e., two colliding inputs* x *and* y *such that* h(x) = h(y)*), and that given a specific hash-value* y*, it is computationally infeasible to find an input (pre-image)* x *such that* h(x) = y.*"* [47] Therefore, in order to find the input value to a given hash value, only a brute force attack can be used, because hash functions cannot be reversed.

2.3.1.2 Key-Value Storage

A hash table stores key-value pairs: A value is stored under a key that is derived using a hash function. The value can be retrieved using the key under which it has been stored. Distributed hash tables work in the same way, but divide the hash table into multiple parts stored at different peers [24]. The value stored in the DHT can be a file or location information, for instance. In this thesis, the terms "content item" or "item" are used for the value of the key-value pairs that are stored in the DHT.

2.3.1.3 Identifiers for Peers and Content Items

In addition to the peer identifiers mentioned above, DHTs introduce identifiers for the content items that are used as the keys under which the content items are stored. DHTs also define how to calculate the distance between two identifiers. The peers whose IDs are closest to a content item's ID are responsible for this item. In theory, a single responsible node would suffice; however, in order to cope with nodes leaving the network, most DHT algorithms use several peers for replication [24].

Peers and content items share the same identifier address space, usually denoted as integers from 0 to 2^n-1. The same identifier can be assigned to both a peer and a content item. Based on the length of the hash values generated by widely used hash functions such as MD5 and SHA-1, common values for *n* are 128 and 160, resulting in 2^{128} or 2^{160} possible identifiers [24]. Due to this vast address space and the fact that hash functions are designed to avoid collisions of hash values, it is assumed that no two peers share the same identifier. Two peers with the same ID can still be distinguished by their IP address and port, so two peers having the same ID is not a problem. However, as hashing is usually used to determine the ID of a content item, a collision resolution method has to be defined for the rare case of a hash collision in order to still be able to differentiate two distinct content items having the same hash value.

2.3.1.4 Terminology

A "*common API for structured peer-to-peer overlays*" [44] is introduced by Dabek et al. in [44]. Accordingly, the process of identifying the nodes that are responsible for a certain content item and routing a message to them is called "Lookup". Storing a content item is called "Put", retrieving it is called "Get". The process of joining a network is referred to as "Bootstrap". These terms are used in this thesis for the respective procedures as well.

2.3.2 Distributed Hash Table Layers

Dabek et al. [44] define layers of DHTs and the basic interfaces between them. Figure 6 shows these layers: The lowermost layer is the underlay network layer. As already explained in 1.2, it is used for message transmission and reception and can consist of any arbitrary kind of network; however, in most cases, it will be a TCP/IP network such as the Internet. The authors differentiate the overlay layer into Key-Based Routing (KBR) and the Distributed Hash Table (DHT). OverSim [39], the simulation framework that is used in this

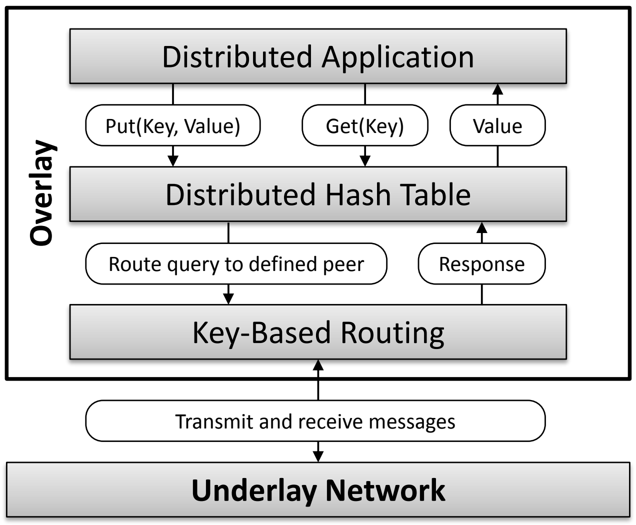


Figure 6: DHT layer model (based on [24])

thesis for the evaluation, reflects this in its class hierarchy. As shown in the figure, the KBR layer is used to route a message to the correct destination node(s). The P2P algorithm (e.g., Kademlia) defines how they are identified. The DHT functionality operates on top of the KBR layer and defines the storage procedures for the content items, e.g., number of replicas, lifetime of content items, etc. As explained before, the application interacts with the DHT by sending Put(key, value) messages to store content items and Get(key) messages to retrieve them. As TrustedKad is designed to be application-independent, it operates on the KBR and DHT layers only and does not use application-specific information to determine a node's trust value.

The layer structure shows that Put and Get processes can be divided into two parts: The first part is the Lookup operation that identifies the responsible node(s). Afterwards, the actual Put or Get operation is performed with the node(s) identified by the Lookup. There are several ways of performing Lookups and storage operations that are described in the following two sections.

2.3.3 Iterative vs. Recursive Routing

There are two basic routing types in P2P networks that can be used to perform a Lookup operation: Recursive routing and iterative routing, with recursive routing having several subtypes. If iterative routing is used, a node queries one other node for nodes that are closer to the target. It evaluates the response and repeats querying until it has found its target. The querying node has the full control over which node is queried next. It is also possible to send out more than one query at a time to accelerate the process [48]. Figure 7 shows an example for iterative routing: Node A queries node B. B informs A that C is closer to the target than itself. A decides to query C, which informs that D is closer to the target. A queries D which informs A that there are no closer nodes to the target than D, and the Lookup ends.

If recursive routing is used, the querying node sends its request to one other node which in turn forwards the query to another, closer node. The query is forwarded node by node until the target is reached. The querying node has no control over the next hops. The different kinds of recursive routing differ in the way in which the reply is routed: The target node can either contact the querying node directly (as shown in Figure 7), send the reply along the path the query has taken before or route the message [48].

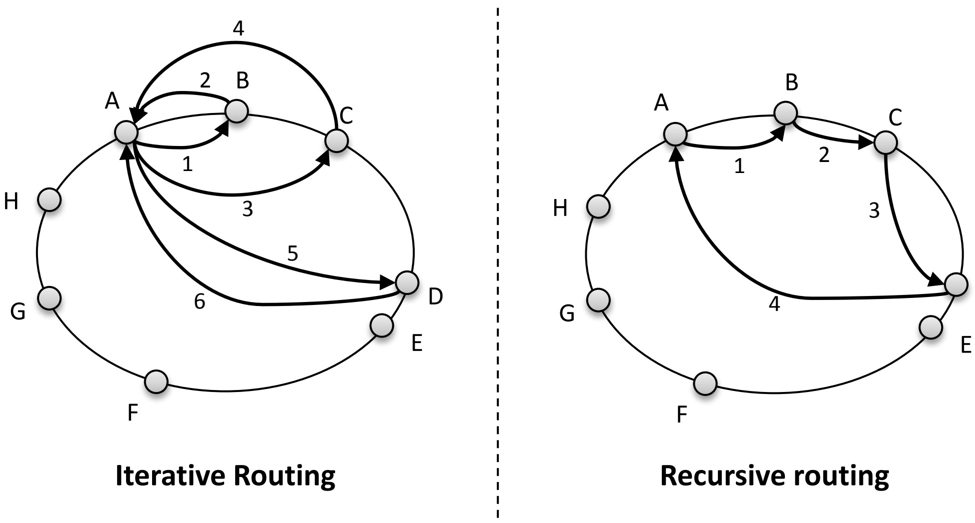


Figure 7: Iterative and recursive routing

Only iterative routing can be used for trust-based routing because a node must be able to control the choice of the next hop. TrustedKad uses Kademlia as a basis, which also uses iterative routing.

2.3.4 Direct vs. Indirect Storage

When the responsible node(s) have been identified, the Put or Get operation can take place. A DHT can store data in two possible ways: Directly or indirectly. Direct storage means that the content item is stored at the nodes that are responsible for the key. This is feasible only with small content items that do not require a lot of storage space and transmission capacity, because the content items have to be transferred to all nodes that store a replica of the item [24].

In order to optimize the content retrieval when direct storage is used, the Get process is split in two phases: In the first phase, all responsible nodes are queried for the hash of the content item's value. Based on the hashes, one node is chosen to be queried for the actual value (e.g., by using a majority vote). In the second phase, only the selected node is asked for the value of the content item. This two-phase approach significantly reduces traffic, but also opens new attack possibilities that are analyzed in Chapter 6.

Indirect storage only stores pointers to the actual content item at the responsible nodes. This way, the content item does not need to be transferred to all replica nodes. Indirect storage requires an additional "hop" for content retrieval, as the responsible nodes need to be queried first, and the nodes that actually store the item can be contacted only afterwards. Filesharing applications use indirect storage, as the shared files can be very large [24].

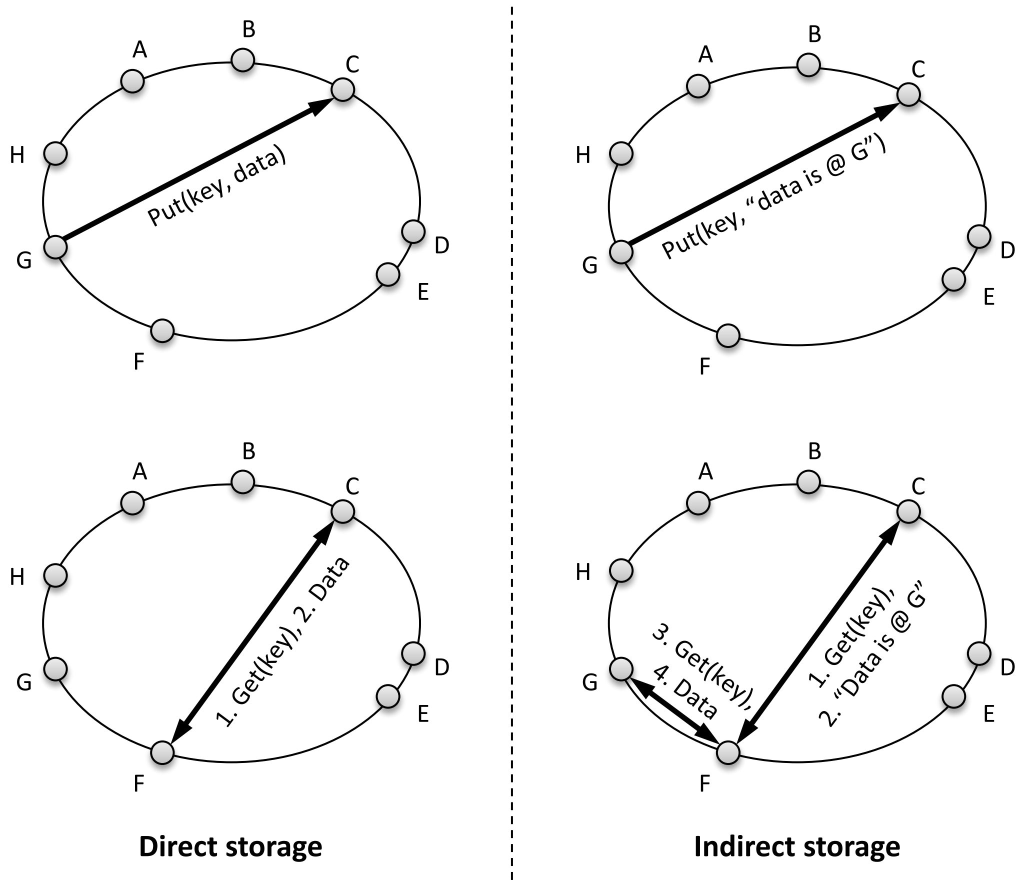


Figure 8: Direct and indirect storage (based on [24])

Figure 8 shows the difference: Node G wants to publish a content item for which node C is responsible. With direct storage, the content item is stored at C. With indirect storage, G informs C that it stores the content item. When node F tries to retrieve the content item with direct storage in place, it can retrieve it from node C and the process ends. With indirect storage, F only learns that the content item is stored at G. F then needs to ask G for the actual data [24].

2.4 Kademlia

In this section, the Kademlia algorithm is presented as it has been designed by the authors Maymounkov and Mazières. Only the relevant parts are described in this thesis; several details (e.g., splitting of *k*-buckets in order to cope with highly unbalanced trees) are left out and can be found in [21].

The author of this thesis has analyzed the functioning of the eMule Kad implementation in his diploma thesis [49] by reviewing Kad's source code. The relevant differences between the original paper and the Kad implementation are also explained in this section to show the adaptations and settings required for using Kademlia in a filesharing network with millions of concurrent users.

2.4.1 Basics

Kademlia uses 160-bit integers for node and content IDs. The node IDs can be chosen freely both in Kademlia and in Kad. As Kad has been designed as a structured successor of the unstructured filesharing network eDonkey, it uses 128-bit integers as identifiers to preserve compatibility with the 128-bit-long MD4 file hashes of the eDonkey network. Because Kad is a filesharing network, it uses indirect storage, so only pointers (called source information) are stored at the responsible nodes. As with all DHTs, content items are stored on nodes whose IDs are close to the content items' IDs.

Kademlia and Kad use $\alpha = 3$ parallel queries to accelerate the Lookup process and to cope with node failures. In addition, the way the nodes are stored in a node's routing table and removed from it is designed to be resistant to certain denial of service (DoS) attacks (as explained in Section 2.4.3) [21]. So the basic Kademlia algorithm already includes several security measures to counter simple attacks.

Maymounkov and Maziéres have identified several weaknesses of previously presented DHT algorithms such as Chord [13] and Pastry [36]. The authors aim to minimize the number of configuration and maintenance messages by using a symmetric distance metric for the identifiers and by including the sender's node ID in every message. This enables a Kademlia node to use incoming messages from other nodes to gain information for its routing table for subsequent queries of its own. This is not possible in Chord, as its routing scheme is asymmetric, so nodes "*[…] do not learn useful routing information from queries they receive*" [21].

Pastry uses two different distance metrics, one at the beginning of a Lookup, and another one for the last hops in the vicinity of the target. They are not compatible to each other, so nodes that are close to each other by one metric can be farer away by the other. Kademlia only uses one metric for all next hop decisions [21].

2.4.2 XOR Metric

The distance of two IDs is calculated using the bitwise XOR (eXclusive OR, $\oplus$) operation: $d(x, y) = x \oplus y$. The result of the XOR operation is interpreted as an integer – the smaller its value, the closer are the IDs. The operands can be swapped without changing the result, and the result is always greater than or equal to 0. XOR is unidirectional, which

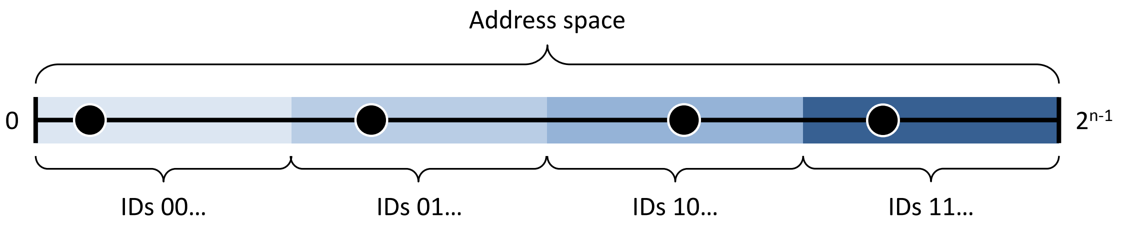


Figure 9: Address space with four uniformly distributed nodes and their responsibility areas

"*[…] ensures that all lookups for the same key converge along the same path, regardless of the originating node*" [21].

From the interpretation of the XOR result as an integer, it follows that the more bits at the beginning of the two 160-bit-long IDs are identical, the closer the IDs are: If the first bit of one ID is not equal to the first bit of the other ID, the distance amounts to at least 2^{159} if all other bits are equal. If the first bits are equal and the second bits differ, the minimum distance is 2^{158} and the maximum distance is 2^{159}-1, if all bits except the first one differ:

$$\begin{aligned} d(0\ldots,1\ldots) &= 0\ldots \oplus 1\ldots \\ &= 1\ldots \\ &= 2^{159} + \mathrm{x}, 0 \le \mathrm{x} \le 2^{159} - 1 \end{aligned}$$

$$\begin{aligned} d(00\ldots,01\ldots) &= 00\ldots \oplus 01\ldots \\ &= 01\ldots \\ &= 2^{158} + \mathrm{x}, 0 \le \mathrm{x} \le 2^{158} - 1 \end{aligned}$$

So the first *n* bits of a node ID should be equal to the first *n* bits of the IDs of the content items it is responsible for, with *n* depending on the size of the network. In his diploma thesis [49], the author of this thesis has found out that in the Kad network with an estimated size of about four million concurrent nodes at that time, at least the first 13 and up to the first 25 bits of node ID and content item ID have been identical. This matches the estimated network size: Assuming a uniform distribution of the node IDs, the first $\log_2(4{,}000{,}000) \approx 22$ bits are relevant for the distance between a node ID and a content item ID and therefore also for the determination of the responsible nodes of a content item.

Figure 9 shows this graphically for a network consisting of four uniformly distributed nodes (resulting in $\log_2(4) = 2$ relevant bits): It is assumed that each of the four possible combinations of the first two bits is used by one node's ID. So the responsibility for one quarter of the address space is assigned to one node based only on the first two bits of its ID; the other bits are irrelevant because no other closer node exists.

2.4.3 Routing Table Structure

The routing table of a Kademlia node consists of so-called *k*-buckets. The variable *k* denotes how many nodes can be stored in one bucket. The Kademlia paper gives *k* = 20 as an example, Kad uses *k* = 10. Each bucket stores nodes that have a certain distance from the node, as there is one bucket per bit of the node ID: A bucket contains the nodes that share the same prefix as the node itself, beginning with 0 shared prefix bits up to *n*-1 shared prefix bits. So, for example, the first bucket contains *k* nodes that do not have the first bit of their identifier set like the node itself (e.g., if the node's identifier starts with a

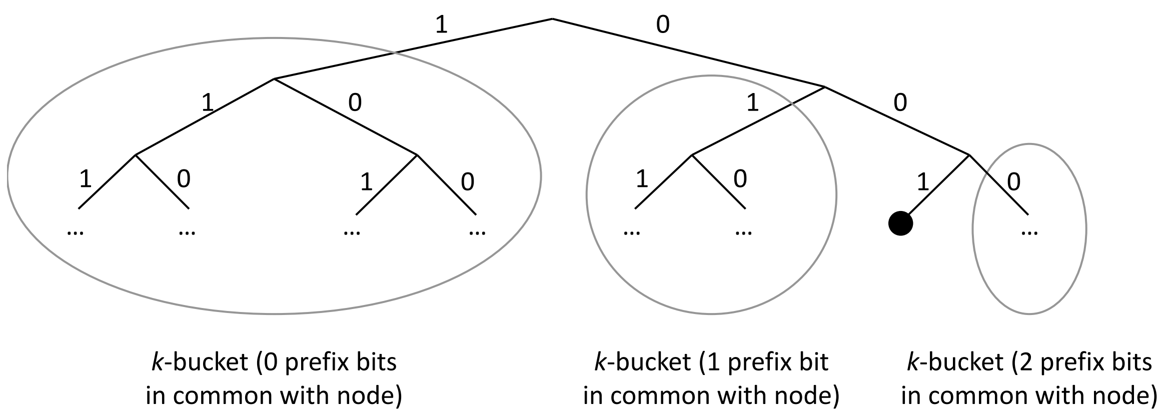

1
0
1
0
1
0
1
0
1
0
1
0
1
0
…
k-bucket (0 prefix bits in common with node)
k-bucket (1 prefix bit in common with node)
k-bucket (2 prefix bits in common with node)

Figure 10: The first three *k*-buckets of node 001... [21]

1-bit, the other nodes' identifiers start with a 0-bit). The second bucket contains *k* nodes that have set the first bit like the node itself, but the second bit differs, etc.

Figure 10 shows a binary tree representation of the identifier address space. It shows three *k*-buckets of the node whose ID begins with 001. The buckets contain other nodes that have 0, 1 and 2 prefix bits in common with the node, respectively.

This structure of the routing table leads to the fact that a node knows many nodes close to itself and fewer nodes, the greater the distance to them becomes. The first bucket with no shared prefix bits contains up to *k* nodes that are in the other half of the tree. So the first routing hops cross comparatively large distances, while the last hops are small.

The *k*-buckets are sorted by the time the nodes have been seen for the last time, with the most-recently seen node at the tail and the least-recently seen node at the top. When a message is received from a node, the *k*-bucket it is stored in is updated according to this rule. If the node is new, it is inserted into the appropriate *k*-bucket if there are less than *k* nodes in it. If the *k*-bucket is full, the least-recently seen node contained in it is pinged. If

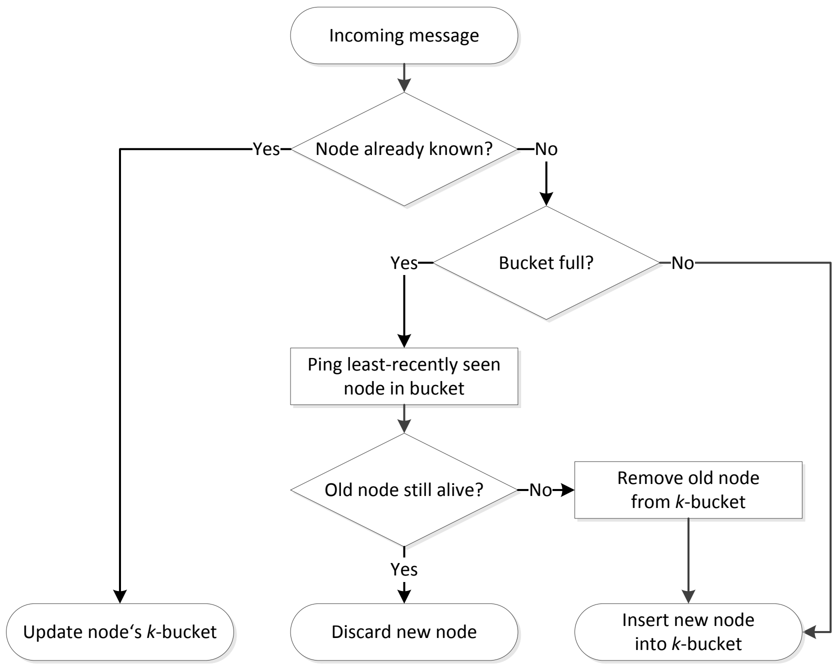


Figure 11: Flowchart of the handling of an incoming message (based on [50])

it fails to respond, it is removed from the *k*-bucket and replaced by the new node. If it replies, the new node is discarded. This procedure creates the aforementioned DoS protection, as only nodes that are not reachable any longer will be removed from the routing table. Malicious nodes cannot replace nodes that still reply to queries. Figure 11 shows this procedure graphically.

In order to reduce the number of Ping messages, the authors propose to use a replacement cache. It shall contain nodes that can replace nodes which are currently in a full *k*-bucket. When the node attempts to contact a node in a full *k*-bucket and that node does not respond, a node is taken from the replacement cache to replace the non-responding node. This way, separate Ping messages are no longer required [21].

2.4.4 Message Types and Content Storage and Retrieval Procedures

In [21], Maymounkov and Mazières define four basic remote procedure calls (RPCs): Ping, Store, Find_Node and Find_Value. Ping is used to determine whether a node is still online. Find_Node queries another node for nodes closer to the target. The response contains a list of up to *k* nodes that are the closest nodes to the target ID the responding node knows about. The Store RPC stores a key-value pair in the network. In order to find the nodes on which to store the key-value pair, Store first performs a Lookup using the Find_Node RPC repeatedly (as described in Section 2.4.6). The Store RPC is then sent to the *k* closest nodes.

To retrieve a content item, the original Kademlia algorithm defines a Find_Value message type. It works in the same way as the Find_Node RPC, as it expects another node to respond with up to *k* nodes that are closer to the target. The difference from the Find_Node message is that if a node stores the desired content item, it shall reply with the value assigned to the target key instead of the node list. So Kademlia only retrieves one replica of the content item [21].

This is not optimal in filesharing networks and from a security point of view: Querying only one node for the replica poses a great opportunity for a malicious node to simply reply with a fake content item. The querying node is unable to detect this because it terminates its query as soon as one replica is retrieved.

Kad does not implement Find_Value in this way, because in filesharing networks, all possible sources of a content item shall be found. Therefore, the *k* closest nodes to the target are all queried. As for the Store RPC, Kad uses the Find_Node RPC to locate these nodes.

2.4.5 Bootstrap Procedure

For joining the network, a node needs to know another node that is already part of the network. The original Kademlia protocol specifies that the joining node performs a Lookup for its own node ID to fill its routing table and to inform other nodes about its existence [21]. Kad uses a special Bootstrap message that queries another node for information about other nodes. The response to this message contains up to 20 nodes [49].

2.4.6 Lookup Procedure

Kademlia uses an iterative Lookup procedure by which a node has full control over the choice of the next hop. Figure 12 shows how a lookup is performed: A node begins a Lookup by choosing the α closest nodes from the *k*-bucket that is closest to the target ID. If that bucket does not contain enough nodes, neighboring buckets are also taken as a source. The node sends α parallel Find_Node messages to the α chosen nodes. The nodes contained in the response messages are inserted into a temporary Lookup list which is

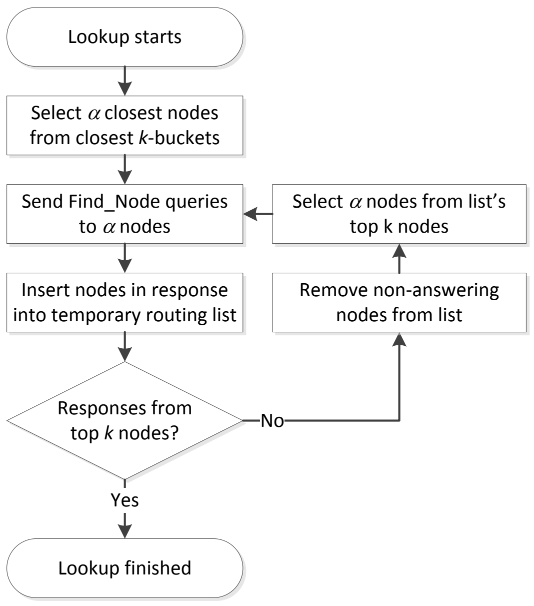

Lookup starts
Select α closest nodes from closest *k*-buckets
Send Find_Node queries to α nodes
Select α nodes from list's top k nodes
Insert nodes in response into temporary routing list
Remove non-answering nodes from list
Responses from top *k* nodes?
No
Yes
Lookup finished

Figure 12: Flowchart of a Kademlia Lookup procedure

kept sorted in a way that the node closest to the target is at the top of the list. The node continues to send α parallel Find_Node queries to the top *k* nodes of the temporary Lookup list until these top *k* nodes have all been queried and have not returned any more closer nodes [21]. In addition, the nodes contained in the response message are inserted into the appropriate buckets of the routing table if possible.

2.5 Security Challenges of Structured Peer-to-Peer Systems and Trust Systems

Research has shown that DHT algorithms are vulnerable to various attacks. Urdaneta et al. state in [32] that the main attack types against DHTs are the Sybil Attack [31], eclipse attacks and attacks on the routing and storage mechanisms. They are briefly introduced in this section. Trust and reputation techniques are also prone to several basic attacks that are explained in this section as well. The measures with which TrustedKad counters the attacks are explained in Chapter 4.

2.5.1 Sybil Attack

Douceur has coined the Sybil attack in [31]. It describes an attack in which one attacker creates several logical identities. The name "Sybil attack" is derived from the main character Sybil of the book "Sybil" [51] who suffers from schizophrenia.

The Sybil attack interferes with the redundancy mechanisms of DHT algorithms: If the replicas that are supposed to be stored on different physical entities are instead stored on the same physical entity that has several logical identities, the redundancy is lost. Furthermore, the attacker can control parts of the DHT using several logical identities, either for monitoring purposes only or in order to perform an Eclipse attack.

Urdaneta et al. present and discuss several countermeasures against the Sybil attack [32]. However, together with Douceur, they conclude that a central authority is the only reasonable countermeasure against the attack. Depending on the actual environment of a P2P system, other countermeasures may also be reasonable: If it can be assumed that all peers have about the same computing power, crypto puzzles can pose an option [52]. Crypto puzzles require a node to (usually repeatedly) solve cryptographic problems, e.g., finding random numbers that result in a hash value with the first *n* bits set to zero. This way, the number of fake identities that can be created by an attacker shall be limited. A more detailed description of crypto puzzles can be found in Section 3.1.

2.5.2 Eclipse Attack

An attack during which the attacker attempts to control a part of the network is called "Eclipse attack". The Sybil attack can be used to perform an Eclipse attack; however, this is not required, because the attacker could also take over other physical entities for its purposes (e.g., by hacking them). According to Urdaneta et al., the attack is also known as "routing table poisoning" because the attacker attempts to place its own nodes into other nodes' routing tables. The goal is to control all messages sent to and from a set of nodes. The Eclipse attack can be used to prepare routing and storage attacks [32].

In order to counter the Eclipse attack, the node identifiers must not be freely selectable. If they were, attackers could choose the region of the ID space they want to control and can concentrate (Sybil or hacked legitimate) nodes there. In addition, the routing table of a node should not be easy to influence, which is the case for Kademlia: As described in 2.4.3, nodes are only removed from the routing table if they are not responding to queries any longer. So, poisoning a node's routing table with fake nodes is difficult [53].

2.5.3 Attacks on Routing and Storage

Attacks on routing and storage can be performed in a variety of ways. Urdaneta et al. state that these "*[…] attacks may attempt to prevent a lookup request from being successful. For example, an attacker may refuse to forward a lookup request. It may forward it to an incorrect, non-existing, or malicious node. Finally, it may pretend to be the node responsible for the key. Another possibility for an attacker is to route requests correctly, but deny the existence of a valid key or to provide invalid data as a response. These attacks are generally classified as routing attacks, which try to disrupt routing, and storage attacks, which attempt to provide bogus responses to queries*" [32]. In addition to these attacks, malicious nodes may attempt to overwrite existing content items with fake content.

The common defense against such attacks is redundancy: Several replicas of content items are stored in the network, and several paths are used for routing [32]. Kademlia already performs both, but it is still susceptible to these attacks, so additional countermeasures have to be used. A digital signature of content items by the storing node, combined with a check that subsequent changes of the content item originate from the same node, counters attacks that attempt to change previously stored content items.

The simulation framework OverSim that is used for evaluation provides several routing and storage attack types which have been extended for the purposes of this thesis. The detailed descriptions of the attacks and their extensions follow in Chapter 5.

2.5.3.1 Network Partitioning

Partitioning of a network means that there is no longer one single network, but several of them that exist concurrently without interconnections. It is the result of a successful attack on the Bootstrap process of a node. If the Bootstrap is attacked, the malicious node can control the bootstrapping node's view of the network.

Figure 13 shows six network partitions created by malicious nodes: Inoffensive nodes that bootstrap from another inoffensive node stay in the same partition (as shown in the green, orange and purple partitions). Malicious nodes form a border between partitions because they do not return information about other existing nodes, but only about non-

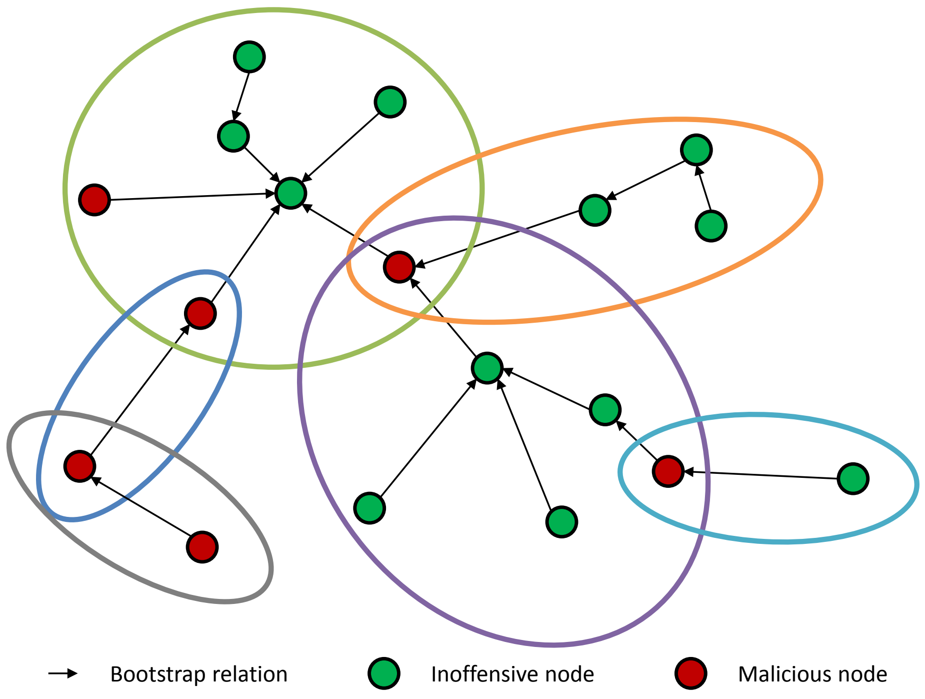


Figure 13: Schema of several network partitions

existing fake nodes. It is possible that partitions consist of only two malicious nodes if one malicious node has bootstrapped from another malicious node, as shown in the dark blue and grey partitions.

If a content item is published in one partition, the other partitions cannot find it due to the missing links between the partitions. Even if a node in another partition knew the content item's ID, e.g., from a directory containing valid content item IDs, the attempt to retrieve it would fail. The impact of network partitioning is analyzed in detail in Chapter 6.

In reality, a malicious node probably attacks both routing and storage operations. In the simulations conducted for this thesis, however, there are configurations in which malicious nodes only attack routing operations. Under such circumstances, content items may be available in more than one partition. This is explained in Section 6.3.1.1.

2.5.4 Attacks on Trust Systems

Just as DHTs have got inherent weaknesses, so do trust systems. Hoffman et al. identify five different types of such attacks in [54]: Self-promoting, whitewashing, slandering, orchestrated attacks and denial of service attacks.

2.5.4.1 Self-Promoting Attack

Self-promoting means that an attacker attempts to increase its own reputation by giving itself positive ratings, e.g., by the means of Sybil nodes. Another way is to manipulate its own reputation data: If it manages this information itself and in a non-authenticated way, it is possible to modify the information. If the trust system does not require a proof of interaction, the self-promoting attack is facilitated. Trust systems that only use positive ratings are especially vulnerable to this attack.

The attack is usually performed by colluding nodes that generate interactions among themselves and give each other positive ratings for the interactions. This way, they could even succeed in countering negative ratings they receive from legitimate nodes [54].

2.5.4.2 Whitewashing Attack

Whitewashing means that a node that has received too many negative ratings leaves the network and returns with a new and non-rated identifier to dispose of its negative ratings. If identifiers can be changed easily, an attacker can create a new identity when the reputation of its old identity is too bad. Trust systems that only use negative ratings are vulnerable to whitewashing because newcomer nodes have the same reputation as nodes that participate in the network for a long time and have behaved well.

Another form of the whitewashing attack is a changing behavior of the node: The attacker behaves correctly first to receive positive ratings and then abuses its high reputation to act maliciously. Nodes that behave this way are called traitor nodes.

In order to make the whitewashing attack more effective, it can be combined with the self-promoting attack: If performed concurrently, the effective duration of the attacks will increase. The whitewashing attack can be countered by limiting users from switching identities, e.g., with a Sybil attack countermeasure, and by giving more weight to more recent ratings to counter traitor nodes [54].

2.5.4.3 Slandering Attack

Slandering is the opposite of self-promoting: Malicious nodes give negative ratings to inoffensive nodes in order to decrease their reputation so that their own reputation increases relative to them. Just as the self-promoting attack, the slandering attack is easily conducted if the system does not authenticate the ratings, and it is usually performed with several colluding nodes.

The countermeasures against traitor nodes and the slandering attack interfere with each other: In order to counter traitor nodes, more recent ratings should receive more weight. However, this renders a slandering attack more effective, so a balance has to be found. Limiting the amount of identifiers one physical attacker can obtain impedes the slandering attack [54].

2.5.4.4 Orchestrated Attack

In an orchestrated attack, the attacker combines several of the aforementioned attacks in a coordinated manner. One possible example is the oscillation attack [55]: The colluding nodes are divided into several groups that perform different attacks at different times, e.g., one group behaves correctly and another group acts maliciously. So the relative reputation of the inoffensive group increases. If at the same time the inoffensive group gives positive ratings to the malicious group, the decline of their reputation is slowed down. When the reputation of the malicious group nodes is too low, the groups swap their behavior to balance the reputation values. Another group that uses the slandering attack to decrease the reputation values of legitimate nodes could also be used together with the two other groups [54].

Just as with the attacks explained before, limiting the number of logical identities a physical attacker can obtain impedes also orchestrated attacks because in order to be effective, they require a large amount of nodes that are controlled by one attacker.

2.5.4.5 Denial of Service Attack

A denial of service attack is typically conducted against centralized elements if the trust system uses any. If the ratings are stored centrally, an attack against this central element can disable the trust system. Then, if the routing is implemented in way that all nodes are regarded as trustworthy if the trust storage does not respond in time or at all, malicious nodes can be used as well. Otherwise, if all nodes would be regarded as non-trustworthy if the trust storage is not available, the network would cease functioning [54].

The fewer central elements exist in a network, the fewer attack targets it presents for a denial of service attack, so content items and ratings should be stored in a distributed way. A denial of service attack that targets any arbitrary node of a P2P network cannot be averted by the P2P network because it can only be handled by the underlay (if at all). But if there are several replicas available, the attack needs to be distributed as well and therefore requires even more resources.

3 Related Work

This chapter presents the current state of research of security and trust systems in P2P networks. As a representative for security systems that do not use trust mechanisms, S/Kademlia [53] is chosen, as it is specifically targeting to improve Kademlia. A lot of the existing work regarding trust systems addresses unstructured networks or mentions structured networks only for the storage of the trust information. As an example, a trust system designed by Mekouar [56] that operates in an unstructured hybrid P2P network is presented in detail. Considering widely used P2P applications, however, the structured systems prevail, so the applicability of Mekouar's concept to structured networks is discussed. Afterwards, several often-cited trust-based approaches are briefly presented. A short excursus about mobile ad hoc networks (MANETs) shows that trust-based routing is also analyzed in related network types. Both MANETs and peer-to-peer networks are dynamic networks lacking a central authority and can therefore share approaches to enhance their security. At the end of this chapter, the previously published papers of the author of this thesis are introduced.

S/Kademlia and Mekouar's concept have been chosen because they are both described comprehensively in the dissertation theses of their respective authors. Both concepts have been published at peer-reviewed conferences before as well.

3.1 S/Kademlia

Baumgart and Mies present S/Kademlia in [53]; Baumgart adds more details in his dissertation thesis [48]. S/Kademlia aims to improve Kademlia's security by countering several known attacks described in the previous chapter: The authors propose to use a secure identifier generation process to impede Sybil and Eclipse attacks. Additionally, other nodes shall not be able to steal a node's node ID. To achieve this, S/Kademlia uses asymmetric key pairs: Using digital signatures, every message is authenticated, proving the possession of the matching private key. The node ID consists of the hash value of the public key.

In order to counter the Sybil and Eclipse attacks, the authors propose to use either a central certification authority to sign a node's public key to prove its identity or to use crypto puzzles. For the latter variant, they introduce two types of crypto puzzles: A static one that denies the free generation of the node ID and a dynamic one that increases the difficulty of creating a large amount of Sybil nodes.

The node ID is generated as the hash of the public key of a node. This way, a node cannot choose its node ID freely. However, by using brute force, it is still possible to generate a node ID that is in a desired range. The static crypto puzzle is used to increase the difficulty of acquiring such a node ID: It calculates a hash of the node ID and demands that it begins with c_1 bits that are set to zero. If the node ID, and with it also the public key, does not fulfill this requirement, a new key pair has to be generated until the requirement is met.

The authors of S/Kademlia assume that the available computing power of a node increases over time so that the static crypto puzzle is not difficult enough at some moment in time. Therefore, they introduce a second, dynamic crypto puzzle. It uses a parameter c_2 which can be increased over time to cope with increasing computing power. It requires a node to generate a random number X and to calculate the hash of NodeID $\oplus$ X. If the first c_2 bits of that hash are set to zero, the puzzle is solved; if not, the process is repeated with another random number. Figure 14 shows the functioning of the two crypto puzzles. The puzzles can be verified with O(1) complexity, but the solving of the crypto puzzles has $O(2^{c_1} + 2^{c_2})$ complexity [53].

Additionally, S/Kademlia uses several disjoint routing paths during a lookup. Each routing path uses α parallel queries, so the messaging overhead is increased. The authors state that this way, a lookup is successful as long as at least one neighbor node is not malicious.

Furthermore, the authors introduce a sibling table. Every node manages one sibling table that contains the s other nodes that are closest to it. For the evaluation of S/Kademlia, $s = 16$ has been used. The sibling table is used to accelerate the final phase of a lookup so that fewer nodes need to be queried for closer nodes [53].

As mentioned before, the usage of crypto puzzles is controversial: If the P2P network operates in a homogeneous environment regarding computing power, crypto puzzles can be a countermeasure against the Sybil attack. If the computing power of the nodes differs by several orders of magnitude, though, another solution is required. Because TrustedKad is designed as a generic framework, it cannot assume homogeneity of the nodes, so it supports using different types of Sybil attack countermeasures (introduced later in Sec-

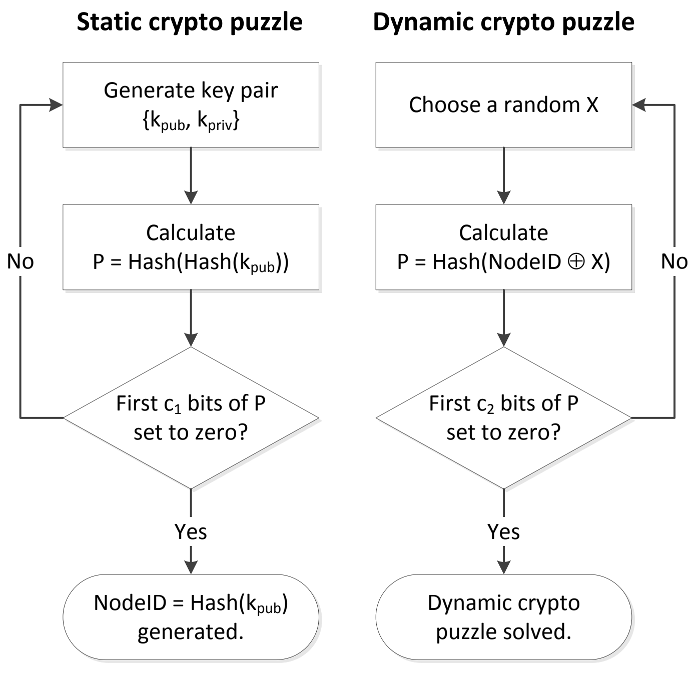

Static crypto puzzle
Dynamic crypto puzzle
Generate key pair {k_pub, k_priv}
Calculate P = Hash(Hash(k_pub))
First c1 bits of P set to zero?
No
Yes
NodeID = Hash(k_pub) generated.
Choose a random X
Calculate P = Hash(NodeID ⊕ X)
First c2 bits of P set to zero?
No
Yes
Dynamic crypto puzzle solved.

Figure 14: Static and dynamic crypto puzzles of S/Kademlia [53]

tion 4.4).

3.2 Trust Systems for Peer-to-Peer Networks

Research of trust mechanisms and P2P networks has taken place for almost a decade now. Marti and Garcia-Molina give an overview of existing trust systems for P2P systems in [57] and analyze their components: They divide trust systems into three functionalities which they call “information gathering”, “scoring and ranking” and “response”. For the first one, “information gathering”, the identification of a node is essential. Based on this, information about a node can be shared between nodes. The definition of a policy how to handle unknown nodes is the functionality’s last part.

“Scoring and ranking” defines the way ratings are given, collected, combined and evaluated. The results lead to the selection of the peers that are interacted with and which ones are avoided. The third functionality “response” defines how to reward inoffensive nodes and how to punish malicious ones. In their discussion of these functionalities, the authors give hints on how to secure their operation and how to reward inoffensive nodes and to punish malicious ones.

Gómez Mármol and Martínez Pérez give an overview of several trust and reputation systems for P2P networks in [58]. They identify three basic scenarios which trust and reputation models should consider: The first is a static one in which there is no churn. It is considered as a starting point, but unrealistic. The second is a dynamic one which includes churn. The authors call the third one “oscillating”, which means that the nodes can change their behavior over time. The model should be able to quickly cope with that as well. This scenario is the hardest to detect and therefore appears to be the most promising one for an attacker. However, TrustedKad successfully counters also malicious nodes that change their behavior; the results are presented in Chapter 6.

3.2.1 Example of a Trust System for a Peer-to-Peer Filesharing Network

In order to give an example of a P2P trust system, the system designed by Mekouar and explained in detail in her dissertation thesis [56] shall be presented. Before, Mekouar has published several frequently-cited papers, journal articles and book chapters on trust and reputation in P2P systems (such as [59,60]). Her concept targets a hybrid P2P filesharing network, so the network consists of supernodes and leafnodes. Afterwards, other P2P trust systems are outlined.

In her thesis, Mekouar presents a P2P trust system that operates in a hybrid filesharing system. She uses three different indicators to rate a node’s trustworthiness: Authentic behavior, credibility behavior and contribution behavior. The authentic behavior value rates the validity of the content item the node sends when queried. The credibility behavior value denotes how reliable the ratings of a node are. The contribution behavior value is specific to filesharing, as it measures how well a node contributes to the system by calculating the ratio of its up- and downloads.

The supernodes manage the ratings. So after every transaction, a node sends the rating of that transaction to its supernode. This way, the supernode gains a behavior overview of all of its leafnodes. In addition, the supernode forwards the rating to the supernode of the other involved node so that the values for that leafnode are updated as well. A supernode periodically transmits a leafnode’s trust information to the leafnode in a digitally signed form. This way, a leafnode does not lose all of its trust information if its supernode fails, but is able to transfer it to its new supernode.

3.2.1.1 Authentic Behavior

The author explains that not only a simple transaction counter, increased by one after each transaction, should be used for the authentic behavior value. Instead, the size of the transferred file or also other upload resource usage (e.g., processing, bandwidth, time) should be used by the supernode to evaluate its leafnodes:

```
if ($Download\ reported\ as\ successful$) then
  $D_i^+ = D_i^+ + (file\ size)$
else
  $D_i^- = D_i^- + (file\ size)$
end if
```

Mekouar explains that a simple difference-based approach to assess the behavior of a leafnode *i* is not sufficient because the difference $AuthB_i = D_i^+ - D_i^-$ would not consider the total amount of the transferred data; $AuthB_i$ denotes the authentic behavior value, $D_i^+ \geq 0$ the sum of the file sizes of successful downloads and $D_i^- \geq 0$ the sum of the file sizes of downloads resulting in invalid files. The author gives the following example of two nodes A and B with $D_A^+ = 40$, $D_A^- = 20$, $D_B^+ = 20$ and $D_B^- = 0$. If only the difference would be considered, both nodes would have the same authentic behavior value:

$$D_A^+ - D_A^- = 40 - 20 = 20 = 20 - 0 = D_B^+ - D_B^-$$

However, the two nodes should not be considered as equal because node B has not sent any invalid information while node A has done so. So node B should be preferred.

Therefore, the author proposes to also consider the sum of the values:

```
if ($D_i^+ + D_i^- \neq 0$) then
  $AuthB_i = \frac{D_i^+ - D_i^-}{D_i^+ + D_i^-}$
else
  $AuthB_i = 0$
end if
```

This results in authentic behavior values in the range of [-1; 1], with -1 denoting only unsuccessful downloads and 1 denoting only successful downloads. The authentic behavior values of the example nodes would be $AuthB_A = \frac{D_A^+ - D_A^-}{D_A^+ + D_A^-} = \frac{40 - 20}{40 + 20} = \frac{1}{3}$ and $AuthB_B = \frac{D_B^+ - D_B^-}{D_B^+ + D_B^-} = \frac{20 - 0}{20 + 0} = 1$, so node B would be preferred. TrustedKad follows this argument and uses a similar formula to calculate trust values (as explained in Section 4.6).

3.2.1.2 Credibility Behavior

The credibility behavior value denotes whether a peer lies about the outcome of a transaction, which means that the peer gives a negative rating although the result has been correct or vice versa. Mekouar proposes to detect such lying peers by comparing the rating with the expected rating. This comparison results in a ratio denoting how many ratings of a node were as expected compared to the total number of ratings ($CredB_i$ is the credibility behavior value, N_i is the total number of downloads, N_i^* is the number of downloads with an unexpected rating)[7]:

[7] Please note that Mekouar does not define $CredB_i$ for $N_i = 0$.

$$CredB_i = 1 - \frac{N_i^*}{N_i}$$

The author then changes the formula of the transaction counter that is used to calculate the authentic behavior value of a leafnode to incorporate the credibility of the node that gives the rating:

```
if ($Download\ reported\ as\ successful\ by\ node\ j$) then
  $D_i^+ = D_i^+ + CredB_j \times (file\ size)$
else
  $D_i^- = D_i^- + CredB_j \times (file\ size)$
end if
```

This way, the file size of the transferred file shall be multiplied with the ratio, so that the full file size is only taken into account if the rating node has never given an unexpected rating.

3.2.1.3 Contribution Behavior

The contribution behavior value rates how useful a node is for other nodes. According to Mekouar, it shall identify freeriders in a filesharing system. Freeriders are nodes that only download files, but do not upload any. The author argues that if only the reputation assessment is used, a freerider could first upload files to gain good ratings. When its reputation is good enough, it could stop uploading and use its good reputation for downloading only. So the concept aims to continuously reward uploading peers and to punish peers that cease uploading.

The author further states that not only peers that actually upload shall be rewarded, but also peers that are available for upload shall receive credit. This way, newcomer nodes shall be able to enter the system. Otherwise, the already existing nodes would receive all requests and the new nodes would not be able to gain reputation.

The availability rating compares a node's availability to the average availability of all nodes connected to the same supernode. *Available$_i$* is the number of times a node *i* was available for upload, *NbrPeers$_{sup}$* denotes the number of peers connected to supernode *sup*, *Average$_{sup}$* is the mean availability of all nodes that are connected to supernode *sup*. Then, the *Availability$_i$* of a peer *i* is calculated as follows[8]:

$$Average_{sup} = \frac{\sum_k Available_k}{NbrPeers_{sup}}$$

$$\text{if } (Average_{sup} > 0) \text{ then } Availability_i = \text{Min}\left(\frac{Available_i}{Average_{sup}}, 1\right)$$

This ensures that a node that is available for uploading receives credit for that.

The second part of the contribution behavior value is the involvement value. It is defined as follows with U_i^+ and U_i^- denoting the sums of the file sizes that have been uploaded correctly (+) and incorrectly (-):

```
if ($D_i^+ + D_i^- \neq 0$) then
  $Involvement_i = \frac{U_i^+ - U_i^-}{D_i^+ + D_i^-}$
else
  $Involvement_i = U_i^+ - U_i^-$
```

[8] Please note that Mekouar does not define Availability$_i$ for Average$_{sup}$ = 0.

```
    end if
    $Involvement_i = \mathrm{Min}(Involvement_i, 1)$
```

This way, the *Involvement_i* of a peer is the ratio between the peer's uploads and downloads. Freeriders have low values, as they do not upload a lot of data. Malicious nodes should even have negative values, as they upload more invalid data than valid data.

The contribution behavior value *$ContrB_i$* is computed based on the availability and the involvement of a peer:

```
$a = Availability_i$
if $(Involvement_i < 0)$ then
    $b = -1$
else
    $b = Involvement_i$
end if
$c = \mathrm{Max}(\mathrm{X} \times a + \mathrm{Y} \times b, 0)$
$ContrB_i = \mathrm{Min}(c, 1)$
```

The two factors *X* and *Y* are application-dependent and greater than or equal to 0. They give a weight to the two parts of the contribution value, availability and involvement.

3.2.1.4 Discussion of Applicability to Structured Networks

Mekouar's concept relies on two assumptions: It makes extensive use of filesharing-specific data and it uses the supernodes as trust management nodes. Both properties cannot be used to design a general-purpose trust mechanism operating in a structured P2P network: A P2P network such as Skype, for instance, that is used as a directory service to look up IP addresses of users to establish a direct connection cannot use up- and download ratios. Supernodes do not exist in structured P2P networks, so other means have to be found to manage the trust information.

However, the argumentation regarding difference-based calculation of the authentic behavior value still holds true: Not only the difference should be considered, but instead a ratio based on the total amount of data. As the usage of the file size or any other resources as an increment of behavior assessment is application-specific, only the number of transactions can be used by TrustedKad. Section 4.6 describes how TrustedKad computes trust values.

The supernodes are not specifically considered in [56]. They are assumed to be trustworthy, and their operation is not evaluated, which is a potential weakness. In hybrid networks, routing is performed via the supernodes. Due to the fact that they are regarded as trustworthy, the author does not specify a method to rate them for correct routing.

In structured networks, however, the routing is performed by all nodes, and it is both a basic and a vital functionality of the network. So the correctness of a node's routing behavior has to be rated and the ratings have to be considered during routing decisions.

Additionally, the supernodes are used to manage the trust information about the leafnodes. As there are no supernodes in a structured network, the trust information has to be stored in a distributed way.

3.2.2 Other Peer-to-Peer Trust Systems

As hinted by the overview publications mentioned above [57,58], there are several existing trust systems for P2P networks which are briefly introduced in the following.

EigenTrust [61] and FuzzyTrust [62] are two frequently cited trust algorithms. They work similar to the eBay rating system [30], as every transaction can be rated as either positive or negative. Both define methods of deriving trust information from these values. EigenTrust uses a distributed way to calculate a global trust value for each node with pre-trusted peers at the root, whereas in FuzzyTrust, every node has its own set of fuzzy rules to derive the trust value. Just as in Mekouar's concept, the trust value is based on the number of positive and negative experiences and has a value in the range [-1; 1].

There are two P2P trust systems that operate in P-Grid [63], a structured P2P network that has no wide-spread real-world implementations: Xiong and Liu present PeerTrust in [64]. PeerTrust uses a complex distributed scheme to calculate trust values. The other system is presented by Aberer and Despotovic in [65]. It is a complaint-based system that either regards a node as fully trustworthy or fully untrustworthy. The DHT is used to store trust information. In order to avoid infinite loops of lookups for trust information, it uses an upper limit of the search depth.

The system presented by Galuba et al. in [66] targets unstructured and structured P2P networks in which messages are routed recursively. It is proposed to be used not only by P2P networks, but also by wireless ad-hoc networks and sensor networks. These types of networks need to rely on recursive routing, while structured P2P networks usually do not: Kademlia uses iterative routing, for example. As explained before in Section 2.3.3, iterative routing enables a node to choose the next hop on its own instead of leaving the choice to other nodes.

Several approaches attempt to use information gained from social networks to form trusted groups of nodes. Examples for such systems are SPROUT [67] by Marti et al. and SocialTrust [68] by Li et al. Social-network-based approaches imply that each peer has got its own view of trusted and untrusted nodes, so there is no global trust. Personalized EigenTrust [69] extends EigenTrust (see above) by eliminating the global pre-trusted peers and replacing them by peers chosen individually by each peer based on social network connections.

In contrast to systems that use social networks, TrustMe [70] by Singh and Liu emphasizes the anonymity of the nodes. TrustMe aims to hide a node's trust management nodes from itself so that it cannot attempt to manipulate its trust value. The trust management nodes of a node are therefore randomly selected by a bootstrap server and not determined by an algorithm, and the trust values are queried by using broadcasts. Similarly, SuperTrust [71] by Dimitriou et al. aims to preserve the privacy and anonymity of transactions. It operates in a hybrid P2P network. In contrast to Mekouar's concept, the authors state that malicious superpeers can be identified by SuperTrust.

3.3 Excursus: Trust-Based Routing in Mobile Ad Hoc Networks

Research about trust-based routing is also taking place for other network types in which nodes communicate in a peer-to-peer way without a central authority. Mobile ad hoc networks (MANETs) are one example: They consist of nodes that are spread geographically and communicate wirelessly. The nodes are not necessarily stationary, but may also be moving. The network is usually spread in a way that does not allow every node to communicate directly with every other node as it is possible in P2P networks. Instead, if source and destination node cannot communicate directly, the messages have to be forwarded by intermediate nodes. Depending on the size of the network, there may be several paths between the source and the destination.

P2P networks and MANETs are subject to similar attacks, as both networks lack central authorities. Their routing operations can be harmed, e.g., by malicious nodes giving false routing information. In addition, the Sybil Attack can also be conducted in MANETs.

One current example of trust-based countermeasures against such attacks is presented by Xia et al.: In [72], the authors propose a trust-based approach that determines the trustworthiness of the paths between the source and the destination node as a result of correct message forwarding behavior. A node is able to check if its message is forwarded correctly because it can overhear the transmission of the neighbor node when it forwards the message. The trust value of the neighbor node is calculated as percentage of correct forward operations. Then, the trustworthiness of a path is determined by multiplying the forwarding trust values of the nodes along the path. The most trustworthy path is chosen to send the message to the destination.

While the approach of the most trustworthy path may be applicable to certain P2P networks, the way of identifying malicious behavior by overhearing another node's communication cannot be transferred to P2P networks. Because P2P nodes cannot overhear each other's communication, other means of identifying malicious behavior need to be found.

3.4 Prior Work of the Author

The author of this thesis engages himself in security of structured P2P networks since 2008. In order to assess the current state of (classic) security measures, the Kad network has been taken as a basis at the start. Fortunately, the source code of eMule is open source and is therefore freely available for analyses. During that time, the eMule developer community has been quite active and has apparently been monitoring the research concerning the Kad network: A method of how to perform Eclipse attacks in the Kad network published by Steiner et al. in [73] has been defeated by a new version of eMule. A close analysis of the source code of the new version has revealed other weaknesses, and so a new way of conducting an Eclipse attack in the Kad network has been designed and evaluated successfully by the author of this thesis in cooperation with a Bachelor student. The results are published in [1].

The mentioned security measures of the eMule client are rather static: They base on IP address filters that only allow a limited number of nodes per subnet to be stored in the routing table and also use blacklists to avoid whole IP subnets. These kinds of countermeasures are not able to cope with nodes that change their behavior over time. So, in the following publications, the focus is set to trust mechanisms to be able to identify malicious nodes without relying on IP filters or other static measures.

The usage of trust mechanisms in structured DHT networks interferes with the assignment of content items to nodes: If content items are stored only on trustworthy nodes and the opinions about which nodes are trustworthy vary between different nodes, content items might be inaccessible for some nodes if they are stored only on nodes they do not regard as trustworthy.

So first of all, a basic study has been performed in cooperation with a Master student to analyze whether a Kademlia-based network is still functional if a fraction of the nodes does not use all existing nodes for their operations. For this study, simplifying assumptions have been made:

- There are two types of nodes in the network:
    - "Cert" nodes with a certificate and
    - "NoCert" nodes without a certificate.

- Cert nodes never behave maliciously and are regarded as trustworthy.
- NoCert nodes are not trusted.

The simulations have revealed that Cert nodes are still able to retrieve content published by NoCert nodes. The reverse case (NoCert nodes attempting to retrieve content published by Cert nodes) has also been successful. The results are published in [2].

The previous environment actually does not use trust mechanisms due to its simplifications: It relies on certification instead of reputation gained by positive or negative experiences in the past. Hence, a concept of how to apply trust mechanisms to a Kademlia-based DHT has been developed that constitutes the beginnings of TrustedKad and is published in [4].

Based on this preliminary version of TrustedKad, routing trust values have been analyzed with respect to their generation, distribution and their impact on improving Kademlia's resilience against attacks on the routing processes. These first tests have been successful, and the results are published in [3]. The final version of TrustedKad, its implementation in a simulation environment and the results of the simulations are presented in the following chapters.

4 TrustedKad

In this chapter, the components of TrustedKad are motivated and explained in detail. A brief discussion of the overhead created by TrustedKad and of optimization opportunities regarding the overhead concludes the chapter.

4.1 Parameters

The original Kademlia uses the two variables *k* and α: *k* signifies the number of nodes in a routing table bucket and the number of nodes in the temporary Lookup list; α indicates the number of Lookup messages that are sent in parallel. When retrieving a content item, Kademlia uses the first version it receives and terminates the Lookup. This is not reasonable because it simplifies attacks on storage, so several replica nodes should be queried.

Kad and also TrustedKad use more differentiated variables that enable a node to configure several settings independently from each other: The parameter *k* still denotes the size of a routing table bucket and the parameter α still denotes the number of parallel messages during a Lookup. In addition, two new parameters are introduced: *numLookupResult* denotes the number of nodes that a Lookup process shall return when it finishes and *numReplica* indicates the number of replicas that shall be stored or retrieved.

4.2 Asymmetric Key Pair and Certificate for Sender Authentication

The identity of a node should be unforgeable, i.e., it should be impossible to impersonate another node. In addition, nodes should be identifiable without having to rely on potentially changing information such as the node ID or the IP address. Usually, an asymmetric key pair is used to achieve this (e.g., see Pastry [36], S/Kademlia [53], TrustGuard [55], PeerTrust [64]): The node can be identified by its public key and can prove its identity by using its private key to encrypt or digitally sign its messages. So, TrustedKad requires every node to have a public/private key pair.

Using the key pair, a node creates a self-signed certificate for itself that contains the following information:

- The node's public key,
- the time of the generation of the certificate,
- the node's underlay address, typically its IP address and port, and
- anti-Sybil-attack information (explained in Section 4.4).

The address and port are included so that a node can directly contact another node if it gets to knows its certificate, e.g., from a Lookup response message. The node ID is not part of the certificate, as it is derived from the certificate (explained in Section 4.3).

Every message that a node sends has to contain its node ID and its certificate and has to be digitally signed using the sender's private key. The receiving node can check the digital signature of the message using the public key contained in the certificate in the message. Additionally, the sender address and port have to match the data inside the certificate and the node ID has to match the certificate.

If an attacker controls several nodes at different physical locations, it would be possible to have its nodes share logical identities. The inclusion of address and port in the certificate prevents an attacker from doing so, because if the messages do not originate from the correct address, they are discarded. This procedure is also recommended by Castro et al. in [52].

4.3 Node ID Generation and Invalidation

The node ID is crucial for the functioning of structured P2P networks, as it is used to assign responsibility for content items to a node. If the node ID can be freely chosen by a node, it can declare for which content it is responsible. This has to be avoided; otherwise, the network is vulnerable to a multitude of attacks such as routing and storage attacks and the Eclipse attack [32].

S/Kademlia [53] and PeerTrust [64] recommend using a hash of the public key as the node ID because it is computationally infeasible to generate an input value for a given hash value. This way, the node cannot choose its ID freely: Forging a node ID is impossible because the correctness of the node ID generation can easily be verified by other nodes by calculating the hash value of the public key. In addition, using the result of a hash function as node ID ensures that the IDs are distributed uniformly across the address space because hash functions are designed to generate uniformly distributed hash values [47].

In [38], Condie et al. induce churn into a P2P network to prevent nodes from being responsible for a certain part of the identifier space for too long a time. This way, the influence of malicious nodes decreases because they cannot manage "their" items indefinitely.

TrustedKad combines both approaches: The time of the generation of a node's certificate is included in the certificate. It is used to invalidate the certificate after a certain amount of time. The node ID is calculated by hashing the certificate as shown in Figure 15. So if the certificate expires, the node ID does as well. A new certificate with a new timestamp has to be generated, leading to a new node ID. So even if a node is available for a long period of time at the same address and port, it has to use different node IDs and is therefore responsible for different content items because its logical location changes regularly. This would not be possible if the node ID was a hash value only of the IP address as in Chord [13] or only of a public key as in S/Kademlia [53] or PeerTrust [64]: This would result in static node IDs.

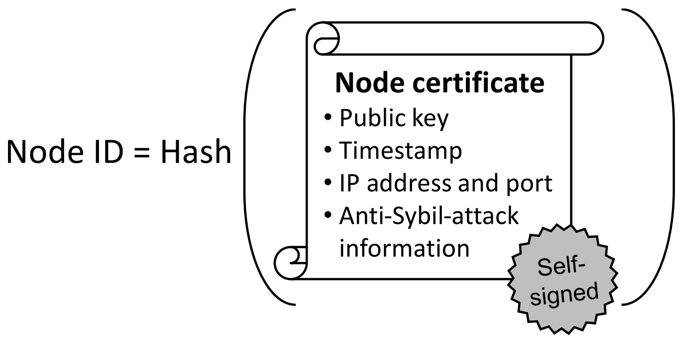


Figure 15: Node ID generation

Using a node's certificate, other nodes are able to verify the ID of the node. The only way for an attacker to generate a desired node ID is by using brute force approaches. Even if an attacker tried to generate a desired node ID this way (by varying the key pair, IP address, port and/or the timestamp included in the certificate), he would have to repeat that process regularly due to the certificate invalidation.

The lifetime of the certificates needs to be adjusted to the (estimated minimum) network size. The smaller the network is, the more often the node ID needs to change: The fewer nodes there are in the network, the shorter is the common prefix of the IDs of two neighboring nodes and so the greater is the distance between them (cf. 2.4.2). This makes it easier for an attacker to generate a certificate that results in a node ID that is closest to the desired ID. This leads to the fact that the smaller the network is, the quicker an attacker can become responsible for a desired part of the address space. So the lifetime of the node ID has to be shorter, the smaller the network is: If an attacker that uses brute force to create a certificate close to a desired ID succeeds in doing so, the certificate is valid only for a comparably short time, so the attacker has to repeat the brute force attack more often than in a large network.

4.4 Sybil Attack Prevention

The Sybil attack can render a peer-to-peer network inoperative: If an attacker is able to control a significant share of the network, all security measures of the network become useless. Two examples for attack approaches are the insertion of a large amount of Sybil nodes into the network or the take-over of other legitimate nodes. So a countermeasure against the Sybil attack is mandatory for every P2P network.

The problem is that Douceur, who has given the attack its name in [31], and Urdaneta et al., who have analyzed a multitude of countermeasure proposals in [32], both come to the conclusion that the attack cannot be completely countered without a central authority. Furthermore, the approaches discussed in [32] are applicable to special environments only: Crypto puzzles, for instance, rely on the fact that all devices in the network have about the same amount of computing power; if this was not the case, slower machines would need much longer to join the network than faster ones, and if an attacker manages to control a large amount of computing power, he can easily create Sybil nodes. Even if there is a functioning Sybil attack prevention system in action, it is still possible for an attacker to take over other legitimate nodes by hacking them.

4.4.1 TrustedKad's Modular Approach

Regardless of whether an attacker uses Sybil nodes or taken-over legitimate nodes, he will most likely use them for an attack in which the controlled nodes collude to manipulate content items. The evaluation results presented in Chapter 6 show that TrustedKad is able to cope with such colluding malicious nodes to a certain extent.

In addition, TrustedKad supports to use an arbitrary Sybil attack countermeasure that suits the application and network environment in which TrustedKad is deployed. This way, the protection against Sybil attacks can be further improved. It is required to implement a suitable anti-Sybil-attack mechanism to prevent nodes from creating fake nodes with fake cached trust information (cf. 4.8.3).

As stated before, the anti-Sybil-attack information is included in the certificate of a node. This information could be a solution to a crypto puzzle or the digital signature of a certificate authority (CA) that confirms that the physical user of the node uses only one logical identity. If possible, the Sybil attack countermeasure should also include the node's IP address and port in addition to its public key to prevent the exchange of asymmetric key pairs between nodes as explained in Section 4.2. If a CA is used, its signature can easily include the IP address and the port; if crypto puzzles are used, IP address and port can be included in the input of the hash function. The anti-Sybil-attack information must have a limited lifetime so that an attacker cannot accumulate valid signatures for different nodes over time.

It can be argued that if a CA is used for Sybil attack prevention, it could also be used to replace the distributed trust management system. However, the usage of a central authority against the Sybil attack requires interaction with it only once during the Bootstrap and once every time the node ID changes. If it would manage the ratings and trust values as well, it would require considerably more resources because any transaction of any node in the network would cause trust queries during the transactions and ratings after the transaction.

4.4.2 Countermeasure Against the Whitewashing Attack

The Sybil attack countermeasure is also a countermeasure against the whitewashing attack: When a node receives too many negative ratings and then switches to a new identity, this is called whitewashing attack (cf. 2.5.4.2). To do so, the node has to generate a new key pair and a new certificate. However, the Sybil attack countermeasure prevents that a node can do this repeatedly: The CA has to limit new number of logical identities per physical user and time, and the crypto puzzle approach requires the solution of new puzzles for every new identity. Other Sybil attack countermeasure approaches also prevent the user from changing identities repeatedly and thereby prevent whitewashing attacks.

4.5 Types of Ratings and Trust Values

TrustedKad rates actions and uses these ratings to identify inoffensive and malicious nodes. A single rating can either be positive or negative. The ratings are combined to a trust value (cf. 4.6): Malicious nodes are supposed to have low trust values, while inoffensive ones shall have high trust values.

As explained in Section 2.3.1, the four basic DHT actions are Bootstrap, Lookup, Put and Get. Bootstrap and Lookup messages can be regarded as equal in the decision if another node is trustworthy, as both of them are used to find new nodes in the network. Hence, nodes that are trustworthy for Lookups can also be used for Bootstraps.

A separate rating for Put and Get is not reasonable, as it is impossible to evaluate if a Put operation has been successful: The storing node could delete the content item immediately after receiving a positive rating for storing it or anytime later without any node noticing and therefore also without any punishment. Hence, successful Put operations should not have any influence on the trustworthiness of a node. From this, it follows that only nodes that have proven to deliver correct content items during retrieval should be used to store content items on.

In TrustedKad, a node is used a lot more often for routing than for content storage or retrieval: The final *numLookupResult* nodes are all used for routing and storage/retrieval. The nodes used during the Lookup until the final *numLookupResult* nodes are found are only used for routing. So, if there was only one type of rating for both routing and storage, the node's routing behavior would have a greater influence on the trust value than the storage behavior.

TrustedKad therefore uses two trust values to assess a node's trustworthiness: A routing trust value that is used for Bootstrap and Lookup operations and a storage trust value that is used for Put and Get operations. Sections 4.10 and 4.11 describe in detail how ratings are given for routing and storage procedures, respectively.

TrustedKad operates on the KBR and DHT layers, independently from the application, so it does not offer any trust values that are based on application-dependent behavior such as the contribution behavior value of Mekouar [56]. In most cases, application-dependent trust values are not required: If the application offers different qualities of service to other nodes depending on their routing and storage trust values, nodes are encouraged to route and store content items correctly, which should be enforced (as explained in Section 4.14). However, if an application still requires additional trust values, TrustedKad can easily be extended to handle them and forward them to the application layer for interpretation.

4.6 Computation of Trust Values and Usage of Thresholds

TrustedKad takes both positive and negative ratings in consideration to determine a node's trustworthiness; if only positive or only negative ratings were taken into account, the system would be susceptible to common attacks on trust systems that are presented in Section 2.5.4. A formula similar to the ones used by FuzzyTrust [62] and Mekouar [56] (as shown in Section 3.2.1.1) is used to compute the trust values of a node. In contrast to the other two algorithms, TrustedKad introduces a grace phase for newcomer nodes: A node is treated as trustworthy as long as it has not yet received a certain amount of ratings. If the computed trust values were used from the first rating on, a node whose first rating is negative would never get the chance to earn better ratings, as a single rating that is negative results in the worst possible rating and therefore in ignoring the node.

The routing trust value of a node is computed as follows:

$$T_{R,i} = \begin{cases} 1, & if\ R_{R,i}^{+} + R_{R,i}^{-} \leq g_R \\ \dfrac{R_{R,i}^{+} - R_{R,i}^{-}}{R_{R,i}^{+} + R_{R,i}^{-}}, & otherwise \end{cases}$$

$T_{R,i}$ denotes the routing trust value of node i, $R_{R,i}^{+}$ the number of positive routing ratings of node i, $R_{R,i}^{-}$ the number of its negative routing ratings, with $R_{R,i}^{+}$, $R_{R,i}^{-} \geq 0$, and g_R the number of grace routing ratings, with $g_R > 0$ for normal operations. g_R is set to 0 during Bootstrap (cf. 4.10.1); the formula also applies then and sets the routing trust value to 1 only for nodes without any routing rating.

Correspondingly, the storage trust value is calculated with this formula:

$$T_{S,i} = \begin{cases} 1, & if\ R^{+}_{S,i} + R^{-}_{S,i} \leq g_S \\ \dfrac{R^{+}_{S,i} - R^{-}_{S,i}}{R^{+}_{S,i} + R^{-}_{S,i}}, & otherwise \end{cases}$$

Similar to the routing trust value formula, $T_{S,i}$ denotes the storage trust value of node *i*, $R_{S,i}^{+}$ the number of its positive and $R_{S,i}^{-}$ the number of its negative storage ratings, with $R_{S,i}^{+}, R_{S,i}^{-} \geq 0$, and g_S the number of grace storage ratings, with $g_S > 0$.

Both formulas generate trust values in the range of [-1.0; 1.0]. A trust value of -1.0 indicates that a node has only received negative ratings; 1.0 means that a node has received only positive ratings or is a newcomer node that is still in its grace phase. Table 1 shows the trust values that result from selected ratios of positive and negative ratings: For example, a trust value of 0.5 indicates that a node has three times as many positive ratings as negative ones:

$$\text{Trust value:} \quad \frac{R^{+} - R^{-}}{R^{+} + R^{-}} = \frac{75 - 25}{75 + 25} = \frac{50}{100} = 0.5$$

$$\text{Ratio:} \quad \frac{R^{+}}{R^{-}} = \frac{75}{25} = 3$$

The routing and storage trust values are used during Bootstrap/Lookup and Put/Get actions, respectively. Every node uses its own individual routing trust value threshold and storage trust value threshold to determine whether it trusts another node: Only nodes with a trust value equal to or greater than the threshold are used for the node's operations, other nodes are ignored. In Chapter 6, the results of evaluations with different thresholds are presented.

4.7 Distributed Storage of Trust Information

In contrast to P2P networks with central elements, such as the login servers of the Skype network [5] or the supernodes of hybrid P2P networks like the one used by Mekouar [56], structured P2P networks do not have central elements or peers with special functionalities. So, in order to store the ratings and trust values, a distributed solution has to be designed. The complaint-based system presented by Aberer and Despotovic in [65] uses P-Grid [63] to store "*one or more*" [65] replicas of the trust information. TrustedKad uses several trust management nodes that are responsible for storing the ratings of "their" node.

The trust information about a node must not be bound to its IP address and port and nei-

Table 1: Trust values of selected ratios of positive and negative ratings

R^+	R^-	R^+/R^-	Trust Value	R^+	R^-	R^+/R^-	Trust Value
100	0		1.0	0	100	0.000	-1.0
95	5	19.000	0.9	5	95	0.053	-0.9
90	10	9.000	0.8	10	90	0.111	-0.8
85	15	5.667	0.7	15	85	0.176	-0.7
80	20	4.000	0.6	20	80	0.250	-0.6
75	25	3.000	0.5	25	75	0.333	-0.5
70	30	2.333	0.4	30	70	0.429	-0.4
65	35	1.857	0.3	35	65	0.538	-0.3
60	40	1.500	0.2	40	60	0.667	-0.2
55	45	1.222	0.1	45	55	0.818	-0.1
50	50	1.000	0.0				

ther to its node ID because the IP address and port may change and the node ID does change regularly. So the public key (K_{pub}) of a node is used to identify it and to determine the location of its trust management nodes.

4.7.1 Location of the Trust Management Nodes

An attacker must not be able to control the trust management nodes of its node(s). By using the hash of a node's certificate as the node ID, TrustedKad already prevents the free choice of a node ID. However, to avoid any risks of eclipse attacks against trust information, TrustedKad stores trust information at three different locations of the ID space. These locations shall be determined as Hash(K_{pub}), Hash(Hash(K_{pub})) and Hash(Hash(Hash(K_{pub}))). The actual trust management nodes of a node change regularly due to the node ID invalidation. Figure 16 shows a node and its trust management nodes graphically. The locations of the trust management nodes are not distributed evenly because they are determined by hashing. This distributed system does not have a single point of failure or a single target for attacks, and every node is a potential trust management node. It also prevents a node from being able to control its own trust management nodes.

If the nodes at one location report significantly different trust values than the nodes at the other two locations, the single location's information is discarded. A node must not be its own trust management node, so if it would be responsible for its own trust information according to its current node ID, it is ignored.

4.7.2 Identification of the Trust Management Nodes

When a node joins the system, it has to locate its trust management nodes and inform them about its presence. Then, the trust management nodes have to confirm that they are indeed the trust management nodes; if they are not because they know nodes that are closer to the trust management location, they have to inform the correct trust management nodes and the node itself.

No grace ratings and no unchoking (cf. 4.13) are applied when the trust management nodes are identified, just as during a Bootstrap (cf. 4.10.1). In addition, system-wide settings for the trust value thresholds have to be used instead of the node's individual

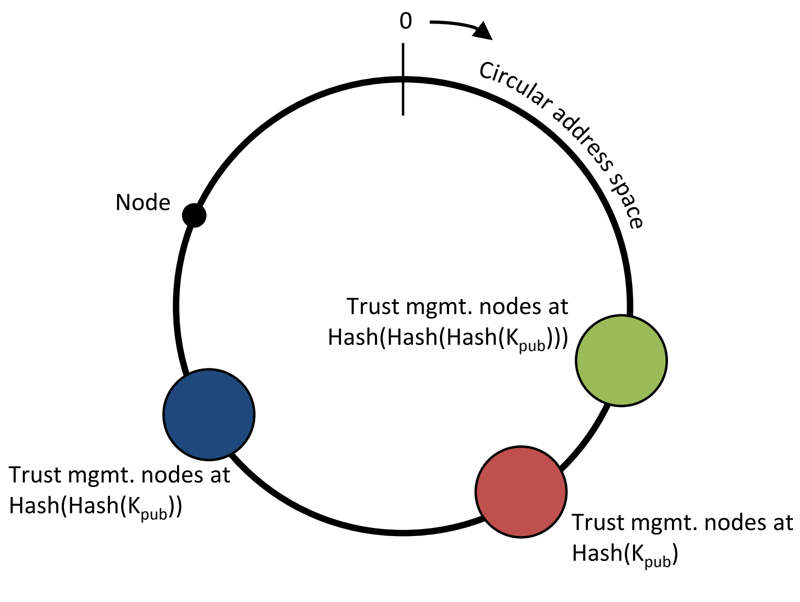

0
Circular address space
Node
Trust mgmt. nodes at Hash(Hash(Hash(K$_{pub}$)))
Trust mgmt. nodes at Hash(Hash(K$_{pub}$))
Trust mgmt. nodes at Hash(K$_{pub}$)

Figure 16: A node and its trust management nodes

thresholds. The actual setting of these global thresholds depends on the system in which TrustedKad is deployed.

4.7.3 Replication of Trust Information

Nodes that store trust information about a node have to periodically replicate and exchange it to also receive ratings that have not reached every trust management node, e.g., due to network failures. As every trust management node needs to be able to prove its assessment (cf. 4.9.2), the original rating messages of the rating nodes are exchanged, not only the current status of the rating counters. This mechanism ensures that the trust management nodes are up-to-date and synchronized, even if one node leaves the system or needs to change its node ID and another node becomes responsible.

4.7.4 Countermeasure Against Self-Promoting and Slandering Attacks

As described in Section 2.5.4, self-promoting and slandering attacks aim to increase the trust values of an attacker's nodes or attempt to decrease the trust values of legitimate nodes, respectively. TrustedKad counters these attacks by allowing each node to rate any other node only once for routing and also only once for storage. The most recent rating that has been given is to be used. Also here, the rating nodes are identified by their public key and not by their node ID.

4.8 Caching of Trust Information

In order to determine whether to interact with another node, e.g., during a Lookup, a node would actually have to ask the other node's trust management nodes for the trust value first. This in turn would require another Lookup to find the trust management nodes, again requiring a Lookup to find the trust values of other involved nodes, etc. The result would be an infinite loop of Lookups. Aberer and Despotovic use a limited search depth as described in [65] and accept to not be able to decide whether another node is trustworthy in some cases. They claim that "*These cases should be rare*" [65] but do not prove it.

Therefore, TrustedKad uses a different approach: It requires the trust management nodes of a node to regularly cache the trust information of the managed node at the node itself. The cached trust information has to be included in every message to enable other nodes to immediately assess the trustworthiness of a node. As explained in Section 3.2.1, Mekouar [56] also caches trust information at the node itself, but only to be able to restore it at another superpeer if a node's own superpeer leaves the network.

4.8.1 Contents of Cached Trust Information

The cached trust information consists of the following:

- Certificate of the trust management node,
- public key of the managed node,
- the number of positive and negative routing and storage ratings and
- a timestamp.

This information is digitally signed by the trust management node to prevent manipulation.

The certificate of the trust management node includes its underlay address, so it can be contacted directly without another Lookup. This way, the trust management nodes can be asked to confirm the trust information and be asked for an update of the information at a later time. In addition, the addresses of the trust management nodes are used to give the ratings.

The public key of the node is included in the cached trust information to prove that the trust information applies to this node. Including the public key of the managed node in the cached trust information is sufficient; the full certificate is not required because it is contained in every message the node sends, i.e., also in the message that contains the cached trust information. The cached trust information contains the rating counters instead of the computed trust values because nodes need to able to calculate the group trust value for storage as described later in Section 4.11.2. In addition, only the knowledge of the counters allows for different numbers of grace ratings to be used: During Bootstrap, the number of grace ratings is set to 0 because the Bootstrap needs to be especially protected (cf. 4.10.1).

The timestamp is included to invalidate the cached trust information after a certain amount of time. This way, a malicious node that behaves correctly until it has reached a certain trust level cannot use the cached trust information attesting high trust values to conduct attacks indefinitely.

4.8.2 Procedures

When a node sends a Lookup response message, the response also has to include the cached trust information of all nodes contained in the response. This way, the node that performs the Lookup can decide which nodes to use as candidate nodes for the next hop.

During the Bootstrap, a node identifies its trust management nodes and asks them for cached trust information. If this is the first time the node joins the network, the trust management nodes return cached information containing zero ratings.

4.8.3 Attacks on Cached Trust Information

A possible attack is that attackers fake cached trust information, either their own or the trust information of fake nodes they list in a Lookup response. Such an attack is only possible if the attacker is able to fake the anti-Sybil-attack information in a node’s certificate. If he is, faking cached trust information still requires the creation of several fake trust management nodes. Each of them must have a certificate, so one key pair per node and other information (timestamp, IP address, port) has to be generated. Then, the fake node IDs and the fake cached trust information are generated.

Any node receiving cached trust information is able to roughly assess whether the trust management nodes contained in the information are indeed responsible for the managed node: The logical location of the trust management nodes is derived from (nested) hash operations using the public key of the managed node. With *n* depending on the (estimated minimum) size of the network, the first *n* bits of the (nested) hash of the public key should be equal to the trust management node’s IDs (as explained in Section 2.4.2).

If this quick check fails, the cached trust information could be faked, so the node should attempt to confirm the cached trust information by querying the trust management nodes. If they are reachable, the current trust information returned by them can be used. If they are not reachable, the node has to be avoided. During a Lookup, instead of querying the trust management nodes, the node could also query other nodes that pass the quick check first or in parallel in order to not lose time with the confirmation.

4.9 Transparent Rating Procedure

It is necessary that a rating can only be given if an interaction has taken place before. If this was not the case, the slandering attack would be simplified because a malicious node could easily give negative ratings to any node without previous interaction.

TrustedKad therefore requires every rating to be authorized by the node that is rated. Every Lookup and Get response message has to contain an acknowledgement of the responding node that the requesting node may rate it. If the message does not contain an acknowledgment, it is discarded. The acknowledgement contains

- the certificate of the querying node,
- the certificate of the responding node,
- the type of action that may be rated (routing or storage) and
- a timestamp.

The acknowledgement is digitally signed by the responding node. It has to be presented to the trust management nodes when giving the rating. The timestamp limits the validity of the acknowledgement.

4.9.1 Format and Transmission of Rating Information

When an operation has taken place, the querying node has to rate the involved nodes. It creates ratings for every node that contain the following:

- The rating (positive or negative),
- the rating acknowledgement which contains the certificates of querying and responding node and the rating type and
- a timestamp.

Each rating is digitally signed and sent to the respective trust management nodes.

4.9.2 Verifiability of Ratings

The trust management nodes have to keep the full rating messages. They have to be able to present them if another node demands proof of the ratings. If only the counters were increased and the ratings discarded afterwards, the trust management nodes could manipulate the counters arbitrarily.

TrustGuard [55] and TrustMe [70] both use digitally signed proofs of transactions as well, but they do not require the proofs to be verifiable by other nodes after they have been sent to the trust management instances. This enables an attacker to manipulate the rating counters at will.

4.9.3 Rating Enforcement

TrustedKad relies on ratings, so it has to be ensured that all interactions are rated. A node therefore forwards incoming Lookup and Get requests to its trust management nodes. If they do not receive a rating after a reasonable amount of time, they give a negative routing/storage rating to the querying node. Instead of the acknowledgement, they use the original message of the querying node as a proof of interaction.

4.10 Securing Bootstrap and Lookup Procedures With Routing Trust Values

This section describes how routing trust values are used to improve the security of Bootstrap and Lookup operations. Definitions of inoffensive and malicious routing behavior are given, and the resulting impediment for attackers is described.

4.10.1 Bootstrap Procedure

The Bootstrap operation needs to be especially protected because in most cases, a joining node uses a single node for the Bootstrap. If that node is a malicious node, it gains full control over the joining node's view of the network.

Therefore, during Bootstrap, the number of grace ratings is set to 0 to avoid using malicious nodes. Unchoking (cf. 4.13) is also disabled. In addition, the routing trust values are confirmed using the trust management nodes of the Bootstrap node. If more than one Bootstrap node is available, several of them should be used, starting with the most trustworthy one.

Due to the cached and verifiable trust information, a newcomer node is able to assess the trustworthiness of potential Bootstrap nodes. The responses of the Bootstrap node contain the cached trust information and information about how to reach the trust management nodes for confirmation. In systems without TrustedKad, this assessment is not possible.

4.10.2 Lookup Procedure

Lookups are performed as described in Section 2.4.6: At the beginning of each Lookup, a temporary Lookup list is created and filled with the *numLookupResult* closest nodes to the target the querying node currently knows. The temporary Lookup list is kept sorted with the closest node at the top and contains only trustworthy nodes, as it is used for the node's own actions.

The top α nodes of the list that have not been queried yet are queried for closer nodes. The nodes contained in the response messages are evaluated for trustworthiness using the cached trust information contained in the responses. The trustworthy nodes are inserted into the Lookup list. The process repeats until the top *numLookupResult* nodes have all been queried and have not returned any closer nodes.

Section 4.11.2 introduces a "most trustworthy" approach for content retrieval. It should be noted that this is not reasonable for routing: DHT routing aims to reach the target nodes with as few hops as possible. If the Lookup list was sorted by trustworthiness, the duration of the Lookup would increase and the Lookup would perhaps not even reach the correct target nodes if the nodes that could lead to them are not among the top *numLookupResult* trustworthy nodes. So the Lookup list is sorted by distance also in TrustedKad.

As mentioned before, when a node receives a Lookup request, its response includes all types of nodes, i.e., also nodes it does not trust itself. The querying node could use another trust value threshold and therefore regard more or fewer nodes as trustworthy than the responding node. In addition, a node always has to answer to a query regardless of whether it trusts the querying node.

4.10.3 Definition of Inoffensive and Malicious Routing Behavior

TrustedKad defines that a node is regarded as acting inoffensively if its Lookup response contributes to find the final *numLookupResult* nodes. If the node's response is not useful, it receives a negative rating. To evaluate the usefulness of the involved nodes, the querying node keeps track about which node informed about which other nodes during a Lookup. Only nodes that have responded to queries can be rated because the querying node needs the acknowledgement to give the rating. Nodes that do not respond to queries on purpose harm themselves as they cannot receive positive ratings to gain trust and achieve higher trust values (cf. 4.14).

A node's usefulness is evaluated in two ways which are referred to as "bottom-up" and "top-down" in the following. The top-down evaluation starts with the final *numLookupResult* nodes and gives a positive rating to every node that has informed about at least one of the final *numLookupResult* nodes. Then, all nodes that have informed about those nodes are given a positive rating as well until the node reaches itself in the hierarchy.

A node that informs only about nodes that are so far away from the target that they are never among the top *numLookupResult* nodes of the Lookup list receives a negative rating: Its response has not led to the final *numLookupResult* nodes because none of its reported nodes is queried. If only the top-down approach was used, a node that informs only about that negatively rated node would also receive a negative rating despite of the fact that is has led to another node that has been queried and that has responded: It is not part of a path leading to the final *numLookupResult* nodes. The same is true for nodes that only inform about malicious nodes that in turn only respond with non-existent nodes.

Therefore, the bottom-up evaluation starts at the node itself and gives a positive rating to every node that has led to at least one other node that has responded to a Lookup query, regardless of whether the nodes contained in the Lookup response have led to the final *numLookupResult* nodes or not.

The bottom-up approach is also required if the Lookup times out: As there are no final top *numLookupResult* nodes, the bottom-up way is the only way by which contributing nodes can receive a positive rating.

If the Lookup finishes regularly, the evaluation is always performed in both ways. A negative rating is given to all nodes that do not receive a positive rating by either procedure. Figure 17 shows the response hierarchy created by the trust evaluations with the querying node at the bottom and the final *numLookupResult* nodes at the top of the figure. The node in the second layer from the bottom on the left only receives a positive rating using the bottom-up evaluation: It has informed only about one other node that in turn has

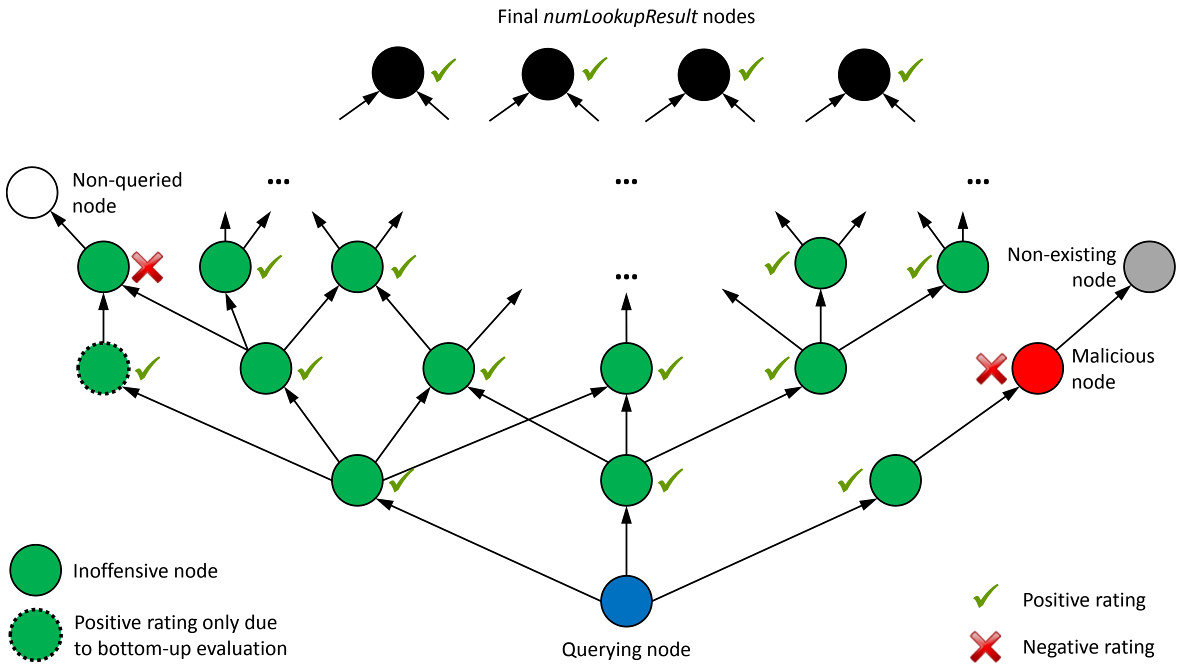


Figure 17: Routing trust evaluation

only informed about a node that is too far away from the target so it has never been queried during the Lookup (and therefore does not receive any rating, see above). The neighboring node on the right that has also informed about the negatively rated inoffensive node receives a positive rating also via the top-down way: It has also informed about other nodes that have led to the final *numLookupResult* nodes.

The first queried node shown on the right would also receive a negative rating without the bottom-up evaluation because it only informs about a malicious node that in turn only responds with non-existing nodes. The final *numLookupResult* nodes receive a positive rating because they are all queried for closer nodes and return nodes that respond to queries, namely the other final *numLookupResult* nodes. The figure also shows that inoffensive nodes that respond to queries without attempting to harm the querying node can receive a negative rating if the nodes they deliver are not close enough to the target.

4.10.4 Resulting Impediment of Routing Attacks

This section first explains how an attack on routing is conducted in a network that does not use TrustedKad: The malicious node responds to a Lookup query only with non-existing nodes to have the Lookup time out. The author has successfully conducted such an attack using a single physical node in the Kad network and has published the results in [1]. Afterwards, the impediments that TrustedKad creates to protect against such an attack are explained.

4.10.4.1 Network Without TrustedKad

The Kad network allows choosing the node ID freely. So when a malicious node receives a Lookup query, it can generate a list of non-existing fake nodes whose node IDs are very close to the target ID, i.e., only the last few bits differ from the target ID. The IP address and port are chosen randomly. The list is sent to the querying node which inserts the fake nodes into its Lookup list. The list is sorted by distance to the target, so the fake nodes are at the top of list because their node IDs are very close to the target. Queries are sent to the fake nodes and no responses are received because the nodes do not exist. Finally, the Lookup times out without finding the target.

4.10.4.2 Network With TrustedKad

If TrustedKad is used, the node IDs of the fake nodes cannot be chosen freely as they are computed as the hash value of the node certificate. As already mentioned in Section 4.8.3, the attack is only possible if the attacker is able to fake the anti-Sybil-attack information of both the fake nodes and their fake trust management nodes. If he is, he still has to put a lot more effort in generating the non-existing fake nodes: A key pair has to be created for every fake node and its certificate has to be signed using the generated key pair.

In addition, several fake trust management nodes have to be created for every fake node, each requiring one key pair, to fake the nodes' cached trust information. As explained in Section 4.8.3, a node can estimate whether the trust management nodes are indeed responsible for their managed node, so a brute force attack has to be conducted for every trust management node to obtain a node ID that passes the estimation.

Furthermore, the attacker cannot prepare the fake information in advance if it wants to attack the whole address space. Instead, the fake information has to be generated dynamically when the Lookup query is received to be able to specifically attack the incoming request.

The fake nodes contained in the response message are probably not close enough to the target ID to fill all top places in the querying node’s Lookup list, if one is among the top *numLookupResult* nodes at all. So, there probably still remain reachable nodes to continue the Lookup.

The required effort to create all the fake information is probably too high to timely generate a response that fulfills all conditions. However, the simulation results presented in Chapter 6 show that even if the attacker manages to create trustworthy fake nodes, TrustedKad reduces the attack’s impact significantly.

4.11 Securing Put and Get Procedures With Storage Ratings

When the Lookup procedure has successfully terminated, all nodes in the Lookup list with a storage rating greater than or equal to the querying node’s individual storage trust value threshold become candidate nodes for the Put or Get operation; all other nodes are discarded. Only nodes that are trustworthy for routing can be candidate nodes for storage operations because nodes that are not trustworthy for routing cannot be among the final nodes, as they are sorted out before. As a result, the candidate nodes for storage are always trusted for both routing and storage.

4.11.1 Put Operation

As explained in Section 4.5, there is no separate rating for Put and Get. Instead, there is a combined storage trust value. Only nodes that are trustworthy for storage are used for the Put operation. TrustedKad stores replicas at the *numReplica* nodes that are closest to the content item ID, so the first *numReplica* nodes are contacted first to store the content. If the Put action does not succeed at one or more of them, the next nodes in the list are used. The Put action is regarded as successful if the content item can be stored on at least one node.

When a publishing node stores a content item using the Put operation, it shall sign the data and leave its certificate. This way, the retrieving node can verify the origin of the content item using the signature. Attacks that store fake content items (e.g., music files that have the name of an existing song but contain only white noise in filesharing networks) have to be handled by the application.

As in the Kad filesharing network, content items have a limited lifetime, so they are deleted from the storing nodes after a certain amount of time. In addition, they are not automatically replicated between the nodes that store them. So if one node leaves the system or has to generate a new node ID, one replica is lost. Therefore, the node that has originally published a content item has to republish it regularly. The node that publishes a content item can decide whether the content item may be changed by other nodes or if only the publishing node itself may change the item.

4.11.2 Get Operation

After receiving the list of candidate nodes determined by the Lookup, the Get process consists of two phases: In the first phase, *numReplica* nodes are queried for a hash of their replica version of the content item using GetHash messages. Queries that reach a timeout or respond with “content item unknown” lead to more candidate nodes being queried because TrustedKad attempts to receive *numReplica* responses to cope with the different views on trustworthiness the nodes can have. Only if the first *numReplica* responding nodes all reply with “content item unknown”, the Get operation is terminated and fails.

If all received hashes are equal, one node is randomly selected for the second phase, the actual transfer of the content item's value. The message type used to query the content item value is called GetValue. If several different hashes are returned, one version has to be chosen. In networks without TrustedKad, a simple majority decision takes place: The hash value that is returned by the most nodes is chosen, the other ones are discarded.

Instead of a majority decision, TrustedKad uses a trust-based approach to solve the problem: The nodes are grouped by the hash values they have returned. Then, group trust values are calculated as follows:

$$T_{S,g} = \begin{cases} 0, & if\ R^+_{S,g} + R^-_{S,g} = 0 \\ \dfrac{R^+_{S,g} - R^-_{S,g}}{R^+_{S,g} + R^-_{S,g}}, & \text{otherwise} \end{cases}$$

Let $T_{S,g}$ be the storage trust value of the node group g, $R_{S,g}{}^+$ the sum of all positive storage ratings of all nodes of the group g and $R_{S,g}{}^-$ the sum of their negative storage ratings. The version with the highest group trust value $T_{S,g}$ is chosen. If there is a stalemate, the version with the largest number of total storage ratings $R_{S,g}{}^+ + R_{S,g}{}^-$ is chosen. If there still is a stalemate, the version of the group with more nodes in it is chosen; otherwise, the choice is random. Please note that in contrast to the evaluation of a single node according to Section 4.6, no grace values are applied when determining a node group trust value.

Table 2 shows an example of two node groups having delivered different hashes of the content item's value. If only the mean of the nodes' storage trust values would be relevant, group B would have been chosen. However, group B has a worse group trust value if all positive and negative ratings of the node groups are taken into account. So group A is chosen by TrustedKad.

4.11.3 Definition of Inoffensive and Malicious Storage Behavior

Just as with the routing procedure, contributing nodes receive a positive rating. A node is regarded as contributing if it belongs to the node group that has delivered the chosen version of the content item. All other nodes receive a negative rating, i.e., nodes that have delivered another version of the content item, nodes that have delivered a value that does not match the hash, nodes that have responded to the GetHash request but not to the GetValue request and also nodes that have stated that the content item does not exist if at least one other node returns a content item. If all queried nodes state that the content item does not exist, then no ratings are given. In order to prevent trust management nodes from giving a negative rating to a node because it does not give a rating, a special rating is sent to them indicating that the content item does not exist.

A possible attack of a malicious node is to return the correct hash of a content item but a fake value afterwards. This can be easily detected by the querying node by calculating the

Table 2: Comparison of storage ratings of two example node groups

Node Group A			
Node	**Positive Ratings**	**Negative Ratings**	**Trust Value**
A	849	197	0.62
B	365	109	0.54
C	1,019	12	0.98
D	342	197	0.27
Total	**2,575**	**515**	**0.67**
Mean of the nodes' trust values			*0.60*

Node Group B			
Node	**Positive Ratings**	**Negative Ratings**	**Trust Value**
G	1,067	350	0.51
H	418	28	0.87
I	15	4	0.58
Total	**1,500**	**382**	**0.59**
Mean of the nodes' trust values			*0.65*

hash of the value: If it does not match the previously received hash, the value is faked and another node is queried for the correct value. However, this attack enables a malicious node to falsely obtain a positive rating: If it delivers the correct hash and is not queried for the (fake) value, then it receives a positive rating. The results of this attack are shown in Chapter 6.

4.11.4 Resulting Impediment of Storage Attacks

This section explains how attacks on storage are conducted in non-TrustedKad networks and how TrustedKad impedes them.

4.11.4.1 Network Without TrustedKad

The common way to choose the correct version of a content item is a majority decision. S/Kademlia is an example of a network that uses it [48]. The majority decision simplifies attacks on the content retrieval operation, especially if there is no measure in place that keeps nodes from choosing their own ID. Only $\left\lceil\frac{numReplica+1}{2}\right\rceil$ nodes are required to successfully control the content item's value; if the attacker is able to easily position its nodes at the content item's location by choosing the node IDs, only this number of nodes is required. If the attacker cannot choose the node IDs freely, he creates as many nodes as required until he has enough nodes at the target location. This can be simplified by a Sybil attack.

4.11.4.2 Network With TrustedKad

Instead of the majority decision, TrustedKad uses storage trust values to choose the correct version of a content item. Using the aggregated number of ratings of the nodes in each group, the most trustworthy node group is chosen to retrieve the content item. It is possible that this "group" consists of only one node, while there is another version delivered by several nodes. However, if the storage trust value of the single node is higher than the group's storage trust value, the version of the single node is chosen.

As a result, more nodes are required to conduct the attack: All replica nodes have to be malicious. In addition, each of them must have a storage trust value above the threshold. If not all replica nodes are controlled by the attacker, his nodes furthermore have to form the most trustworthy node group. Conducting the attack is additionally impeded by the fact that TrustedKad does not allow the free choice of the node ID. The Sybil attack countermeasure increases the effort even more.

4.12 Concealment of the Content Item ID During Get Operations

A possible attack on the Get operation is that a malicious node claims to store any arbitrary content item [32]: If queried for a content item ID with a GetHash message, it always responds with the hash of a fake content item. If there are several of such nodes and if they collude by returning the same fake content item, they might be able to significantly disrupt also TrustedKad's trust-based approach.

In order to counter this attack, TrustedKad requires that nodes have to prove that a content item has been stored on them before. To achieve this, the actual ID of the content item that shall be retrieved must not be disclosed to any other node but has to be concealed so that no node is able to claim knowledge of the content item unless the item has truly been stored on it. This concealment of the content item ID significantly improves the security of the network as shown later in Section 6.5.2.

4.12.1 Changes to Lookup and Get Operations

In DHTs, the Lookup process usually uses the ID of the content item to determine the responsible nodes. This way, the ID is disclosed to all involved nodes, which has to be avoided. Fortunately, it is not necessary to use all bits of the ID to find the responsible nodes: As shown in the author's diploma thesis [49] and explained in Section 2.4.2, in a network with four million concurrent users, only the first $\log_2(4{,}000{,}000) \approx 22$ bits of the IDs of a node and the content item have to match; the remaining bits are irrelevant for the determination of the responsible nodes. TrustedKad takes advantage of this and only uses the first 64 bits of the content item ID to find the responsible nodes, which suffices for networks that consist of up to $2^{64} \approx 1.8 \times 10^{19}$ nodes. The remaining bits of the Lookup target ID are chosen randomly as shown in Figure 18.

When the responsible nodes have been identified, GetHash messages are used to obtain the hashes of the content item versions. The content item ID must not be disclosed, so the querying node conceals it in the messages: The node combines the content item ID with its node ID using XOR and hashes the result, which is used as the Get target ID. Note that the first 64 bits of the Lookup target ID and the Get target ID probably do not match, so this ID can only be used for GetHash, but not during the Lookup.

The queried node performs the same operation with the querying node's ID and all content items it stores and checks whether one of the results matches the Get target ID. If so, it responds to the query and includes the content item ID in the response. This way, the querying node can check whether the response refers to the correct content item. Afterwards, the original content item ID can be used in the GetValue message.

4.12.2 Resulting Impediment of Attacks on Get Operations

With concealment of the Get target ID, a malicious node cannot claim to be responsible for any content item: Figure 19 shows that a malicious node that does not know the queried content item from a previous Put operation cannot respond with a fake hash if the

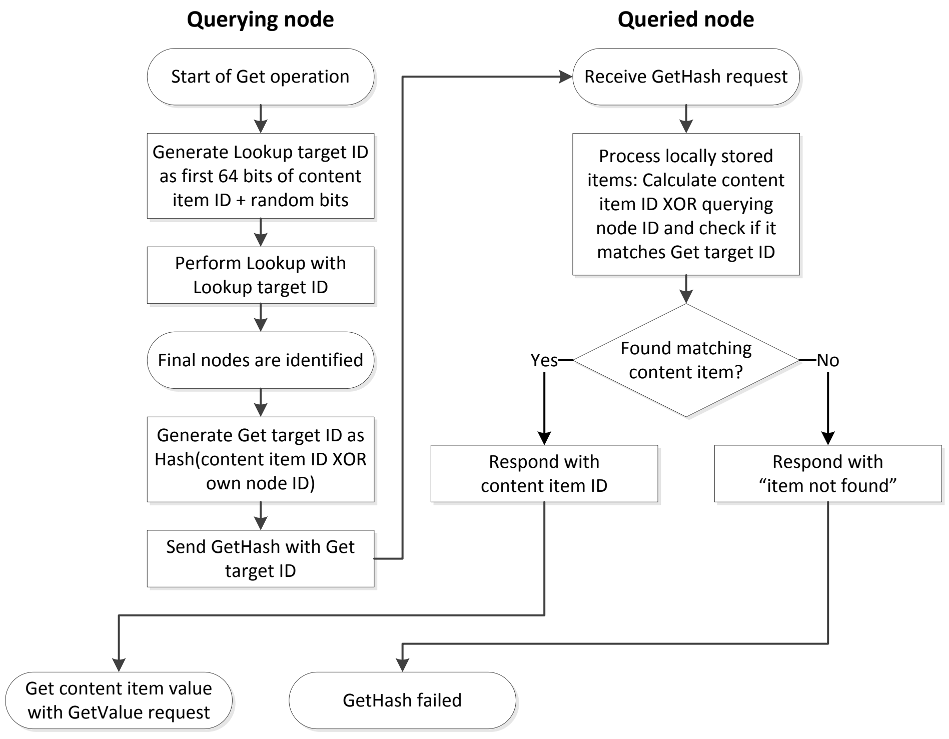

Querying node
Queried node
Start of Get operation
Generate Lookup target ID as first 64 bits of content item ID + random bits
Perform Lookup with Lookup target ID
Final nodes are identified
Generate Get target ID as Hash(content item ID XOR own node ID)
Send GetHash with Get target ID
Receive GetHash request
Process locally stored items: Calculate content item ID XOR querying node ID and check if it matches Get target ID
Found matching content item?
Yes
No
Respond with content item ID
Respond with "item not found"
Get content item value with GetValue request
GetHash failed

Figure 18: Flowchart of Lookup and GetHash operations with concealed content item ID

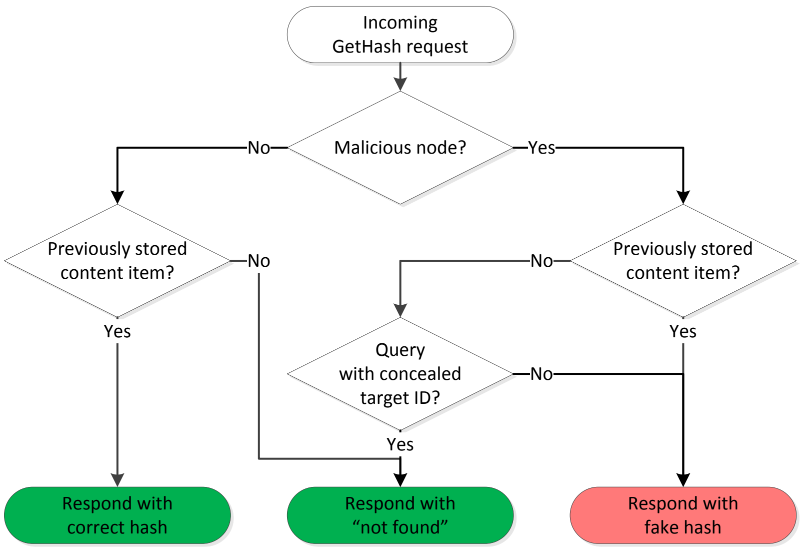

Incoming GetHash request
Malicious node?
No
Yes
Previously stored content item?
No
Yes
Previously stored content item?
No
Yes
Query with concealed target ID?
No
Yes
Respond with correct hash
Respond with "not found"
Respond with fake hash

Figure 19: Handling of incoming GetHash message with and without concealment of target ID

Get target ID is concealed. If the ID is not concealed, the malicious node can attack all incoming requests by responding with a fake hash. This attack is successfully averted by the concealment of the ID. Inoffensive nodes do not change their behavior, regardless of whether the ID is concealed or not.

The combination of the content item ID and the node ID prevents that the malicious node uses the received Get target ID to query its neighbor nodes for the real content item ID: If only a hash of the content item would be used, the malicious node could query its neighbor nodes by using a replay attack after it has received the first query from another node. The correct content item ID is contained in the response message, so the malicious node could respond correctly to all subsequent queries. Therefore, the Get target ID needs to be concealed in a dynamic way dependent on the querying node. If the malicious node replays the Get target ID obtained from another node, the queried node cannot find a matching content item because the node IDs of the node that has sent the original query and the malicious node differ.

Due to the fact that hashing is irreversible, an attacker can only use a brute force attack on a received Get target ID to obtain the content item ID from which it has been derived. The concealment of the content item ID during the Get operation prevents malicious nodes from attacking every Get request. Only requests for content items that have been previously stored on the malicious node can be attacked.

4.13 Unchoking Feature

The definitions of inoffensive and malicious behavior can result in negative ratings also for inoffensive nodes: If an inoffensive node responds to a Lookup query with nodes that are never among the top *numLookupResult* nodes, it receives a negative rating. The same is true if an inoffensive node does not store a content item that is stored on other nodes or if it only stores an older version.

These events occur only occasionally and do not pose a problem in most cases – the results in Chapter 6 show that most inoffensive nodes do not receive a significant amount of negative ratings. However, if high thresholds are used, it is possible that a node's trust values fall below the thresholds. If a node's trust values have once fallen below the thresholds, it is unable to restore its reputation by behaving well because it is not used any longer for any requests and therefore cannot receive any positive ratings anymore.

The only way to recover from such a situation is to obtain a new identity. This, however, must not be legitimate behavior because malicious nodes do the same during a white-washing attack. In addition, the Sybil attack countermeasure should not allow such a behavior.

Therefore, TrustedKad uses a so-called unchoking mechanism. The term “unchoking” is borrowed from BitTorrent’s “optimistic unchoking”: BitTorrent regularly gives a small amount of upload capacity to a randomly selected other node to assess whether the new node offers more download capacity than one of the nodes that currently receive uploads. If this is the case, the old node is abandoned and the new one is used instead [22].

A node uses a certain unchoking probability: This probability denotes during how many trustworthiness evaluations an untrusted node is regarded as trustworthy to give it the opportunity to gain a higher trust value. So if the probability is set to 1%, 1 of 100 trustworthiness evaluations would allow an untrusted node to be used despite its too low trust value. If the evaluated node is trustworthy anyway, the process continues normally.

The unchoking feature allows inoffensive nodes to recover from having received too many negative ratings accidentally. The results in Chapter 6 show that unchoking slightly decreases the success rates that can be achieved. It is required nevertheless because creating a new identity to get rid of negative ratings must not be legitimate behavior under any circumstance: This would enable malicious nodes to conduct attacks until their trust values are too low and then simply start all over again with a new identity.

4.14 Scope, Incentives and Penalties

TrustedKad operates on the KBR and DHT layers. This means that it is application-independent and does not handle application-specific requirements. It also means that TrustedKad cannot give any incentive or penalty to a malicious node: TrustedKad can only ensure that the KBR and DHT operations, i.e., Bootstrap, Lookup, Put and Get, are handled in a secure way. Penalizing malicious behavior on the KBR or DHT layer, e.g., by not responding to incoming queries, must not be performed because this could result in a partitioning of the network. Instead, the application must reward inoffensive behavior and penalize malicious behavior, e.g., as described by Marti and Garcia-Molina in [57]. The ways in which this can be performed have not been analyzed in detail because TrustedKad does not operate on the distributed application layer, so only several brief examples are given in the following.

- A possible way of penalizing malicious behavior in a filesharing application could be to provide a lower upload bandwidth to a node with a low trust value. This encourages the node to contribute to the network to receive a better service. This system can also be combined with existing reward systems on the application layer.
- If the chosen Sybil attack countermeasure includes a differentiation to which degree it is certain that the node is not a Sybil node, in the example of the filesharing application, the bandwidth could depend on that Sybil attack countermeasure as well: If crypto puzzles are used, the provided bandwidth could be the higher the more puzzles have been solved.
- Nodes still in the grace phase should receive a lower quality or even no service at all to prevent whitewashing attacks. Also after the grace phase, the service quality can be linked to the total number of ratings to further secure against the attack.

4.15 Overhead Introduced by TrustedKad and Optimization Potentials

TrustedKad introduces a considerable overhead, especially if it is implemented exactly as described in the previous sections. In the following, the overhead of TrustedKad is presented and optimization potentials are shown and discussed. It should be noted, though, that the optimizations may weaken TrustedKad's security and should therefore be thoroughly analyzed before deployment.

4.15.1 Increases in Size

The size of all existing message types increases because every message includes the certificate of the sender node and the cached trust information and is digitally signed. In addition, the Get and Lookup response messages are extended by a rating acknowledgement. To reduce the message sizes, the cached trust information could be sent only on demand: If a node sends a query to a node of which it only has outdated cached trust information, it can demand that the cached trust information shall be included in the response message. If a node receives a query without cached trust information included in it, it can include the demand for the cached trust information in the response, e.g., if the querying node is new to the queried node. An additional message containing the cached trust information is then sent.

Similarly, the size of a routing table entry increases by the certificate and the cached trust information of the entry. In addition, the size of a Lookup response message increases significantly because it contains the certificates and cached trust information of all nodes included in it. To reduce both the size of a routing table entry and of a Lookup response message, a node could only keep, e.g., the cached trust information of the three trust management nodes that are closest to the mean of all trust management nodes of a node. This would significantly reduce the storage space and message size and would still enable a verification of the cached trust information if one or two trust management nodes are no longer reachable.

4.15.2 New Messages and Requirements

Several extra messages are required by TrustedKad: A new message type is used to send ratings to the trust management nodes. Likewise, a new message type is required to query a trust management node for the trust information of its managed node; this message can also be used by the managed node to query its trust management nodes for its own trust information. The trust management nodes also require a message type to exchange the trust information for replication.

According to Section 4.9.1, the trust information has to be sent to all trust management nodes. To decrease the number of messages, only a subset of the trust management nodes could be chosen, e.g., one per location. As the trust management nodes have to replicate the trust information regularly, the new ratings can be exchanged then.

Similarly, to ensure that ratings are given, Section 4.9.3 describes that Get and Lookup messages are to be forwarded to all trust management nodes. This requirement can be lowered by only forwarding the message to a limited number of the trust management nodes. However, it must be ensured that the actual rating reaches all the trust management nodes that have received the forwarded message to prevent a negative rating from them.

The trust information of a managed node also requires additional storage space at trust management nodes to store the ratings. The number of ratings that have to be verifiable could be limited so that older ratings may be discarded by the trust management nodes.

4.16 Summary of the Components of TrustedKad

TrustedKad is intended as a framework for arbitrary applications. It operates on the KBR and DHT layers and provides secure Bootstrap, Lookup, Get and Put operations to the application layer by using trust mechanisms. It is designed to be application-independent and to be used in open networks that do not have a central management element. It consists of the following components:

- Self-signed certificates: Every node must have a self-signed certificate containing a public key, a timestamp, the node's underlay address and anti-Sybil-attack information. This prevents the sharing of logical identities between nodes and identifies a node even after a node ID change.
- Node ID generation and renewal: The node ID is formed as a hash of the node's certificate. This way, the node cannot freely choose its node ID. In addition, the node ID has a limited lifetime and needs to be renewed regularly.
- Sybil attack countermeasure: TrustedKad uses a modular approach that enables the inclusion of arbitrary Sybil attack prevention methods that are suitable for the network in which TrustedKad is deployed.
- Ratings after transactions: The behavior of nodes is rated after each transaction as either positive or negative.
- Trust values: The ratings are combined into trust values that denote the trustworthiness of a node. There are two types of trust values: One is for routing behavior, the other is for storage behavior. Only nodes that are trustworthy for the respective operation are used for it. In addition, during Get operations, the most trustworthy node group is determined and used to decide the version of the content item that shall be chosen if several versions are returned by the queried nodes.
- Trust value threshold: The nodes decide which nodes to interact with in the future based on the trust values and a threshold. The trust value threshold can be set individually by each participating node, which is why every node has to be able to inform about every existing node and not only those nodes it trusts itself. This is a contrast to existing "clean routing table" approaches. Certainly, a node must only use nodes itself trusts for its own operations, so "clean routing processes" are used by TrustedKad.
- Distributed storage of trust information: TrustedKad defines how the storage location of ratings and trust values is determined. The experiences of all nodes in the network can be used to generate trust because TrustedKad creates a trust system in which the trust information is stored at well-defined places. This way, nodes do not have to rely only on direct experiences with other nodes or "friends of a friend" approaches such as web of trust systems or other hierarchical systems, e.g., certification systems. TrustedKad makes all ratings available to all nodes. Hierarchical systems usually only cover parts of a network (e.g., in certification systems, there are usually several certification roots), whereas TrustedKad enables all nodes to use the experiences of all other nodes.
- Caching of trust information: Secure caching of trust values is used to avoid the need to contact the trust management nodes of every node that is to be used during operations.

- Transparent rating system: TrustedKad ensures that nodes cannot give unauthorized ratings and that trust management cannot fake ratings. Every node that gives a rating must prove that an interaction has taken place and every trust management node must maintain the original rating messages to prove the validity of the ratings.
- Identification of inoffensive and malicious behavior: The rules how to identify whether a node behaves inoffensively or maliciously are defined. For routing, the nodes that help to identify the responsible nodes receive a positive rating; all other nodes receive a negative rating. For storage, the nodes that deliver the chosen version of the content item receive a positive rating. Nodes that deliver another version of the content item or respond with “unknown item” receive a negative rating.
- Concealment of the content item ID: TrustedKad conceals the target ID to protect against nodes that attempt to attack all requests they receive by simply claiming responsibility for the queried content item.
- Unchoking mechanism: Unchoking is introduced to give nodes that have received too many negative ratings the possibility to regain trustworthiness without having to create a new identity.

4.17 How TrustedKad Protects Against Common Attacks

TrustedKad takes existing knowledge about the possible attacks on P2P systems and on trust systems into account and uses appropriate countermeasures that have proven themselves in other contexts and independently from each other. It combines these countermeasures and makes adaptations where necessary to harmonize the different measures and create a consistent system. In addition, Kademlia-specific rules are defined by which inoffensive and malicious behavior is recognized and the unchoking feature is introduced which allows a node to recover from too many false negative ratings.

The components of TrustedKad counter the basic attacks against P2P and trust systems that are described in Section 2.5. Table 3 lists these attacks and the respective countermeasures of TrustedKad. Identity theft and impersonation attacks are countered via the certificates and the asymmetric key pairs, so an attacker cannot abuse another node’s certificate for himself. The Sybil attack is mitigated using a modular approach that depends on the operational environment in which TrustedKad is deployed. The Eclipse attack and attacks on routing and storage are countered using the routing and storage trust values: Only trustworthy nodes are used during routing and storage operations, so mali-

Table 3: Overview of common attacks and TrustedKad's respective countermeasures

Attack	**TrustedKad’s countermeasure**
Impersonation attack, identity theft	Sender authentication
Sybil attack	Modular approach based on the operational environment. Examples: Certification authority, crypto puzzles
Eclipse attack	No free choice of the node ID, Routing and storage trust values
Attacks on routing and storage	Routing and storage trust values, concealment of content item ID
Self-promoting attack	Only one rating per node for one node
Slandering attack	Only one rating per node for one node
Whitewashing attack	Sybil attack countermeasure, unchoking mechanism
Orchestrated attack	Sybil attack countermeasure, routing and storage trust values
Denial of service attack	Distributed storage of trust information
Faking of trust information	Ability to verify ratings at the trust management nodes

cious nodes are avoided. The Eclipse attack is also mitigated by denying the nodes the free choice of their node ID. To prevent a malicious node from claiming that it stores every content item it is queried for, the content item ID is concealed during Get operations.

The self-promoting and slandering attacks aim to manipulate the ratings of the attacker's nodes or inoffensive nodes, respectively. Both attacks are countered by the fact that TrustedKad allows only one routing rating and one storage rating per node for one node. In combination with the Sybil attack countermeasure that prevents that an attacker can create an unlimited amount of identities, this countermeasure mitigates the two attacks on the trust values. The Sybil attack countermeasure also prevents whitewashing and orchestrated attacks. The unchoking mechanisms prevents that inoffensive nodes have to act as if they perform a whitewashing attack in order to recover from too many negative ratings. Routing and storage trust values also help against orchestrated attacks: In Chapter 6, several configurations show that TrustedKad is able to identify nodes that change their behavior from inoffensive to malicious.

Denial of service attacks against a central entity that stores trust values are impossible because TrustedKad uses a distributed storage scheme for the trust information. These nodes are located at different positions in the logical address space and most probably are also widely distributed physically. Hence, a denial of service attack against the distributed storage system cannot be concentrated on one physical network location, but needs to be spread across multiple physical locations. The transparent rating system enables nodes to verify that the trust management nodes do not create fake ratings.

5 Implementation

Simulations are performed to evaluate if TrustedKad is able to identify inoffensive and malicious nodes correctly and if this classification increases the success rates of the operations in the network. The simulations are conducted using the OverSim framework presented first by Baumgart et al. in [39] and also later by Baumgart in his dissertation thesis [48].

OverSim is based on OMNeT++ [40], an event-based simulation environment. It is written in C++ and its source code is freely available. The details of OverSim's functioning had to be reconstructed from the source code because, regrettably, the documentation only consists of a brief introduction on the project website, a few comments in the source code itself and the overviews given by [39] and [48].

The framework is designed in a modular way and already includes modules for several DHT algorithms, with Kademlia being one of them. As S/Kademlia [53] has also been presented by Baumgart et al., OverSim includes its improvements as well. OverSim has been chosen because its basis OMNeT++ is widely accepted as a valid simulation environment in the scientific community [40].

In the following, the intention of the simulations is explained. The components of OverSim that are relevant for the evaluation of TrustedKad and the attacks that are performed in the simulations are introduced. For the evaluation, several of the existing components have been changed and extended; these extensions are also presented. Finally, the behavior of the simulated nodes and the management of the trust information are explained in detail.

5.1 Focus of the Simulations

The simulations with OverSim are used to show that TrustedKad achieves its goal of identifying and avoiding malicious nodes and thereby improves the security of the network by mitigating known attacks on routing and storage. All the measures described in Chapter 4 are necessary to create a secure environment: They depend on each other and enhance the network's security in a transparent and distributed way together. However, the key element is the calculation of the trust values and thereby the identification and avoidance of malicious nodes, as it forms the basis of TrustedKad. Hence, this aspect is thoroughly analyzed in this thesis to show that TrustedKad successfully mitigates attacks against P2P and trust systems. In order to be able to analyze the trust value calculation without any disturbance caused by effects of other components, the other components are partly replaced, e.g., by central components.

Additionally, the simulations only include components that are necessary to prove TrustedKad's functioning in order to reduce the resource usage of the simulations. For example, the simulated nodes do not generate and verify key pairs and certificates. Instead, the effects of the keys and certificates are simulated: For example, randomly generated node IDs are assigned to all nodes, also the fake ones.

The simulations do not implement the distributed storage of the trust information, but use a central trust management instance instead. The reason for this is that the analyses aim to show that the definitions of inoffensive and malicious behavior during routing and storage operations are reasonable and lead to an improvement of the network's performance. The effectiveness of these rules has to be analyzed first and without any interference that could, e.g., be caused by the distributed storage of the trust information. The same is true for other effects such as node ID changes and churn, which are not included in the simulations either. Due to the fact that the trust information is stored centrally, the caching of the trust information is also not included in the simulations, as well as the trust information replication and verification features.

TrustedKad is designed to be application-independent, so a simple DHT test application is used which uses direct storage of content items. This is comparable, e.g., to a VoIP network such as Skype that stores small pieces of information such as the IP address and port at which the user can be reached for a direct voice (or video) connection.

5.2 Introduction to OverSim

OverSim is designed in a modular way. The modules resemble the DHT layer structure as introduced in Section 2.3.2. Due to the modularity, the class hierarchy of OverSim makes intense use of inheritance: For example, algorithm classes such as the Kademlia class are derived from a base class called BaseOverlay and override the relevant methods of it. The same is true for application classes which are derived from the BaseApp class. For the same reason, a multitude of helper classes exists, e.g., representing messages or routing table entries. Additional helper classes have been defined for TrustedKad, e.g., to represent ratings and the trust information of a node.

Figure 20 shows the relevant classes of OverSim. The SimpleUnderlay class forms the underlay network. It does not simulate the complete TCP/IP protocol stack including routers; instead, it uses a database of the results of Internet latency measurements to delay the transmission of messages between nodes [48]. OverSim would allow using the INET framework [74] of OMNeT++ as well to simulate the complete protocol stack and routing, but this is not necessary to evaluate TrustedKad and would only waste computing time.

An overlay node in OverSim consists of several modules: The Kademlia module implements the Kademlia algorithm and the routing table. It contains methods to define which nodes are responsible for which other nodes or that perform the Bootstrap. The IterativeLookup class is derived from the base class AbstractLookup. An instance of it is created for every Lookup that is performed by a node; this instance controls the Lookup process by sending Lookup requests and handling the responses. Incoming Lookup requests that originate from IterativeLookup instances of other nodes are handled by the Kademlia class. Communication between two nodes always uses the methods provided by the SimpleUnderlay class.

OverSim supports using up to three layers for applications; for the evaluation of TrustedKad, only two of them are required. The DHT module is operating on the lowest of

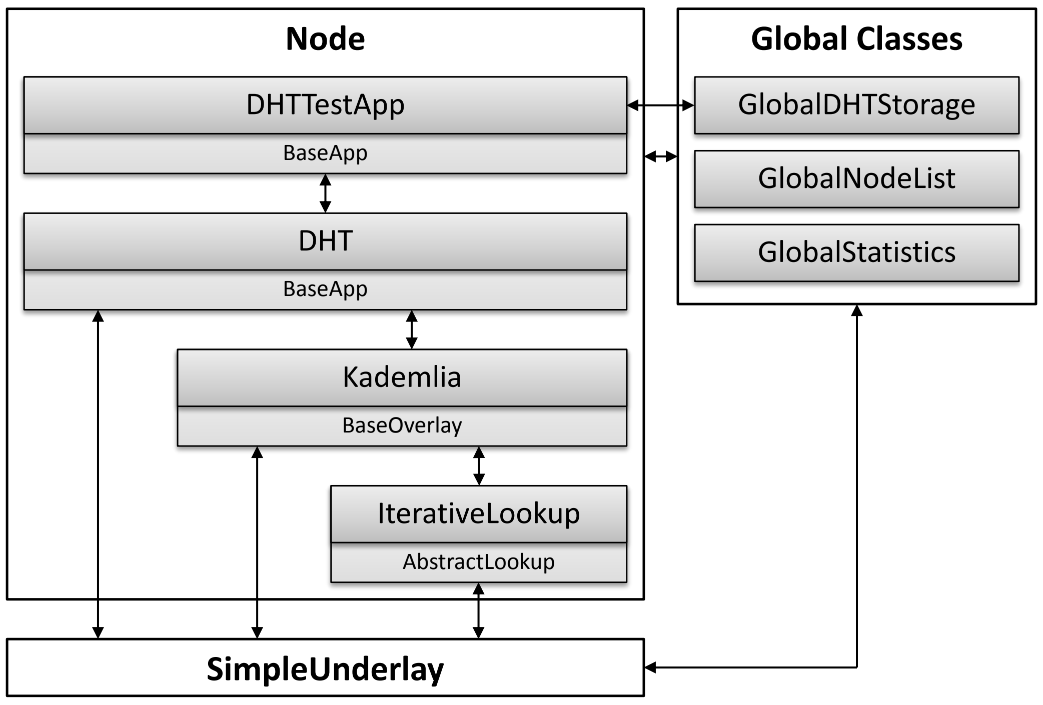


Figure 20: Relevant OverSim classes (based on [48])

these layers. It handles the internal requests from the DHTTestApp module and also responds to incoming Get and Put requests from other nodes.

There are several global classes that are used for maintenance purposes: The GlobalNodeList class contains information about all existing nodes and is used to get a Bootstrap node for a joining node, for example. It also contains the central trust information storage. The GlobalStatistics class collects statistical information from all other classes (including the SimpleUnderlay class) and stores it at the end of each simulation run. The GlobalDHTStorage class belongs to the DHTTestApp and is used to store information about all content items that exist in the DHT to enable nodes to query only for existing content items and to compare the received content item value with the originally stored one.

5.3 Attacks Performed in the Simulations

The simulations are used to show that TrustedKad's trust mechanisms protect against known attacks on routing and storage operations. These attacks are aimed against the four basic DHT operations: Bootstrap, Lookup, Put and Get.

5.3.1 Attacks Against Bootstrap and Lookup Operations

As explained in Section 4.10.1, the Bootstrap needs to be especially protected because if a node joins the network via a malicious node, the malicious node can control the view the joining node has of the network. This usually results in a network that consists of several partitions.

Kademlia uses a Lookup operation during the Bootstrap to learn about other nodes in the network, so the attacks that are directed against the Lookup process also harm the Bootstrap process. Lookups are also used by Put and Get operations to identify the responsible nodes, so if the Lookup fails, so does the following operation.

There are two basic attacks against Lookup operations: Discarding the Lookup request and delivering false routing data. By not responding to Lookup requests, an attacker can harm the network because the querying node waits for the response for a certain amount of time. If this happens several times during a Lookup, the Lookup timeout limit can be exceeded which causes the Lookup to fail. Kademlia is quite resilient against this attack type because it uses several parallel requests: Hence, if one request times out, the results of the other parallel requests still help in identifying the target nodes [48].

So compared to discarding incoming Lookup requests, responding with nodes that do not contribute to finding the target nodes causes more harm to the routing process. The attack is most effective if the response contains non-existing nodes that are very close to the target so that they take the first places in the Lookup list and are queried next. Then, all of these queries time out and the Lookup fails because the total Lookup timeout is exceeded [48]. This is also the way in which the attack works that has been published by the author in [1]. If TrustedKad is used, this most effective form of the attack is no longer possible because the node IDs cannot be chosen freely. This greatly reduces the impact of this attack.

For the analyses, the malicious nodes respond with fake nodes whose node IDs are randomly distributed. As TrustedKad uses anti-Sybil-attack information in the certificates and frequent re-generation of the node ID, these fake nodes are identified as such and ignored. However, also a worst-case scenario is analyzed in which the malicious nodes are able to fake the anti-Sybil-attack information in the cached trust information as described in Section 4.8.3 and possess a large amount of computing resources to perform the attack sketched in Section 4.10.4.2.

5.3.2 Attacks Against Put and Get Operations

Even if the Lookup performed for the Put or Get operation is successful, the operation can still fail due to attacks of the responsible nodes. Attacks against the Put operation are easy to conduct: A malicious node can send a positive response after receiving a Put request and delete the content item stored on it immediately afterwards. This is the reason why TrustedKad does not use a separate trust value for Put operations, but only a joined storage trust value: Only nodes that have proven to return content items correctly shall be used to store new items on. In addition, this attack against the Put operation does not differ from an attack against the Get process in which the malicious nodes responds that it does not know the queried content item, so it is not analyzed separately [48].

If a malicious node only claims that the queried content item does not exist, the attack is not successful if the querying node finds at least one inoffensive node that responds with the correct content item. However, the attack's impact can be increased by returning fake content items, especially if the malicious nodes collude: In a system that uses a majority vote to decide which content item version is the correct one, there would have to be more inoffensive nodes than malicious ones to have the attack fail. It is possible that a single inoffensive node does not suffice anymore as it is the case if the malicious nodes only claim that the content item does not exist [48].

TrustedKad does not use a majority decision, but instead chooses the content item version that is returned by the most trustworthy node group (cf. 4.11.2). The effectiveness of this approach is analyzed in Section 6.5.

In the simulations, colluding and non-colluding malicious nodes that deliver fake content items are analyzed. The effectiveness of the concealment of the target ID during Get operations (cf. 4.12) is also shown by comparing it to operations that do not conceal the target ID.

5.4 Adaptations to OverSim

The existing code of OverSim has been changed and extended in many ways. Several classes have been added to represent ratings, trust information of a node and to evaluate a node’s trustworthiness. The output of statistical data about both existing and added components has been enhanced; for example, a more detailed error reporting has been added to be able to analyze the different reasons for the failure of Put and Get requests. The existing classes and methods have been extended so that they use trust information when interacting with other nodes and that the ratings are given after the interactions.

In addition to the new classes and actions required by TrustedKad, a lot of the details of the nodes’ behavior have been changed or extended. Basically, OverSim already contains the different attack types that are evaluated for this thesis. However, several extensions have been included: For the storage attack, the original OverSim only uses colluding nodes, so the option of having non-colluding malicious nodes has been added. Also the option to send the hash of the original content item but a fake value when asked for it with a GetValue request has been added for this thesis. This required the addition of a verification of whether the value matches the previously chosen hash; this verification is not performed in the original OverSim. Another option has been added to only attack Get requests if the queried content item has been stored on the malicious node before.

The available options of the attack intensity of malicious nodes have also been enhanced: The original OverSim only supports malicious nodes that attack all the time. For this thesis, options have been added that have malicious nodes attack only some requests sent to them or only after a certain amount of time. This allows checking whether TrustedKad is able to identify changing behavior of malicious nodes that is used to improve the node’s own trust value.

The invalidNodesAttack (cf. 5.5.4) has been changed to incorporate the fact that the nodes cannot choose their node IDs freely: Before, the node IDs of the non-existing nodes have been chosen to be very close to the target ID, which is not possible any longer in TrustedKad due to the hashing of the certificate to create the node ID.

The reaction to Put and Get responses has also been changed: Previously, it has been sufficient if half of the required amount of nodes have responded for a Put or Get operation to succeed. With TrustedKad enabled, this does not suffice any longer because only responding nodes are rated, so at least *numReplica* candidate nodes are queried. In addition, TrustedKad attempts to store and retrieve *numReplica* replicas, so if some requests time out or some Get responses state that the content item does not exist while others return an existing item, more queries are sent until either *numReplica* responses have been received or there are no more candidate nodes left to query.

Lookup operations have been changed to return twice as many nodes as requested. This is required if the Lookup is performed for Put or Get operations because there can be nodes among the returned nodes that are not trusted for storage operations. In order to cope with that, twice as many nodes are returned so that half of the nodes can be filtered out without reducing the number of candidate nodes for the actual storage operation.

Due to the fact that a node can choose not to use the node provided by the GlobalNodeList for Bootstrap because it is not regarded as trustworthy, it has been necessary to incorporate a functionality to retry a Bootstrap. This was not needed before because the Bootstrap has been performed with the provided node in any case. Additionally, an option has been added to enforce that Bootstrap operations are only performed using inoffensive Bootstrap nodes in order to prevent network partitioning. It should be noted that this option does not necessarily result in all Bootstraps succeeding at the first try: Even if the node is inoffensive, its routing trust value can still be too low so that the bootstrapping node refuses to use it.

Most of the changes and extensions have been implemented in a modular way so that they can be enabled and disabled independently from each other. This allows for a distinct analysis of the effectiveness of each of TrustedKad's components.

5.5 Behavior of the Nodes

In the following, the behavior of the simulated nodes is explained. The DHTTestApp initiates the Put and Get operations (and thereby also Lookups) that are evaluated in Chapter 6. The operations that originate from a node are explained as well as the handling of incoming messages.

Malicious nodes behave correctly in every way except for the attack they conduct. This means that they perform Lookup, Put and Get requests in the same way as inoffensive nodes do. In addition, malicious nodes that only attack routing processes do not attack storage processes and vice versa.

5.5.1 DHTTestApp

The DHTTestApp of OverSim is used to evaluate TrustedKad. Each node has one DHTTestApp instance that generates content items with random IDs and values and stores them in the DHT and in the GlobalDHTStorage. The Put process is performed in regular intervals, controlled by a timer.

A Get process is initiated by every node in regular intervals as well: For this, the hash of a randomly selected content item is chosen from the GlobalDHTStorage and the node attempts to retrieve its value. This way, only existing content items are attempted to be retrieved, leading to an expected success ratio of 100%.

Figure 21 graphically shows both the Put and the Get process of DHTTestApp. At the end of each process, statistical information is recorded using the GlobalStatistics module.

5.5.2 Performing a Bootstrap Operation

The first operation a node has to conduct is the Bootstrap operation. Figure 22 illustrates the actions that are performed during a Bootstrap: If the node is the first node in the network, the node behavior is set to inoffensive. No further action is required and the node regards itself as bootstrapped.

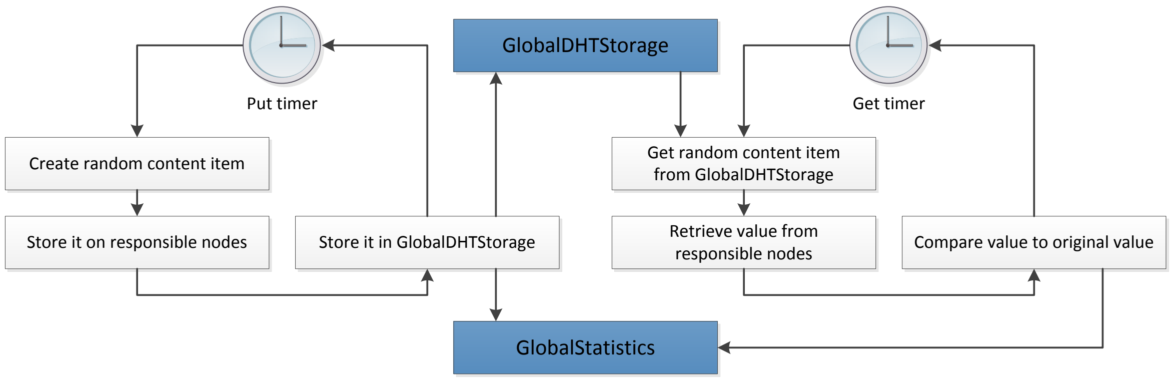

Put timer
GlobalDHTStorage
Get timer
Create random content item
Get random content item from GlobalDHTStorage
Store it on responsible nodes
Store it in GlobalDHTStorage
Retrieve value from responsible nodes
Compare value to original value
GlobalStatistics

Figure 21: Flowchart of the DHTTestApp processes

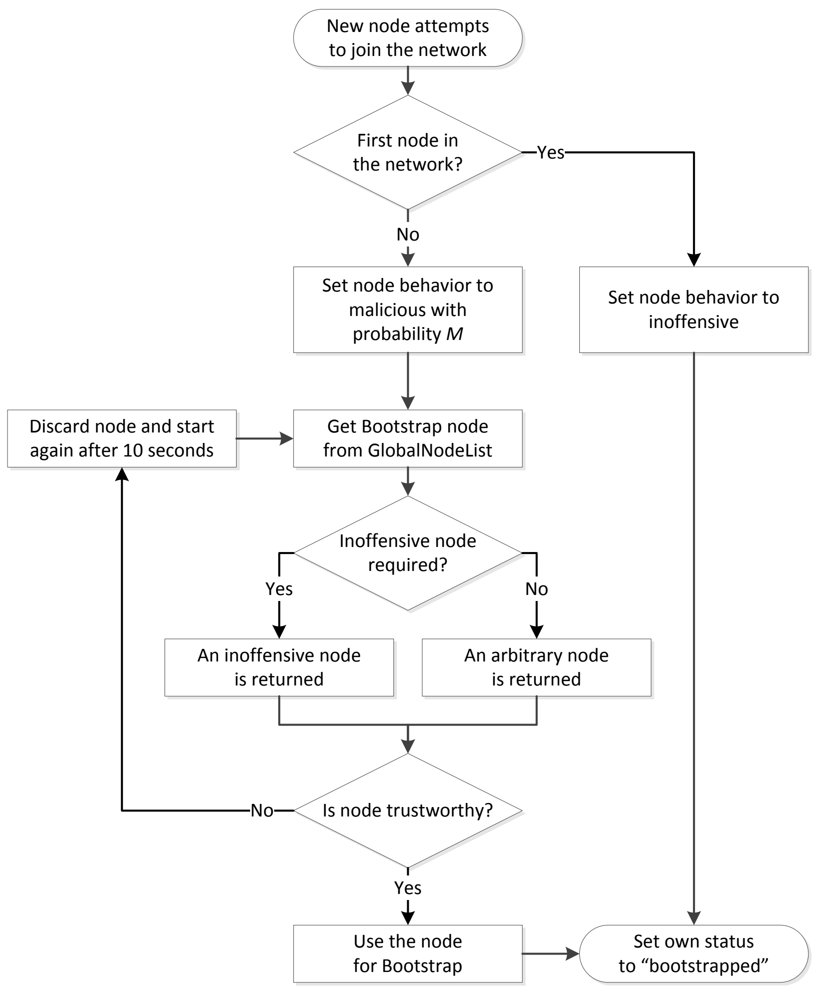

New node attempts to join the network
First node in the network?
Yes
No
Set node behavior to malicious with probability *M*
Set node behavior to inoffensive
Discard node and start again after 10 seconds
Get Bootstrap node from GlobalNodeList
Inoffensive node required?
Yes
No
An inoffensive node is returned
An arbitrary node is returned
No
Is node trustworthy?
Yes
Use the node for Bootstrap
Set own status to "bootstrapped"

Figure 22: Flowchart of the Bootstrap operation

From the second node on, the node behavior is set to malicious with a probability of M; M denotes the fraction of malicious nodes in the network. Then, a node retrieves an existing node from the GlobalNodeList for the Bootstrap. A configuration parameter has been added to be able to configure that only inoffensive nodes are returned by GlobalNodeList for Bootstrap operations to prevent network partitioning. The Bootstrap candidate node is checked for trustworthiness. As explained in Section 4.10.1, no grace ratings (and no unchoking) are used during the Bootstrap, so g_R is set to 0. This leads to the fact that nodes without any routing rating (such as when the second node in the network attempts to use the first node for Bootstrap) are regarded as trustworthy. As soon as a node has received at least one routing rating, the trust value is calculated and evaluated without any grace ratings. If the node is not trustworthy, it is discarded and the process repeats after 10 seconds.

If the node is trustworthy, it is used for the Bootstrap. In order to fill its own routing table, the node sends a Lookup query with its own node ID as target ID to the Bootstrap node. Choosing its own node ID as target ID enables the node to learn about other nodes in its vicinity and especially about its logical neighbors, as the final nodes in the Lookup list will be the nodes that are closest to it. Finally, the node changes its status to "bootstrapped".

5.5.2.1 Discussion of the Partitioning Probability and Number of Partitions

The probability that a new partition is created when a node joins the network depends on whether its bootstrap node is malicious. If it is malicious, a new partition is created; if the bootstrap node is inoffensive, an existing partition is extended by the new node. The expectation value of the number of partitions can be calculated as described in the following.

The first node is always an inoffensive node. All other nodes can be malicious with a probability of M. The first node of the network forms the first partition. The second node uses the first node for the Bootstrap and extends the first partition. From then on, a new partition can be created during every Bootstrap if the bootstrap node is a malicious one. Therefore, the number of partitions depends on the number of malicious nodes: The more malicious nodes are contained in a network, the more partitions there will be.

For example, if the second node that joins the network is malicious and the third node bootstraps from the second one, a second partition is created. For the Bootstrap process of the third node, the probability that a new partition is formed is

$$\frac{1}{2} \times 0 + \frac{1}{2} \times M = \frac{1}{2} \times M.$$

Both existing nodes are chosen as bootstrap nodes with the same probability of $\frac{1}{2}$. The first node is not malicious; therefore, no new partition is created if it is chosen. With a probability of M, the second node is malicious. If it is, a new partition is created.

The expectation value of the number of partitions after the third node has joined the network is $1 + \frac{1}{2} \times M$: One partition exists from the start and the probability that a second partition is created is $\frac{1}{2} \times M$. For M = 5%, this would result in an expectation value of 1.025 partitions after the third node has joined the network.

In general, the probability that a new partition is formed when the n-th node joins the network can be calculated as $\frac{n-2}{n-1} \times M$: There are n-1 nodes in the network. The first one is not malicious, so only n-2 can be malicious with a probability of M. Furthermore, the expectation value of the number of partitions in a network with N nodes can be computed as follows:

$$Expectation\ Value = 1 + \sum_{n=3}^{N} \frac{n-2}{n-1} \times M$$

The limit value of the fraction is $\lim_{n\to\infty} \frac{n-2}{n-1} = 1$. Therefore, the expectation value can be estimated as follows:

$$Estimation\ of\ Expectation\ Value = 1 + \sum_{n=3}^{N} 1 \times M = 1 + (N-2) \times M.$$

So, for 1,000 nodes and M = 5%, the expectation value of the number of partitions is about $1 + (1000 - 2) \times 0.05 = 50.9$.

5.5.3 Performing a Lookup Operation

Lookups are initiated by methods in the Kademlia class and its base class BaseOverlay. They create an IterativeLookup class instance for every Lookup that performs and controls the actual Lookup operation. Lookups are required by Put and Get operations and by routing table maintenance processes. The number of nodes that shall be returned by the

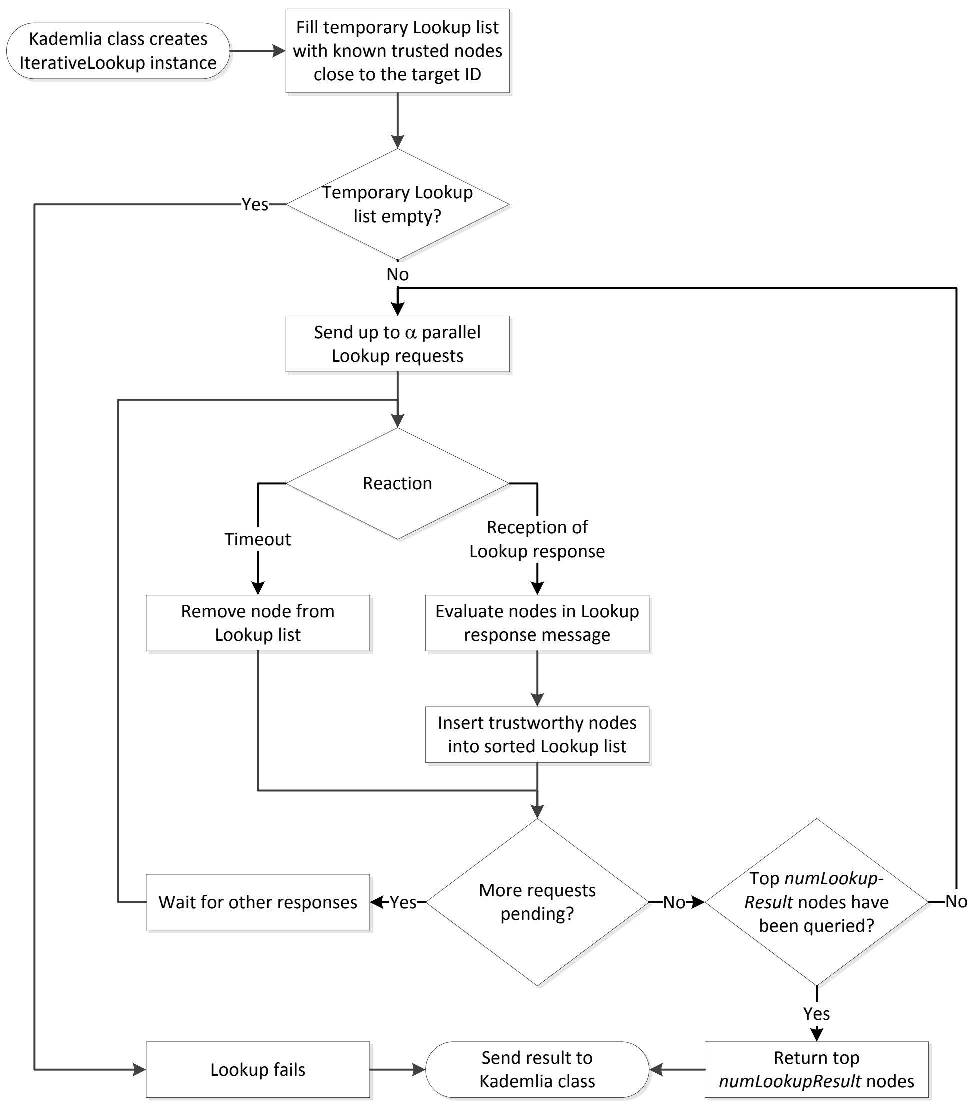

Kademlia class creates IterativeLookup instance
Fill temporary Lookup list with known trusted nodes close to the target ID
Temporary Lookup list empty?
Yes
No
Send up to α parallel Lookup requests
Reaction
Timeout
Reception of Lookup response
Remove node from Lookup list
Evaluate nodes in Lookup response message
Insert trustworthy nodes into sorted Lookup list
Wait for other responses
Yes
More requests pending?
No
Top numLookup-Result nodes have been queried?
No
Yes
Lookup fails
Send result to Kademlia class
Return top numLookupResult nodes

Figure 23: Flowchart of the Lookup operation

Lookup is determined by the respective operation for which the Lookup is performed, e.g., it depends on the number of replicas from which the content item shall be retrieved. However, in order to cope with untrusted nodes, a Lookup attempts to identify twice as many nodes as requested. For the simulations for this thesis, four replicas of a content item are stored and retrieved, so the number of nodes returned by the IterativeLookup class instance is eight. This way, half of the nodes can be untrusted without impeding the operation, enabling the node to handle networks with about 50% of malicious nodes.

As shown in Figure 23, at the beginning of a Lookup, the temporary Lookup list is filled with the nodes that the querying node has in its routing table and that are closest to the target ID. Due to the fact that the routing table may contain untrusted nodes, the Lookup list is filtered so that it only contains nodes that are trustworthy for routing. If no trustworthy candidate nodes are known, the Lookup is cancelled.

For the evaluations, the Kademlia parameter α that denotes the number of parallel queries is set to Kademlia's default of 3. When 3 requests have been sent, the next requests are sent out when all 3 responses have been received or the requests have timed out. The trustworthiness of the nodes contained in the responses is evaluated and only trustworthy nodes are inserted into the sorted Lookup list. In several configurations, the fake nodes generated by malicious nodes with the invalidNodesAttack parameter enabled are regarded as trustworthy during the simulations. This way, it can be analyzed whether TrustedKad can counter the attack even if the attacker is able to fake the trust information as described in Section 4.8.3.

OverSim stores twice as many nodes in the Lookup list as shall be identified for the operation for which the Lookup is performed. So, the list contains four times the number of nodes requested by the calling method. As eight nodes shall be returned in the simulations for this thesis (*numReplica* = 4), the Lookup list contains up to 16 nodes. Information about more nodes than required is maintained in order to cope with non-answering nodes.

When the first eight nodes have all been queried and do not change anymore, the Lookup terminates and returns these nodes. OverSim's default timeouts for the whole Lookup process and single requests are 10 and 1.5 seconds, respectively, which are used for this thesis as well.

If a Lookup fails, it is impossible to tell whether it fails due to not enough trustworthy nodes or due to not enough nodes at all, as the route to the target nodes can differ from the start depending on the trust values of the nodes, so it is not possible to differentiate between "not enough nodes" and "not enough trustworthy nodes" in the statistics.

5.5.3.1 Rating Procedure for the Lookup Operation

For the rating procedure, two lists are maintained during a Lookup:

- Source list: Contains information about which node has informed about which other node.
- Temporary rating list: Contains the ratings that are given to the involved nodes at the end of the Lookup.

When a response is received from another node, this node is added to the temporary rating list with a default negative rating. All the nodes it informs about are added to the source list (if not already contained in it), and the node is added as a source for those nodes. Additionally, the temporary ratings of the nodes that have informed about the responding node are changed to positive, as they have informed about an existing and responding node (see "bottom-up rating" explanation in Section 4.10.3).

At the end of a Lookup, the source list is used to change the temporary ratings of the nodes that have informed about the result nodes to positive. According to the "top-down" rating approach, this is repeated until the node that gives the ratings reaches itself in the hierarchy. Finally, the temporary rating list is used to give the ratings to all involved nodes.

5.5.4 Handling of an Incoming Lookup Request

Incoming Lookup requests are handled by the Kademlia class and its base class BaseOverlay. Three parameters can be used to configure the behavior of malicious nodes:

- ignoreLookupsAttack: If set to true, the node does not respond to incoming Lookup requests. Due to the parallel Lookups, this attack is not very effective in a Kademlia-based network, so it is not performed during the attacks that are evaluated in this thesis. In addition, a node that does not respond to Lookup requests is removed from the routing table as it is regarded as offline.
- isClosestAttack: If set to true, the response contains only the malicious node itself indicating that the node claims to be responsible for the content item and does not know any closer nodes.
- invalidNodesAttack: If set to true, the response contains only non-existing nodes.

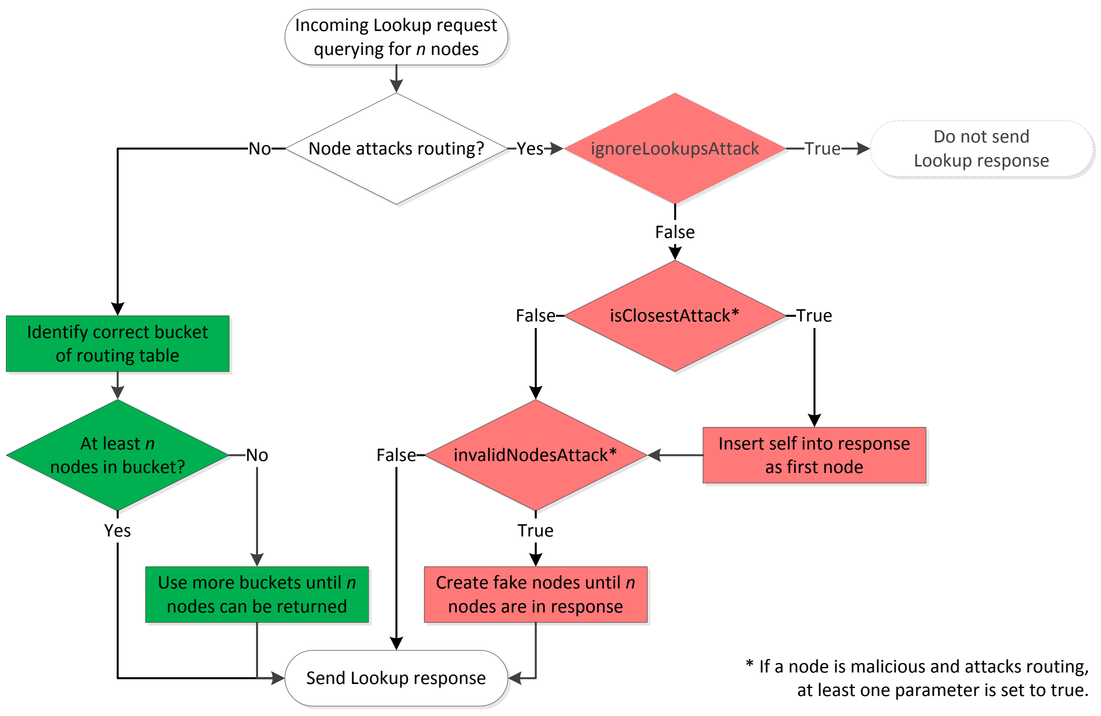

Incoming Lookup request querying for *n* nodes
Node attacks routing?
No
Yes
ignoreLookupsAttack
True
Do not send Lookup response
False
isClosestAttack*
False
True
Identify correct bucket of routing table
At least *n* nodes in bucket?
No
Yes
invalidNodesAttack*
False
Insert self into response as first node
True
Use more buckets until *n* nodes can be returned
Create fake nodes until *n* nodes are in response
Send Lookup response
* If a node is malicious and attacks routing, at least one parameter is set to true.

Figure 24: Flowchart of the handling of an incoming Lookup request

- isClosestAttack and invalidNodesAttack: If both parameters are set to true, the response contains the malicious node itself and non-existing nodes, but no other existing nodes.

An inoffensive node uses the distance between the target ID and its own node ID to determine which bucket of the routing table contains the required next hop nodes for the querying node. If there are not enough nodes in that bucket to return as many nodes as demanded by the Lookup request, more buckets are used until enough nodes can be returned. Figure 24 presents the handling of incoming Lookup requests graphically.

OverSim's handling of incoming Lookup requests has been extended to incorporate TrustedKad's improvements described in Section 4.10.4: In a network without TrustedKad, the malicious nodes can freely choose the node IDs of the fake nodes, so the IDs are chosen to be very close to the target so that they take the top positions of the Lookup list. In TrustedKad, however, the node IDs cannot be chosen freely, so the attack has been changed to take random node IDs for the fake nodes (because as explained in Section 5.1, certificates are not fully implemented in the simulation to save computing resources).

5.5.5 Performing a Put Operation

When the DHTTestApp initiates a Put request due to a timer event, it calls a method of the DHT class to perform the Put operation. The DHT class uses the Kademlia class to execute the Lookup which in turn creates a new instance of the IterativeLookup class that actually carries out the required Lookup. Figure 25 illustrates the Put operation.

The DHT class demands *numReplica* = 4 candidate nodes from the Lookup. As mentioned before, the Lookup attempts to locate up to 2 × *numReplica* = 8 nodes in order to cope with nodes that are not trustworthy for storage operations.

The nodes identified by the Lookup are filtered so that only nodes that are trustworthy for storage remain as candidate nodes. If the Lookup fails to find any trustworthy responsible nodes, this is reported to the DHTTestApp which records this operation as failed. If

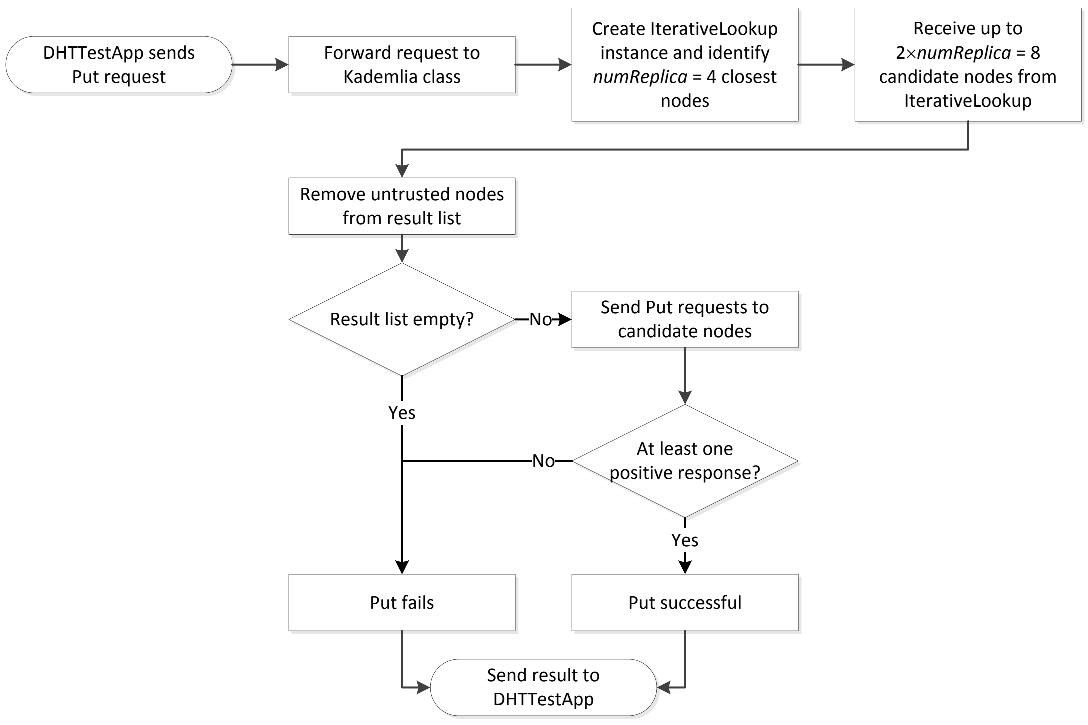

DHTTestApp sends Put request
Forward request to Kademlia class
Create IterativeLookup instance and identify *numReplica* = 4 closest nodes
Receive up to 2×*numReplica* = 8 candidate nodes from IterativeLookup
Remove untrusted nodes from result list
Result list empty?
No
Send Put requests to candidate nodes
Yes
At least one positive response?
No
Yes
Put fails
Put successful
Send result to DHTTestApp

Figure 25: Flowchart of the Put operation

at least one candidate node is found, the DHT class sends Put requests to the candidate nodes to store the content item on them. Up to *numReplica* Put requests are sent, starting with the node whose ID is closest to the content item ID.

If at least one node replies with a positive response, the content item is stored in the GlobalDHTStorage and the Put operation is recorded as successful. If no node responds positively, the Put operation fails.

No storage ratings are given during a Put operation because a node can respond positively and delete the content item immediately afterwards as explained in Section 4.5. Routing ratings are given at the end of the required Lookup process.

5.5.6 Handling of an Incoming Put Request

The DHT class handles incoming Put requests. All nodes, inoffensive and malicious ones, act in the same way as shown in Figure 26: The content item contained in the Put request is stored in the local storage of the node and a positive response is returned. Attacks on storage are performed only during the processing of Get requests.

5.5.7 Performing a Get Operation

Just as a Put operation, a Get operation is also initiated by the DHTTestApp due to a timer event. The DHTTestApp randomly chooses one content item from the GlobalDHTStorage

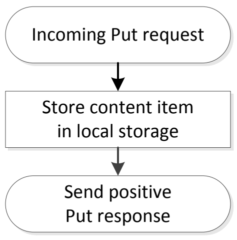

Incoming Put request
Store content item in local storage
Send positive Put response

Figure 26: Flowchart of the handling of an incoming Put request

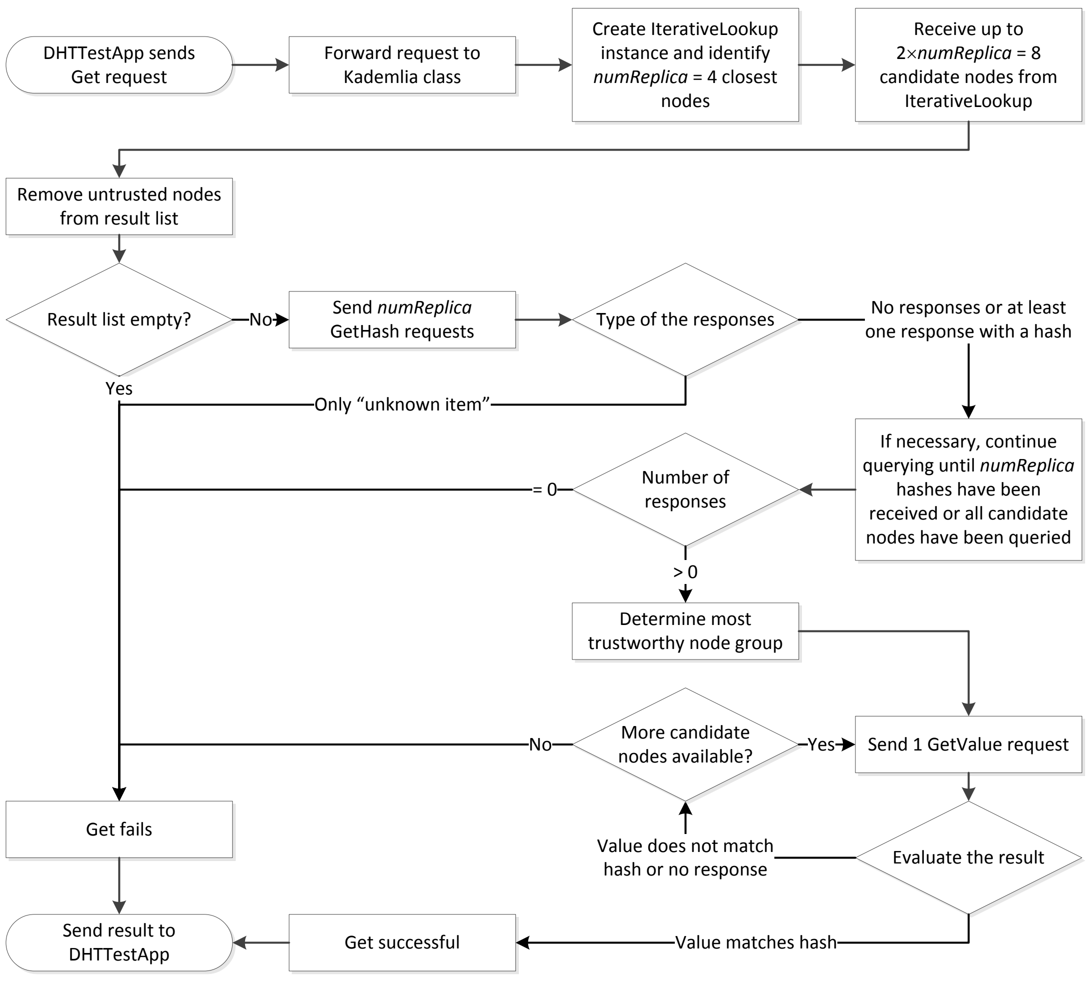

DHTTestApp sends Get request
Forward request to Kademlia class
Create IterativeLookup instance and identify *numReplica* = 4 closest nodes
Receive up to 2×*numReplica* = 8 candidate nodes from IterativeLookup
Remove untrusted nodes from result list
Result list empty?
No
Send *numReplica* GetHash requests
Type of the responses
No responses or at least one response with a hash
Yes
Only "unknown item"
If necessary, continue querying until *numReplica* hashes have been received or all candidate nodes have been queried
Number of responses
= 0
> 0
Determine most trustworthy node group
More candidate nodes available?
No
Yes
Send 1 GetValue request
Get fails
Value does not match hash or no response
Evaluate the result
Send result to DHTTestApp
Get successful
Value matches hash

Figure 27: Flowchart of the Get operation

and asks the DHT application to retrieve its value. Figure 27 illustrates the Get operation.

The DHT class uses the Kademlia class to identify the *numReplica* = 4 closest nodes to the content item ID. Similar to the Put operation, the DHT class creates a new IterativeLookup instance to perform the Lookup and to locate up to 2 × *numReplica* = 8 candidate nodes; more nodes are returned by IterativeLookup to cope with nodes that are not trustworthy for storage operations.

The DHT class removes nodes from the result list that are not trustworthy for storage. If the result list is empty afterwards, the Get operation fails. Otherwise, GetHash requests are sent to the first *numReplica* candidate nodes. More nodes are queried if a node does not respond to the GetHash request.

The nodes that respond with "unknown item" are counted separately. If all of the first *numReplica* nodes respond with "unknown item", this is taken as the result. This is recorded as a failed Get operation, as only existing content items are attempted to be retrieved. The failure reasons are also recorded so that it is possible to analyze the ratios of the failure reasons.

If at least one of the first *numReplica* nodes replies with a hash, more nodes are queried for the hash until *numReplica* nodes have responded with a hash or until all 2 × *numReplica* nodes have been queried. If none of the 2 × *numReplica* candidate nodes responds at all, the Get operation fails.

If at least one node has responded with a hash, the most trustworthy node group is chosen to be queried for the value of the content item as explained in Section 4.11.2. A GetValue message is sent to one of the nodes of the group. The value returned by it is verified by calculating the hash of the delivered value and comparing it to the hash previously received by the GetHash responses. If the value matches the hash, the Get operation is successful. If the node does not respond or if the value does not match the hash, the next node in the node group is queried. If no node responds or returns a matching value, the Get operation fails.

5.5.7.1 Rating Procedure for the Get Operation

Two lists are maintained during a Get process to give the ratings at the end of the operation:

- A list of all received hashes and the nodes that have sent them and
- a list of all nodes that have responded to a GetValue request with a value that does not match the hash or that have not responded to the GetValue request.

At the end of the Get operation, all nodes that have delivered the hash of the chosen version of the content item receive a positive rating, except for nodes that have not answered the GetValue request for that hash. All nodes that have returned other hashes, a mismatching value or no value at all receive a negative rating.

If the chosen content item version is not the originally stored one, the nodes still receive a positive rating because the querying node is unable to detect this – the detection is only possible in the simulation. In the evaluation chapter, these Get operations are shown as "false positives".

5.5.8 Handling of an Incoming Get Request

Incoming Get requests are handled by the DHT class. In order to simplify the illustration,

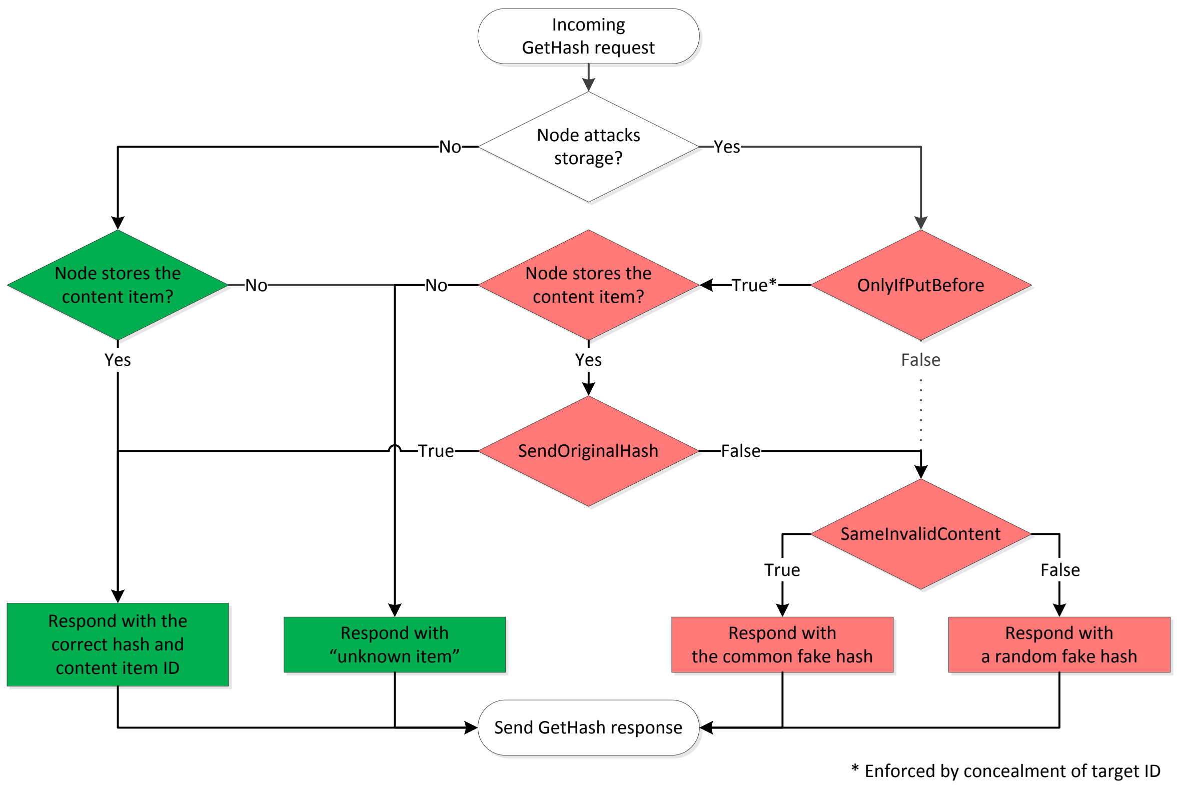

Incoming GetHash request
Node attacks storage?
No
Yes
Node stores the content item?
No
No
Node stores the content item?
True*
OnlyIfPutBefore
Yes
Yes
False
True
SendOriginalHash
False
SameInvalidContent
True
False
Respond with the correct hash and content item ID
Respond with "unknown item"
Respond with the common fake hash
Respond with a random fake hash
Send GetHash response
* Enforced by concealment of target ID

Figure 28: Flowchart of the handling of an incoming GetHash request

the GetHash and the GetValue processing is split into two figures, although both message subtypes are handled by one method in the source code. So, Figure 28 shows the processing of a GetHash message and Figure 29 the processing of a GetValue message.

Malicious nodes that attack storage operations by delivering fake content items can act in different ways and can be configured by the following options:

- OnlyIfPutBefore: If set to true, a malicious node only responds with fake content if the content item has been stored on it before. This option is set to true for the simulation runs that evaluate TrustedKad because due to the concealment of the content item ID introduced in Section 4.12, a malicious node can only perform its attack if the content item has been stored on it before. If it has not, the node responds with "unknown item".
- SendOriginalHash: Requires OnlyIfPutBefore. If set to true, a malicious node returns the hash of the original content item if queried for the hash, but returns a fake content item if asked for the value.
- SameInvalidContent: If set to true, all malicious nodes send the same fake content item hash or value. Otherwise, each of them sends a randomly generated one.

When an inoffensive node receives a GetHash request, it responds with the hash of the content item if it has been stored on it previously; otherwise, it responds with "unknown item", as shown in Figure 28. If the node attacks storage operations, the OnlyIfPutBefore parameter is evaluated first. If it is set to true and the node does not store the content item, it can only respond with "unknown item": If TrustedKad is used, a malicious node that does not store the queried content item cannot respond with a fake hash because the content item ID is concealed. If the malicious node stores the content item, it responds with the original hash if SendOriginalHash is set to true. The effects of this attack are shown in Section 6.5.4. If SendOriginalHash is set to false, a fake hash is sent.

A malicious node attacks all incoming requests regardless of whether the item has been

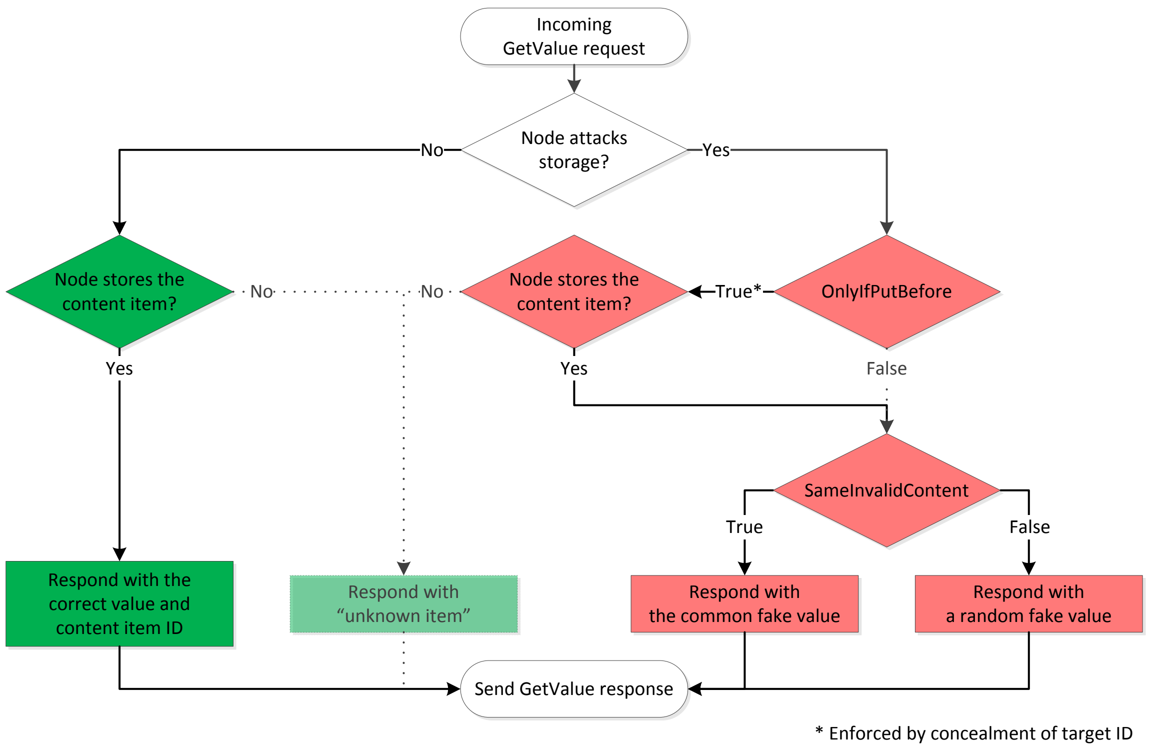

Incoming GetValue request
Node attacks storage?
No
Yes
Node stores the content item?
No
No
Node stores the content item?
True*
OnlyIfPutBefore
Yes
Yes
False
SameInvalidContent
True
False
Respond with the correct value and content item ID
Respond with "unknown item"
Respond with the common fake value
Respond with a random fake value
Send GetValue response
* Enforced by concealment of target ID

Figure 29: Flowchart of the handling of an incoming GetValue request

stored on it before if OnlyIfPutBefore is set to false. If the malicious nodes collude (SameInvalidContent = true), then all of them respond with the same fake content item; otherwise, each one uses its own fake content item.

A node should only receive a GetValue query if it has previously responded to a GetHash request and is among the node group that is chosen by the group trust value evaluation (with TrustedKad) or the majority decision (without TrustedKad). So the option of responding with "unknown item" is included in Figure 29 for completeness only. Inoffensive nodes respond to a GetValue request by sending the value of the content item. Again, a malicious node always attacks if OnlyIfPutBefore is set to false. If TrustedKad's concealment of the content item ID is used (OnlyIfPutBefore = true), it can only attack GetValue requests if the item has been stored on it before.

5.6 Trust Information Management

As mentioned above, the trust information is stored centrally in the simulations performed for this thesis. The central storage replaces the distributed trust management nodes and the caching of the trust information. The GlobalNodeList class uses the helper classes PeerStorage and PeerInfo: PeerInfo contains the information about one node and PeerStorage contains all PeerInfo objects of all simulated nodes. All three classes have been extended to be able to store trust information. For each node, there is a TrustInformationDetails object that contains four lists which store the positive and negative routing and storage ratings, respectively. This simplifies the calculation of the trust values because only the sizes of the four lists are relevant. If there were only two lists containing routing or storage ratings or even only one list for all routing and storage ratings, these lists would have to be iterated and filtered during every request, which would decrease the performance of the simulation. The lists are implemented as a key-value store ("map"). The node that gives the rating is used as key and the rating is the value. This way, it can be easily ensured that a node can be rated only once by another node. The most recent rating is kept, older ones are discarded.

5.6.1 Storing Trust Information

When a node rates another node, e.g., for routing, it calls the appropriate method of the GlobalNodeList class as shown in Figure 30. The rating is given to the TrustInformationDe-

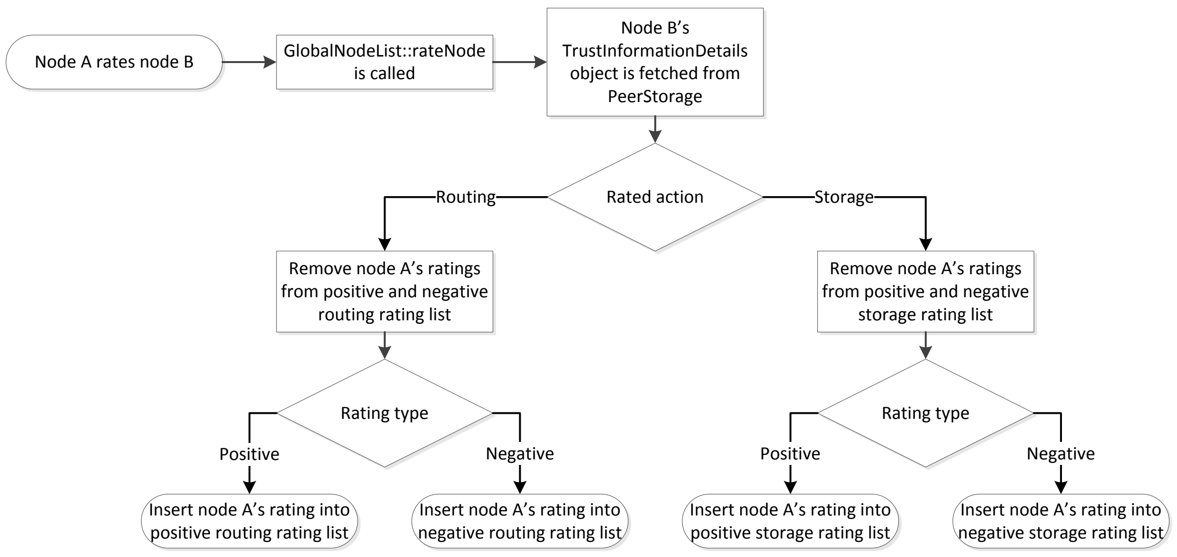

Node A rates node B
GlobalNodeList::rateNode is called
Node B's TrustInformationDetails object is fetched from PeerStorage
Rated action
Routing
Storage
Remove node A's ratings from positive and negative routing rating list
Remove node A's ratings from positive and negative storage rating list
Rating type
Positive
Negative
Rating type
Positive
Negative
Insert node A's rating into positive routing rating list
Insert node A's rating into negative routing rating list
Insert node A's rating into positive storage rating list
Insert node A's rating into negative storage rating list

Figure 30: Flowchart of the rating storage procedure

tails object of the rated node together with the information about which node gives the rating. The TrustInformationDetails object removes any existing rating from the two lists that contains the positive and the negative routing ratings. Afterwards, the new rating is inserted into the correct list.

5.6.2 Retrieving Trust Information

Just as when trust information is stored, the TrustInformationDetails object of a node is fetched from the PeerStorage when its trustworthiness is evaluated. If there is no TrustInformationDetails object for the node in question, this node does not exist because such an object is created for each legitimate node when it joins the network; hence, the call evaluates the trustworthiness of a non-existing fake node. Fake nodes do not have valid cached trust information and are therefore discarded. However, as already mentioned in

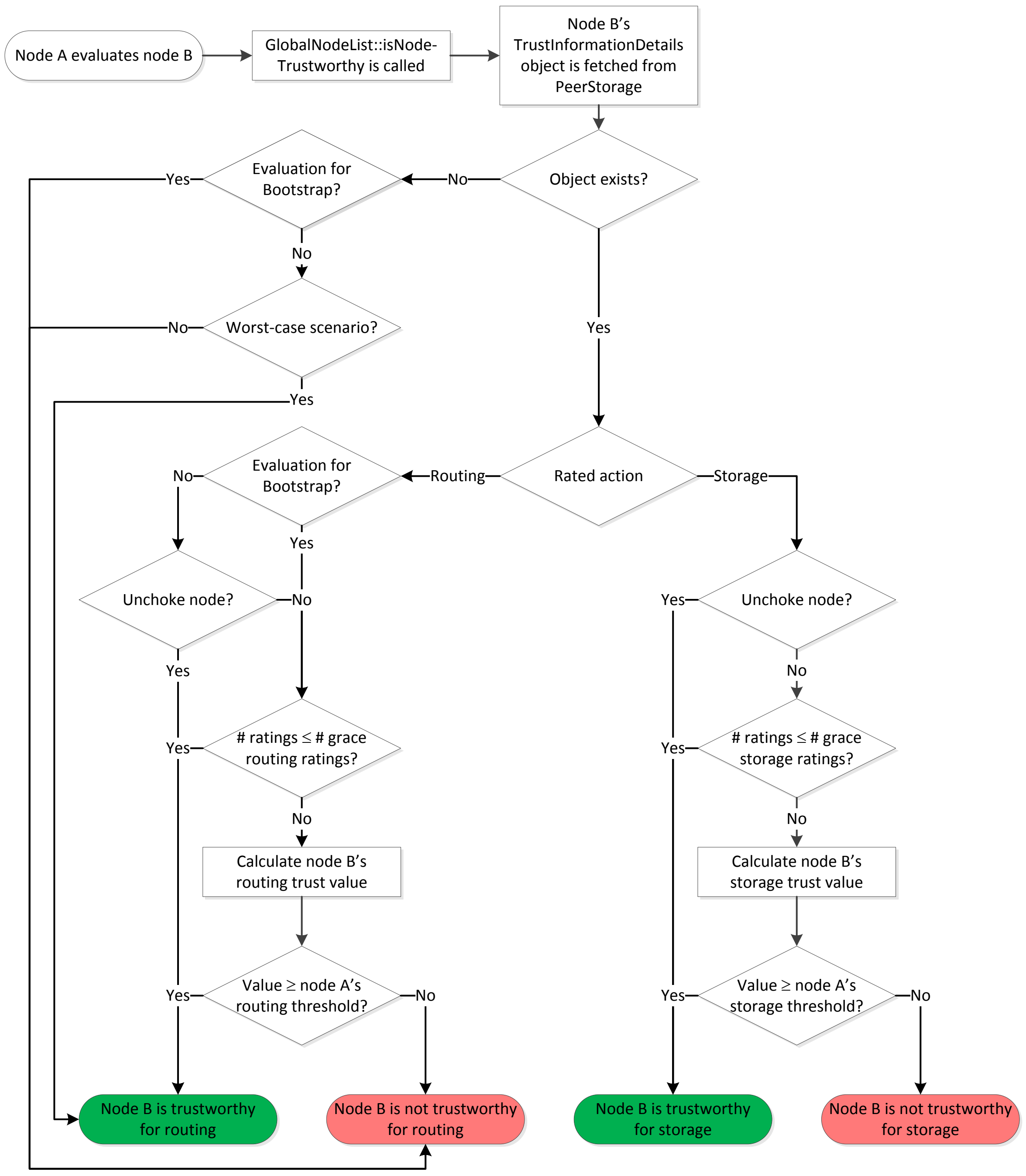

Node A evaluates node B
GlobalNodeList::isNode-Trustworthy is called
Node B's TrustInformationDetails object is fetched from PeerStorage
Object exists?
No
Evaluation for Bootstrap?
Yes
No
Worst-case scenario?
No
Yes
Yes
Rated action
Routing
Storage
Evaluation for Bootstrap?
No
Yes
Unchoke node?
No
Yes
# ratings ≤ # grace routing ratings?
Yes
No
Calculate node B's routing trust value
Value ≥ node A's routing threshold?
Yes
No
Unchoke node?
Yes
No
# ratings ≤ # grace storage ratings?
Yes
No
Calculate node B's storage trust value
Value ≥ node A's storage threshold?
Yes
No
Node B is trustworthy for routing
Node B is not trustworthy for routing
Node B is trustworthy for storage
Node B is not trustworthy for storage

Figure 31: Flowchart of the rating retrieval procedure

Section 5.3.1, also a worst-case scenario is evaluated. In this scenario, the fake nodes are regarded as trustworthy except for when the trustworthiness of a potential Bootstrap node is evaluated: The cached trust information of Bootstrap nodes is verified at the trust management nodes, and this check fails if the cached trust information is not valid. In addition, the number of grace routing ratings during Bootstrap operations is set to 0 (as explained in Section 4.10.1).

Figure 31 shows how a node's trustworthiness is evaluated: If no TrustInformationDetails object exists for the node, it is discarded during evaluations for Lookup and Bootstrap operations in normal scenarios. Only if the worst case is evaluated, the node is regarded as trustworthy for Lookups. The node is still not a Bootstrap candidate node because in a full implementation of TrustedKad, the verification of the cached trust information at the trust management nodes would most probably fail because they do not exist.

If the evaluated peer does exist and the storage trustworthiness is evaluated, the evaluating node first decides whether it unchokes the node; if it does, the trust value is not calculated. Instead, the node is regarded as trustworthy regardless of its trust value. If no unchoking is performed, the total number of storage ratings is checked. If it is below the grace number of storage ratings, the node is regarded as trustworthy (see $T_{S,i} = 1 \; if \; R_{S,i}^{+} + R_{S,i}^{-} \leq g_S$ in Section 4.6). If the node has enough storage ratings, its trust value is calculated and compared to the storage trust value threshold. If it is greater than or equal to the threshold, the node is regarded as trustworthy for storage.

Evaluations of the routing trustworthiness work in the same way except for when it is performed to assess a Bootstrap candidate node: If so, unchoking is not performed and the number of grace routing ratings is set to 0. However, if the node that is evaluated has no routing ratings yet ($R_{R,i}^{+} + R_{R,i}^{-} = 0$), it is regarded as trustworthy during the simulations. This is required, e.g., for the Bootstrap of the second node that joins the network so that it trusts the first node for Bootstrap.

6 Evaluation

This chapter presents the results of the simulations that are conducted to analyze TrustedKad's effectiveness. First, the general settings of the simulation runs and the technical simulation environment are explained. Afterwards, the comparison indicators that are used to evaluate TrustedKad's effectiveness are introduced.

The main part of the chapter discusses the simulation results. For comparison, a baseline of a network that does not use TrustedKad is presented first. Then, analyses of configurations in which the malicious nodes only attack routing or storage operations follow. At the end of the chapter, a configuration is presented in which TrustedKad counters attacks against both routing and storage.

Different variations of the attacks on routing and storage are used to thoroughly analyze TrustedKad's features. In addition, parts of TrustedKad's features are disabled to show their influence on the results.

6.1 Simulation Parameters and Environment

OverSim [39] offers a multitude of parameters to configure simulation runs. In this section, the parameters that are relevant for the simulations in this thesis and their settings are introduced. In the following, the technical simulation environment is presented.

6.1.1 OverSim Parameters and Settings

In Table 4, OverSim's relevant parameters and their settings are listed. The network consists of 1,000 nodes. This number of nodes is the setting of OverSim's example configuration "KademliaLarge" and is used by other authors as well [55,56,61]. OverSim is capable of simulating considerably larger networks [39]. However, the simulation time would increase considerably as well and during analyses for [3], the author conducted tests with larger networks which did not yield significantly different results.

The interval between two nodes that join the network is one second, so during the first 1,000 seconds, one node per second attempts to join the network – this is also the default setting of OverSim. The attempt is successful at the first time if the node trusts its randomly assigned Bootstrap node for the Bootstrap operation. If it does not trust it, it retries to join the network after 10 seconds as described in Section 5.5.2.

There is no churn, so no node leaves the network after it has joined it, and after 1,000 seconds, no new nodes join the system. Hence, the initialization phase takes

Table 4: OverSim parameters and settings

Parameter	Setting
Number of nodes	1,000
Type of churn	No churn
Interval between two nodes joining the network	1s
Interval between two join attempts	10s
Initialization phase duration	1,000s
Measurement time	3,000s
Total simulation time	4,000s
Fraction of malicious nodes	0% to 40% (in steps of 5%)
Lifetime of a content item	300s
Interval between two Get operations per node	60s
Message timeout	1.5s
Lookup operation timeout	10s
Packet loss	None
Number of parallel requests (α)	3
Number of stored and retrieved replicas	4
Number of repetitions per parameter combination	30

1,000 seconds, during which no Put and Get operations take place. Afterwards, the measurement phase begins and lasts for 3,000 seconds, resulting in a total simulation time of 4,000 seconds. For [3], the author has conducted tests with shorter and longer runtimes which have not yielded significantly different results. In addition, randomly selected samples of the simulations for this thesis have been analyzed for earlier points in simulation time which also show that there are no significant changes in the last 1,000 seconds of simulation time, so the system is in a steady state at the end of the simulations.

The fraction of malicious nodes is varied from 0% to 40% in steps of 5%. The lifetime of a content item is 300 seconds, i.e., the nodes on which it is stored delete it after 300 seconds. Every node conducts a Get operation every 60 seconds, so in the measurement phase of 3,000 seconds, every node can perform up to 50 Get operations. As already mentioned in Section 5.2, the network delays are based on latency measurements conducted in the Internet. The UDP timeout is set to 1.5 seconds and the timeout of a Lookup operation is 10 seconds; there is no packet loss. All of these settings are the default settings of OverSim.

The number of parallel requests (α) is 3, as it is also used by Kademlia and Kad (cf. 2.4.1). The number of replicas that shall be stored and retrieved is 4. To eliminate effects caused by the selection of random numbers, every parameter combination is run 30 times with different seeds for the random number generator, resulting in 30 different environments regarding, e.g., the distribution of the nodes and content items across the ID space.

6.1.2 TrustedKad Parameters and Settings

Aside from the OverSim parameters, TrustedKad introduces new parameters that are listed in Table 5. To show the influence of malicious nodes on routing attacks and storage attacks, they are analyzed separately, so the malicious nodes attack either routing operations or storage operations. Afterwards, attacks on both operation types are evaluated.

The thresholds for the routing and storage trust values are varied from -1.0 to 1.0 in steps of 0.1 to analyze which thresholds are reasonable, i.e., that increase the success rate and decrease the false positives rate. A reasonable unchoking ratio is determined in Section 6.4.1 for routing and in Section 6.5.1 for storage. Both analyses result in an unchoking ratio of 1%.

Table 5: TrustedKad parameters and settings

Parameter	**Setting**
Type of operation that is attacked	Routing, storage and both
Trust value threshold (routing and storage)	-1.0 to 1.0 (in steps of 0.1)
Unchoking probability (routing and storage)	1%
Number of grace ratings (routing and storage)	10
Attack probability of the malicious nodes	50% and 100%
Start time of attacks	Immediately (0s) and after 2,000s
isClosestAttack during routing attacks	True and false
Only inoffensive Bootstrap node during routing attacks	True and false
Malicious nodes can successfully fake cached trust information during routing attacks	True and false
Colluding malicious nodes during storage attacks	True and false
SendOriginalHash during storage attacks	True and false
Concealment of content ID during storage attacks	True and false
Use most trustworthy node group during storage attacks	True and false

The number of grace ratings depends on the scenario in which TrustedKad is deployed: The node should achieve the number of grace ratings as quickly as possible because during the grace phase, it may receive only a limited service as described in Section 4.14. As samples, the simulations of sections 6.4.2 (attacks only on routing) and 6.5.2 (attacks only on storage) show that inoffensive nodes receive about 460 routing ratings and about 170 storage ratings on an average. Hence, the amount of 10 grace ratings as it is used in the simulations is achieved rather quickly at the beginning of the simulation runs.

The results do not only show TrustedKad with all of its features, but also include configurations in which some of the features are disabled to show that they increase the security of the system. In addition, several attack variations are analyzed to show that TrustedKad is able to protect from them. This leads to a multitude of possible configurations of which a subset is presented in this thesis.

The attack probability is set to 100% in most configurations, so the malicious nodes attack every request they receive. Two configurations, one for routing and one for storage, aim to show that TrustedKad is also able to identify malicious nodes that only attack 50% of the requests directed at them. The same is true for the two configurations in which the malicious nodes attack only after 2,000 seconds of simulation time. By behaving this way, the malicious nodes gain high trust values at first and attempt to abuse those high trust values for conducting attacks.

The isClosestAttack parameter as described in Section 5.5.4 is set to true and false to analyze its influence. By performing attacks on Bootstrap operations, the malicious nodes can partition the network; this is prohibited in some configurations by having new nodes Bootstrap only from inoffensive nodes as described in Section 5.5.2. As stated in Section 4.8.3, attacks on the cached trust information are possible, so for this thesis, configurations in which such attacks are successful are also analyzed.

The attacks on storage operations also have several parameters that have an influence on the results. The main difference between malicious storage nodes is whether they collude or do not collude, i.e., if they deliver the same fake content item or different fake content items. In order to boost their trust values, the malicious nodes can choose to return the hash of the originally stored content item, so this attack variation is also analyzed. The concealment of the content item ID is enabled in most configurations and only disabled to show how it improves the results (by setting OnlyIfPutBefore to false as explained in Section 5.5.8). The same is true for TrustedKad's feature of choosing the most trustworthy node group to retrieve the correct version of the content item.

6.1.3 Used Simulation Configurations

In this section, the simulation configurations that are used for the results presented in this thesis are introduced. The parameters explained in the previous two sections are combined into configurations, so a configuration defines which parameters can take which values. Every parameter combination is repeated 30 times, and if not stated otherwise, the fraction of malicious nodes is varied from 0% to 40% in steps of 5%.

The configurations for this thesis consist of 9 to 882 parameter combinations that are repeated 30 times each. Several baseline configurations only vary the number of malicious nodes from 0% to 40% in steps of 5%, resulting in 9 different parameter combinations × 30 repetitions = 270 simulation runs. The configuration of Section 6.6 in which the malicious nodes attack routing and storage operations consists of 2 different fractions of malicious nodes × 21 different routing trust value thresholds (-1.0 to 1.0 in steps of 0.1) × 21 different storage trust value thresholds × 30 repetitions = 26,460 simulation runs.

Table 6 shows the configurations for Section 6.3 in which the baseline runs are presented. The first five configurations analyze attacks against the routing operations. The first two represent a worst-case scenario in which the nodes claim to be responsible for all items. In the second one, a network partitioning is prevented by only offering inoffensive bootstrap nodes to nodes that attempt to join the network. In the third and fourth configuration, the malicious nodes do not claim to be responsible for the content item. As an example, in the fifth configuration, the malicious nodes only attack half of all requests.

The following three configurations analyze storage attacks. All configurations are run with both colluding and non-colluding malicious nodes. The effects of an attack in which the

Table 6: Parameter settings of baseline configurations

Section	Attack routing	isClosestAttack	Inoffensive Bootstrap node	Attack storage	Colluding malicious nodes	OnlyIfPutBefore	SendOriginalHash	Attack probability	Repetitions	Fraction of malicious nodes [%]	Number of simulation runs
6.3.1.1	True	True	False	False	-	-	-	100%	30	0 - 40	270
6.3.1.2	True	True	True	False	-	-	-	100%	30	0 - 40	270
6.3.1.3	True	False	False	False	-	-	-	100%	30	0 - 40	270
6.3.1.4	True	False	True	False	-	-	-	100%	30	0 - 40	270
6.3.1.5	True	True	False	False	-	-	-	50%	30	0 - 40	270
6.3.2.1	False	-	-	True	True, False	True, False	False	100%	30	0 - 40	1,080
6.3.2.2	False	-	-	True	True, False	True	True	100%	30	0 - 40	540
6.3.2.3	False	-	-	True	True, False	True, False	False	50%	30	0 - 40	1,080
6.3.3.1	True	True	False	True	True, False	True, False	False	100%	30	0 - 40	1,080
6.3.3.2	True	False	False	True	True, False	True, False	False	100%	30	0 - 40	1,080
Total											**6,210**

nodes attack all content items are shown as well as situations in which they only attack items that have been stored on them previously. In addition, a configuration is analyzed in which the malicious nodes respond with the hash of the originally stored content item, but then with a manipulated value. Also for the storage attack configurations, one example is given in which the malicious nodes only attack 50% of the requests.

In the last two configurations, the malicious nodes attack both routing and storage operations. Two exemplary configurations are used to show the difference between attacks with and without the isClosestAttack present. In total, 6,210 simulation runs are performed for Section 6.3.

The configurations for the analysis of attacks against routing with TrustedKad present are shown in Table 7. In all configurations, the routing trust value threshold is varied from -1.0 to 1.0 in steps of 0.1. For the analysis of the routing attacks, 85,050 simulation runs are required in total.

The first four configurations are required for the analysis of the unchoking ratio. They represent a worst-case scenario in which the malicious nodes are able to fake the cached trust information and in which the network is partitioned. The remaining configurations always analyze the effect of the isClosestAttack, so the isClosestAttack parameter is always set to true and false. The configurations for sections 6.4.2 and 6.4.3 differ by the setting for the bootstrap nodes: The effect of the network partitioning is clearly visible in the former section and so a non-partitioned network is presented in the latter section.

The following two sections analyze the behavior of TrustedKad if the malicious nodes are able to fake the anti-Sybil-attack information in the cached trust information. The results of the configuration with isClosestAttack set to true that are used for the unchoking ratio analysis can be used again for Section 6.4.4. In the last two sections, attacks are presented in which the malicious nodes do not always perform their attacks. The worst-case scenario is used also in these two configurations.

Table 7: Parameter settings of routing attack configurations

Sections	**Routing unchoking probability**	**Successfully faked cached trust information**	**isClosestAttack**	**Inoffensive Bootstrap node**	**Attack probability**	**Attack only after X seconds**	**Routing trust value threshold**	**Repetitions**	**Fraction of malicious nodes [%]**	**Number of simulation runs**
6.4.1	0%	True	True	False	100%	0	-1.0 - 1.0	30	0 - 40	5,670
6.4.1, 6.4.4	1%	True	True	False	100%	0	-1.0 - 1.0	30	0 - 40	5,670
6.4.1	5%	True	True	False	100%	0	-1.0 - 1.0	30	0 - 40	5,670
6.4.1	10%	True	True	False	100%	0	-1.0 - 1.0	30	0 - 40	5,670
6.4.2	1%	False	True, False	False	100%	0	-1.0 - 1.0	30	0 - 40	11,340
6.4.3	1%	False	True, False	True	100%	0	-1.0 - 1.0	30	0 - 40	11,340
6.4.4	1%	True	False	False	100%	0	-1.0 - 1.0	30	0 - 40	5,670
6.4.5	1%	True	True, False	True	100%	0	-1.0 - 1.0	30	0 - 40	11,340
6.4.6	1%	True	True, False	False	50%	0	-1.0 - 1.0	30	0 - 40	11,340
6.4.7	1%	True	True, False	False	100%	2000	-1.0 - 1.0	30	0 - 40	11,340
Total										**85,050**

Table 8: Parameter settings of storage attack configurations

Sections	Storage unchoking probability	Colluding malicious nodes	OnlyIfPutBefore	SendOriginalHash	Use most trustworthy node group	Attack probability	Attack only after X seconds	Storage trust value threshold	Repetitions	Fraction of malicious nodes [%]	Number of simulation runs
6.5.1	0%	True, False	True	False	True	100%	0	-1.0 - 1.0	30	0 - 40	11,340
6.5.1, 6.5.2	1%	True, False	True	False	True	100%	0	-1.0 - 1.0	30	0 - 40	11,340
6.5.1	5%	True, False	True	False	True	100%	0	-1.0 - 1.0	30	0 - 40	11,340
6.5.1	10%	True, False	True	False	True	100%	0	-1.0 - 1.0	30	0 - 40	11,340
6.5.3	1%	True, False	False	False	False	100%	0	-1.0 - 1.0	30	0 - 40	11,340
6.5.4	1%	True	True	True	True	100%	0	-1.0 - 1.0	30	0 - 40	5,670
6.5.5	1%	True, False	True	False	True	50%	0	-1.0 - 1.0	30	0 - 40	11,340
6.5.6	1%	True, False	True	False	True	100%	2,000	-1.0 - 1.0	30	0 - 40	11,340
Total											**85,050**

Table 9: Parameter settings of combined routing and storage attack configurations

Sections	Attack routing	Successfully faked cached trust information	isClosestAttack	Inoffensive Bootstrap node	Attack storage	Colluding malicious nodes	OnlyIfPutBefore	SendOriginalHash	Use most trustworthy node group	Routing trust value threshold	Storage trust value threshold	Repetitions	Fraction of malicious nodes [%]	Number of simulation runs
6.6, A.2	True	True	True	False	True	True	True	False	True	-1.0 - 1.0	-1.0 - 1.0	30	5, 20	26,460

In Section 6.5, the malicious nodes perform only attacks on storage operations. The corresponding simulation configurations are shown in Table 8; they also result in 85,050 simulation runs in total. Similar to the routing attack configurations, the storage trust value threshold varies from -1.0 to 1.0 in steps of 0.1. All configurations except for one analyze both colluding and non-colluding nodes.

Also similar to the routing attack configurations, the first four configurations aim to determine the unchoking ratio. The results of the configuration with 1% unchoking ratio are also used to present TrustedKad's effectiveness against storage attacks if all of its features are enabled. In Section 6.5.3, two of TrustedKad's features are disabled to show their usefulness: The concealment of the content item ID and the approach of using the most trustworthy node group to determine the correct content item. The following section aims to show TrustedKad's resilience against attacks in which the malicious nodes reply with the correct content item hash, but with a fake value. It also shows the effects this attack has on the nodes' trust values. Because all nodes return the same hash value, a distinction between colluding and non-colluding malicious nodes is not necessary. In the last two configurations, TrustedKad's resilience against malicious nodes that do not always attack is presented.

At the end of this chapter, a configuration is shown in which the malicious nodes attack both routing and storage operations. The configuration is listed in Table 9. Again, a worst-case scenario is chosen: The malicious nodes can fake the cached trust information and perform the isClosestAttack. The network is partitioned and the malicious nodes collude. Both the routing and the storage trust value thresholds are varied from -1.0 to 1.0. To reduce the number of simulation runs, only fractions of 5% and 20% malicious nodes are analyzed.

It should be noted that all diagrams in one section are derived from the same simulations. For example, the data for all diagrams in Section 6.6 and Annex A.2 is derived from the 26,460 simulation runs shown in Table 9.

6.1.4 Technical Simulation and Evaluation Environment

OMNeT++ is available for several platforms with Windows and Linux being the most common of them. However, out of these two, OMNeT++ can run in 64-bit mode only on Linux. During tests, several simulation runs have required more than 4 GiB of RAM, which is only supported by the 64-bit variant (RAM: Random Access Memory; GiB: gibibytes, 1 GiB = 2^{30} bytes). So the simulations are performed with the Linux implementation of OMNeT++.

Each simulation run takes at least five minutes and can last more than an hour. In order to reduce the time required by the simulations, the runs are distributed across several machines with 57 CPU cores in total. The used CPU types are Intel Xeon, Intel Core2Duo, Intel Core i5 and AMD Athlon 64 x2. Even with these 57 cores available, the 202,770 simulation runs that are used for this thesis take more than 21 days.

A Java-based server and Perl-script-based clients are used to distribute the simulation runs to the machines and to collect the simulation results. The simulation environment is also used for other simulations which have a higher priority, so the distribution software includes a detection of other simulations. If another simulation is started on a machine, the OverSim simulations are cancelled immediately and put back into the simulation run queue on the server. As soon as the machine is free again, OverSim simulations are distributed to that machine again. A manual exclusion of machines and also cores is possible as well.

The simulation results are imported into a Microsoft SQL Server 2008 database (SQL: Structured Query Language). Microsoft Excel 2010 is used to evaluate the data and to create the diagrams. As with the simulations, the memory requirements enforce the usage of the 64-bit version of Excel, because the 32-bit version only supports the usage of less than 2 GiB. In order to automate the analyses, VBA macros (Visual Basic for Applications) are used.

6.2 Comparison Indicators for Evaluations

In the following, the comparison indicators that are used to evaluate TrustedKad's effectiveness are introduced. The main indicators are the success rates of the Put and Get operations including the false positives rate, and the trust values of the nodes. Other indicators such as the number of involved nodes and the duration of the operations are also presented. Finally, the gathering of the results is explained and several remarks to the diagrams that follow afterwards are given.

6.2.1 Success Rates of Put and Get Procedures

TrustedKad aims to improve the success rates of the operations and to decrease the false positives rate, so these rates are main indicators. As explained before in Section 5.5.5, a Put operation is regarded as successful if the content item is successfully stored on at least one node. The operation fails if the Lookup does not return any reachable nodes or if none of the returned nodes responds positively to the Put request. However, the latter reason cannot occur in the simulations because the malicious nodes do not attack incoming Put requests – every node responds positively to an incoming Put request.

The choice of the content item that shall be retrieved is random. Analyses of samples of the results show that the random choice leads to a long-tail distribution of item access: Most items are accessed either never or one time; a few items reach up to 30 accesses.

According to Section 5.5.7, a node regards a Get operation as successful if it can retrieve a version of the content item that matches the hash of the GetHash phase. If this is the case, all nodes that have sent the same hash receive a positive rating. However, a positive rating for a node does not necessarily mean that the operation is successful: If the attack on the storage is successful, the node gives a positive rating although it has chosen a fake version of the content item.

Hence, the statistical recording of the Get operation is more differentiated: It is recorded as successful if the node chooses the originally stored version of the content item. If it chooses a faked version or if it believes that the item does not exist, this is recorded as a false positive. Other possible reasons for failing Get operations are that the Lookup does not return any nodes that could be queried or that the hash of the content item value does not match the hash received in the GetHash phase.

It should be noted that Get requests only attempt to retrieve content items that have been stored successfully before. Therefore, the maximum Get success ratio is always 100%, regardless of the Put success ratio.

6.2.2 Trust Values

In order to assess the trustworthiness of a node, TrustedKad uses trust values and thresholds. So, the trust values that the nodes have at the end of the simulation are a main indicator for the effectiveness of TrustedKad. The trust values of the inoffensive nodes should be close to 1.0, but at least greater than or equal to the trust value threshold. Similarly, the trust values of the malicious nodes should be close to -1.0 or at least lower than the threshold.

Based on the trust values, the fractions of trusted nodes that are malicious are determined by analyzing how many nodes are trusted in total and which fraction of them is malicious. Please note that this indicator does not denote the fraction of the malicious nodes that are regarded as trustworthy – instead, it considers only the trusted nodes and determines the fraction of malicious nodes among them.

6.2.3 Involved Nodes

In addition to the aforementioned main indicators, several other indicators are analyzed as well. One of them is the number of involved nodes: It denotes how many nodes are queried during a Put or Get operation. Due to the parallel querying of the nodes during a Lookup, the term “hop count” cannot be applied to Kademlia. The numbers of involved nodes are analyzed also for verification purposes: If, e.g., a malicious node attacks routing operations by informing about non-existing nodes, the requests sent to those nodes run into a timeout. Due to the Lookup timeout, the number of involved nodes should decrease with increasing intensity of the attack because more and more requests run into a timeout. Therefore, they take more time, leading to fewer possible requests that can be sent.

6.2.4 Duration of Put and Get Procedures

Also due to timeouts, the duration of Put and Get operations should increase with increasing routing attack intensity. If, e.g., only storage operations are attacked by delivering fake versions of the content item, the duration should remain unchanged because no timeouts are provoked by that attack.

6.2.5 Number of Replicas

The number of replicas should always be four for the Put operation if only storage attacks are conducted. During routing attacks, it is lower if fewer nodes are found during the Lookup. For Get operations, the number denotes how many nodes deliver the chosen version of the content item. For both routing and storage attacks, it decreases with increasing attack intensity because either fewer nodes are found or several versions of the content item are identified.

6.2.6 Levels of Aggregation

OMNeT++ offers different levels of aggregation for the simulation results that are also explained in the documentation [40]. The default is to use the so-called scalar output. For scalar results, the samples of the indicator are aggregated during the simulation runs. Only the aggregated result is stored at the end of the simulation, e.g., the mean, the standard deviation, the minimum and the maximum. So there is one result per repetition of each parameter combination. Due to this, the exact distribution of the sample values cannot be reconstructed.

In order to analyze the distribution of the sample values, the so-called vector output can be used. For vector output, every single sample value of an operation is stored as a result. This considerably increases the storage requirements of the simulation results: Instead of storing one aggregated result, the individual results of 1,000 nodes are stored.

For the main indicators Put and Get success rates and the trust values, the vector output is used. Only this way, the median and quartiles can be determined and further analyses of the results are possible.

Table 10 shows an example for the aggregation levels based on the Get success rate. The Get success rate is calculated based on the Get operations that each node performs. About 50 Get operations are conducted by each node. At the end of the simulation, the node calculates its own Get success rate by determining the fraction of successful operations compared to the total number of operations. This results in 1,000 results per repetition for the vector output. For the scalar output, the mean, standard deviation and several other statistical indicators are calculated. Only the aggregated values are stored; the 1,000 original sample results are discarded. So for each of the 9 to 882 parameter combinations of a configuration, up to 30,000 vector results are stored for main indicators such

Table 10: Example of OMNeT++ aggregation levels

Aggregation level	Description	Output level
Get operation	50 per node	
Node	1,000 nodes in the network	Vector
Repetition	30 repetitions per parameter combination	Scalar
Parameter combination	9 to 882 per configuration	
Configuration		

as the Get success rate; for other indicators, 30 scalar results are stored.

6.2.7 Intention of the Graphs

In the following sections, the results of the simulations are presented. Most of the figures include the results of all parameter combinations of a configuration. The intention is not to discuss every data point; instead, the dependency of the indicator from parameter variations such as the trust threshold and the fraction of malicious nodes shall be shown.

For the main indicators, the median and the 25% and 75% quartiles are given instead of the mean and the standard deviation (the median is the 50% quartile). This way, the graphs show a better characteristic of the underlying data: The marker indicates the point at which 50% of the values are either lower or greater. 25% of the underlying data values are lower than the lower error bar, 25% are greater than the upper error bar, and 50% of the underlying data values are between the two error bars. In addition, the median is more resistant against outliers than the mean.

Furthermore, only the median is able to give meaningful information about the trust values and therefore the trustworthiness of the nodes. The analysis of the data shows that the trust values are not normally distributed (cf. figures 52, 78, 101 and 110). Hence, the mean value cannot determine the fraction of nodes that is regarded as trustworthy: If, e.g., the routing trust value threshold is set to 0.4 and the routing trust values have a mean of 0.5, this could still mean that more than half of the nodes have a trust value below the threshold due to a skewed distribution of the underlying data. In contrast, the median and the quartiles state the minimum trust value that 25%, 50% and 75% of the nodes have. This allows for analyzing which fraction of the nodes is regarded as trustworthy.

The median is also given for the success rates because they are also not normally distributed (cf. figures 53 and 79). In addition, both the trust values and the success rates have a limited value range of -1.0 to 1.0 or 0% to 100%. The author believes that due to these limited value ranges, the specification of the median and the quartiles is more meaningful than giving, e.g., a mean of 98% and a standard deviation of 5%. This would result in the upper limit of the interval being 103%, which is out of the value range. With the median and the quartiles, the actual distribution of the results is represented in a better way.

In order to determine the quartiles and the median, the vector output has to be collected during the simulations because OMNeT++ does not offer to export them in scalars. Due to the large effort required to determine the medians and quartiles, the mean and standard deviation are given for the other indicators.

6.3 Baseline

In this section, the results of configurations without TrustedKad are presented as a baseline to which TrustedKad is compared in the following sections. In these runs, no trust mechanisms are active, which means that all other nodes are trusted. As security measures, only replication of the content item on several nodes during Put operations is performed and majority decisions are used during a Get operation to identify the correct version of the content item. The Lookup process is not protected, because the measures used by Kad are susceptible to attacks as shown by the author in [1]. In addition, it is possible to choose the node ID freely because Kad does not use any algorithm to prevent this. With this configuration, the security level is similar[9] to the one of the Kad network.

A subset of all possible parameter combinations is analyzed in this section to introduce the effects of different attack types and parameters. During routing attacks, a malicious node can claim to be the closest node to the desired item ID (see "isClosestAttack" in Section 5.5.4), so the effects of this parameter are shown in this section. Network partitioning (cf. 2.5.3.1) has a great influence on the functioning of a P2P network, so configurations with and without network partitioning are analyzed.

During a storage attack, nodes can collude, so the effects of colluding and non-colluding malicious nodes are shown. In addition, a malicious node can respond to a GetHash request with the hash value of the original content item and send fake data as a response to a subsequent GetValue request. The results of this attack are also shown in this section.

For routing and storage attacks, one example configuration is shown in which the malicious nodes only attack 50% of the incoming requests. At the end of the section, a configuration is presented in which malicious nodes attack both routing and storage operations.

6.3.1 Routing Attacks Only

Five example configurations with different attack parameter settings are analyzed in this section. In the first two scenarios, the isClosestAttack parameter is set, so a malicious node responds that it knows no other node closer to the target than itself. This leads to the fact that the malicious node is probably contained in the final top *numLookupResult* nodes if no other inoffensive node returns enough closer nodes. In the following two scenarios, this behavior is deactivated to show the effect on the results. In the final section, a configuration in which the nodes attack only half of all incoming requests is presented.

6.3.1.1 Routing Attacks With isClosestAttack in a Partitioned Network

For this configuration, the malicious nodes answer each Lookup request with non-existing nodes and also insert themselves into the response messages as possible storage nodes of the desired content item. The node IDs of the non-existing nodes are chosen to be close to the target ID. In addition, the malicious nodes attack the Bootstrap process, which results in network partitioning.

Figure 32 shows the Put and Get success rates. The median Put success rates decrease if the fraction of malicious nodes M is greater than 0%, but never fall below 85%. The failed Put operations occur when nodes query only non-existing nodes they have received from inoffensive nodes. These inoffensive nodes have previously stored the non-existing nodes in free spaces of their routing tables when they received a response from a malicious node that contained those non-existing nodes. If only malicious nodes are used to store the content item on, the Put operation succeeds: The node is reachable, and every node stores every item it receives.

The median of the Get success rate rapidly decreases even for low fractions of malicious nodes: It drops to 62% with only 5% malicious nodes present and is 2% or less for $M \geq 20\%$. This shows that routing attacks are very effective even for low fractions of malicious nodes.

[9] The Kad filesharing network uses indirect storage, whereas the network simulated for this thesis uses direct storage. Most functions (e.g., choice of the node ID, bootstrapping and routing) work the same way in both network types. However, the question of how to choose a version of a content item does not arise in filesharing networks: There is no single version of the content item, but a multitude of possible sources for the same file instead.

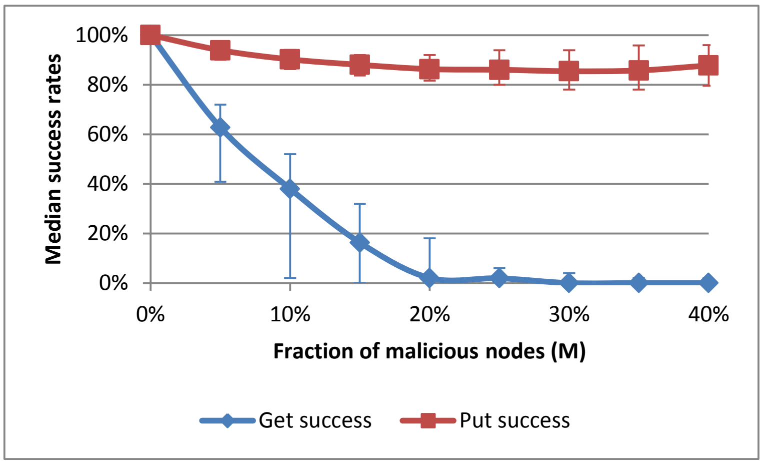


Figure 32: Median Put and Get success rates

A small fraction of the queries fails because all requests time out, but most of the Get operations fail because all queried nodes state that the content item does not exist. This happens because of the network partitioning: Content items that are published in one partition can usually only be found in the same partition if malicious nodes attack both routing and storage operations.

However, due to the fact that the malicious nodes attack only routing procedures but not storage ones in this simulation configuration, a malicious node returns the correct content item if it has been stored on it before. Therefore, the median Get success rates are larger than could be expected based on the estimated number of partitions (cf. 5.5.2.1): For M = 5%, an estimated number of about 50 partitions would mean that only approximately every 20th content item would be in the same partition as the querying node, resulting in a maximum possible Get success rate of only 5%.

The reason for the larger median Get success rates in this simulation is shown in Figure 33. It shows three network partitions: Partition A consists of nodes 1 to 5, partition B consists of nodes 5 to 10, and nodes 10 to 14 form partition C. One possible way of the development of this network is shown by the indicated bootstrap relations.

If node 1 would store four replicas of a content item on nodes 2 to 5, the content item would be stored on a malicious node, node 5. If a node from partition B would attempt to find that content item and if node 5 would be among the nodes that are queried for the content item, the item would be retrieved despite the fact that is has been published by a node in partition A. This is because in this section, the malicious nodes attack only routing procedures, so node 5 responds with the content item that has previously been stored on

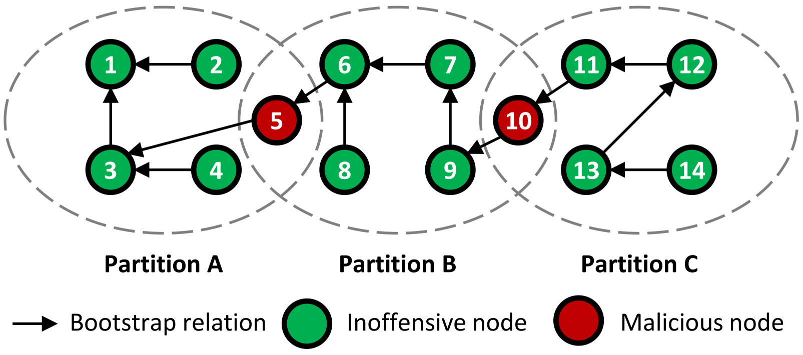
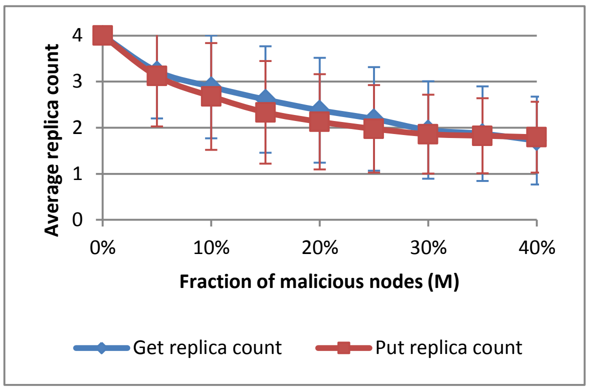
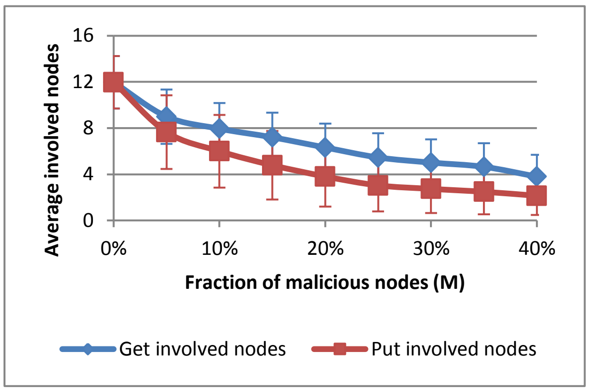


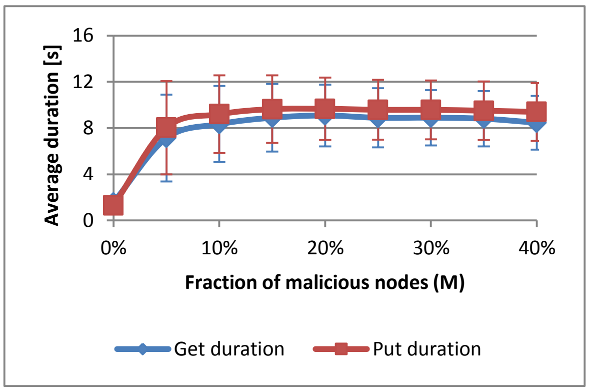
Figure 33: Three network partitions



Replica count



Number of involved nodes



Duration

Figure 34: Average replica counts, involved nodes and durations

it. Please note that nodes 11 to 14 are not able to retrieve content items stored on node 5: Node 10 only informs about non-existing nodes and therefore separates partitions B and C.

A malicious node can form an arbitrary number of partitions and because it does not attack storage, it makes content items that are stored on it available in all partitions it has created. Because of this, the median Get success rates are larger than expected.

Figure 34 shows the average replica count, number of involved nodes and duration of the Put and Get processes. The first two decrease with increasing M; the greatest drop for both is between 0% and 5% malicious nodes, similar to the Get success rate. The duration increases and also has the greatest rise between 0% and 5% malicious nodes.

The number of Get replicas is higher than the number of Put replicas. This is irritating at first, as it should not be possible to retrieve more replicas than have been stored previously. The reason is that the malicious nodes claim to be close to the content item that shall be published and only inform about non-existing nodes. So it is possible that a node attempts to query two non-existing nodes previously stored in its routing table and a malicious node as the first three nodes. When queried, the malicious node informs about other non-existing nodes close to the queried content item and claims that it is close to the item. This leads to a termination of the lookup, as the requests to the non-existing nodes time out. As a result, only one replica of the content item is stored, in fact on the malicious node. This can happen with two or three malicious nodes in the first round as well, leading to two or three stored replicas. For simplicity, this phenomenon will be referred to as “single-replica phenomenon” from here on.

This Put operation is counted as successful and decreases the mean Put replica count because only one replica is stored instead of the desired four. It can be assumed that the stored content item cannot be found by any other node, as the malicious node probably is not located at the correct storage location of the content. (If it is found anyway, the originally stored item will be returned, as no storage attacks occur in this configuration; this results in a successful Get operation.)

So the Get operation for the only replica fails in most cases. In contrast, the content items which are stored at the correct locations with four replicas can be retrieved in subsequent Get processes. This raises the mean Get replica count above the mean Put replica count, as only positive processes are counted.

The single-replica phenomenon also explains why the number of involved nodes is greater and the mean duration is lower in Get processes than in Put processes: The mean number of involved nodes during Put processes is lowered by the single-replica Put operations, as they only have one involved node, namely the malicious one on which the content is stored. The mean Get duration is lower than the Put duration because the nodes can be reached quicker during the successful Get requests. During the single-replica Put operations, a lot of timeouts occur that increase the mean Put duration.

6.3.1.2 Routing Attacks With isClosestAttack in a Non-Partitioned Network

In the second scenario, network partitioning is prevented by enforcing that each node bootstraps from an inoffensive node as explained in Section 5.5.2. Compared to the partitioned network presented in the previous section 6.3.1.1, the Put success rate shown in Figure 35 is slightly lower because more timeouts occur when only non-existing nodes are asked to store the content item. The spread between the quartiles is reduced because in this configuration, there is only one large network in which all nodes have got the same number of nodes they can communicate with. In the previous configuration with different network partitions of different sizes, this is not the case, so the results spread more.

The Get success rate is higher than before: Without network partitioning, the effect of the routing attacks are weakened. Figure 35 shows, however, that the Get success rate still decreases more than the fraction of malicious nodes increases. Nevertheless, the attack loses a lot of its impact: Before, the Get success rate was almost at 0% with only 20% malicious nodes present. Now, it is still at 12% even with 40% malicious nodes in the network because the content item cannot be stored in an unreachable network partition. This proves that it is vital to protect the Bootstrap operation as already stated in Section 4.10.1.

Figure 36 shows that the Get replica count has not changed significantly. The number of

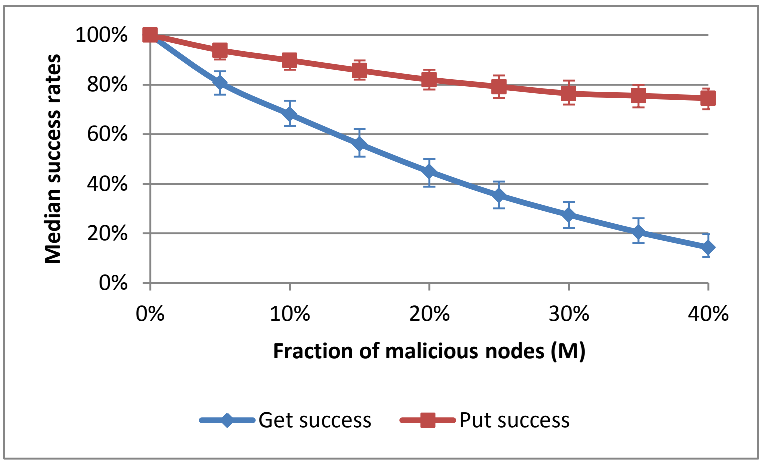
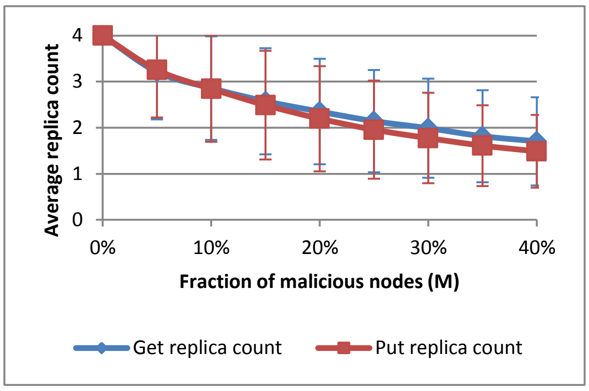
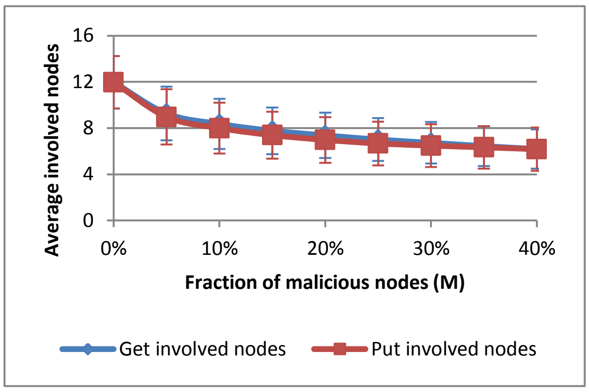


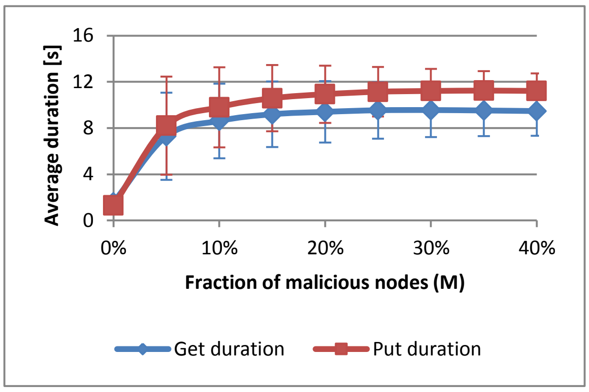
Figure 35: Median Put and Get success rates



Replica count



Number of involved nodes



Duration

Figure 36: Average replica counts, involved nodes and durations

involved nodes during Get operations has increased slightly, as has the mean duration. The number of Put replicas decreases stronger while the numbers of involved nodes and the mean durations have increased. The reason is that there is no partitioning anymore: More responding nodes can be found during the processes, which increases the number of involved nodes and the duration.

6.3.1.3 Routing Attacks Without isClosestAttack in a Partitioned Network

In contrast to the previous two configurations (cf. 6.3.1.1, 6.3.1.2), in this scenario, the malicious nodes only inform about non-existing nodes, but do not claim to be close to the content item. Figure 37 shows that the median Put success ratio first decreases and

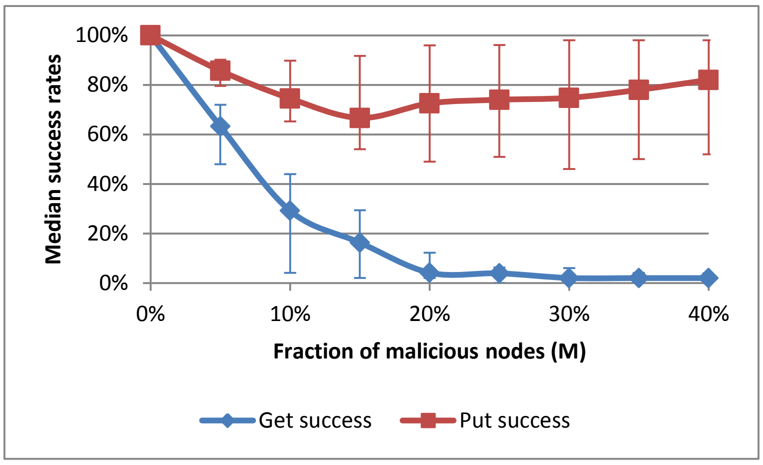
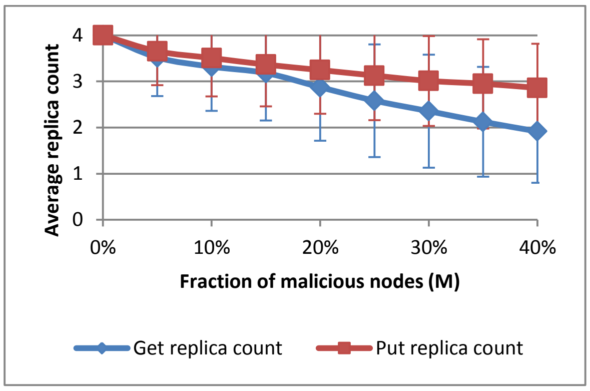
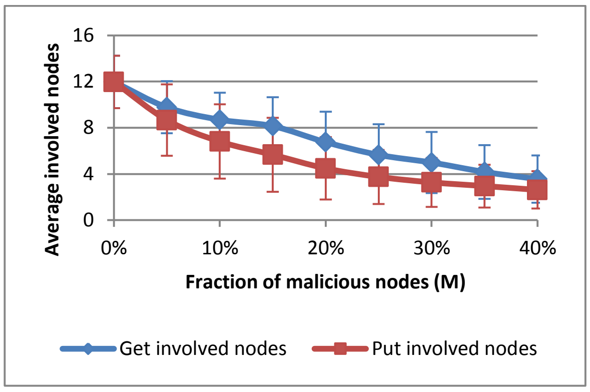


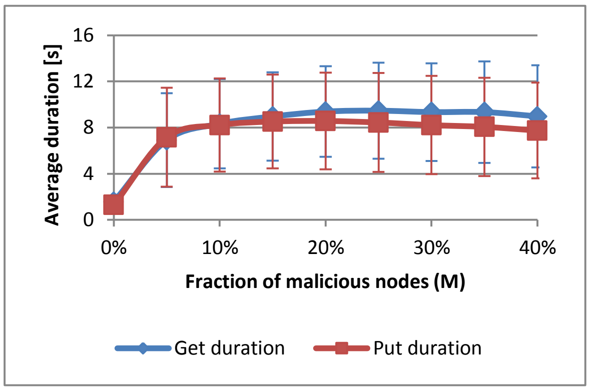
Figure 37: Median Put and Get success rates



Replica count



Number of involved nodes



Duration

Figure 38: Average replica counts, involved nodes and durations

slightly increases again with increasing malicious node ratio. There is a large spread between the 25% and 75% quartiles. This large variance between different nodes (or node groups) results from network partitioning and the fact that a malicious node does not claim to be responsible for the content item. When a node tries to store content and queries a malicious node for closer nodes, the malicious node returns only non-existing fake nodes. In contrast to Section 6.3.1.1, in this scenario, the malicious node does not claim to be close to the content item and therefore does not include itself in the response. If the top *numLookupResult* nodes are fake ones, the query times out and the node is not able to store the content on any node.

The Get success ratio increases only slightly compared to Section 6.3.1.1. Figure 38 shows that the number of Get replicas stays about the same, whereas the number of Put replicas is increased. This shows that the fraction of single-replica Put processes is decreased: Without the isClosestAttack, only non-existing nodes are contained in the response of a malicious node – with the isClosestAttack, the malicious node is also included in the response.

Due to the attacks, not all replica nodes are found during the Get operation, so the Get replica count is significantly lower than the Put replica count. The number of involved nodes during Put processes slightly rises, and the average duration of a Put process is shorter because more responding nodes are found and fewer timeouts occur.

6.3.1.4 Routing Attacks Without isClosestAttack in a Non-Partitioned Network

In the next scenario, the network is not partitioned as in the previous configuration (cf. 6.3.1.3), but the malicious nodes still do not claim to be close to the content item. Figure 39 shows that the Put success rate decreases strongly with increasing malicious

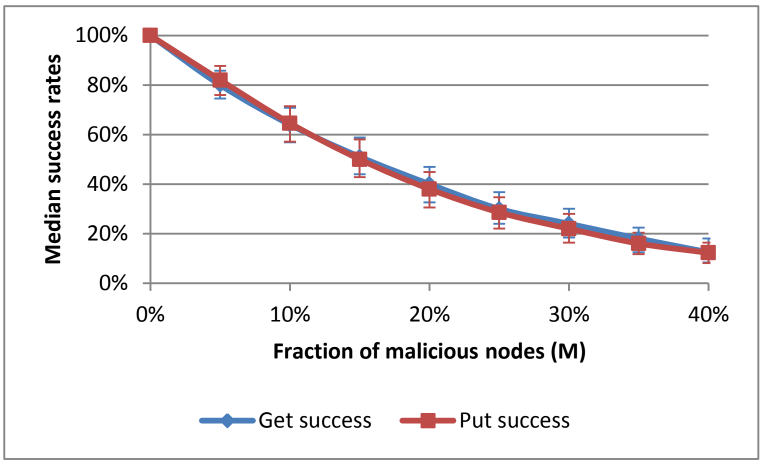


Figure 39: Median Put and Get success rates

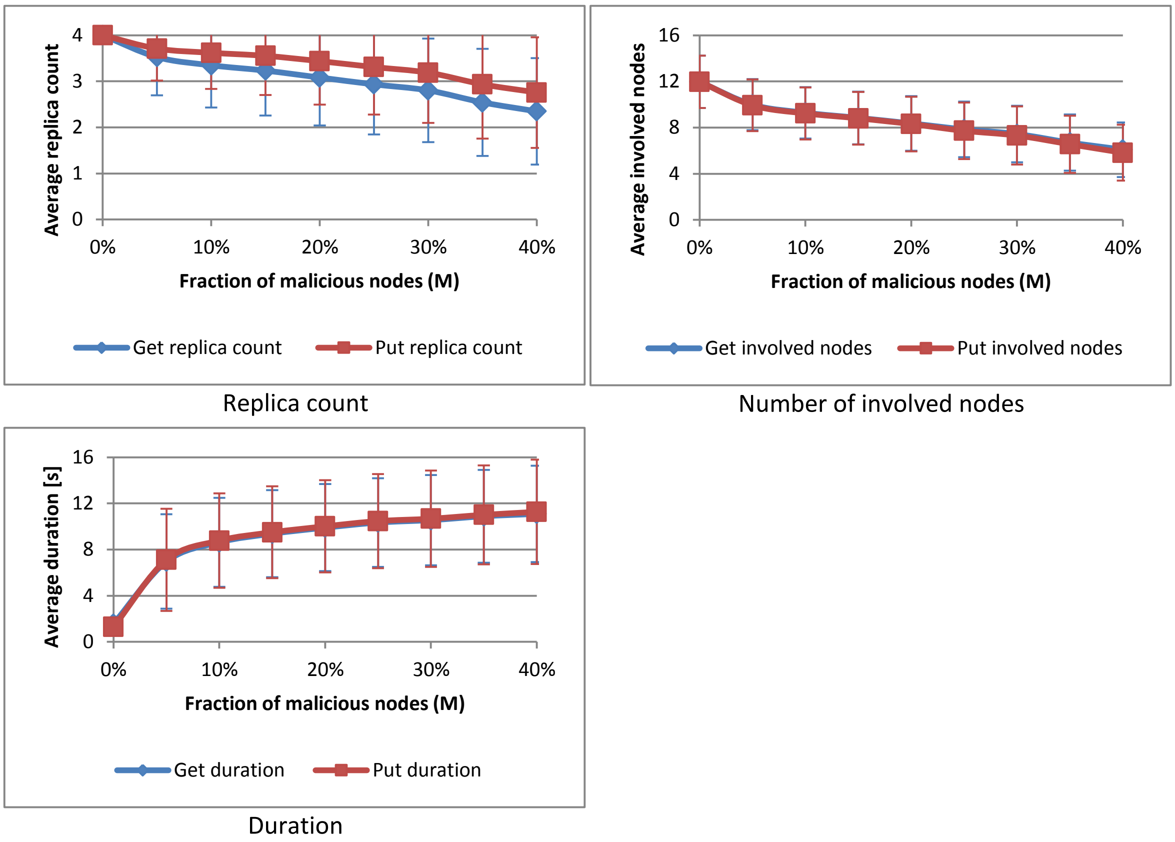


Figure 40: Average replica counts, involved nodes and durations

node ratio and does not spread as wide as before. Again, the impact of the malicious nodes is disproportional to their fraction. It is important to notice that in this scenario, the Get success rate is almost identical to the Put success rate for all malicious node ratios: This shows that without network partitioning and without malicious nodes claiming to be close to the content item ID, the success rate of a Lookup only depends on the ratio of malicious nodes.

Figure 40 shows that also the mean number of involved nodes and the mean duration are nearly identical. Only the number of Get replicas is lower than the number of Put replicas, which shows that, with increasing malicious node ratio, fewer of the replicas can be found.

6.3.1.5 Routing Attacks With isClosestAttack and 50% Attack Probability

In the final baseline configuration for routing attacks, the malicious nodes attack only half of the requests they receive; during the other half, they respond correctly. This drastically reduces the effect of network partitioning, because the inoffensive nodes can create links between network partitions when the malicious nodes behave correctly. This way, partitions can be combined.

So, as shown in Figure 41, the Put success rate is similar to the Put success rate of the configuration without network partitioning in Section 6.3.1.2; the Get success rate is even greater: For M = 40%, it is still at 37% compared to 15% without network partitioning and 100% attack probability. This happens because it is sufficient if only one queried node

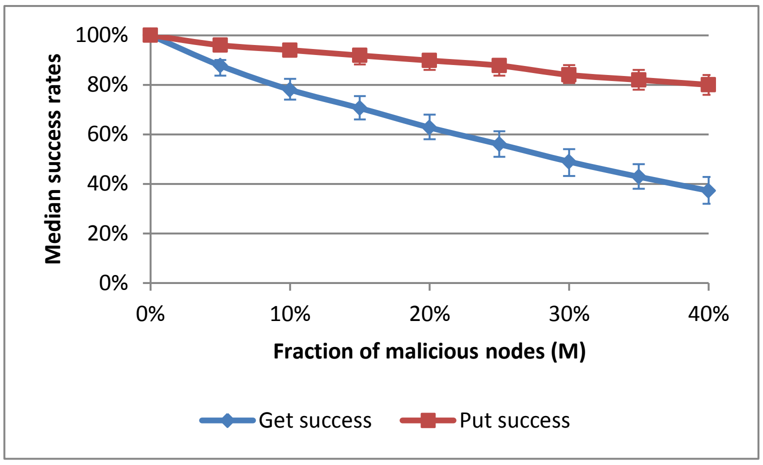


Figure 41: Median Put and Get success rates

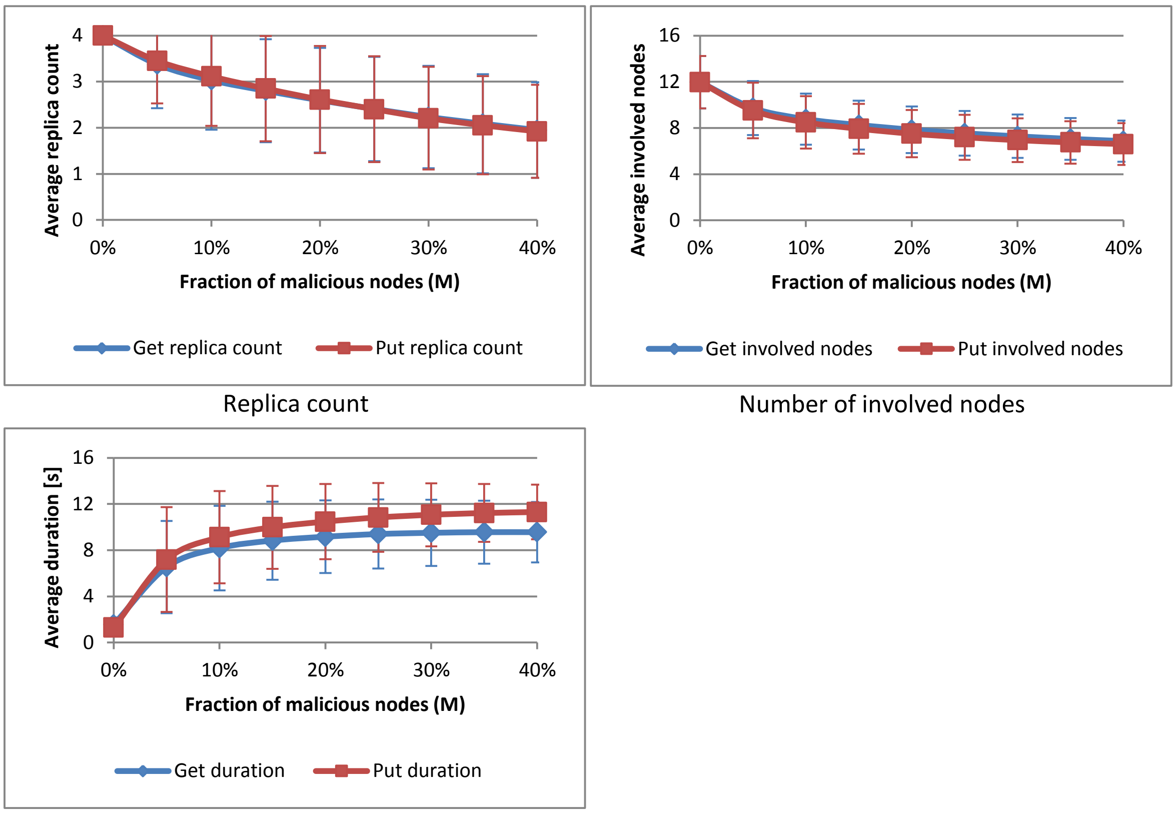


Figure 42: Average replica counts, involved nodes and durations

behaves correctly, either because it is inoffensive or because it is a non-attacking malicious node.

As shown in Figure 42, the replica counts and numbers of involved nodes increase slightly compared to the configuration without network partitioning and 100% attack probability due to the non-attacking malicious nodes. The Put and Get durations are about the same.

6.3.2 Storage Attacks Only

After analyzing the effects of routing attacks against an unprotected network, simulations in which the malicious nodes conduct attacks only on the storage are analyzed in this section. The malicious nodes attack the Get operation by delivering fake content items, i.e., they deliver random data instead of the correct data. Different parameters can influence the effectiveness of the attack: The malicious nodes can act independently or can collude to increase their impact ("SameInvalidContent"). They can attack any arbitrary content item by claiming that they store it or they can only attack items previously stored on them ("OnlyIfPutBefore"). It is also possible that a malicious node responds to a GetHash value with the correct hash of the original content item, but then sends fake data as a response to the GetValue query ("SendOriginalHash"). The differences for this parameter are shown here in detail because they are important for the analysis of TrustedKad.

6.3.2.1 Storage Attacks Using a Fake Hash

In the first configuration, the malicious nodes respond with fake content items and a matching hash, so the hash does not match the hash of the original content item. Figure 43 shows the success ratios of Put and Get processes. The Put success rate is always 100% for colluding and non-colluding nodes (so the two graphs overlap), because there are no attacks on the routing process. Hence, all nodes are reachable and store any item they are queried to store. With increasing fraction of malicious nodes, the Get success rates decrease because the majority decision incorrectly chooses the fake content item. If the malicious nodes collude, i.e., deliver the same fake content item, the attack is more effective as if they act on their own. However, it should be noticed that even with 40% colluding malicious nodes, the Get success rate is still at 71%. So the first impression of the storage attack is that it is effective, but only has significant impact with a high fraction of malicious nodes.

Additionally, the ratio of false positives is indicated. A false positive occurs if the content item that is regarded as valid at the end of a Get process is not the item that was originally stored. The node that performs the Get process is not able to recognize this false positive. Therefore, with TrustedKad enabled, a positive storage rating would be given to the

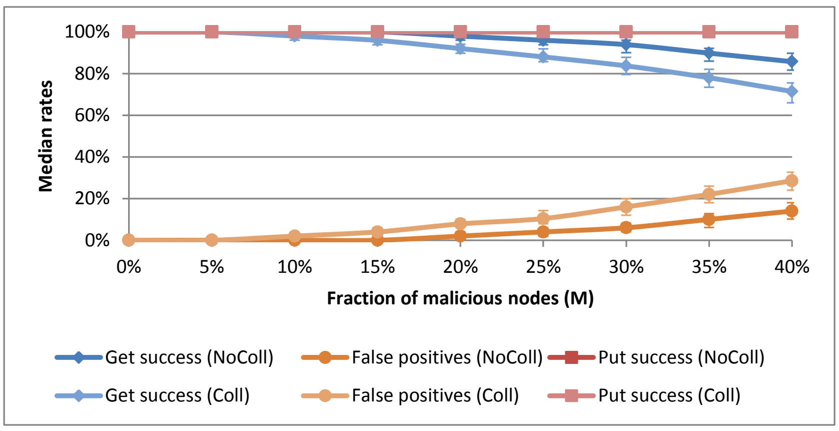
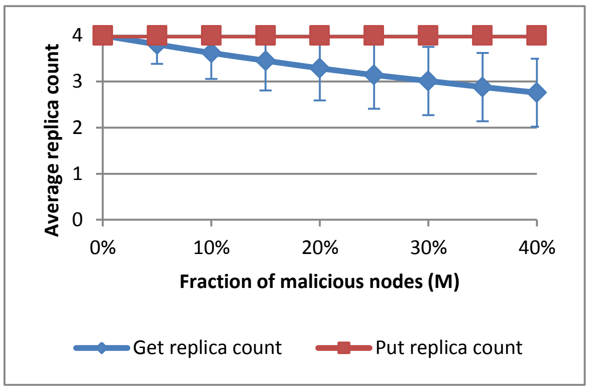
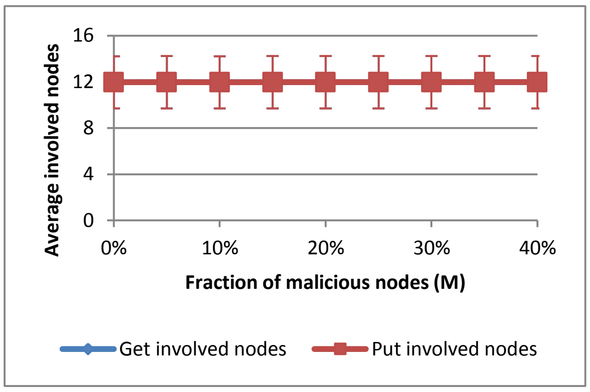


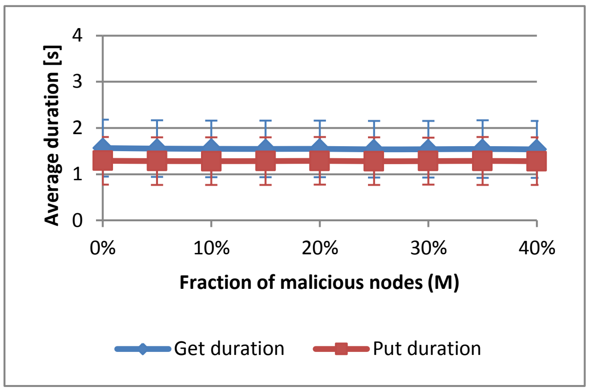
Figure 43: Median Put and Get success and false positives rates



Replica count



Number of involved nodes



Duration

Figure 44: Average replica counts, involved nodes and durations

node(s) delivering the fake content item. All failing Get processes are caused by false positives.

The attack is conducted both with malicious nodes attacking only content previously stored on them and with them attacking every request they receive. However, the results do not differ significantly, so the mean value is shown in figures 43 and 44.

Figure 44 shows the replica count of Put and Get processes. As expected, the demanded four replicas are always stored during a Put process. This is valid for all simulations in which only storage attacks are performed. The number of available replicas during a Get process linearly decreases with increasing fraction of malicious nodes that deliver their fake content instead of the original one. The reason for this is that the more often different versions of a content item are found, the fewer replicas of the chosen version exist.

The number of involved nodes shown in Figure 44 and the duration of the operations shown in are independent from the amount of malicious nodes. This confirms that no routing attacks are present because there are no timeouts: During a storage attack, the nodes respond immediately, in contrast to routing attacks during which timeouts occur when non-existing nodes are queried. Please note that the scale of the duration diagram is changed: The maximum is at 4s instead of 16s in figures 44 and 46. As the values for colluding and non-colluding malicious nodes do not differ significantly, the figures show their mean.

6.3.2.2 Storage Attacks Using the Original Hash

If the malicious nodes send the hash of the original item, they respond with the same hash value as the inoffensive nodes do during the GetHash phase of the Get process. Therefore, the querying node regards this as only one version of the content item. This attack type is introduced as “SendOriginalHash” attack in Section 5.5.8.

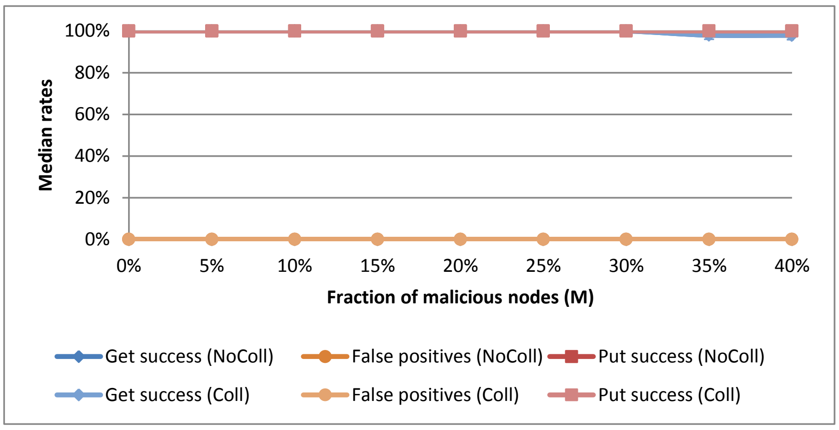
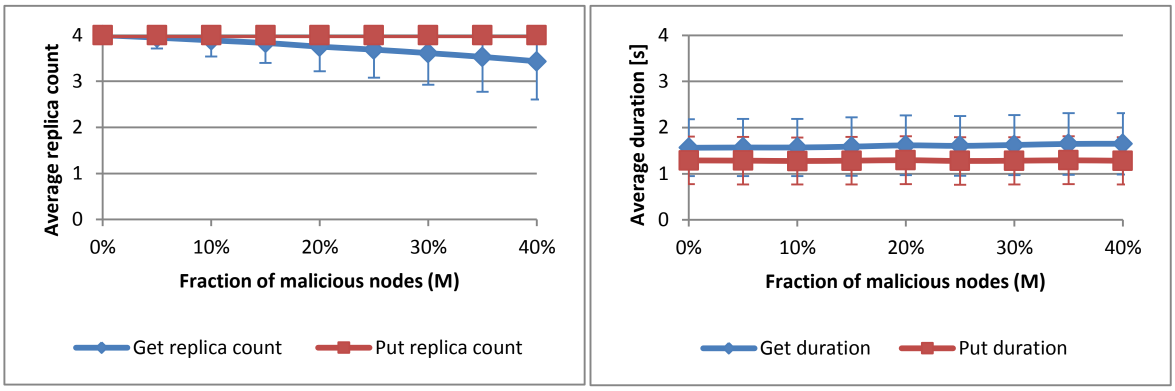


Figure 45: Median Put and Get success and false positives rates



Replica count Duration

Figure 46: Average replica counts and durations

In the GetValue phase, the querying node asks for the full content item. After downloading it from one randomly selected node, it verifies the hash. If the content has been obtained from a malicious node, the hash check fails because the fake data does not match the hash of the original data. So the node queries the next node for the content item until either one node delivers a matching content item or no node is left to query. The Get operation succeeds if a matching content item is received and fails otherwise. This attack type is easy to detect, but can have an influence on the trust values of the malicious nodes if TrustedKad is used as shown later in Section 6.5.4.

The results in Figure 45 show that no false positives occur because this is impossible in this configuration: Fake content does not match the original hash, so it will be discarded. This is also the reason why the results for colluding and non-colluding nodes do not differ: Fake content is always identified.

It is notable that the Get success rate decreases only slightly even at rather high fractions of malicious nodes: The median is lower than 100% only for 35% and 40% of malicious nodes. This is because only one version of the content item is found during the GetHash phase, as the malicious nodes also return the original hash. Then, the Get operation is successful if only one inoffensive node is among the final nodes.

Under the assumption that the routing process always leads to the responsible nodes and that the IDs of the content items are equally distributed, the theoretical probability that at least one of the four nodes answering the GetHash request is inoffensive can be computed with the following formula of the hypergeometric probability distribution:

$$1 - \frac{\binom{Total\ number\ of\ nodes - Number\ of\ inoffensive\ nodes}{4}}{\binom{Total\ number\ of\ nodes}{4}}$$

For 35% of malicious nodes, this results in a success probability of $1 - \frac{\binom{1000-650}{4}}{\binom{1000}{4}} \approx 98.5\%$. For 40% malicious nodes, the probability is about 97.5%; for lower fractions of malicious nodes, the success probability is at least 99%. The simulation results match the theoretical values.

Conducting such an attack therefore seems to be very unattractive, as it has virtually no impact. However, Section 6.5.4 shows the impact of this attack on the storage trust values in a network using TrustedKad.

The numbers of involved nodes do not differ from the numbers from the runs with fake hash values, so the figure is omitted. However, Figure 46 shows that the number of replicas has increased. This is a result from the fact that the full content item value is only fetched from one node. As soon as the hash check succeeds, the query finishes and the other potential replicas that have been determined in the GetHash phase are not verified. As a result, the number of available replicas seems to be higher than it really is, as the malicious nodes also send the original hash in the GetHash phase and can therefore not be identified as malicious. Because the values do not differ significantly if the malicious nodes collude or do not collude, the mean is given in the figure.

The duration of the Get processes slightly increases compared to the previous configuration (cf. 6.3.2.1). This can be explained by the failing hash value checks: Asking the next potential replica node for the full content after a hash check has failed takes time.

6.3.2.3 Storage Attacks Using a Fake Hash and 50% Attack Probability

Figure 47 shows the success rates if the malicious nodes only attack 50% of all requests directed at them. The Put success rate is again at 100% for both colluding and non-colluding nodes. The Get attack impact (1 - success rate) now is less than half as large as with an attack probability of 100% (as shown before in Figure 43). The Get success and false positives rates are stretched: Their median values for 5%, 10%, 15% and 20% malicious nodes and an attack probability of 100% are about the same as the values for 10%, 20%, 30% and 40% for an attack probability of 50% for both colluding and non-colluding nodes. Also in this configuration, colluding nodes have a greater impact on the success rate than non-colluding nodes.

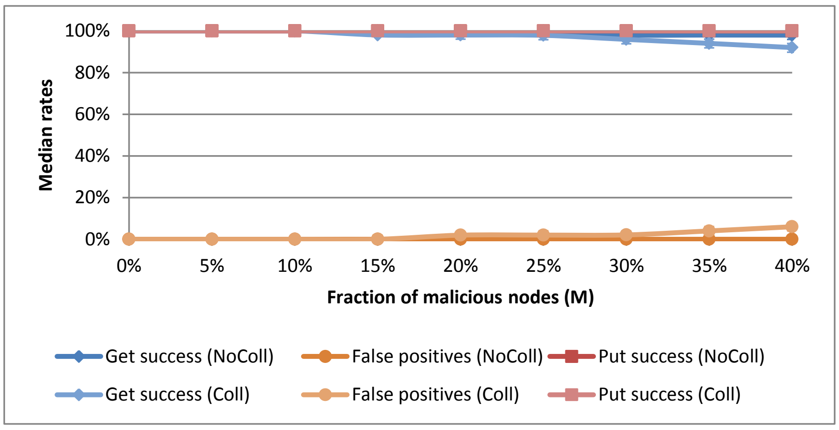


Figure 47: Median Put and Get success and false positives rates

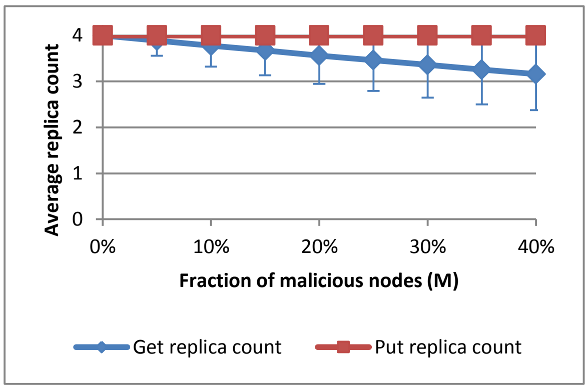


Figure 48: Average replica counts

The numbers of replicas shown in Figure 48 also increase compared to the configuation in which the malicious nodes always attack. The numbers of involved nodes and the duration of the requests do not change compared to Figure 44, so the figures are omitted.

6.3.3 Combined Routing and Storage Attacks

In this section, the results of simulation runs in which the malicious nodes attack both routing and storage operations are presented. Two main configurations are presented as examples. As the routing attacks have a much greater impact on the success rates than the storage attacks, the isClosestAttack parameter is varied between the two presented configurations.

The results of simulation runs without network partitioning and with only 50% attack probability have been analyzed as well, but are not presented in detail in this thesis because they do not yield new insights compared to the previous configurations in which either routing or storage attacks occur (cf. 6.3.1, 6.3.2).

6.3.3.1 Attacks With isClosestAttack

Figure 49 shows the rates of Get and Put success and the false positives. The false positives are shown for attacks in which the malicious nodes attack any arbitrary content item and also for attacks in which they only attack items that have been stored on them previously (see parameter “OnlyIfPutBefore” explained in Section 5.5.8).

It can be seen that the effects of both attack types are combined: As the routing attack has a much greater impact even for low fractions of malicious nodes, the Put and Get success rates of the combined attack are similar to the ones of the routing attacks

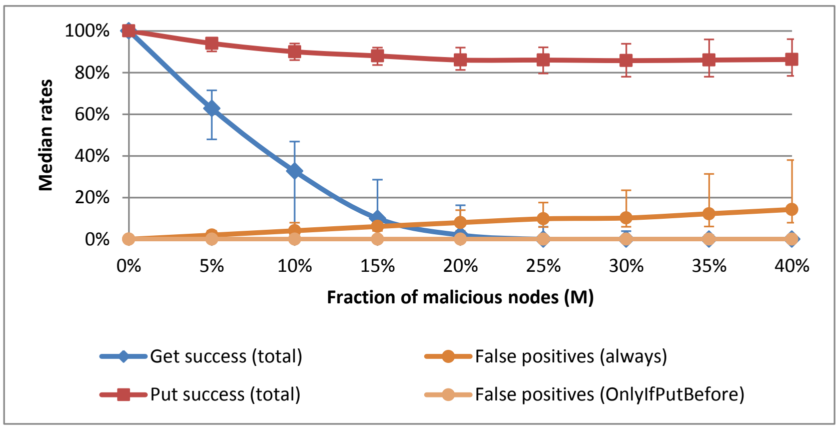


Figure 49: Median Put and Get success and false positives rates

(cf. 6.3.1.1). Because of the fact that the Put operations are always successful if only storage attacks are conducted, the Put success rates are actually equal. However, the Get success rates of the combined attacks are even lower than the routing attack success rates alone, so the storage attacks further decrease them: If the querying node manages to avoid paths with malicious routing nodes, the content item can still be stored on malicious nodes that harm the Get operation, so the failure rates of both attack types are added.

In contrast to the configurations in which only storage attacks occur, the amount of false positives now differs between the configurations in which malicious nodes attack all Get requests and the ones in which they attack only if a Put has been performed on them previously. In the latter case, the false positives rate is always zero, because in almost all cases, the queried item has not been stored on the malicious node before, either due to the routing attacks or due to the network partitioning. But if the malicious nodes attack all Get requests, the rate increases with an increasing fraction of malicious nodes. The values for 25% malicious nodes mean that no Get request was successful (0% Get success rate) and that about 10% of the failed Get requests are caused by successful storage attacks resulting in false positives. The rest fails due to the attacks on the routing process.

The reason why there is a difference now is that due to the attacks on the routing, the “responsible” nodes identified during the Put process are most likely not the same ones that are found by the Lookup of the Get operation. So the content is stored on other nodes, and as the malicious storage nodes do not attack requests for content items that have not been stored previously, no false positives occur.

The replica counts, numbers of involved nodes and durations are about the same ones as if only routing attacks are performed, so the figures are omitted. Due to the dominance of the routing attack effects, the configuration with the SendOriginalHash attack enabled does not differ significantly from the results of this configuration, so these results are also not presented in detail.

6.3.3.2 Attacks Without isClosestAttack

The difference between the previous configuration (cf. 6.3.3.1) and this one is that the isClosestAttack is not performed, so malicious nodes do not claim to be close to the content item. It should be kept in mind that the network is partitioned in both the previous and this configuration.

Just as before, also in this configuration, the corresponding routing attack configuration presented in Section 6.3.1.3 dominates the Put and Get success rates as shown in

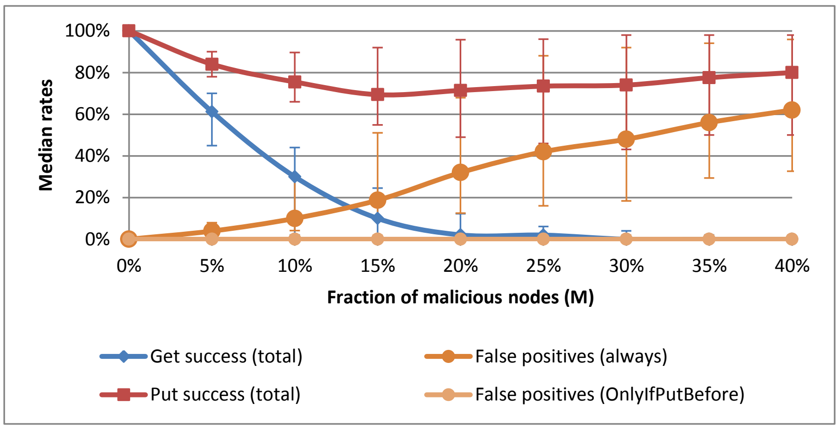
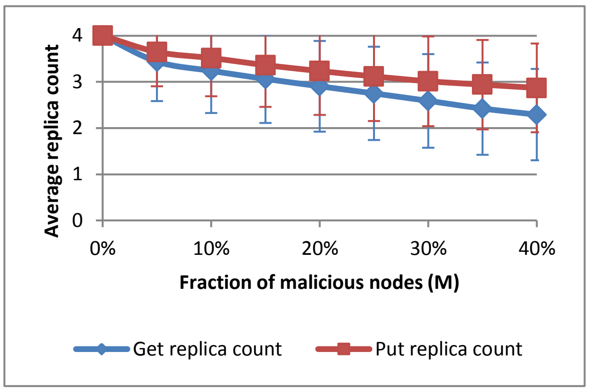
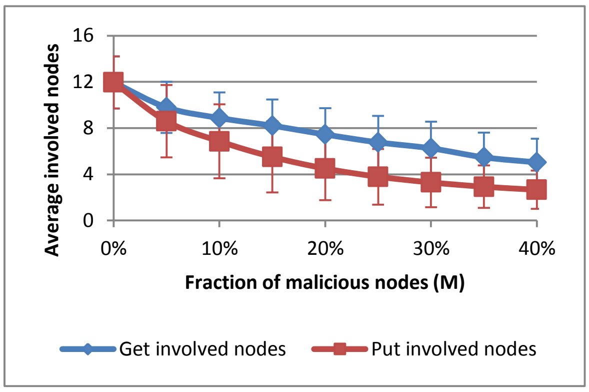


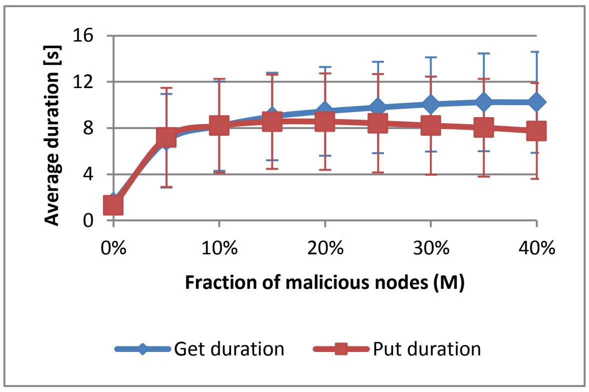
Figure 50: Median Put and Get success and false positives rates



Replica count



Number of involved nodes



Duration

Figure 51: Average replica counts, involved nodes and durations

Figure 50. Again, the Put success rates including the spread between the quartiles are almost identical and the Get success rates are slightly lower now. Also as before, there are no false positives if OnlyIfPutBefore is active.

The great difference occurs when OnlyIfPutBefore is not active, i.e., when the malicious nodes attack all requests directed at them. Compared to the previous configuration (cf. 6.3.3.1), the fraction of Get requests that fail due to false positives is significantly larger. The difference is that the malicious nodes do not claim to be close to the content item any longer, as they do in the previous configuration. The isClosestAttack leads to the fact that the malicious nodes are queried more often, because they are repeatedly inserted into the Lookup list. By this, more random non-existing nodes are "created" that are added to the routing table of the querying node. When that node receives a Lookup request afterwards, it informs about those non-existing nodes, causing the Lookup to fail due to a timeout. That way, the probability that a Get operation fails without any existing node being queried at all is larger. So the fraction of Get requests that fail due to timeouts is greater than in this configuration. Therefore, there are more false positives in this configuration than in the previous one.

Figure 51 shows that the values for the Put operations are about the same as if only routing attacks are conducted compared to Figure 38. The values for Get operations are all increased slightly for larger fractions of malicious nodes. This shows that fewer timeouts occur and more existing nodes are queried during the Get operations.

6.4 How TrustedKad Protects Against Routing Attacks

The network used for the simulations is similar to the Kad network, so the baseline results give an impression how vulnerable one of the widely used P2P networks is. Especially attacks against routing procedures have a large impact on the functioning of the network.

In this section, TrustedKad's measures against routing attacks are evaluated. First, the unchoking mechanism is analyzed and a reasonable routing unchoking ratio is defined for the remaining simulation runs. It should be mentioned that it is not possible to differentiate the results between "no nodes found" and "no trustworthy nodes found" because the usage of the trust values changes the Lookup process from the beginning: The choice which nodes are inserted into the temporary Lookup list depends on trust values and has an influence on the result of the Lookup.

The results of "replica count", "involved nodes" and "duration" are not presented in this section. In the baseline configurations, they are used to further explain and verify the effects of the attacks. Samples of these results of the simulations runs presented in this and the following sections have been analyzed and have not yielded unusual results, so they are omitted in this thesis.

The routing trust value threshold (RT) is varied from -1.0 to 1.0 in steps of 0.1. The storage trust value threshold is always set to -1.0, so all nodes are trustworthy for storage. This does not have any impact on the results, because no storage attacks are conducted. If not mentioned otherwise, the results for RT = -1.0 are similar to the corresponding baseline configuration as shown in Section 6.3.1. Throughout Section 6.4, the term "trust value" is used as a short form for "routing trust value".

The first two simulation configurations show TrustedKad's performance in a scenario in which all features are enabled (one configuration simulates a partitioned network, the other one a non-partitioned network). Afterwards, two worst-case scenarios are presented in which it is possible for an attacker to create fake nodes, i.e., the anti-Sybil-attack measure does not work. Furthermore, the fake cached trust information of non-existing nodes designates them as trustworthy so that all nodes attempt to use them for routing. The last two sections present results of example configurations in which the malicious nodes do not attack all the time, but only 50% of all requests or after 2,000 seconds.

6.4.1 Unchoking Analysis

Before the actual evaluations are performed, the unchoking ratio that is to be used is determined. As explained before in Section 4.13, unchoking aims to enable inoffensive nodes to recover from accidentally received negative ratings. If too many negative ratings are received, a node cannot recover from this because it is not used any longer and cannot receive any positive ratings any more. However, unchoking slightly worsens the success rates, because it leads to the situation that also malicious nodes that deserve the negative ratings are used from time to time. So the unchoking ratio must be chosen carefully.

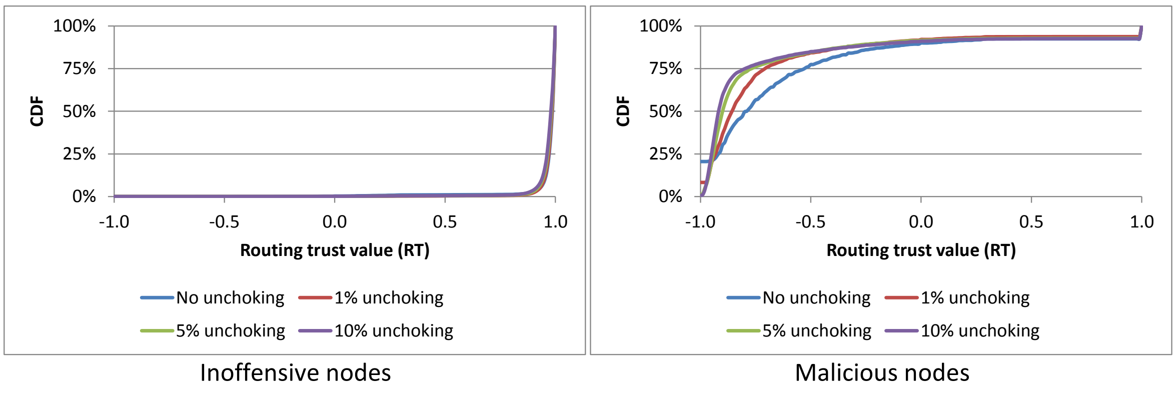


Figure 52: CDF of the routing trust values of inoffensive and malicious nodes

For the determination of a reasonable setting, a configuration that represents a worst-case scenario is used: The fake non-existing nodes are considered trustworthy because the cached trust information delivered by the malicious nodes passes the checks. In addition, the isClosestAttack is performed and the network is partitioned.

As an example, the fraction of malicious nodes is set to 20% and the routing trust value threshold is 0.3, denoting that in order to be trusted, a node must have about twice as many positive ratings as negative ones (cf. 4.6).

During the simulation runs, the mean and standard deviation (in parentheses) of the number of performed unchoking operations by each of the 1,000 nodes is 4.6 (5.5), 20.5 (23.4) and 40.3 (45.5) for 1%, 5% and 10% unchoking ratio, respectively. So with 1% unchoking ratio, about 4,600 times per simulation run, a node is used despite having a too low trust value.

Figure 52 shows the cumulative distribution functions (CDF) of the trust values of inoffensive and malicious nodes in the test configuration. For the inoffensive nodes, there is almost no difference between the unchoking ratios: More than 95% of the inoffensive nodes have a trust value of at least 0.9; for 1% unchoking, the value is best: Only 2% of the inoffensive nodes have a trust value below 0.9.

For the malicious nodes, the results without unchoking are the clearest ones: More than 20% of them have a trust value of -1.0, i.e., they have only negative ratings. For 1% unchoking, this is still true for 8% of the malicious nodes. Also for 1% unchoking, most of the malicious nodes (94%) have a trust value below the trust value threshold of 0.3; the fractions for the other settings are 93% (5% unchoking) and 92% (no and 10% unchoking), respectively. As a result, inoffensive and malicious nodes can be clearly identified using the trust values.

In Figure 53, the CDF of the Get success rates of the four settings are shown. The configuration without unchoking has the best results, as more nodes achieve higher Get success rates. The medians of the configurations without and with 1% unchoking are about the same at 63%, while the other two configurations reach a median of about 54%. The jumps in the graphs result from the fact that most nodes perform 50 Get operations during the simulation time, so the greatest changes of the number of nodes occur at multiples of $\frac{1}{50}$.

For further analyses, the difference between the median trust values of the configurations without and with 1% unchoking are shown in Table 11; a negative value means that the value for 1% unchoking is lower than the one for no unchoking. The median trust values of the inoffensive nodes do not differ significantly (the only non-zero value at M = 35% and RT = 0.1 is caused by rounding). The median trust values of the malicious nodes decrease slightly in the configuration with 1% unchoking. This happens because

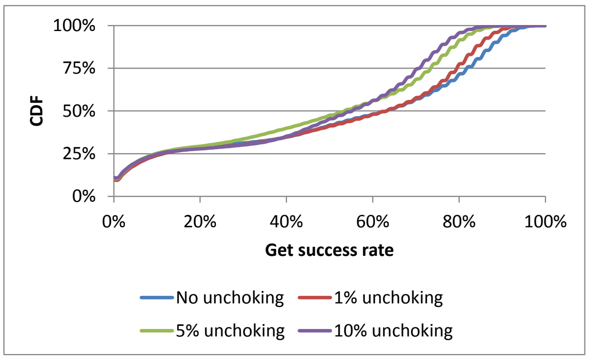


Figure 53: CDF of the Get success rates

Table 11: Differences between median routing trust values of no unchoking and 1% unchoking

Inoffensive nodes										
		Fraction of malicious nodes								
		0%	5%	10%	15%	20%	25%	30%	35%	40%
Storage trust threshold	0.0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-0.01	0.00
	0.2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Malicious nodes										
		Fraction of malicious nodes								
		0%	5%	10%	15%	20%	25%	30%	35%	40%
Storage trust threshold	0.0		0.06	-0.02	-0.03	-0.05	-0.06	-0.10	-0.10	-0.08
	0.1		0.06	-0.03	-0.03	-0.06	-0.07	-0.10	-0.09	-0.06
	0.2		0.06	-0.03	-0.03	-0.05	-0.07	-0.10	-0.13	-0.08
	0.3		0.06	-0.02	-0.02	-0.06	-0.07	-0.12	-0.13	-0.06
	0.4		0.07	-0.03	-0.03	-0.05	-0.07	-0.10	-0.13	-0.04
	0.5		0.06	-0.03	-0.03	-0.05	-0.09	-0.11	-0.09	-0.09
	0.6		0.00	-0.03	-0.04	-0.08	-0.09	-0.10	-0.12	-0.04
	0.7		-0.04	-0.06	-0.08	-0.11	-0.08	-0.14	-0.06	-0.10
	0.8		-0.07	-0.11	-0.14	-0.12	-0.22	-0.18	-0.18	-0.04

Table 12: Differences between median Get success rates of no unchoking and 1% unchoking

Get success rate										
		Fraction of malicious nodes								
		0%	5%	10%	15%	20%	25%	30%	35%	40%
Storage trust threshold	0.0	0%	0%	-2%	-4%	-4%	-4%	2%	0%	0%
	0.1	0%	-2%	-2%	-4%	3%	-2%	2%	0%	0%
	0.2	0%	-2%	-2%	-3%	-3%	-1%	2%	0%	0%
	0.3	0%	-2%	-2%	-2%	0%	2%	4%	2%	0%
	0.4	0%	-2%	-2%	-2%	-4%	-5%	1%	2%	0%
	0.5	0%	0%	-2%	-5%	-5%	0%	-2%	0%	0%
	0.6	0%	0%	-2%	-3%	-4%	0%	0%	0%	0%
	0.7	0%	-2%	-2%	1%	6%	-2%	0%	0%	0%
	0.8	0%	0%	0%	-4%	-2%	0%	0%	0%	-2%

unchoking leads to the fact that the actually untrusted nodes are used during routing. However, as they are again identified as malicious and rated negatively, their trust values decrease further. Only for M = 5% and RT ≤ 0.5, the configuration without unchoking has better results. This is explained by the fact that the median trust value is already at -1.0 in these scenarios.

Table 12 shows that also the Get success rates of the two configurations do not differ significantly: For some configurations, they improve with unchoking, for others, they worsen.

As a result of the unchoking analysis, an unchoking ratio of 1% is used during routing trust value evaluations. As explained before, unchoking nodes is necessary. The results show that lower unchoking ratios yield better results, so the ratio chosen for the remaining tests is 1%, because it is closest to the configuration without unchoking and even outperforms it in some cases.

6.4.2 TrustedKad in a Partitioned Network

In this and the following section, configurations are analyzed in which all of TrustedKad's measures are enabled and function successfully. The subsequent two sections present results of configurations in which the malicious nodes are able to fake the cached trust information of the non-existing nodes to show that TrustedKad is able to improve the security of the network even in such a worst-case scenario.

The results of the configurations with and without the isClosestAttack are similar in this and the following section; the graphs for both parameter settings are included anyway as a reference for the following results in which differences occur. The similarity is expected: In this scenario, the non-existing nodes returned by malicious nodes are identified as fake because the verification of the cached trust information fails. Therefore, they are not added to the temporary Lookup list and only existing nodes are contained in it. If the

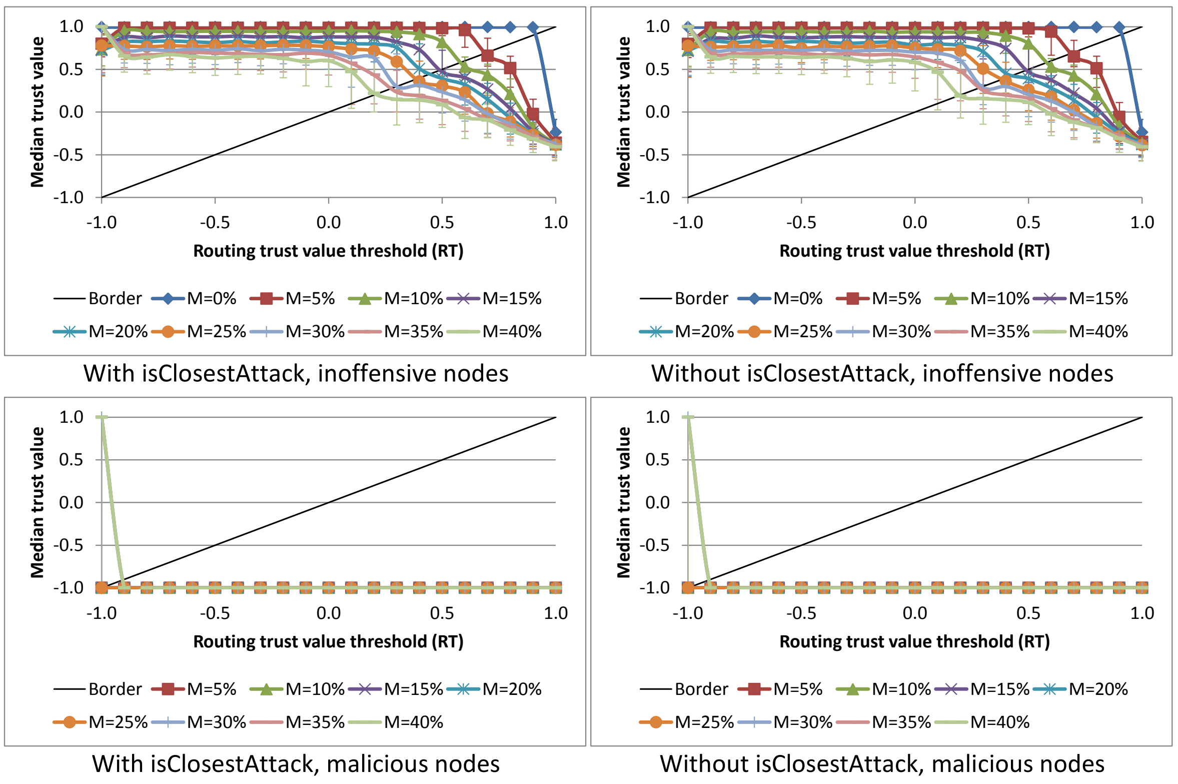


Figure 54: Median trust values of inoffensive and malicious nodes

cached trust information can be faked successfully, the Lookup list also contains non-existing nodes, which leads to timeouts. Then, with isClosestAttack, a malicious node is inserted into the Lookup list repeatedly, which results in a difference between the configurations with and without isClosestAttack as described in the subsequent two sections.

The line “border” in the diagrams is the border between the trusted zone and the untrusted zone: Values above the line indicate trustworthiness; nodes with a value below the line are not trusted. So if the median is below the line, at least 50% of the nodes are not trustworthy. Ideally, all quartiles of the inoffensive nodes are on or above the line, and all quartiles of the malicious nodes are below the line.

The median trust values of the malicious nodes are -1.0 in almost all cases as shown in Figure 54, so they are clearly identified as malicious. This is also true for the inoffensive nodes for all M and RT $\leq$ 0.2. For larger RT, the median trust values decrease below the trust threshold towards the minimum of -0.4; this happens the earlier, the larger M is. The reason why the inoffensive nodes receive negative ratings is as follows: When a node

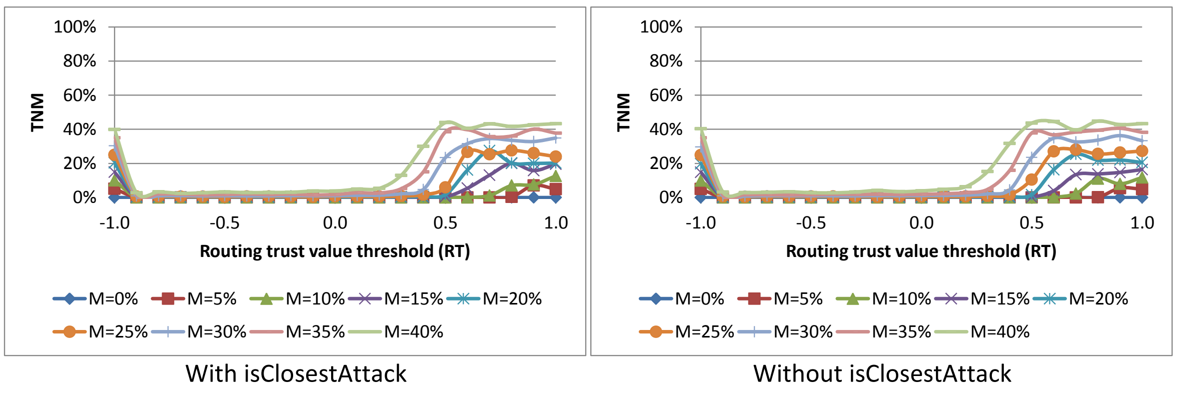
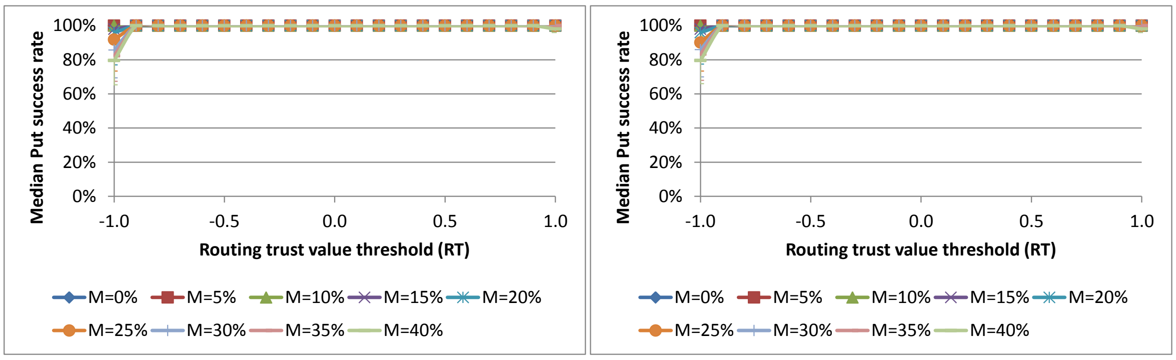


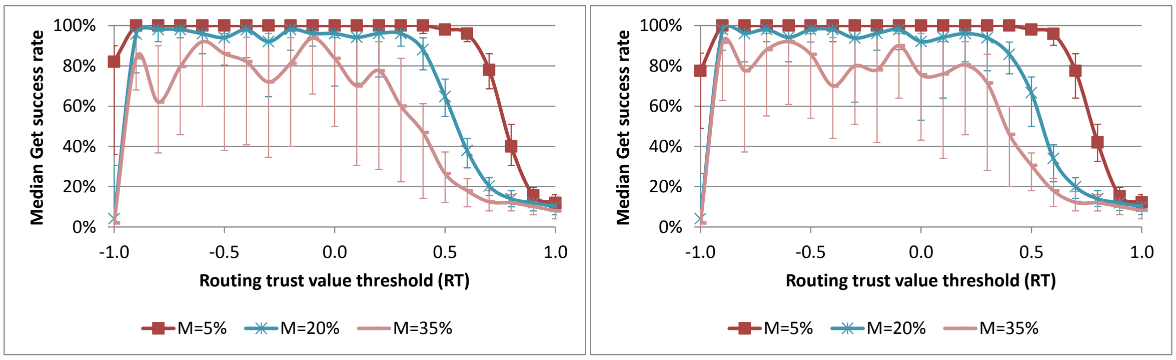
Figure 55: Fractions of trusted nodes that are malicious (TNM)



With isClosestAttack

Without isClosestAttack

Figure 56: Median Put success rates



With isClosestAttack

Without isClosestAttack

Figure 57: Median Get success rates

performs a Lookup, it inserts all nodes that are included in the response message into its routing table regardless of their trust value because it needs to be able to inform about all nodes (cf. 4.10.2). When it receives a Lookup request from another node afterwards, it responds with nodes from its routing table. Among these nodes, there are also non-existing fake nodes it received during its own previous Lookups. Because of this, the inoffensive node does not lead to the final node or even to any responding other node and therefore receives a negative rating.

Starting at a trust threshold of 0.3 with M ≥ 30%, the median trust value of the inoffensive nodes is below the threshold. This means that less than half of the inoffensive nodes are regarded as trustworthy. However, Figure 55 shows that even in these cases in which the median trust values of the inoffensive nodes are lower than the threshold, more than half of the trusted nodes are inoffensive. The only reason why there are trusted malicious nodes is because they are still in the grace phase: The trust values of the malicious nodes are either -1.0 (not trustworthy) or 1.0 (still in grace phase). Due to the fact that fewer inoffensive nodes are trusted for larger RT, the fraction of trusted nodes that are malicious increases. The values for RT = -1.0 are included for completeness: All nodes are regarded as trustworthy, so the fraction of trusted nodes that are malicious is equal to M.

In Figure 56, it can be seen that all Put operations are successful except for when all nodes are trusted regardless of their trust value (RT = -1.0). So TrustedKad enables the nodes to find other nodes on which they can store their content items even if the network is partitioned. As the results for RT = -1.0 are similar to the results of the corresponding baseline runs (cf. 6.3.1.1, 6.3.1.3), TrustedKad improves the functioning of the network significantly: Regardless of the fraction of the malicious nodes, the Put operations are always successful.

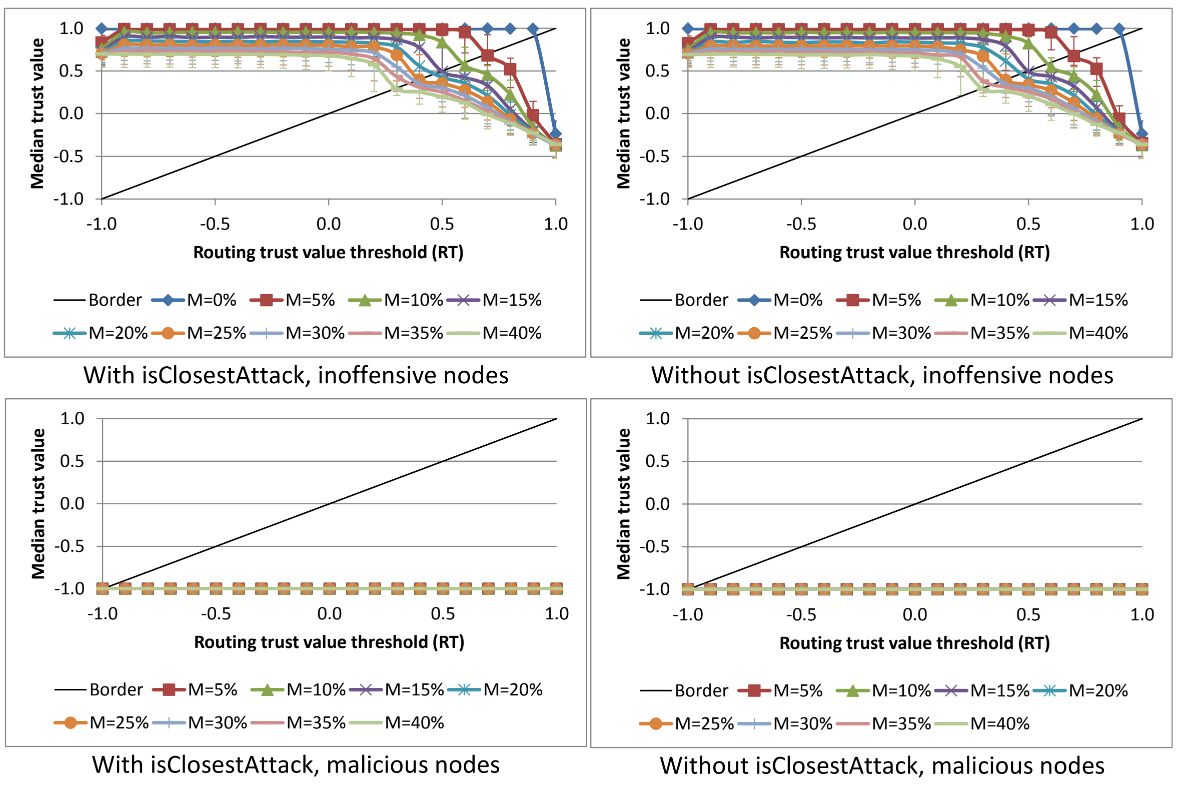
Figure 57 shows the median Get success rates for M = 5%, M = 20% and M = 35%; the other results can be found in Annex A.1.1. Also here, the Get success rates improve significantly compared to the baseline results. For M = 5%, it is greater than 95% for RT ≤ 0.6 with the baseline being less than 70%. Due to the network partitioning, the results have a large spread between the quartiles, as it is visible in the corresponding baseline simulations in sections 6.3.1.1 and 6.3.1.3. As explained in Section 6.3.1.1, the queried content items cannot be found if they have been published in an unreachable partition. This effect grows stronger with an increasing fraction of malicious nodes, as more partitions are formed. Also because of this, the median fluctuates more the larger the fraction of malicious nodes is.

Nevertheless, the Get success rates increase significantly compared to the baseline runs. This is because the creation of network partitions is limited considerably by TrustedKad: During Bootstrap operations, a grace value of 0 is used during the computation of routing trust values (cf. 4.10.1). Therefore, as soon as a node has received its first routing rating, it is evaluated if it is trustworthy for routing operations before it is used as Bootstrap node. So nodes that have received too many negative ratings are not used for Bootstrap. This way, the number of partitions is limited significantly because if malicious nodes are used less often for Bootstrap operations, fewer partitions are created.

6.4.3 TrustedKad in a Non-Partitioned Network

In the previous section 6.4.2, the network is partitioned. This section analyzes a network in which only inoffensive nodes are used for the Bootstrap, leading to a non-partitioned network. TrustedKad enables a node to enhance the security of its Bootstrap operation by verifying the cached trust information of a node at its trust management nodes as described in Section 4.10.1. So the normal case for a network that uses TrustedKad should be that it is not partitioned.

Figure 58 shows that the median trust values of the inoffensive nodes are slightly greater



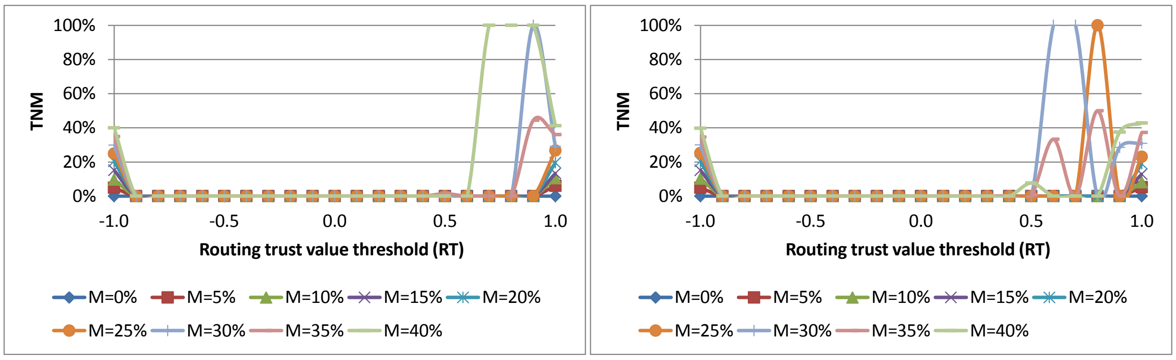
With isClosestAttack, inoffensive nodes
Without isClosestAttack, inoffensive nodes
With isClosestAttack, malicious nodes
Without isClosestAttack, malicious nodes

Figure 58: Median trust values of inoffensive and malicious nodes

than the ones of the previous section. The decrease of the lines, however, starts at about the same values for M and RT. The median trust values of the malicious nodes are now always -1.0 without exceptions.

The fractions of trusted nodes that are malicious are shown in Figure 59. In almost all cases, no malicious nodes are trusted. However, there are several outliers in the figure: They occur in parameter combinations in which the total amount of trusted nodes is very low – in several combinations, there are no trusted inoffensive nodes at all, resulting in outliers in which the fraction of trusted nodes that are malicious is 100% if there is a single malicious node that is still in its grace phase. In the previous configuration (cf. 6.4.2), there are significantly more nodes in total that are trusted, so outliers do not occur.

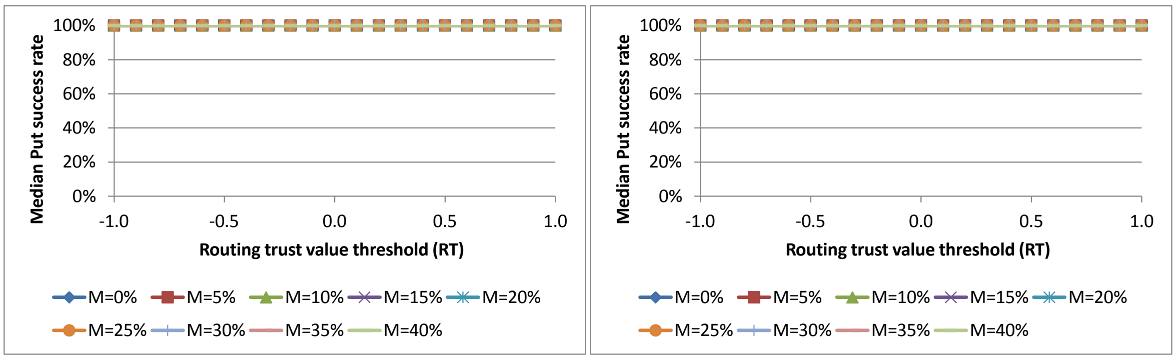
The results of the Put operations shown in Figure 60 are similar to the previous configura-



With isClosestAttack

Without isClosestAttack

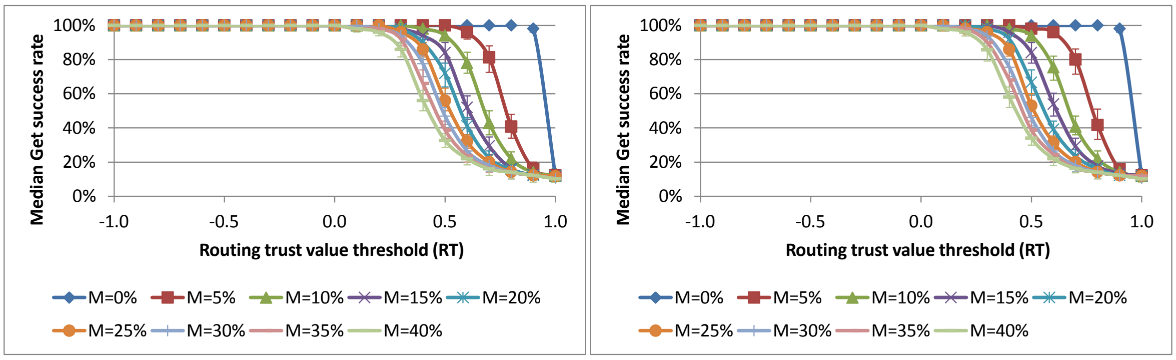
Figure 59: Fractions of trusted nodes that are malicious (TNM)



With isClosestAttack

Without isClosestAttack

Figure 60: Median Put success rates



With isClosestAttack

Without isClosestAttack

Figure 61: Median Get success rates

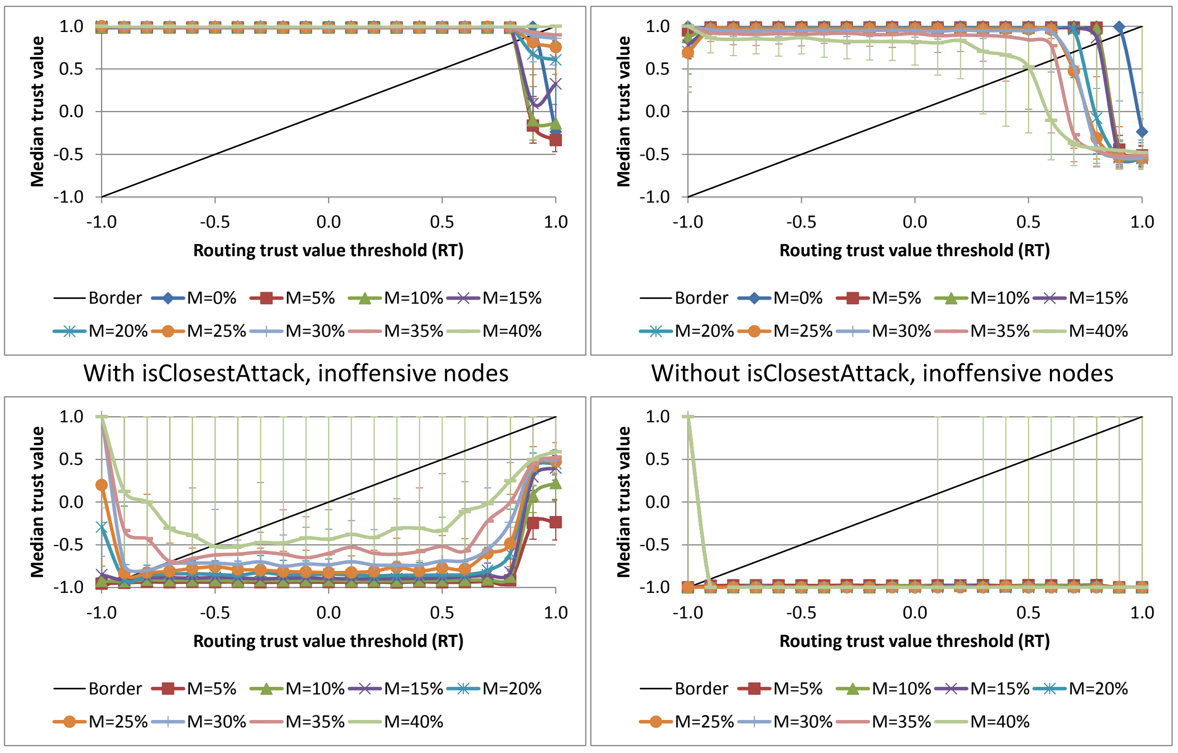
tion. However, now, the median Put success rate is 100% even for RT = -1.0. This is due to the fact that every node knows at least one inoffensive node, namely its bootstrap node.

The median Get success rates shown in Figure 61 are significantly greater than in the partitioned network. In addition, they do not have such large spreads between the quartiles as before, so all lines can be included in the figure. As with the Put success rates, even for RT = -1.0, the median Get success is 100%. This shows that TrustedKad is able to significantly improve the security of the network. The median Get success rates begin to decrease at about the same value of RT as when the median trust value of the inoffensive nodes starts to decrease as well. So the number of trustworthy nodes decreases quickly, and with it the Get success rate, because the nodes are not able to reach the nodes on which the content item has been originally stored by the publishing node.

6.4.4 TrustedKad in a Partitioned Network With Successfully Faked Cached Trust Information

The results of the previous two sections 6.4.2 and 6.4.3 show that TrustedKad is able to significantly improve the Get success rates compared to the baseline. However, as explained in Section 2.5.1, the Sybil attack and therefore the creation of fake cached trust information cannot be completely averted. So in the following configurations, the non-existing nodes contained in the Lookup response messages of the malicious nodes are regarded as trustworthy. The querying nodes therefore attempt to contact those nodes. The effects of this worst-case scenario are analyzed in this and the following section, starting with a partitioned network.

Figure 62 shows the median trust values of inoffensive and malicious nodes. It contains two different configurations, one with isClosestAttack enabled and one without. In both configurations, malicious nodes do not inform about other existing nodes, but only about non-existing nodes. In the configuration with isClosestAttack, the malicious nodes additionally declare themselves as responsible for the content item's ID space and insert



With isClosestAttack, inoffensive nodes

Without isClosestAttack, inoffensive nodes

With isClosestAttack, malicious nodes

Without isClosestAttack, malicious nodes

Figure 62: Median trust values of inoffensive and malicious nodes

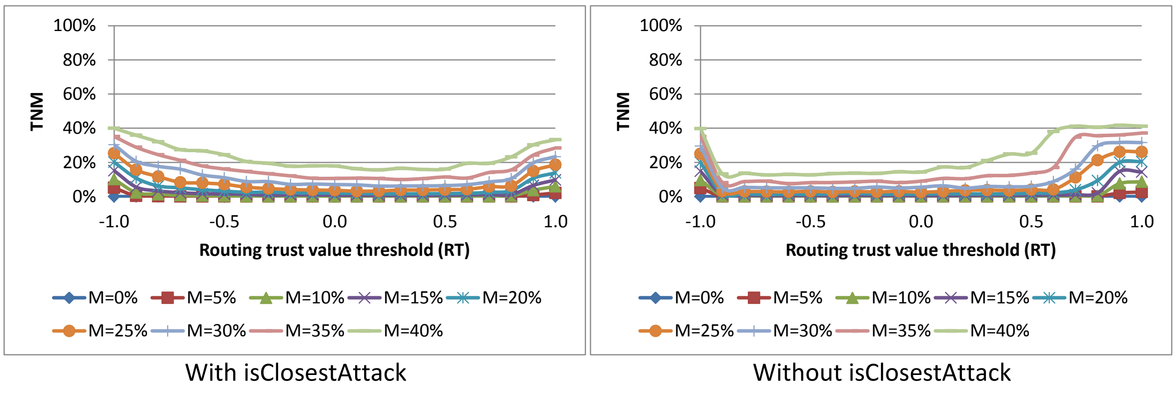


Figure 63: Fractions of trusted nodes that are malicious (TNM)

themselves into the response list as the only existing node in it. The results of the configurations with and without isClosestAttack differ now because the malicious nodes' responses contain one reachable node if the attack is conducted and no reachable node if not.

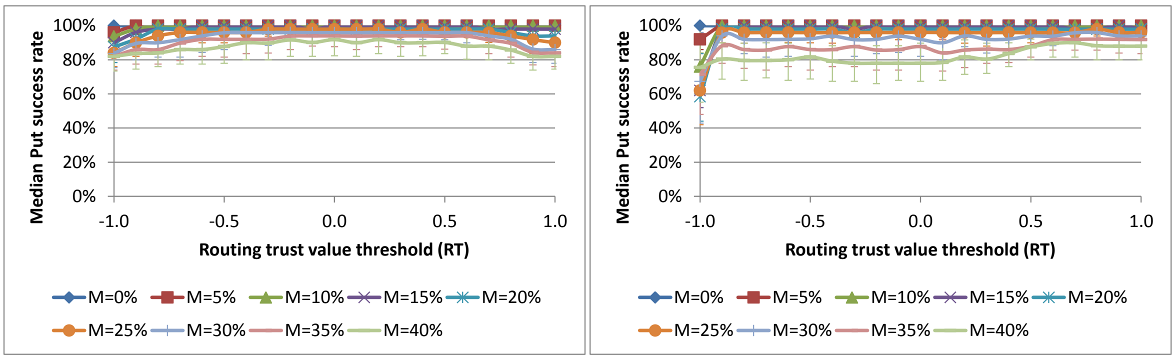
With isClosestAttack, the inoffensive nodes are clearly identified as such for RT ≤ 0.8. The median trust values of the malicious nodes are below the border for -0.5 ≤ RT ≤ 1.0. For M ≤ 35%, this is also true for the 75% quartile for -0.1 ≤ RT ≤ 0.6, and for -0.9 ≤ RT ≤ 0.8, the median trust value of the malicious nodes is below 0, which indicates more negative than positive ratings.

Without isClosestAttack, inoffensive nodes are also clearly identified, but with slightly greater spread than with isClosestAttack. Their median trust value decreases earlier for M ≥ 20%. This is because with the isClosestAttack, the malicious nodes always insert themselves in the response list, which leads to them being queried repeatedly during one Lookup operation. Without the isClosestAttack, the malicious nodes only inform about non-existing nodes and are not queried several times per Lookup. So without the isClosestAttack, more inoffensive nodes are potentially queried and can get false negative ratings, e.g., for only informing about non-existing nodes that they previously inserted into their routing tables.

The malicious nodes are identified more clearly than with isClosestAttack present: Their median trust value is always -1.0 for RT ≥ -0.9; only for M ≥ 30% and RT = -1.0, all nodes, inoffensive and malicious ones, have a median trust value of 1.0. The 75% quartile of the median trust values is also -1.0 in most cases; the only exceptions are M = 40% and RT = 0.1 and RT ≥ 0.3, M = 35% and RT ≥ 0.7 and M = 30% and RT ≥ 0.9. So in part, the isClosestAttack is successful, as it increases the median trust values of the malicious nodes.

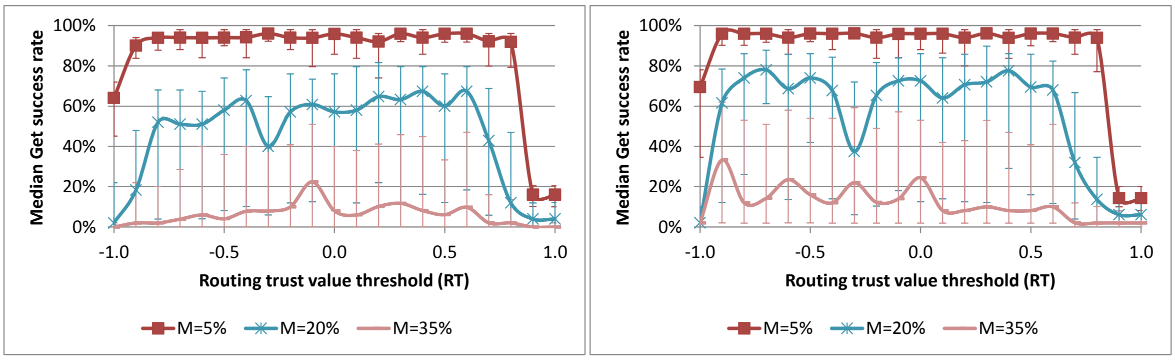
However, Figure 63 shows that the fraction of trusted nodes that are malicious is always below the fraction the malicious nodes have in total. Only without isClosestAttack and large M and RT, the fraction of trusted nodes that are malicious is one or two percentage points larger than the fraction of malicious nodes in total. For the vast majority of parameter combinations, the fraction is well below 10%.

In Figure 64, the median Put success rates are shown. Compared to the baseline (about the same as the values for RT = -1.0; cf. 6.3.1.1, 6.3.1.3), they have increased in all cases, so TrustedKad is able to counter the routing attacks in scenarios with network partitions and with and without isClosestAttack.



With isClosestAttack Without isClosestAttack

Figure 64: Median Put success rates



With isClosestAttack Without isClosestAttack

Figure 65: Median Get success rates

The Get success rates are only shown for fractions of malicious nodes of 5%, 20% and 35% to retain clarity in Figure 65. The diagrams for all fractions of malicious nodes can be found in Annex A.1.2. TrustedKad improves the Get success rates significantly: For M = 5%, the median Get success rate is greater than 90% for $-0.9 \leq RT \leq 0.8$; the baseline was only at 63% (cf. 6.3.1.1). Even with 35% malicious nodes present, the median Get success rate is greater than the value of 0% of the baseline; the 75% quartiles are often even greater than 40%. For the attacks without isClosestAttack, the tendency is the same with slightly greater values in almost all cases. For greater M, the spread between the quartiles increases because of the network partitioning. The difference is shown in the following section in which a non-partitioned network is analyzed.

The reason for the greatly improved Get success despite the successfully faked cached trust information rates is that with TrustedKad, the node IDs of the non-existing nodes cannot be chosen freely any longer. So now, the non-existing nodes are not necessarily the closest nodes to the target ID. Therefore, the possibility that there is still an existing node among the top nodes of the Lookup list after a malicious node has sent information about fake nodes is much greater than before. The baseline values shown in Sections 6.3.1.1 and 6.3.1.3 are generally even worse than the values for RT = -1.0 for the same reason.

Due to the fact that more existing nodes are asked for content items, almost no Get operations fail anymore because no responding node can be found. Instead, almost all operations fail because the existing nodes respond that the item does not exist. This is caused by the network partitioning and the fact that in this configuration, all nodes would respond with the correct content item if it was stored before because no storage attacks are conducted.

6.4.5 TrustedKad in a Non-Partitioned Network With Successfully Faked Cached Trust Information

In this section, the effects of routing attacks on a non-partitioned network in which the malicious nodes can successfully fake the cached trust information are analyzed. Compared to the previous section 6.4.4 with a partitioned network, the median trust values of both inoffensive and malicious nodes spread less as shown in Figure 66. With the isClosestAttack, the median trust values of the inoffensive nodes are also at 1.0 for RT ≤ 0.7. Afterwards, they decrease slightly stronger. As a balance, the median trust values of the malicious nodes also decrease significantly, as all quartiles are in the untrusted zone (except for RT ≤ -0.9). So the malicious nodes are clearly identified.

If the malicious nodes do not perform the isClosestAttack, this situation is clearer yet: Even the 75% quartiles of the trust values of the malicious nodes are never greater than -0.95. As before in the partitioned network, the median trust values of the inoffensive nodes decrease earlier; but compared to that configuration, they are greater now

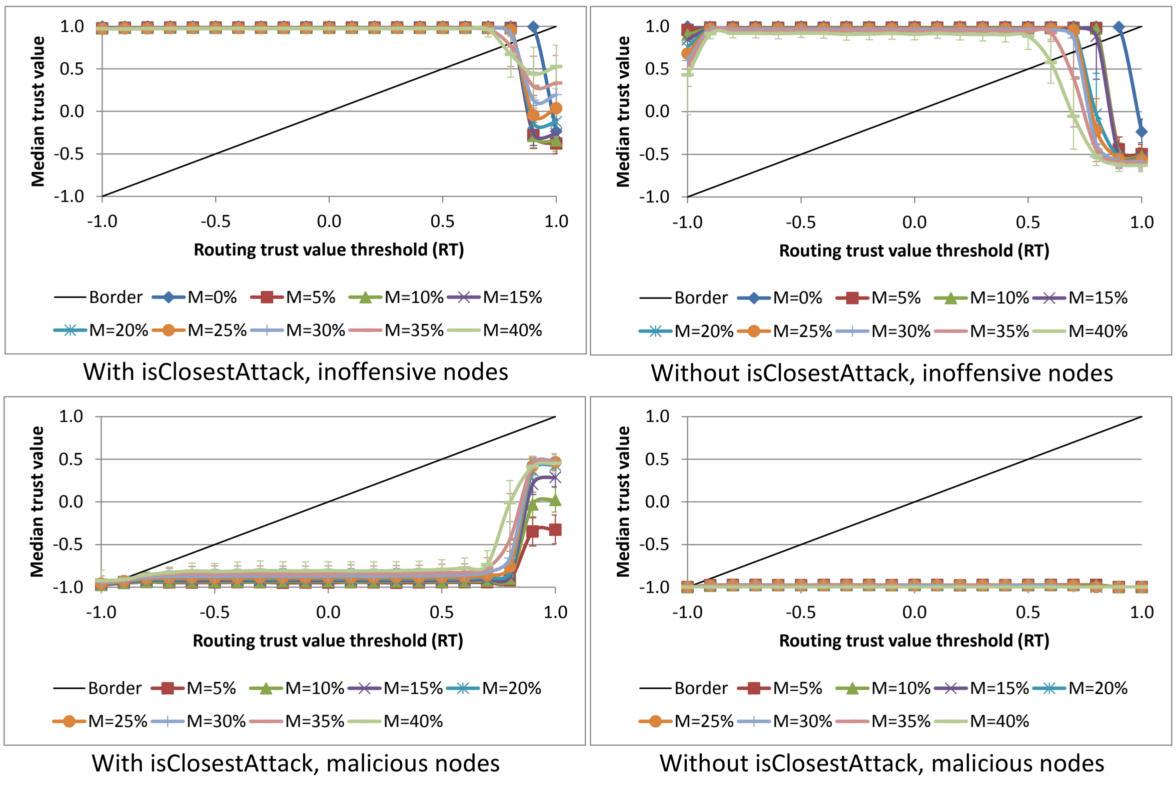


Figure 66: Median trust values of inoffensive and malicious nodes

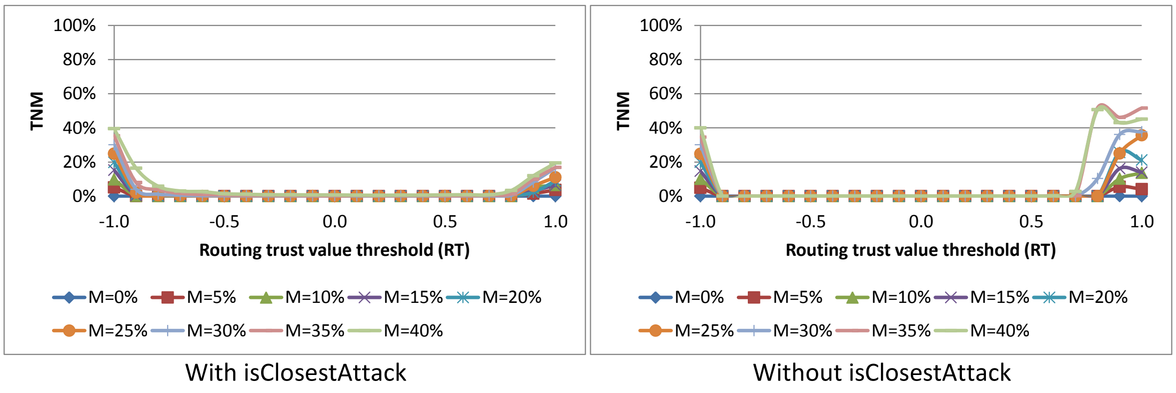


Figure 67: Fractions of trusted nodes that are malicious (TNM)

and decrease later for M ≥ 25%.

Figure 67 shows that the fractions of trusted nodes that are malicious are considerably lower than in the partitioned network; they are close to 0% in most cases. The increases towards RT = 1.0 result from the fact that the total number of trusted nodes rapidly decreases because the median trust values of the inoffensive nodes quickly fall below the threshold for these large RT. So a much smaller fraction of malicious nodes that has a trust value above the threshold is required to have a significant effect on the network.

The result that the nodes perform better in the non-partitioned network can also be observed in the median of the Put success rates in Figure 68, as they also increase. As in the previous section, they increase compared to the baseline (cf. 6.3.1.2, 6.3.1.4).

The Get success rates can be shown for all analyzed fractions of malicious nodes in Figure 69 because they only spread slightly and significantly less than in the partitioned network. In addition, they are also considerably greater, especially for large fractions of malicious nodes. So the nodes on which the content item has been stored can be found in many cases. Thus, the effect of the routing attacks with successfully faked cached trust information is diminished if at least the Bootstrap process can be protected successfully so that no network partitioning occurs.

This example shows very impressively how important the protection of the Bootstrap operation is. Both with and without the isClosestAttack, the threshold up to which the results are acceptable is RT = 0.7; larger thresholds result in much fewer nodes that are above the threshold and therefore also in worse Get success rates.

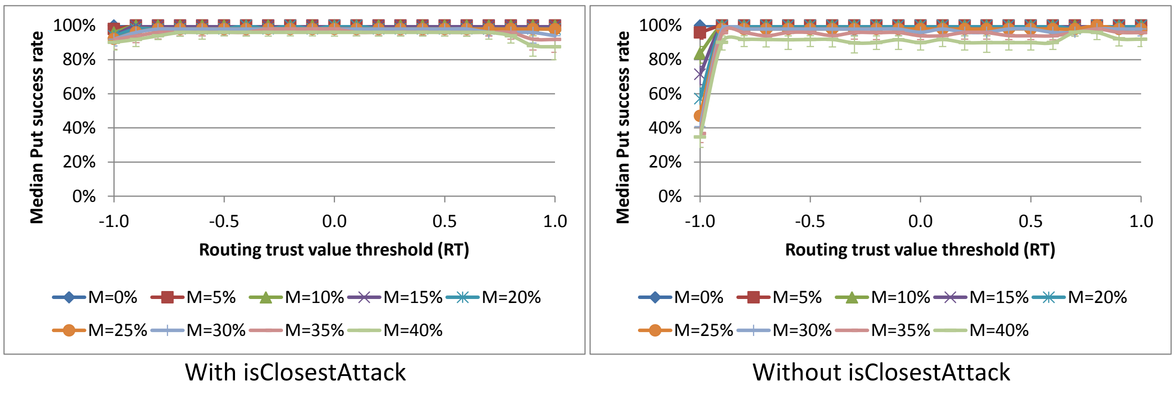


Figure 68: Median Put success rates

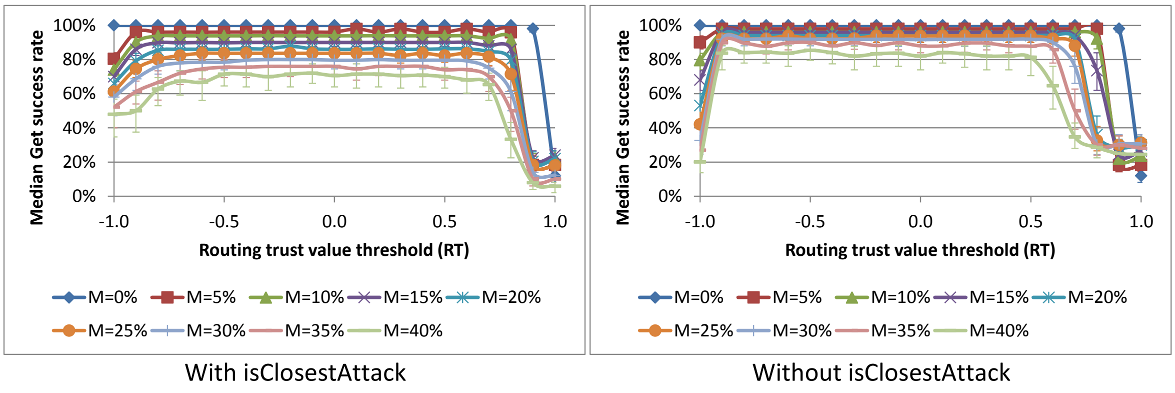
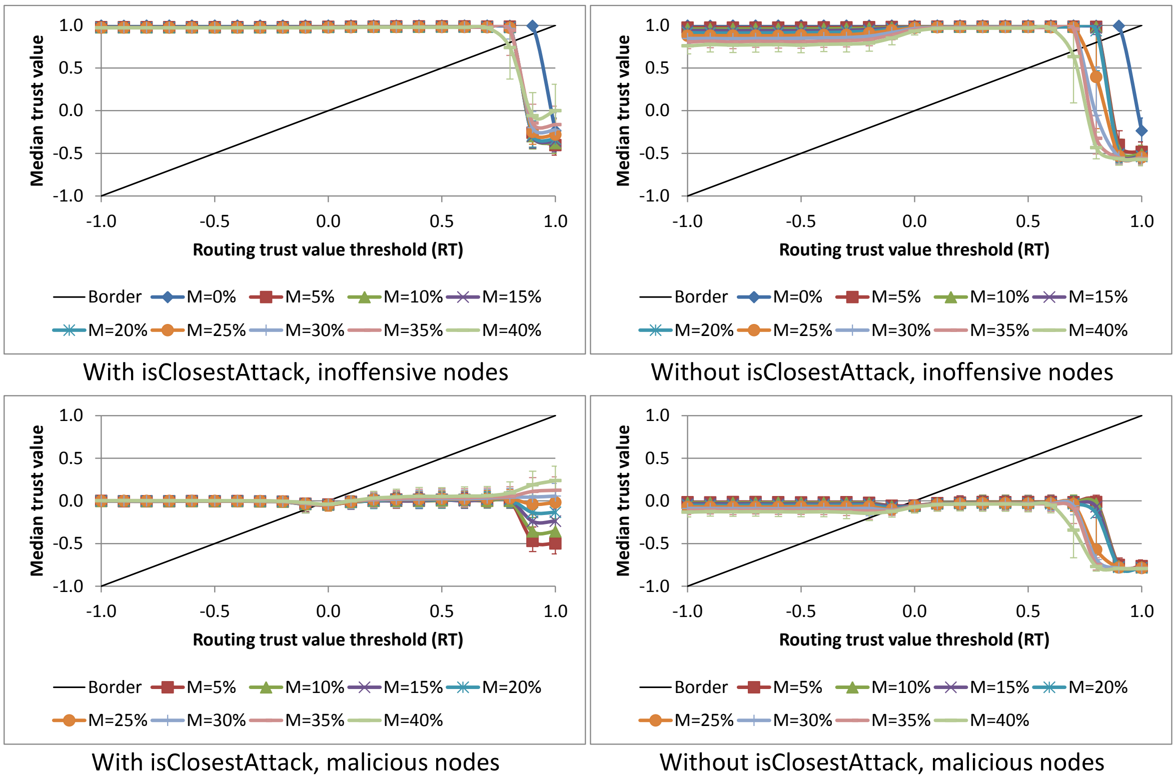


Figure 69: Median Get success rates

6.4.6 Attack Probability 50% With Successfully Faked Cached Trust Information

In all previous configurations presented in Section 6.4, the malicious nodes attack all incoming requests. Now in this configuration, the malicious nodes attack only half of them. The cached trust information is successfully faked as in Sections 6.4.4 and 6.4.5. The network is partitioned at first, but due to the fact that the nodes only attack in 50% of all cases, the partitions become connected when a malicious node responds with existing nodes when it does not attack instead of only with non-existing ones.

Compared to the results shown in Section 6.4.5 in which the malicious nodes always attack, Figure 70 shows that the median trust values of the inoffensive nodes in the configuration with isClosestAttack are about the same up to RT = 0.8. As could be expected, the median trust values of the malicious nodes are greater than before because the nodes only attack half of the requests. This also explains why the median trust value is close to 0: It means that a node has received about as many positives ratings as negative ones,



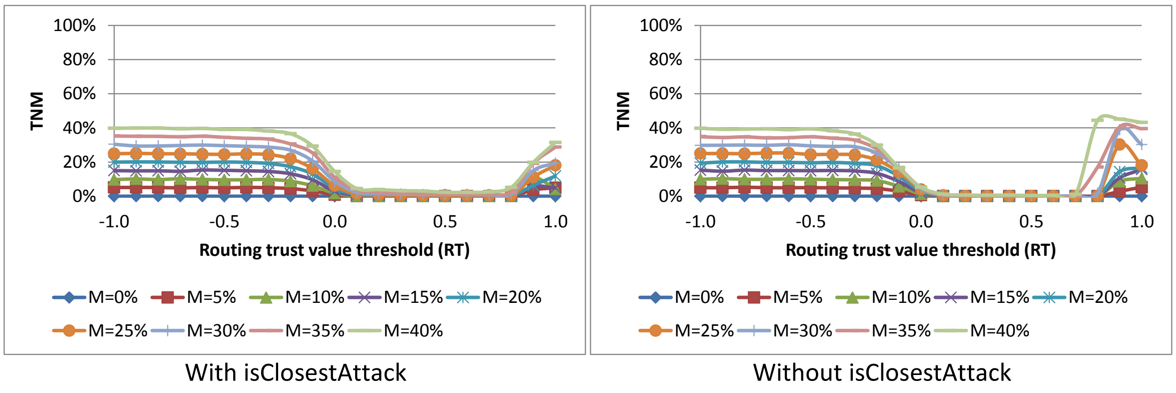
With isClosestAttack, inoffensive nodes

Without isClosestAttack, inoffensive nodes

With isClosestAttack, malicious nodes

Without isClosestAttack, malicious nodes

Figure 70: Median trust values of inoffensive and malicious nodes



With isClosestAttack

Without isClosestAttack

Figure 71: Fractions of trusted nodes that are malicious (TNM)

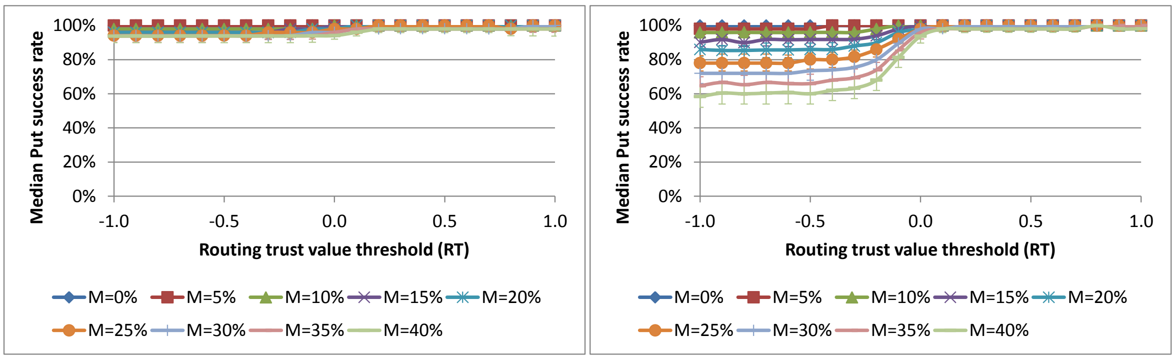
which should be the result if a node only attacks half of the requests.

For RT ≥ 0.9, the median trust values of the inoffensive nodes decrease as with 100% attack probability. The trust values of the malicious nodes spread. Due to the fact that they are only queried if the unchoking process allows it, they can improve their trust value. Therefore, the median trust values increase in the configurations with larger fractions of malicious nodes: In these configurations, fewer nodes are trusted in total, so the unchoking mechanism chooses a malicious node for unchoking more often than in the configurations with fewer malicious nodes.

In the parameter combinations without isClosestAttack, the median trust values of inoffensive and malicious nodes have a greater spread for RT < 0.0: The malicious nodes do not claim to be the closest nodes to the content item and are only queried once during the routing, so it is more likely that they are not contained in the final result list. This way, they are harder to identify as malicious. This changes for RT ≥ 0.0 because then, the median trust values of the malicious nodes are below the threshold and they are queried less frequently. So the inoffensive nodes receive more positive ratings. The median trust values of both kinds of nodes decrease towards RT = 1.0: The number of false negative ratings given to the inoffensive nodes is too high, so the total number of trusted nodes decreases.

Figure 71 confirms this: for RT < 0.0, the number of trusted nodes that are malicious is about equal to M. Beginning at about RT = 0.0, the number of trusted nodes that are malicious is close to zero until the total number of trusted nodes decreases.

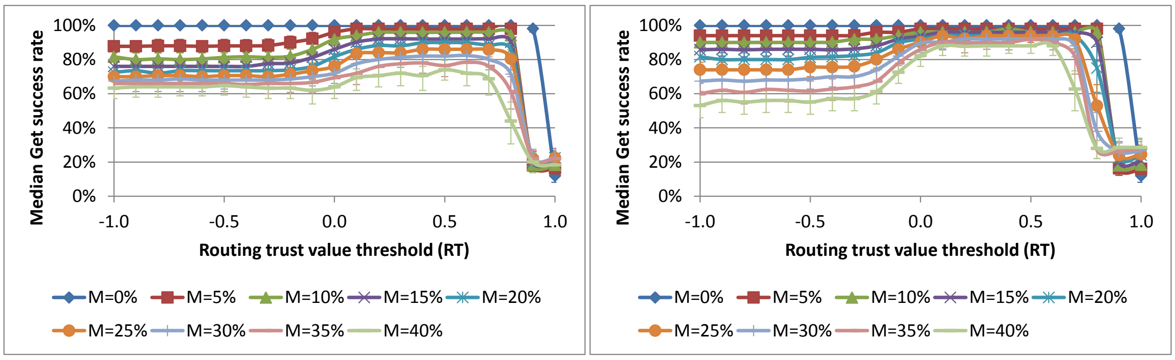
The median Put success rates are shown in Figure 72. For RT ≥ 0.0, they are close to 100% for both isClosestAttack settings because one node is sufficient for the Put operation to



With isClosestAttack

Without isClosestAttack

Figure 72: Median Put success rates



With isClosestAttack

Without isClosestAttack

Figure 73: Median Get success rates

be successful. This can also be a malicious node, which is the reason why the median Put success rate are only slightly lower than 100% with isClosestAttack and lower RT: The malicious nodes always insert themselves into the result list. This way, in most cases, there is at least one node among the final result nodes. This is not the case without the isClosestAttack: Due to the fact that nearly all malicious nodes are trusted for RT < 0.0, the Put success rate is rather low: They are trusted and, hence, their routing attacks succeed.

Figure 73 shows that the Get success rates are greatly improved compared to the configuration with 100% attack probability in a partitioned network as shown in Section 6.4.4. In addition, all parameter combinations can be shown in the figure because the results do not spread as much as before. Also compared to the results of a non-partitioned network in which the nodes always attack as presented in Section 6.4.5, the Get success rates improve slightly. The attacks with 50% attack probability are more successful for thresholds below 0.0; however, choosing such a threshold means that it is acceptable if a node receives more negative than positive ratings, which is not reasonable.

In summary, the malicious nodes reach their goal of achieving a better trust value. The drawback of attacking only in 50% of all cases is that the network is not harmed as much as if the nodes attacked always, so the Put and Get success rates improve.

However, the malicious nodes are still clearly identified, and because of this, the overall success rates improve considerably compared to a partitioned network. The impact of the attacks is less than half as strong as when the malicious nodes attack always. The Put and Get success rates also improve compared to a non-partitioned network, so TrustedKad successfully counters this attack variation.

6.4.7 Malicious Nodes Attack after 2,000 Seconds With Successfully Faked Cached Trust Information

In this configuration, the nodes do not attack during the first half of the simulation time – they act as traitor nodes as described in Section 2.5.4.2. The goal of this attack variation is to receive positive ratings at the beginning and to exploit the high trust values afterwards to conduct attacks. Due to the fact that the nodes do not attack at the beginning, the network is not partitioned. It should be kept in mind that the results show the situation after 4,000 seconds of simulation time, so the success rates contain the results from the first 2,000 seconds in which no attacks occur and the results of the last 2,000 seconds in which attacks are conducted. The trust values and fractions of malicious trusted nodes are taken at the end of the simulation time.

The reason for giving the combined results of both phases is to be able to compare the results in this section to, e.g., the results of the previous Section 6.4.6, in which the nodes attack from the start, but only 50% of all requests. In addition, the combination of the non-attacking and the attacking phase shows the "net present value"[10] of the attack variation for the malicious nodes and whether it is worthwhile to behave inoffensively for a certain amount of time so that the attacks might be more effective afterwards.

The median trust values of the malicious nodes shown in Figure 74 indicate that this attack variation does not improve the impact of routing attacks. After the total simulation time of 4,000 seconds, the malicious nodes are clearly identified: Beginning at RT = -0.3 with isClosestAttack and RT = -0.4 without it, their median trust value is below the threshold, so they are not trusted for routing. Because of this, they are not able to receive

[10] In finance, the net present value shows the value of a series of incoming and outgoing cash flows.

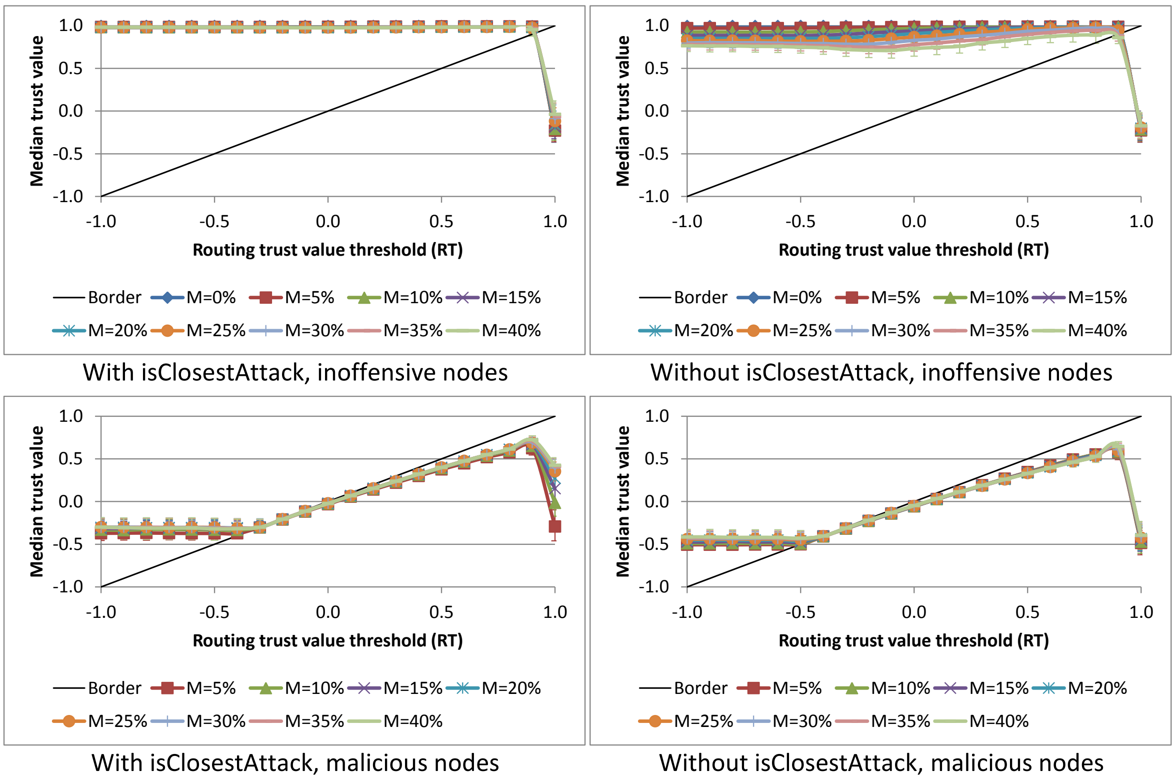


Figure 74: Median trust values of inoffensive and malicious nodes

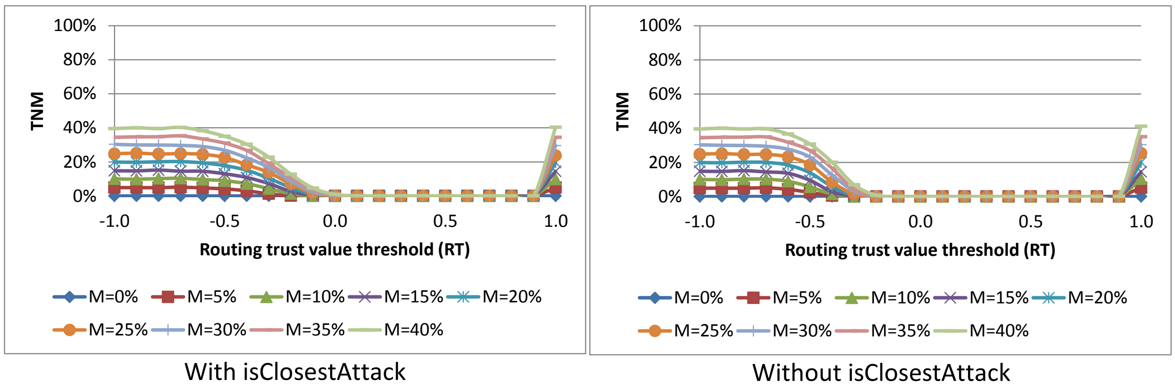
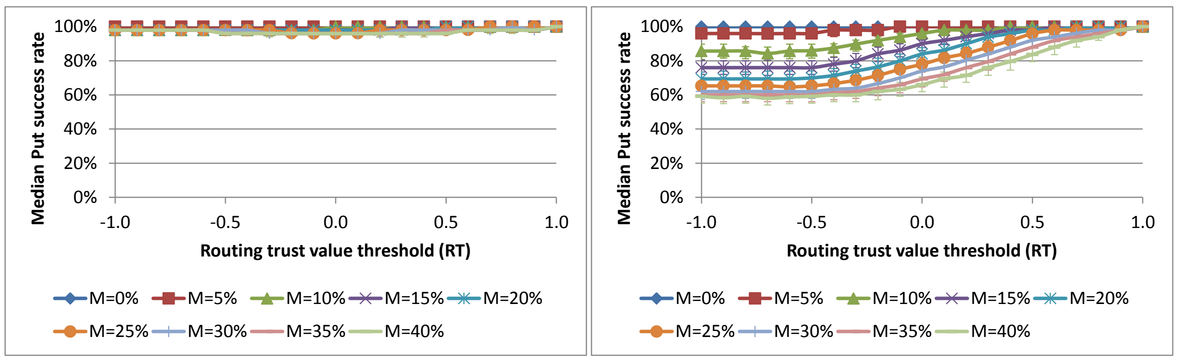


Figure 75: Fractions of trusted nodes that are malicious

any more ratings, so their trust values will stay below the threshold. From time to time, they may receive requests due to unchoking; however, this will further decrease their trust value (cf. 6.4.1).

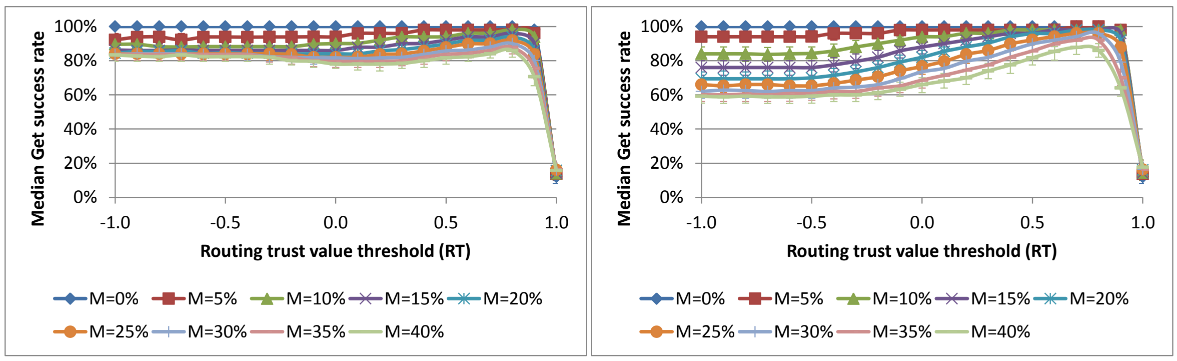
For lower values of RT, the median trust values of the malicious nodes are between -0.3 and -0.4 and -0.4 and -0.5, respectively, which denotes that they have about two to three times as many negative ratings as positive ones. As in the previous configuration with 50% attack probability (cf. 6.4.6), the inoffensive nodes have a median trust value of 1.0 with isClosestAttack and a median trust value that spreads from about 0.7 to 1.0 without isClosestAttack. This value improves the clearer the malicious nodes are identified and the higher the trust value threshold is set.

Figure 75 shows that similarly to the previous configuration, the fraction of malicious trusted nodes is 0 for 0.0 ≤ RT ≤ 0.9 with isClosestAttack; without it, it is 0 even for lower RT because the trust values of the malicious nodes are lower. This shows that TrustedKad



With isClosestAttack Without isClosestAttack

Figure 76: Median Put success rates



With isClosestAttack Without isClosestAttack

Figure 77: Median Get success rates

is able to cope with malicious nodes that attempt to cover themselves by behaving well at first and then start to attack.

The median Put success rates are shown in Figure 76. With isClosestAttack, the Put success rates are always close to 100% because the malicious nodes always insert themselves into the result list if they attack. This way, at least one node on which the content item can be stored is found during most operations. Without isClosestAttack and for low values of RT, the situation is similar to the previous configuration (cf. 6.4.6). However, it can be seen that the attack variation has a greater impact on the Put success: In this configuration, all malicious nodes start acting maliciously at the same time after having earned high trust values. The malicious behavior can be identified immediately and is penalized with negative ratings, but it still takes time for the trust value of a malicious node to fall below the threshold. So, the attack is more effective at first than always attacking half of all requests. But at the end of the simulation, TrustedKad has identified the malicious nodes, so they will be avoided afterwards, leading to even higher success rates for new Put operations.

The same is true for the Get success rates shown in Figure 77. In the configuration with isClosestAttack, they are greater than the success rates with 50% attack probability because of the unimpeded first 2,000 seconds. Without the isClosestAttack, the results are worse: As the malicious nodes do not insert themselves into the result list, they are not queried even if they store the desired content item (remember that malicious nodes only attack routing operations, so they return the correct item if queried for it). Still, also the Get success rates are considerably better than the corresponding baseline configurations (cf. 6.3.1.5).

6.5 How TrustedKad Protects Against Storage Attacks

In the previous section 6.4, TrustedKad's effectiveness to counter attacks against the routing process is evaluated. In this section, attacks against the storage operations are analyzed. Hence, in all configurations that are presented in this section, the malicious nodes do not conduct any routing attacks and the routing trust value threshold (RT) is set to -1.0. The storage trust value threshold (ST) is varied from -1.0 to 1.0 in steps of 0.1. Similar to the previous section, the term "trust value" is used as a short form for "storage trust value" throughout Section 6.5.

As before, the unchoking ratio is determined at first, followed by the simulation results. The exemplary configurations are chosen to show that TrustedKad's approaches of the most trustworthy group and the concealment of the node ID are useful and lead to improved results. In addition, the impact of the SendOriginalHash attack is evaluated as well as attacks by malicious nodes that attack only 50% of all requests or after 2,000 seconds.

6.5.1 Unchoking Analysis

As before for routing attacks in Section 6.4.1, a reasonable unchoking ratio needs to be determined also for storage attacks. Again, unchoking ratios of 0%, 1%, 5% and 10% are analyzed for M = 20% and ST = 0.3. For the analyses, only configurations with non-colluding malicious nodes are used. For 1%, 5% and 10% unchoking ratio, the mean and standard deviation (in parentheses) of the number of unchoking operations conducted by every node is 2.1 (2.6), 10.4 (10.9) and 20.7 (21.4), respectively.

Figure 78 shows the CDFs of the trust values. For the malicious nodes, the results of the different unchoking ratio settings are about the same. For all configurations, 99% of the malicious nodes have a trust value that is lower than or equal to the threshold 0.3, so

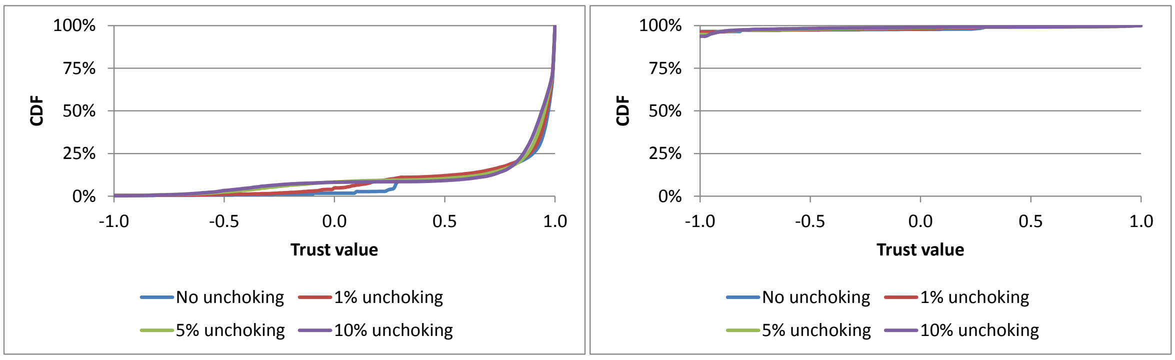


Figure 78: CDF of the trust values of inoffensive and malicious nodes

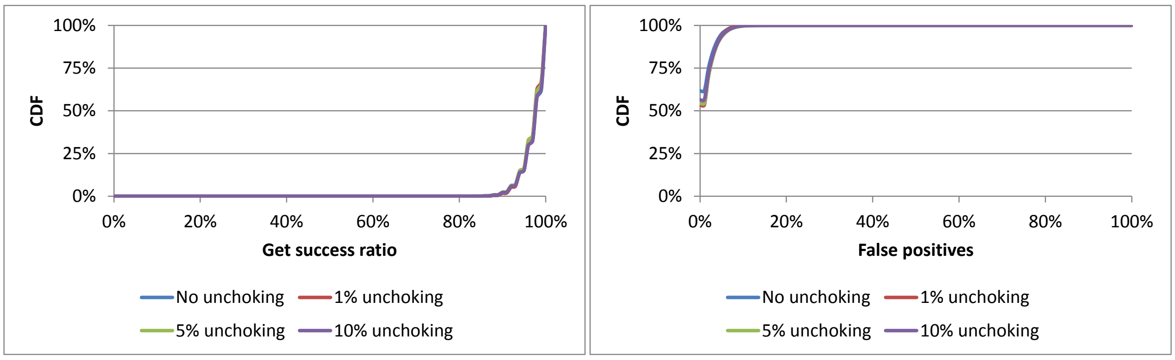


Figure 79: CDF of Get success rates and the false positives rates

they are correctly identified. About the same is true for the inoffensive nodes: 89% to 91% of the inoffensive nodes are correctly identified as trustworthy. The trust value distribution below the threshold is better if 1% unchoking is used because more nodes have a trust value that is close to the threshold: The closer to the threshold the trust value is, the sooner the node can be regarded as trustworthy again because it needs fewer positive ratings to balance its negative ratings.

As shown in Figure 79, there are no significant differences between the unchoking rates for the Get success rates and the false positives rates. The Get success rates are virtually identical. The false positives rates differ only in the amount of nodes that have not received any false positives. So, for further analyses, a storage unchoking ratio of 1% is used.

Table 13 shows the differences between median storage trust values of the configurations without unchoking and with 1% unchoking ratio. A positive value means that the value for 1% unchoking is higher than the corresponding value for no unchoking. The median trust values of the malicious nodes do not differ at all because in both configurations, they are -1.0. The median trust values of the inoffensive nodes differ the more, the higher the threshold is chosen and the more malicious nodes are present. However, the amount of trustworthy nodes is almost equal as shown in Figure 78. In that figure, it is also visible that, e.g., the 5% percentile differs considerably between the configurations. Nevertheless, the amount of trusted nodes is almost equal, and the distribution of the 1% unchoking ratios is more advantageous than the distribution of 5% and 10% unchoking. The difference becomes visible in Table 13 because with increasing ST and M, this behavior shifts from the 5% percentile to higher percentiles such as the median (which is the 50% percentile).

As shown in Table 14, there is almost no difference in the rates of false positives and only

Table 13: Differences between median storage trust values of no unchoking and 1% unchoking

Inoffensive nodes										
		Fraction of malicious nodes								
		0%	5%	10%	15%	20%	25%	30%	35%	40%
Storage trust threshold	0.0	0.00	0.00	0.00	0.00	0.00	0.00	-0.01	-0.01	0.00
	0.1	0.00	0.00	0.00	0.00	0.00	0.00	-0.01	0.00	0.00
	0.2	0.00	0.00	0.00	0.00	0.00	-0.01	-0.01	0.00	0.01
	0.3	0.00	0.00	0.00	0.00	-0.01	-0.01	0.00	0.02	0.03
	0.4	0.00	0.00	0.00	0.00	-0.01	0.01	0.04	-0.04	-0.12
	0.5	0.00	0.00	0.00	-0.01	-0.02	-0.05	-0.13	-0.16	-0.20
	0.6	0.00	0.00	0.01	-0.06	-0.13	-0.18	-0.23	-0.23	-0.26
	0.7	0.00	-0.06	-0.15	-0.20	-0.24	-0.26	-0.27	-0.28	-0.31
	0.8	0.00	-0.15	-0.21	-0.23	-0.25	-0.26	-0.26	-0.26	-0.29

Malicious nodes										
		Fraction of malicious nodes								
		0%	5%	10%	15%	20%	25%	30%	35%	40%
Storage trust threshold	0.0		0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.1		0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.2		0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.3		0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.4		0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.5		0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.6		0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.7		0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.8		0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 14: Differences between median Get success rates and false positives rates of no unchoking and 1% unchoking

Get success rate										
		Fraction of malicious nodes								
		0%	5%	10%	15%	20%	25%	30%	35%	40%
Storage trust threshold	0.0	0%	0%	2%	2%	2%	2%	2%	2%	4%
	0.1	0%	0%	2%	2%	2%	4%	2%	4%	4%
	0.2	0%	0%	2%	2%	2%	2%	2%	2%	6%
	0.3	0%	0%	2%	2%	2%	2%	4%	4%	4%
	0.4	0%	0%	2%	2%	2%	2%	2%	4%	4%
	0.5	0%	2%	2%	4%	2%	4%	3%	4%	3%
	0.6	0%	2%	4%	4%	2%	2%	3%	3%	2%
	0.7	0%	3%	1%	1%	-1%	1%	-1%	-1%	-1%
	0.8	0%	-1%	-2%	-4%	-3%	-5%	-5%	-3%	-4%

False positives										
		Fraction of malicious nodes								
		0%	5%	10%	15%	20%	25%	30%	35%	40%
Storage trust threshold	0.0	0%	0%	0%	0%	0%	0%	0%	0%	0%
	0.1	0%	0%	0%	0%	0%	0%	0%	0%	1%
	0.2	0%	0%	0%	0%	0%	0%	0%	0%	0%
	0.3	0%	0%	0%	0%	0%	0%	0%	0%	0%
	0.4	0%	0%	0%	0%	0%	0%	2%	0%	0%
	0.5	0%	0%	0%	0%	0%	-2%	0%	0%	0%
	0.6	0%	0%	0%	0%	0%	0%	0%	-2%	0%
	0.7	0%	0%	0%	0%	0%	0%	0%	0%	0%
	0.8	0%	0%	0%	0%	0%	0%	0%	0%	0%

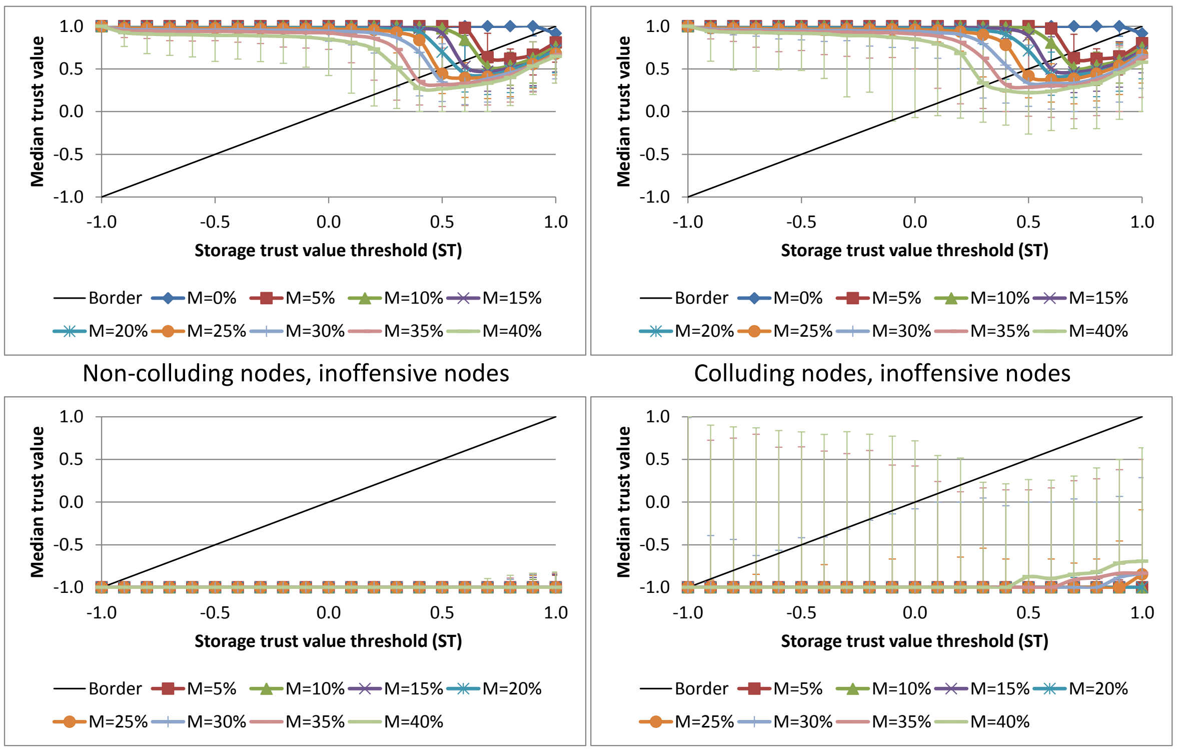
minor differences in the Get success rates. As explained before, unchoking nodes is necessary. The results show, however, that lower unchoking ratios yield better results not only against routing attacks, but also against storage attacks, so the ratio chosen for the remaining tests is 1%.

6.5.2 TrustedKad

In this section, TrustedKad's effectiveness against storage attacks is analyzed. All of the countermeasures presented in Chapter 4 are used: The content item provided by the most trustworthy group is chosen (as explained in Section 4.11.2), and due to the concealment of the content item ID, malicious nodes cannot attack any arbitrary content item they are queried for, but only items that have been previously stored on them (as explained in Section 4.12).

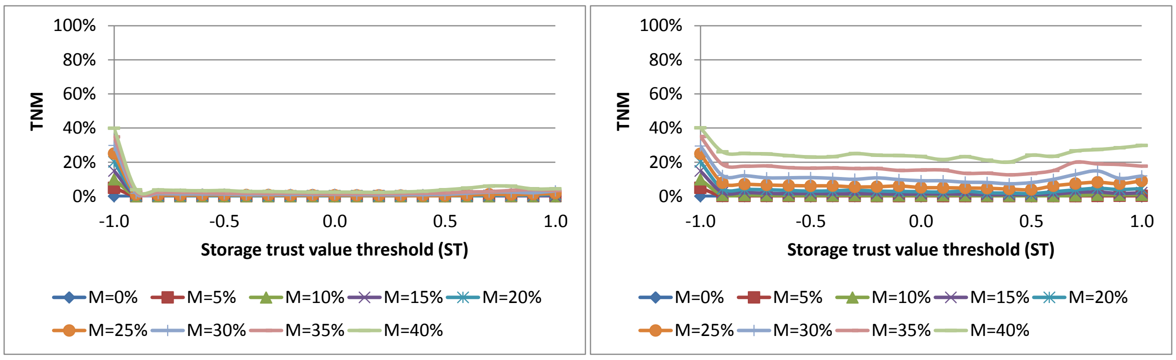
In Figure 80, the median trust values of inoffensive and malicious nodes are shown for colluding and non-colluding nodes. With non-colluding nodes, the inoffensive nodes' median trust values are above the threshold for all fractions of malicious nodes for ST ≤ 0.3; for M = 5%, this is true up to ST = 0.6. It is also visible that even the 25% quartile never falls below 0.0, so most of the inoffensive nodes do not receive more negative ratings than positive ones. The median stays close to the threshold, so if an untrusted inoffensive node that is unchoked and used during a Get process, it has a higher trust value than an unchoked malicious node, so the choice of the content item based on the most trustworthy node group is still correct. The malicious nodes are clearly identified as such: For all ST, their median trust values are -1.0, and only for large M and ST, the 75% quartile is greater than -1.0.

The results are almost the same for the configuration with colluding nodes: Depending on M, the median trust values of the inoffensive nodes fall below the threshold between ST = 0.4 and ST = 0.7. Due to the collusion of the malicious nodes, the spread between the 25% and 75% quartiles increases because some of them are able to manipulate the choice



Non-colluding nodes, inoffensive nodes
Colluding nodes, inoffensive nodes
Non-colluding nodes, malicious nodes
Colluding nodes, malicious nodes

Figure 80: Median trust values of inoffensive and malicious nodes



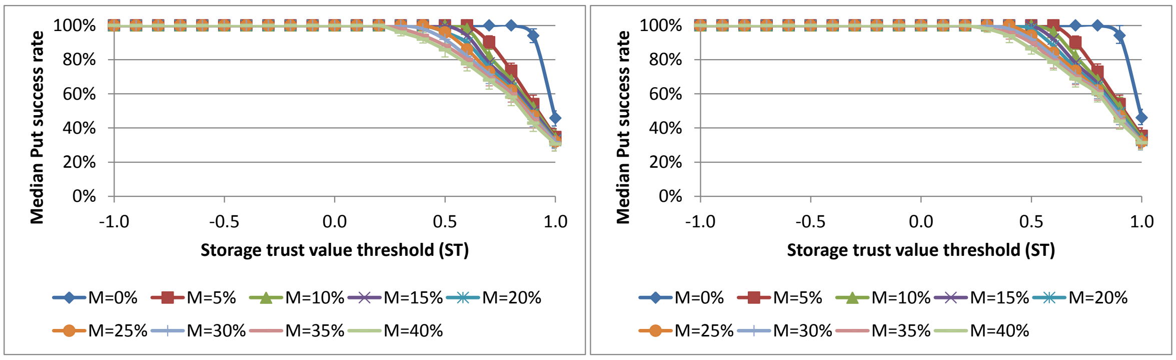
Non-colluding nodes Colluding nodes

Figure 81: Fractions of trusted nodes that are malicious (TNM)

of the content item in their grace phase: In this phase, they are regarded as trustworthy, so malicious and inoffensive nodes both have a trust value above the threshold. If the malicious nodes collude, they deliver only one fake version of the content item, so the chance to win the most trustworthy node group decision is greater than if they delivered several fake content items. However, due to the concealment of the content item ID, this is only possible if the content item has been stored on them before. Almost all of the median trust values of the malicious nodes are still at -1.0, and only for M ≥ 30%, the 75% quartiles reach values above 0.0. So despite their collusion, the malicious nodes are still clearly identified.

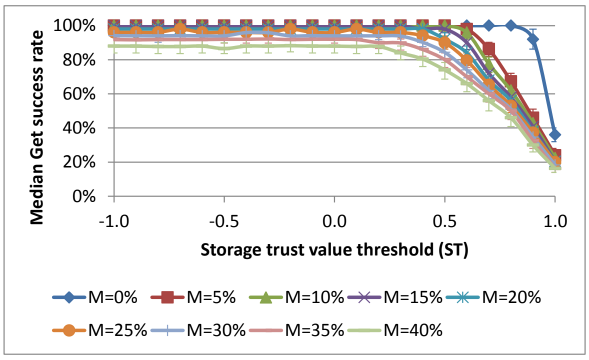
The fraction of trusted nodes that are malicious is very small for non-colluding nodes as shown in Figure 81: For M ≤ 15%, it is always 0% (except for ST = -1.0), and even for M = 40%, the maximum is 6%. The attack is significantly more effective if the malicious nodes collude; nevertheless, TrustedKad is still able to keep the amount of trusted nodes that are malicious considerably lower than M: For M ≤ 30%, the fraction is not greater than 15%, and for M ≤ 20%, it is 5% at most.

The median Put success rates are shown in Figure 82. They do not differ significantly in dependence of whether the nodes collude or not. The rates are greater than 90% for all M and ST ≤ 0.4. For larger ST, they decrease. This happens because the total number of trusted nodes decreases quickly for ST ≥ 0.6: The fewer nodes are trusted in total, the greater is the probability that there is no node in the final Lookup result list that is trusted for storage. If this is the case, the Put operation fails. For example, for ST = 0.8 and M ≥ 5%, only 6% to 10% of all nodes are trustworthy for storage. Still, the median Put success rate is at 58% to 73% (values for non-colluding malicious nodes), so in these cases, at least one of the final nodes is trustworthy. The success rates are that large despite

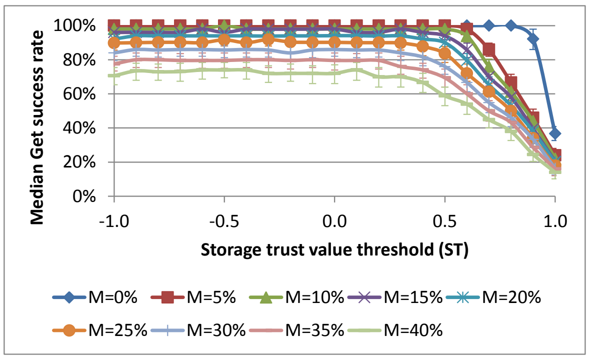


Non-colluding nodes Colluding nodes

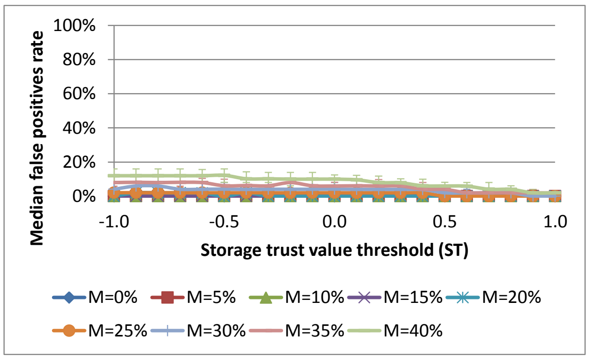
Figure 82: Median Put success rates



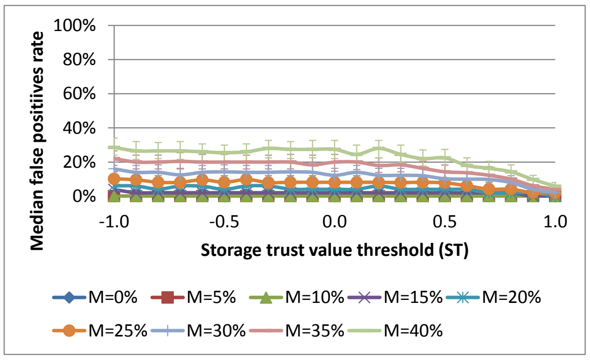
Non-colluding nodes, Get success rate



Colluding nodes, Get success rate



Non-colluding nodes, false positives rate



Colluding nodes, false positives rate

Figure 83: Median Get success and false positives rates

the low numbers of trusted nodes because the final node list contains up to 16 nodes to cope with situations in which many nodes are trusted for routing but not for storage (as explained in Section 5.5.2.1). The results also match the theoretical results of the hypergeometric probability distribution formula shown in Section 6.3.2.2.

Figure 83 shows the median Get success and the false positives rates. A Get operation is regarded as successful if the node is able to retrieve the content item that has been stored originally. A false positive denotes that a node believes to have found the correct content item, but actually has chosen a wrong version of it, which should be avoided in any case.

For colluding and non-colluding nodes, the Get success rates are better than the baseline results for ST ≤ 0.3 (cf. 6.3.2.1). For larger ST, the Get success rates start to decrease. The reason for the failing Get success operations is that a node has not found any trustworthy node to retrieve the content item from, so it cancels the operation: TrustedKad enables the node to decide to not choose any item from any node because it does not trust the result to be correct. Consequently, TrustedKad leads to a decrease of the false positives rates towards ST = 1.0, as the nodes rather choose to cancel the operation instead of choosing a fake content item. So the Get success rates decrease only because the nodes decide to cancel the operation and not because they accept wrong data.

6.5.3 Majority Decision, No Content ID Concealment

The effectiveness of TrustedKad's functions "most trustworthy node group" and "concealment of the content item ID" is proven in this section. Hence, the nodes perform a majority decision instead of the most trustworthy node group approach. Additionally, the malicious nodes attack every Get request directed at them instead of only the ones concerning content items that have been stored on them previously.

Compared to TrustedKad with all features activated, the median trust values of the inoffensive nodes shown in Figure 84 decrease earlier and stronger: They fall below the threshold already for lower ST than before. In addition, the medians are lower and the quartiles spread more, so there are more inoffensive nodes with lower trust values. The effect is greater if the malicious nodes do not collude: If they collude, they are able to gain high trust values during their grace phases as shown in Section 6.5.2, so TrustedKad does not identify them as clearly as if they do not collude. However, the majority decision and the missing concealment of the content item ID still decrease the median trust values of the inoffensive nodes.

The comparison of the median trust values of the malicious nodes yields even greater differences. For non-colluding nodes, the median trust values are always -1.0 for TrustedKad. Now, in this configuration, they rise up to -0.17, and the 75% quartiles even reach values above 0.0. The difference is even larger for colluding malicious nodes: Before, the median trust values are rarely greater than -1.0. But without the two TrustedKad func-

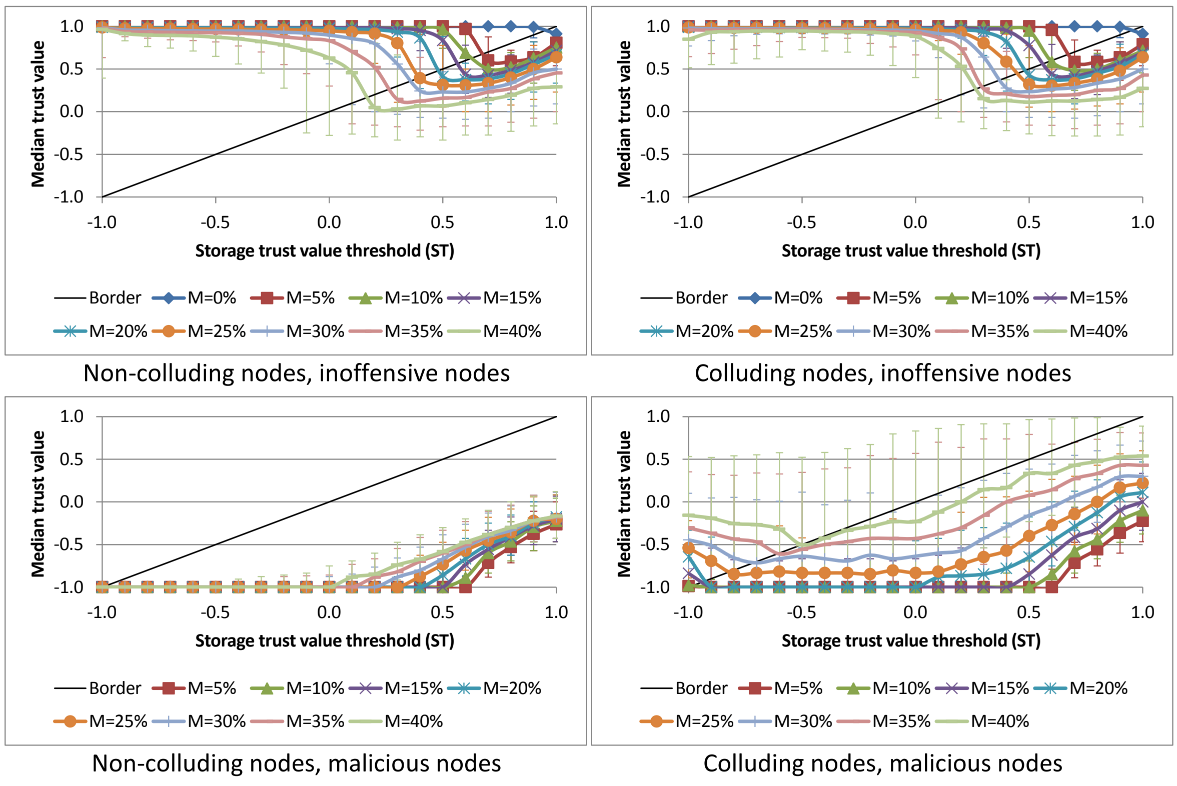


Figure 84: Median trust values of inoffensive and malicious nodes

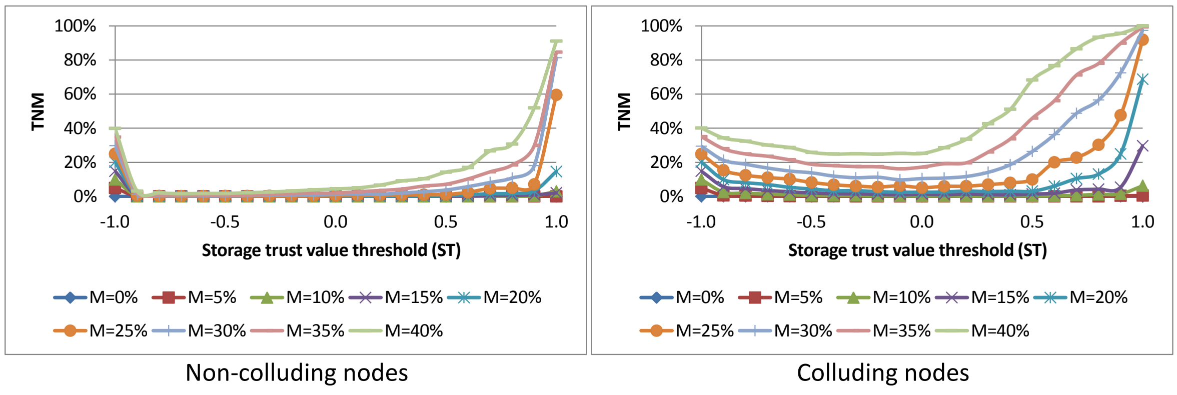
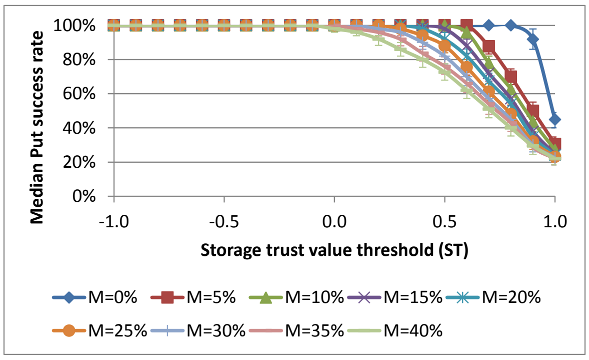


Figure 85: Fractions of trusted nodes that are malicious (TNM)

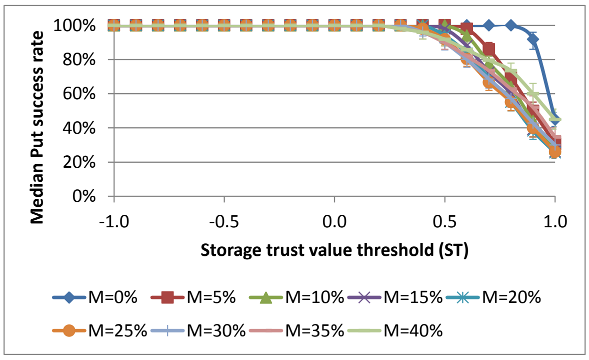
tions, they rise up to 0.5 for large fractions of malicious nodes; even for only 5% of malicious nodes, they rise up to -0.22.

The colluding malicious nodes can achieve such high trust values because they win the majority decision in many cases and receive positive ratings due to that. In addition, the inoffensive nodes whose correct version is not chosen receive a negative rating. And because the malicious nodes attack all incoming Get requests, they can also win the decision if they do not collude with a higher probability because if only one of the nodes that are queried during a Get operation is inoffensive, the choice of the majority decision is random: There are as many different versions as there are responses. So the malicious nodes achieve larger trust values also if they do not collude.

In Figure 85, the trend continues: The number of trusted nodes that is malicious increases significantly compared to the full TrustedKad. For non-colluding nodes, this fraction increases for M ≥ 20% and towards ST = 1.0, so more and more malicious nodes can disrupt the storage operations. For colluding nodes, this is also true for M = 15%. For M ≥ 15%,

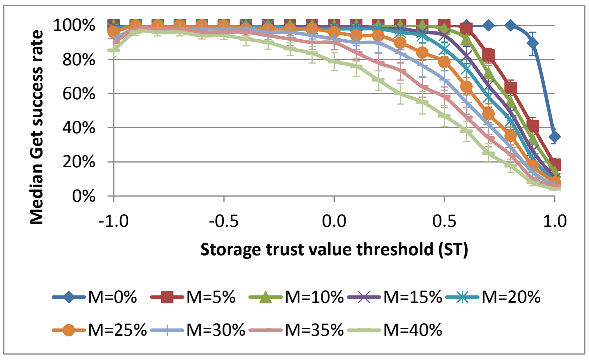


Non-colluding nodes

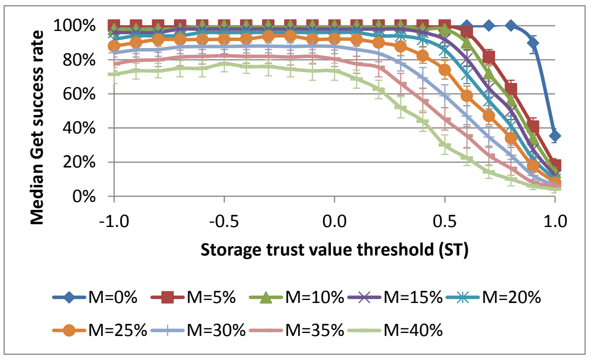


Colluding nodes

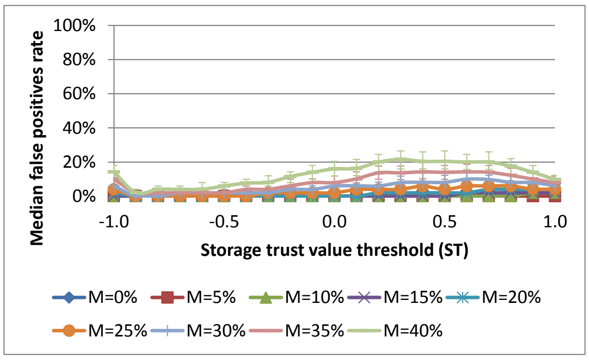
Figure 86: Median Put success rates



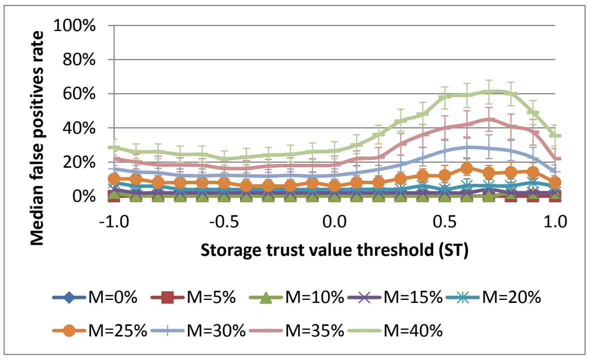
Non-colluding nodes, Get success rate



Colluding nodes, Get success rate



Non-colluding nodes, false positives rate



Colluding nodes, false positives rate

Figure 87: Median Get success and false positives rates

the fraction of trusted nodes that is malicious is even higher than the fraction of the total number of nodes that is malicious. This shows that the two features are required for TrustedKad to be able to protect the network from malicious nodes.

Also the Put success rates shown in Figure 86 are worse than with full TrustedKad with non-colluding malicious nodes. The rates with colluding nodes, however, are only slightly worse because enough “trusted” nodes are found to store the content items on.

As with the Put success rates, also the Get success rates shown in Figure 87 are worse due to the reduced trust values of the inoffensive nodes and the improved ones of the malicious nodes. For both colluding and non-colluding nodes, the decrease of the success rates starts earlier than with full TrustedKad; the higher M, the earlier it starts. Also the minimum is lower than before. In addition, the false positives rates do not decrease steadily as before, but instead increase before decreasing slightly again. For colluding nodes, the maximum fraction of false positives is even considerably greater than M for M ≥ 30%.

Summarizing, the results of this section clearly show that the two features “most trustworthy node group” and “concealment of the content item ID” significantly improve TrustedKad’s ability to protect the network. With a majority decision for the choice of the content item version and with the malicious nodes attacking any arbitrary Get request, the results worsen substantially.

6.5.4 Resilience Against SendOriginalHash Attack

For the remaining configurations, all of TrustedKad’s features are active again. The attack parameter “SendOriginalHash” as explained in Section 5.5.8 is enabled in this configuration. This way, the malicious nodes respond with the hash value of the original content item during the GetHash phase and only send fake content in the GetValue phase. A distinction between colluding and non-colluding nodes is not required: All nodes (inoffensive and malicious ones) send the same hash, so there is only one version of the content item identified in the GetHash phase (instead of at least two as in the previous configurations). In addition, there are no false positives: The fake content is always discarded, as it does not match the hash of the original content item.

The attack can be used by a malicious node to boost its trust value: As only one node is asked for the value of the content item, all nodes that have responded with the same hash receive a positive rating. So if an inoffensive node is asked for the value, also all malicious nodes that are in the most trustworthy node group receive a positive rating.

The attack reaches its goal as shown in Figure 88: The median trust values of the malicious nodes are very high. In the corresponding baseline configuration presented in Sec-

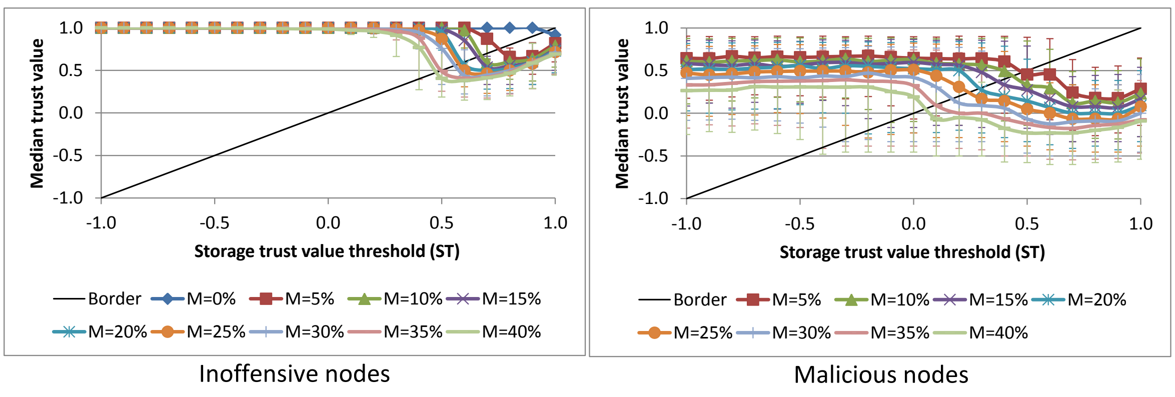


Figure 88: Median trust values of inoffensive and malicious nodes

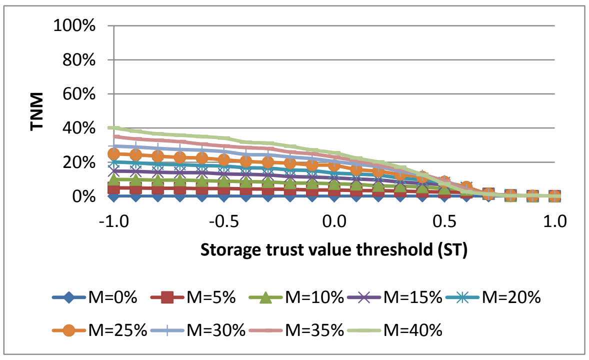
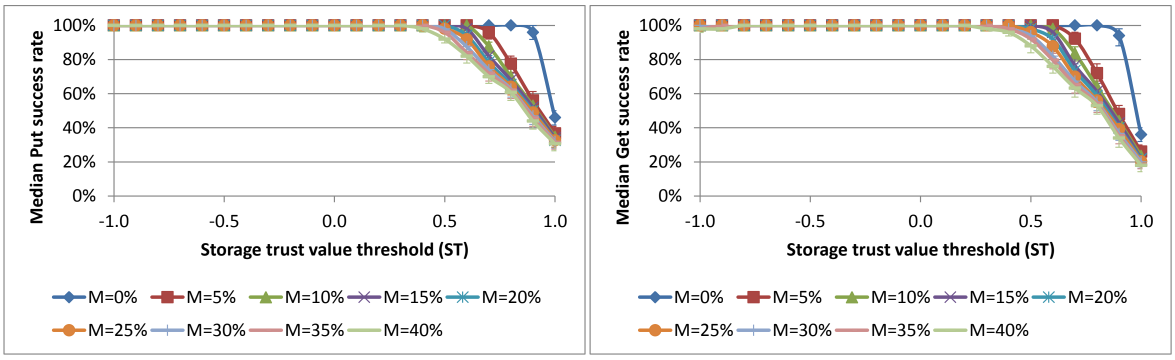


Figure 89: Fractions of trusted nodes that are malicious (TNM)



Put success rates Get success rates

Figure 90: Median Put and Get success rates

tion 6.3.2.2, the number of replicas rises compared to attacks using fake hashes because the nodes are not identified as malicious. For the same reason, in this configuration, the trust values of the malicious nodes increase.

However, the median trust values of the inoffensive nodes improve as well: Now that there is only one version of the content item, there is also only one node group that is queried in the GetValue phase, so the inoffensive nodes cannot accidentally be in one of the groups that are not chosen after the GetHash phase. However, the inoffensive nodes can still receive negative ratings if they do not know the queried item but other nodes do.

The fractions of trusted nodes that are malicious decreases with increasing trust value threshold as shown in Figure 89. With increasing threshold, fewer negative ratings suffice for a malicious node to fall below the threshold, so the fraction of trusted nodes that are malicious decreases. In addition, the total number of trusted nodes decreases as well, so the graphs in Figure 89 decrease more quickly beginning at $ST \approx 0.5$.

Figure 90 shows that the attack effect is not improved by using SendOriginalHash to gain high trust values and then attacking by sending the hash of the fake content: The median Put and Get success rates are higher than in configurations without the SendOriginalHash parameter enabled. In addition, there are no false positives as the fake value can be identified because it does not match the hash. The operations can still fail because the total number of trusted nodes decreases towards $ST = 1.0$.

Summarizing, TrustedKad is able to defend against malicious nodes that use the Send-OriginalHash attack in spite of the fact that the attack is partly successful by enabling the malicious nodes to achieve higher trust values.

6.5.5 Resilience Against Malicious Nodes With an Attack Probability of 50%

The previous sections show that TrustedKad is able to counter attacks of nodes that attack always (cf. 6.5.2, 6.5.3, 6.5.4). In this configuration, the results of a scenario in which the malicious nodes only attack 50% of the requests they receive are presented. With this attack variation, the malicious nodes also aim to increase their storage trust values so that their attacks are more successful. Due to the fact that the malicious nodes attack only half of all requests, the results of colluding and non-colluding malicious nodes are very similar, so they are not differentiated in the following.

Figure 91 shows that the malicious nodes reach their first goal: Their median trust values increase compared to the configuration in which they always attack. Nevertheless, they are still clearly identified, and their median trust values stay below 0.0, so they still receive more negative ratings than positive ones. In addition, the trust values of the inof-

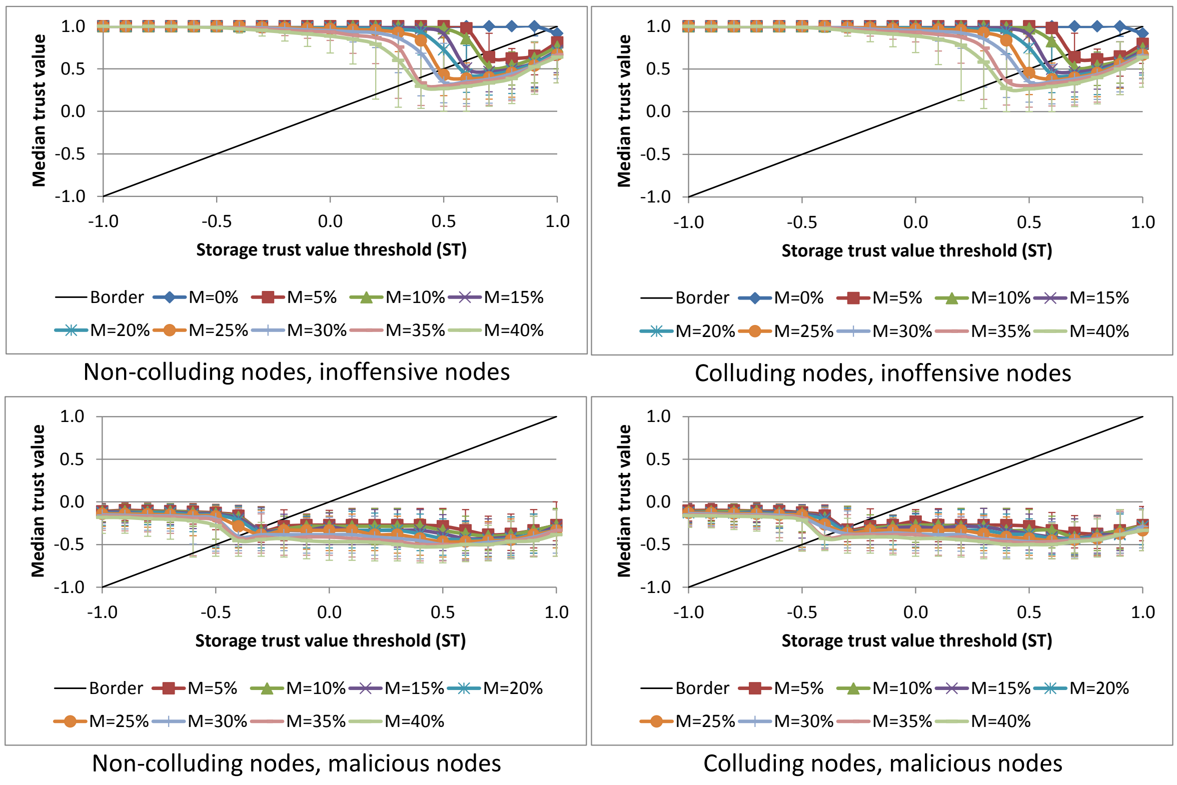


Figure 91: Median trust values of inoffensive and malicious nodes

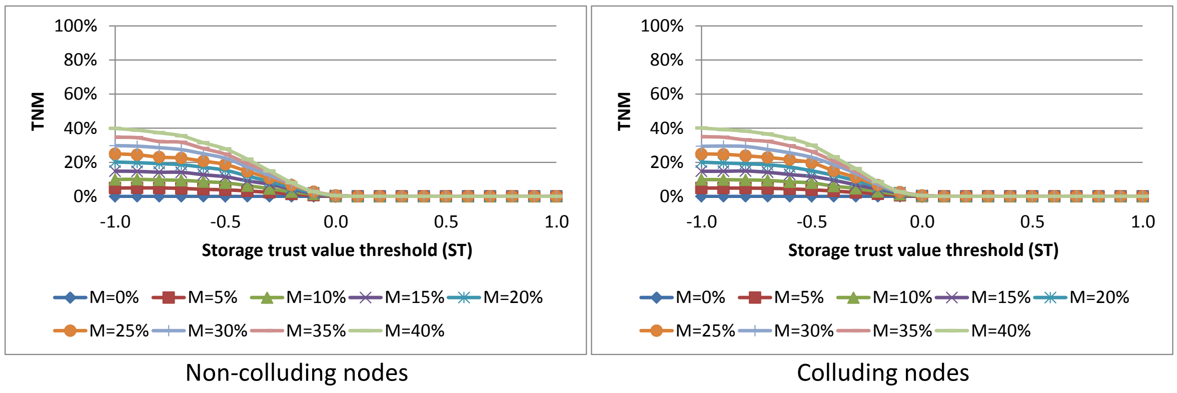


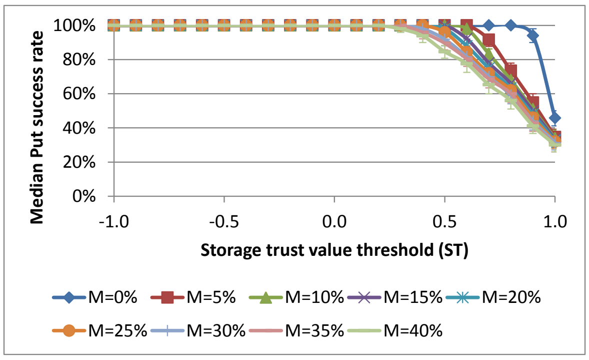
Figure 92: Fractions of trusted nodes that are malicious (TNM)

fensive nodes improve.

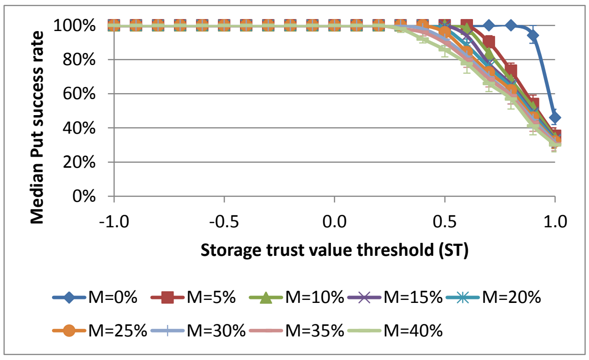
The median trust values of the malicious nodes fall below the trust value threshold already for low values of ST. Hence, the fractions of trusted nodes that are malicious decrease early and are 0% in all cases for ST ≥ 0.1 as shown in Figure 92. For lower ST, the fractions are larger than before in some cases; however, choosing a threshold below 0.0 means to trust nodes that have more negative than positive ratings, which is not reasonable.

The Put success rates shown in Figure 93 are almost the same as the ones for the configuration in which all requests are attacked. So the attack variation does not have any impact on the success of the Put operations. For the Put success, the total number of available trusted nodes is important, and these numbers do not differ significantly for the thresholds whose success rates are below 100%.

Compared to the "always attack" configurations, the Get success rates improve for low and medium trust value thresholds. Also the colluding nodes lose nearly all of their attack

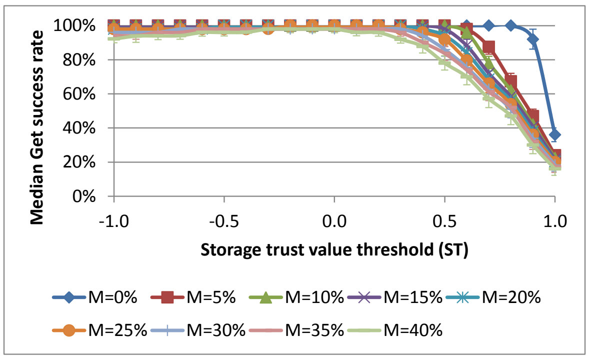


Non-colluding nodes

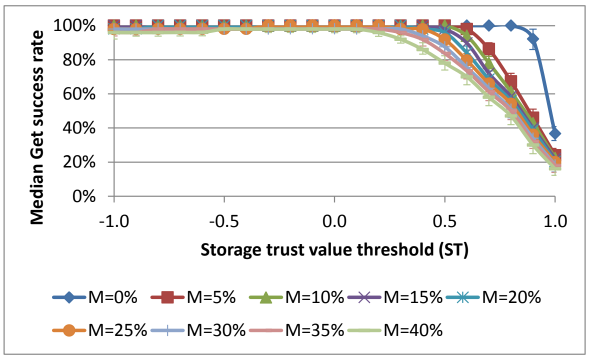


Colluding nodes

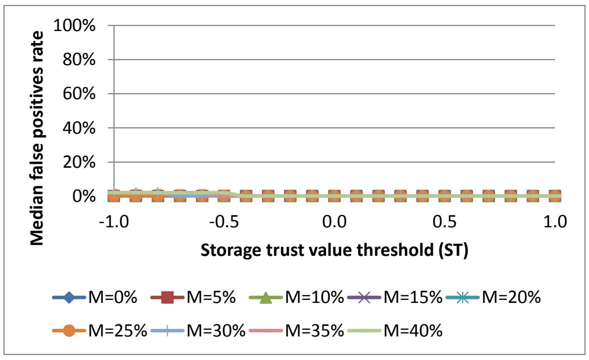
Figure 93: Median Put success rates



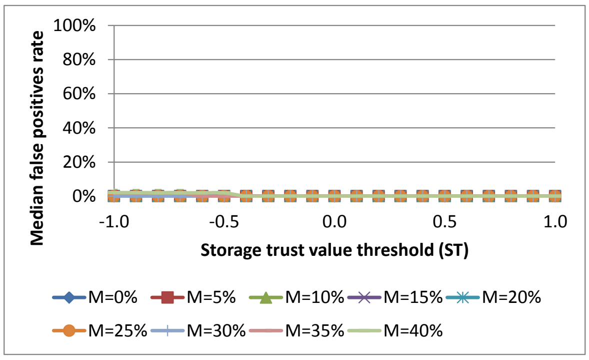
Non-colluding nodes, Get success rate



Colluding nodes, Get success rate



Non-colluding nodes, false positives rate



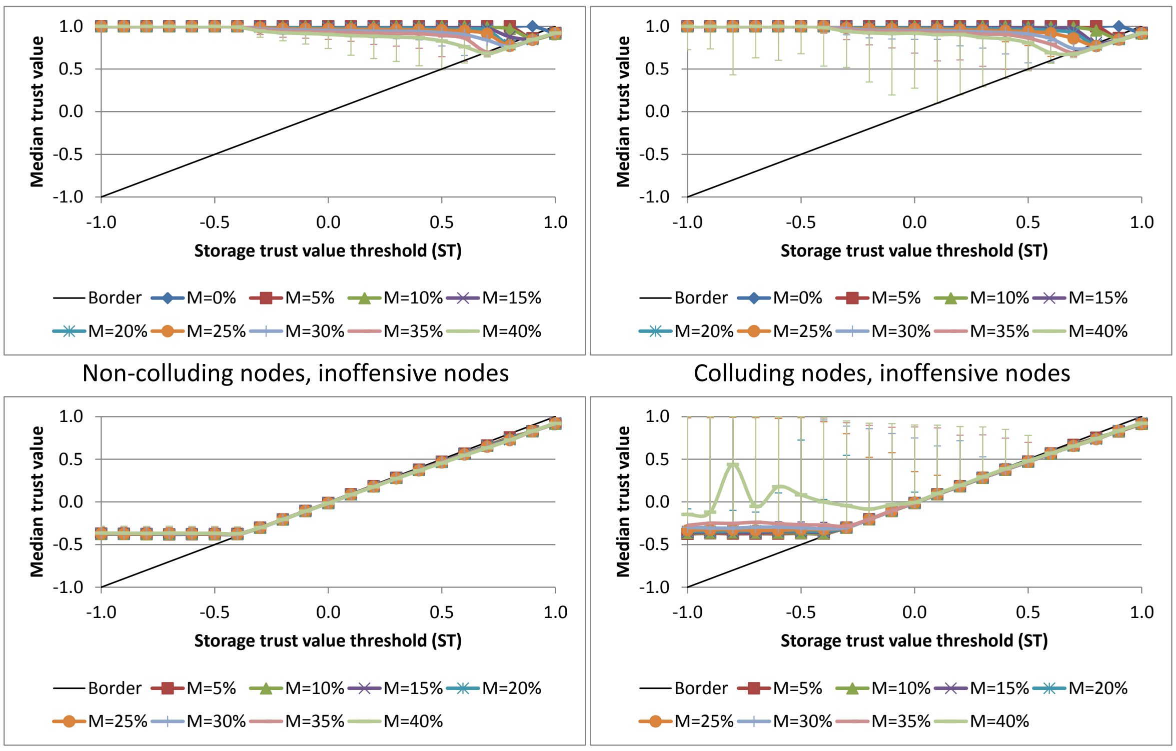
Colluding nodes, false positives rate

Figure 94: Median Get success and false positives rates

impact because their attacks are not coordinated. The increase of the Get success rates is expected because the malicious nodes behave correctly in 50% of all requests. In Figure 94, it is also visible that the malicious nodes do not reach their actual goal of using their increased trust values to harm the network: The false positives rates are always 0% for ST ≥ -0.4. For lower ST, the fractions of trusted nodes that are malicious are large enough to harm the network slightly, but these thresholds are too low to be reasonable.

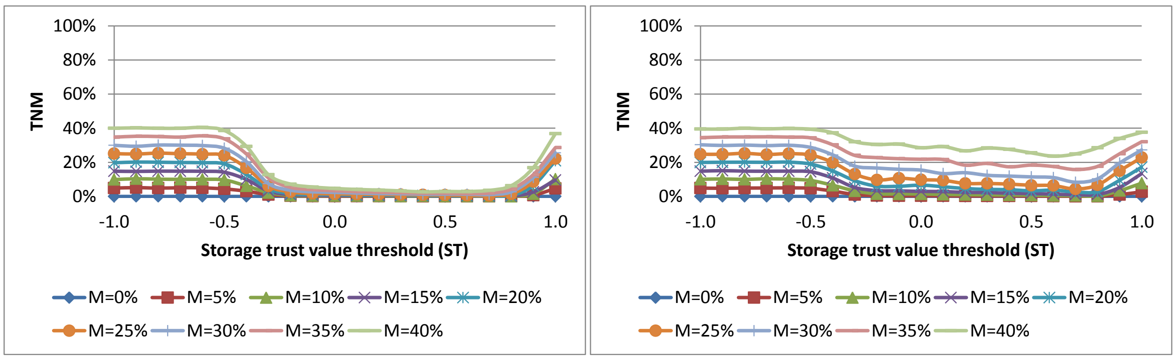
6.5.6 Resilience Against Malicious Nodes That Attack After 2,000 Seconds

The goal of the malicious nodes remains the same in the configuration presented in this section: Gain high trust values in order to harm the network. The difference is that the malicious nodes behave correctly now for the first 2,000 seconds of the simulation time and then attack every request they receive. A worst-case scenario is presented here because all malicious nodes turn malicious at the same time, which requires coordination of



Non-colluding nodes, inoffensive nodes
Colluding nodes, inoffensive nodes
Non-colluding nodes, malicious nodes
Colluding nodes, malicious nodes

Figure 95: Median trust values of inoffensive and malicious nodes

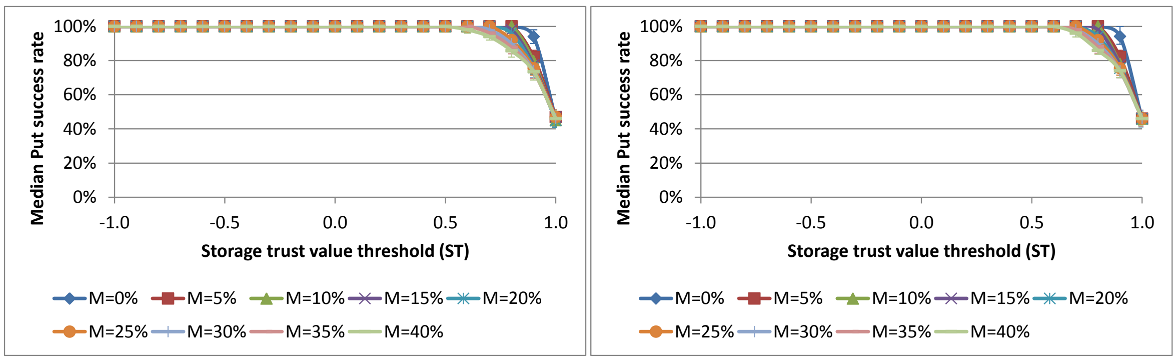


Non-colluding nodes
Colluding nodes

Figure 96: Fractions of trusted nodes that are malicious (TNM)

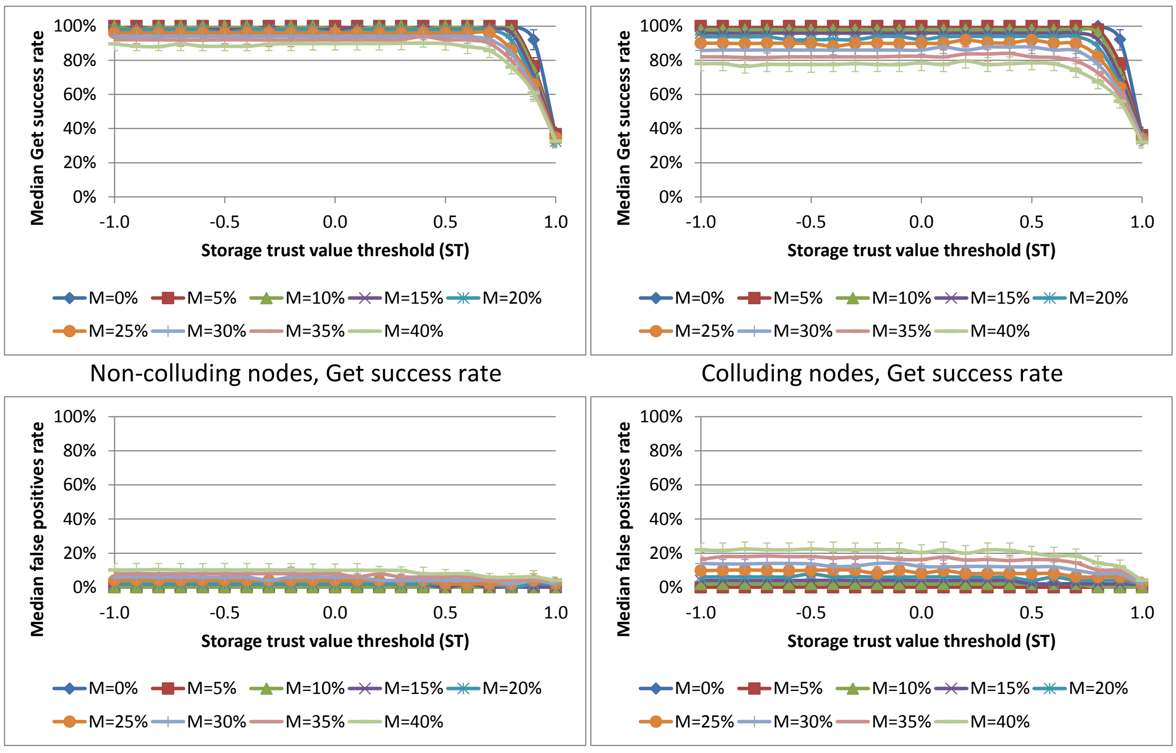
the malicious nodes. As in the corresponding routing attack section (cf. 6.4.7), the malicious nodes deliberately allow the operations to succeed during the first 2,000 seconds. The results shown here are taken at the end of the simulation after 4,000 seconds; because of that, the success rates are a mean between the successful first 2,000 seconds and the impaired operations afterwards. So as in Section 6.4.7, the results show the net present value of the attack variation.

Figure 95 shows the median trust values of the nodes. Again, the median trust values of the inoffensive nodes improve compared to Section 6.5.2. Also the median trust values of the malicious nodes increase. However, for ST ≥ -0.3, the median trust values are lower than the threshold (except for colluding nodes and M = 40%, where this is only true for ST ≥ 0.0). For non-colluding nodes, this is also true for the 75% quartiles; as expected, the colluding nodes are more successful, so the 75% quartiles are partly above the threshold for large M, but also only for ST ≤ 0.6. This clearly shows that TrustedKad is able to identify the changing behavior of the malicious nodes and to rate them negatively. It should be



Non-colluding nodes Colluding nodes

Figure 97: Median Put success rates



Non-colluding nodes, Get success rate Colluding nodes, Get success rate

Non-colluding nodes, false positives rate Colluding nodes, false positives rate

Figure 98: Median Get success and false positives rates

kept in mind that a node may rate another node only once, so this one rating can change from positive to negative. This has a greater impact on the trust value than an additional negative rating, because the total number of ratings does not change. Furthermore, the nodes with trust values below the threshold are not able to receive new ratings because they are no longer used for storage, so their trust values will stay below the threshold (as described for the routing trust values of the malicious nodes in Section 6.4.7).

Figure 96 shows the fractions of trusted nodes that are malicious. For non-colluding malicious nodes, the results are about the same for 0.0 ≤ ST ≤ 0.8. For other thresholds, the values are higher now, so the attack variation succeeds to have more malicious nodes that are regarded as trustworthy for low and very high thresholds. If the malicious nodes collude, this is even true for all thresholds. However, this does not necessarily mean that the attack impact is greater because TrustedKad uses the most trustworthy node group to determine the content item version that is chosen.

Compared to the full TrustedKad configuration (cf. 6.5.2), the Put success rates increase as shown in Figure 97. So despite the attack variation, the nodes are more successful in finding nodes on which the content item can be stored.

The Get success rates are shown in Figure 98. Compared to Section 6.5.2, they are at a higher level and stay there for greater trust thresholds before they decrease. This is as expected because the malicious nodes do not attack during the first 2,000 seconds. The amount of false positives decreases slightly. So, as before with the 50% attack probability, TrustedKad is able to identify the malicious nodes and avert their attack. Because of the clear identification of the malicious nodes, they cannot perform their attack any longer, so for longer simulation times, the Get success and false positives rates would increase even more.

6.6 How TrustedKad Protects Against Combined Routing and Storage Attacks

This last section of the evaluation chapter presents the results of the simulations in which the malicious nodes perform both routing and storage attacks. For the analyses, the routing trust value threshold (RT) and the storage trust value threshold (ST) are varied from -1.0 to 1.0 in steps of 0.1, which considerably increases the number of parameter combinations that are tested. So, as an example, the following settings have been used to create a worst-case scenario which would be used by an attacker who attempts to cause the most damage:

- The network is partitioned,
- the isClosestAttack is enabled,
- the malicious nodes attack all incoming requests all the time,
- the malicious nodes can successfully fake the cached trust information,
- the malicious nodes collude and
- the malicious nodes send the fake hash that matches the fake content.

During the GetHash phase, malicious nodes can only attack content items that have been

-1.0	-0.9	-0.8	-0.7	-0.6	-0.5	-0.4	-0.3	-0.2	-0.1	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0

Trust values

0%	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%	100%

Success rates and fractions

Figure 99: Legend of color coding

previously stored on them because of the concealment of the content item ID.

This scenario is evaluated with 5% and 20% malicious nodes. The tendency of the results is the same, so for reasons of clarity, only the figures for the configuration with 5% malicious nodes are shown here. The figures for 20% malicious nodes can be found in Annex A.2.1.

It should be kept in mind that nodes that are not trustworthy for routing cannot be among the final Lookup result nodes, so their storage trustworthiness does not matter, as it is not even evaluated.

The former configurations vary the fraction of malicious nodes and the trust value threshold of either the routing or the storage trust value; now, both trust value thresholds are varied. Hence, another form of presentation is chosen: In the following, the results are shown in tables that contain 21×21 cells for the possible combinations of RT and ST. The cells are color-coded as shown in Figure 99: The maximum value (1.0 for trust values, 100% for success rates and fractions) is green, the medium value (0.0 or 50%) is yellow and the minimum value (-1.0 or 0%) is red. For orientation, the thresholds 0.0 are indicated in the figures. Only the figures for the medians are included here; the figures for the 25% and 75% quartiles can be found in Annex A.2.2.

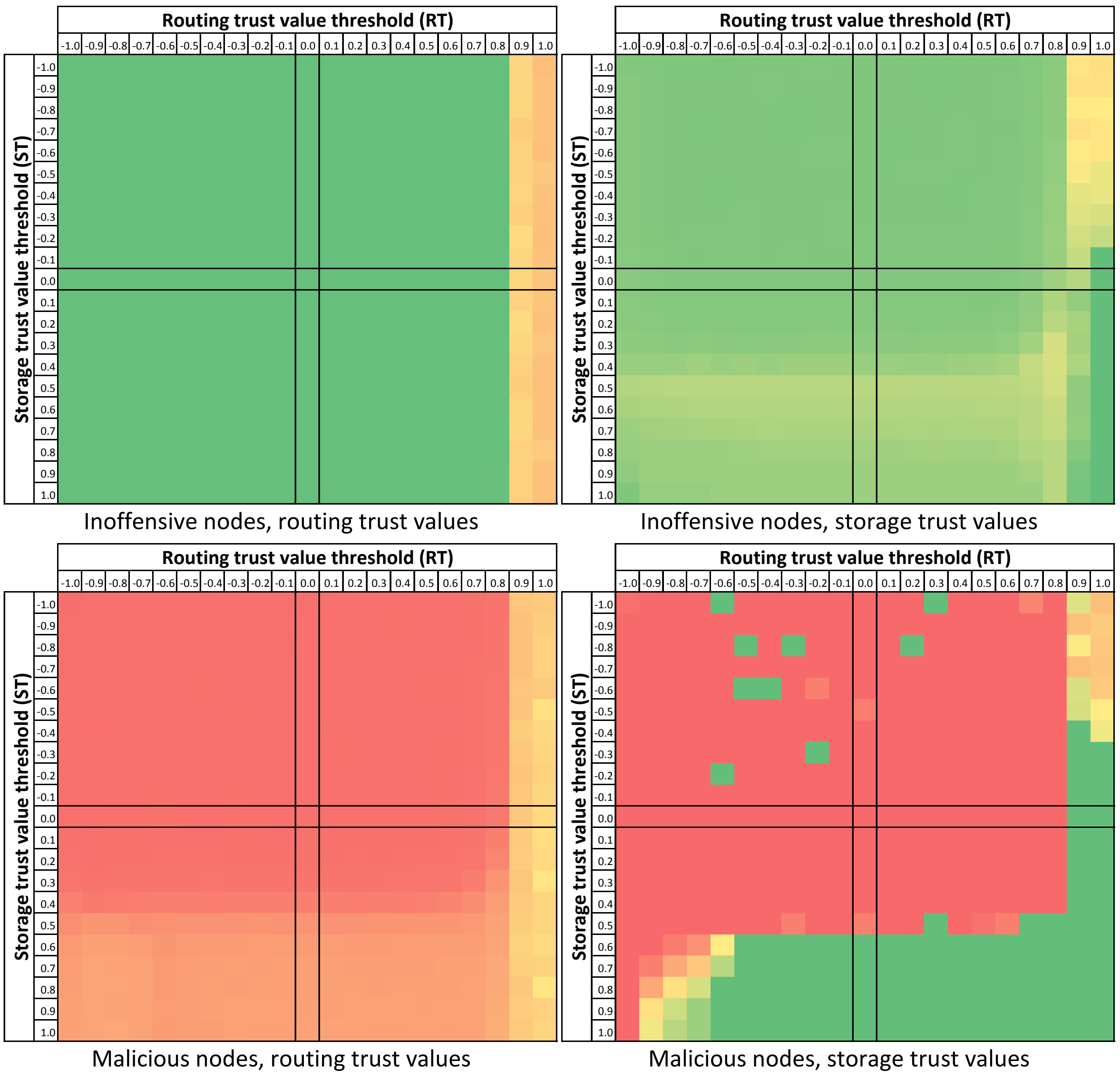


Figure 100: Median routing and storage trust values of inoffensive and malicious nodes (M = 5%)

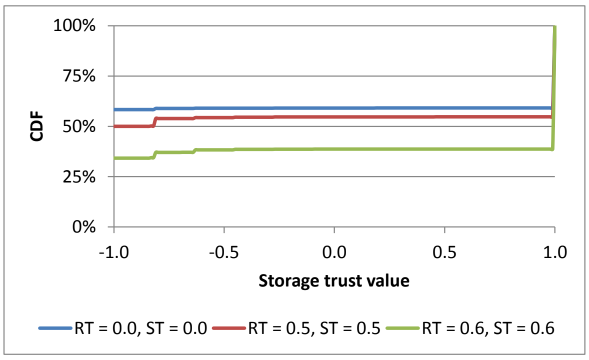


Figure 101: CDF of storage trust values of malicious nodes (M = 5%)

Figure 100 shows the median routing and storage trust values of the inoffensive and the malicious nodes. The routing trust values of the inoffensive nodes are greater than or equal to 0.96 for RT ≤ 0.8, so the decrease of the trust values occurs at the same threshold as if only routing attacks are performed (as presented in Section 6.4.4 and visible in Figure 100 in the line with ST = -1.0).

The storage trust values of the inoffensive nodes also do not vary considerably from the configuration in which only storage attacks occur (as presented in Section 6.5.2 and visible in Figure 100 in the column with RT = -1.0). The decrease of the routing trust values has an effect on the storage trust values as well because only nodes that are trustworthy for routing can be used for storage. So, the trust values vary a lot for RT ≥ 0.9 and also have a large spread between the quartiles as shown in the figures in Annex A.2.2.

For the malicious nodes, the median routing trust values behave in similar ways to the ones of the inoffensive nodes. The median storage trust values, however, show a totally different situation: The malicious nodes have rather large storage trust values which are even larger than the storage trust values of the inoffensive nodes. For M = 20%, the situation is extreme: For almost all thresholds, the median storage trust value of the malicious nodes is 1.0 as shown in Figure 107 in Annex A.2.1.

This happens because many malicious nodes are still in the storage grace phase in which they have a trust value of 1.0. On the other hand, the fact that the median trust values of the inoffensive nodes are slightly lower than 1.0 shows that most of them have left the grace phase and have received considerably more positive ratings than negative ones.

The reason why the malicious nodes are still in the storage grace phase is as follows: The malicious nodes attack both routing and storage and are quickly identified as malicious due to their routing behavior. Therefore, they are not considered for storage operations any longer and cannot earn any more storage ratings. So many of them still have less than 11 storage ratings and are therefore still in the storage grace phase when their routing trust values fall below the routing trust value threshold. However, this does not pose a threat as they are not considered for storage because of their low routing trust values. In addition, due to the fact that they are not considered for storage anymore, they cannot receive more storage ratings and leave the storage grace phase.

The CDFs show that the malicious nodes that have left the grace phase have very low trust values. Figure 101 shows the CDF for configurations in which both thresholds are set

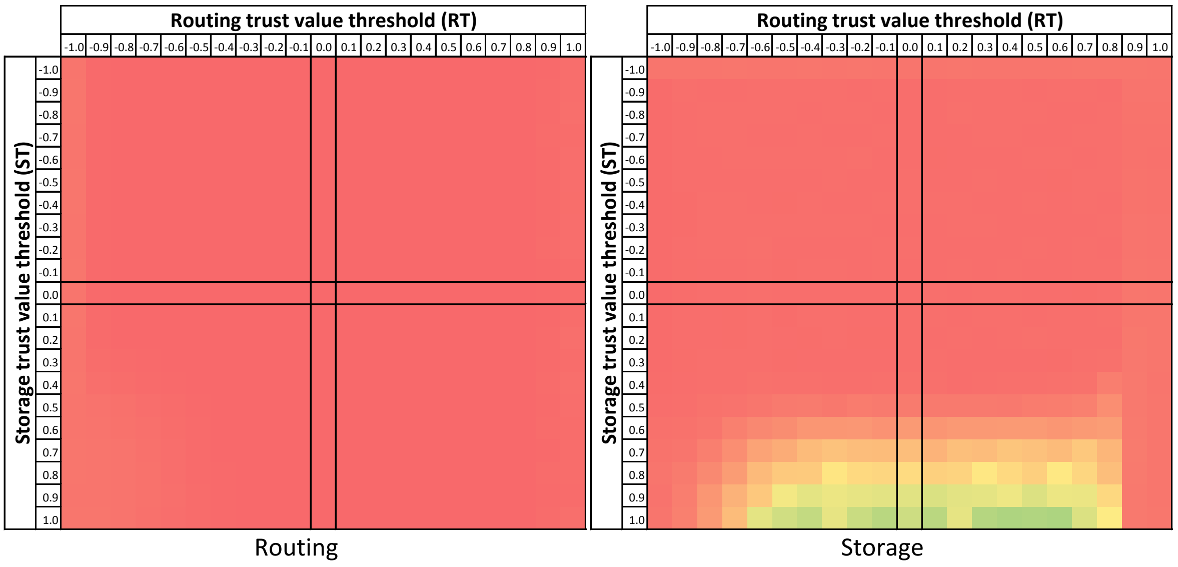


Figure 102: Fractions of trusted nodes that are malicious (M = 5%)

to 0.0, 0.5 and 0.6. The median[11] storage trust values in these configurations are -1.0, -0.92 and 1.0, respectively. All three graphs start at rather high values for -1.0, indicating the nodes that have left the storage grace phase, and stay almost constant for trust values below 1.0, denoting almost no nodes with a trust value between -0.5 and 1.0. The remaining nodes have a trust value of 1.0 because they are still in the storage grace phase.

The jumps in the graphs result from small numbers of positive ratings some of the malicious nodes have received during their grace phase: For example, a malicious node has a trust value of -0.82 if it leaves the grace phase with 11 ratings in total with one positive one among them:

$$T_S = \frac{R_S^+ - R_S^-}{R_S^+ + R_S^-} = \frac{1-10}{1+10} \approx -0.82$$

The fractions of trusted nodes that are malicious are shown in Figure 102. For routing, almost none of the trusted nodes are malicious, so the malicious nodes are identified very well. For storage, the fractions are high only for high storage trust value thresholds and routing trust value thresholds between -0.8 and 0.8. Because there are only 5% malicious nodes in total in the network, the large median storage trust values of the malicious nodes only have an impact on the fractions of trusted nodes that are malicious for these high thresholds.

As in the configurations in which only storage attacks occur, the Put success rates decrease for ST ≥ 0.5 as shown in Figure 103 because the total number of trusted nodes falls as well. Hence, for these thresholds, some of the Put operations fail because no node that is trustworthy first for routing and then also for storage can be found.

[11] In Figure 101, the median for RT = ST = 0.5 seems to be -1.0 because the CDF starts at (-1.0; 50%). This is due to the fact that the underlying results have an even number of elements. Because there is no single element in the middle of the sorted list of the elements that could be chosen as the median, the definition of the median for a set with an even number of elements is the mean value of the two elements that are in the middle. In this case, exactly half of all malicious nodes have a storage trust value of -1.0. The two elements in the middle of the list are -1.0 and -0.83, resulting in a median of -0.92.

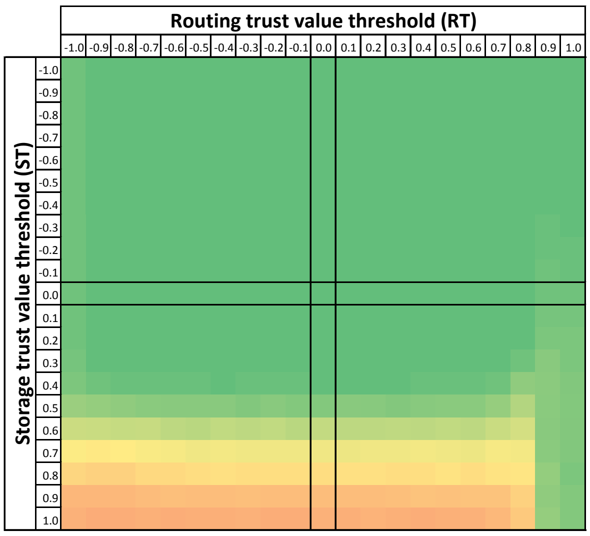
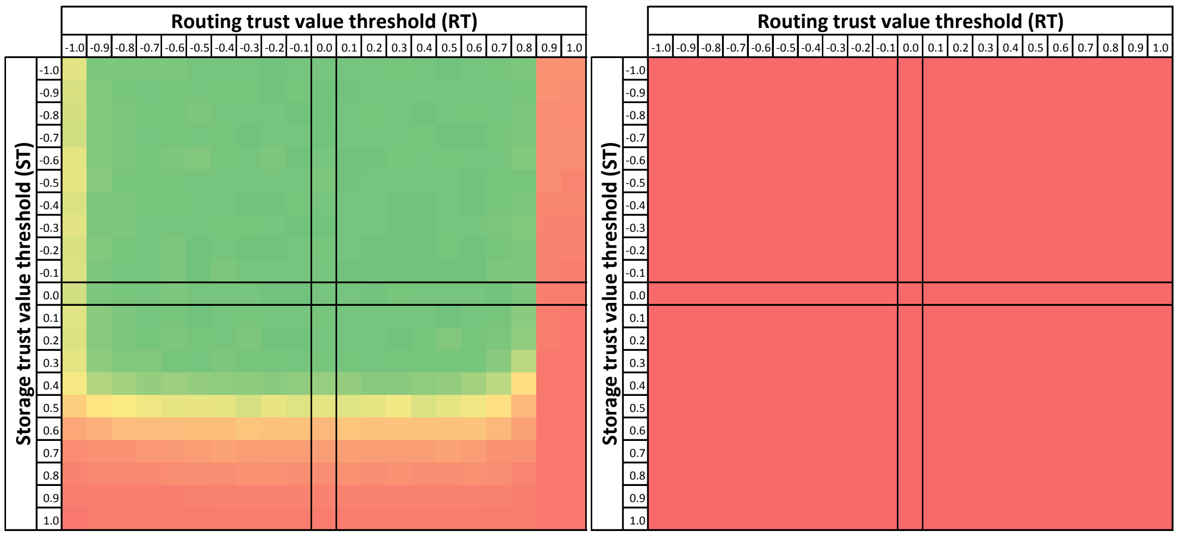


Figure 103: Median Put success rates (M = 5%)



Get success rates

False positives rates

Figure 104: Median Get success and false positives rates (M = 5%)

In spite of the large median storage trust values of the malicious nodes, the false positives rates are always 0% as shown in Figure 104. This happens because the most trustworthy node group approach does not take the trust values into account, but uses the amount of positive and negative ratings the nodes have, so no grace phase is applied (as explained in Section 4.11.2). In addition, a significant fraction of the malicious nodes is soon no longer considered during Get operations because they are already identified as malicious due to their routing attacks (as explained above).

The median Get success rate of the corresponding baseline configuration (cf. 6.3.3.1) is 63% for M = 5%, so TrustedKad still improves the rate for ST ≤ 0.4 and RT ≤ 0.7 in this worst-case scenario. Additionally, the concealment of the content item ID prevents the choice of false positives also for higher thresholds. For M = 20%, the baseline median Get success rate is only 2%. In that configuration, TrustedKad at least matches the baseline Get success rate for all storage trust value thresholds if the routing trust value threshold is greater than -1.0 and exceeds it in many cases. The maximum Get success rate that is achieved in the simulations is 74%, so TrustedKad increases the Get success rate by a factor of thirty-four.

6.7 Summary of the Simulation Results

In this chapter, the results of the simulations that are used to evaluate the functioning and effectiveness of TrustedKad have been presented. First, evaluations of baseline scenarios of an unprotected network have been performed (cf. 6.3). They have shown that despite its robustness, the Kademlia network is still susceptible for attacks. Even small amounts of malicious nodes can have a tremendous effect on the functioning of the network – especially routing attacks in partitioned networks do considerable harm to the network.

Afterwards, configurations in which TrustedKad is used to protect the network against either routing or storage attacks have been presented (cf. 6.4, 6.5). The results have shown that TrustedKad improves the security of the network to a small to large extent, depending on the scenario and the amount of malicious nodes in the network. The simulations have also shown that all of TrustedKad features are required in order to achieve the greatest security improvement, but that TrustedKad can still increase the success rates if a part of its features is successfully attacked. As an example shown in Sections 6.4.4 and 6.4.5, TrustedKad still improves the security of the P2P network if the cached trust information is faked by malicious nodes. In Section 6.5.3, TrustedKad's approaches of the most trustworthy node group and the concealment of the content item ID are deliberately deactivated. This results in a decrease of the success rates, but still improves the network security compared to the unprotected baseline configuration.

Finally, a network has been analyzed in which the malicious nodes attack both routing and storage operations (cf. 6.6). Even in this scenario, TrustedKad is able to improve the security of the P2P network.

Based on the Put and Get success rates of this configuration, the choice of a routing trust value threshold (RT) of about 0.5 and a storage trust value threshold (ST) of about 0.2 appears reasonable. For thresholds up to these values, the success rates of the simulations with 5% and 20% malicious nodes in the network are comparably large; for larger thresholds, they start to decrease quickly. These recommended values for the thresholds also achieve comparably good results in the previous scenarios in which the malicious nodes attack either routing or storage operations (cf. 6.4, 6.5). Although the results of Section 6.6 show that even negative thresholds can achieve good success rates, this is discouraged because nodes with such low trust values have received more negative than positive ratings. In addition, this would result in worse success rates if the malicious nodes attack routing operations during only 50% of all requests as shown in Section 6.4.6.

7 Conclusion and Outlook

In this chapter, TrustedKad and its features are summarized briefly. Afterwards, a résumé of the simulation results is given, followed by an outlook on possible next steps.

7.1 TrustedKad Features

In this thesis, TrustedKad has been introduced and evaluated. TrustedKad extends the Kademlia algorithm by trust mechanisms to improve its security. It uses trust values for routing and storage operations to identify inoffensive and malicious nodes (cf. 4.5, 4.6). In addition, it defines rules that recognize inoffensive and malicious behavior in order to rate the nodes accordingly after every interaction. Based on the trust values, trust value thresholds are used to avoid interaction with malicious nodes (cf. 4.10, 4.11). Furthermore, the unchoking mechanism by which nodes can recover from too many false negative ratings is defined (cf. 4.13) and evaluated (cf. 6.4.1, 6.5.1).

In order to impede attacks on certain content items, TrustedKad uses an algorithm to create verifiable node IDs that cannot be chosen freely by a node (cf. 4.3). They are derived from a self-signed certificate (cf. 4.2) that also includes modular anti-Sybil-attack information created by an algorithm that is chosen to be appropriate for the network in which TrustedKad is deployed (cf. 4.4). In addition, a node's ID has to change regularly so that the node is not responsible for the same logical location indefinitely. Moreover, the actual content item ID is concealed during Get operations to prevent attackers from attacking every Get request that reaches them (cf. 4.12).

TrustedKad uses anti-Sybil-attack information to keep attackers from creating Sybil nodes. However, even if the attacker would succeed in creating Sybil nodes to have them collude in attacks, the fact that the nodes cannot choose their node ID freely considerably impedes such collusion attacks. In large networks (such as the trackerless BitTorrent network with more than 150 million users), a vast amount of Sybil nodes would be required to control certain parts of the ID space. Additionally, the simulations have shown that TrustedKad even protects against colluding malicious nodes (cf. 6.5, 6.6).

The trust values are stored in a distributed way to avoid a bottleneck server and to impede attacks on the trust value storage (cf. 4.7). In order to not decrease the performance of the system, a node's trust information is cached at the node itself (cf. 4.8). The rating procedure is transparent and traceable because every rating has to be proved by presenting the original message of the rated node to show that an interaction has taken place.

The trust management nodes have to be able to prove the ratings by delivering the original rating messages (cf. 4.9). Finally, a node cannot manipulate its own cached trust information because it is signed by the trust management nodes.

7.2 Evaluation Results

In order to evaluate the aforementioned security measures, the key features have been implemented in a simulation environment. For this, OMNeT++ [40] and the OverSim framework [39] have been used. Comprehensive extensions of the existing code have been necessary to include the trust mechanisms. Additionally, the variations of the attacks on routing and storage operations and their implementation have been presented in Chapter 5.

Afterwards, the parameter settings for the simulation runs have been explained and the simulation environment has been introduced (cf. 6.1). Then, the key indicators for the evaluations have been identified: The Put and Get success rates, the false positives rates and the routing and storage trust values of the nodes (cf. 6.2).

7.2.1 Baseline

As a first evaluation step, a baseline has been determined to which TrustedKad is compared. The baseline configurations shown in Section 6.3 do not include trust mechanisms. Their security level is comparable to the level of current real-world implementations of the Kademlia algorithm as used by eMule [23] and BitTorrent [22]. The results have shown that the Kademlia algorithm is still susceptible to routing and storage attacks despite its robust design. Especially the attacks on routing operations have been very effective even for low fractions of malicious nodes; the greatest effect has been achieved if the malicious nodes succeed in attacking the Bootstrap process and therefore in creating several network partitions.

Storage attacks have proven to be less effective than routing attacks, as the Get success rates have been significantly greater compared to the scenarios in which only routing attacks have been performed. During storage attacks, colluding nodes have had a greater influence on the network than non-colluding nodes. If malicious nodes attack both routing and storage, the impact on the success rates is the combination of the impacts of the configurations in which either routing or storage operations are attacked.

7.2.2 Attacks Only Against Routing Operations

After determining the baseline, TrustedKad and its features have been evaluated, starting with attacks only against routing operations in Section 6.4. TrustedKad has clearly identified the malicious nodes in both partitioned and non-partitioned networks. Also, most of the inoffensive nodes have been identified as such in most configurations. As a result, TrustedKad has improved the Get success rate considerably compared to the baseline, especially in non-partitioned networks. Due to the fact that new nodes can verify the trustworthiness of a potential Bootstrap at that node's trust management nodes, TrustedKad significantly impedes network partitioning.

TrustedKad's effectiveness has also been demonstrated in a scenario in which one of its features, namely the caching of the trust information at the node itself, is successfully attacked (cf. 6.4.4, 6.4.5). In this scenario, the malicious nodes have not been identified as such as clearly as before, but still clearly enough. In addition, the inoffensive nodes have been identified even better. Therefore, the Get success rate has still improved compared to the baseline in a partitioned network; in a non-partitioned network, the improvement is considerably larger than in a partitioned network. TrustedKad has also proven to be resistant to attack variations by which malicious nodes attempt to improve their trust values by either attacking only half of all requests or only after collecting positive ratings first (cf. 6.4.6, 6.4.7).

7.2.3 Attacks Only Against Storage Operations

The impact of attacks only against storage operations has not been as large as if routing operations were attacked. Nevertheless, also in this situation, TrustedKad has improved the security of the P2P network as shown by the results presented in Section 6.5. The inoffensive and the malicious nodes have been clearly identified and distinguished from each other even in configurations in which the malicious nodes have colluded. The Get success rates have improved significantly for low and medium trust value thresholds; for higher thresholds, the inoffensive nodes have decided to cancel their operations due to a too small amount of trustworthy nodes. As a consequence, the false positives rates have decreased compared to the baseline. So TrustedKad has improved the security of the network also during storage attacks and reaches its goal of enabling a node to cancel operations instead of conducting them with non-trustworthy nodes.

Variations of the attacks have been conducted also for the storage attacks to demonstrate the usefulness of TrustedKad's features. In one configuration, the features "most trustworthy node group" and "concealment of the node ID" have been disabled (cf. 6.5.3). This has led to a less clear distinction of inoffensive and malicious nodes. Therefore, the Get success rates have worsened and the false positives rates have increased, proving the need for and effectiveness of the two features.

As in the routing attack analysis, the two variations in which the malicious nodes do not always attack are equally well averted (cf. 6.5.5, 6.5.6). In addition, a third attack variation has been performed for the storage attack analysis: In that variation, the malicious nodes have sent the original hash value of the previously stored content item during the GetHash phase. Also this variation aims to obtain higher trust values and to abuse them to deliver fake content items. The first goal of the attack variation has been achieved, as the median trust values of the malicious nodes have increased considerably. However, the Get success and false positives rates have even improved because the fake content item does not match the hash value and is therefore discarded (cf. 6.5.4).

7.2.4 Attacks Against Routing and Storage Operations

A worst-case scenario of a network in which the malicious nodes attack routing and storage operations at the same time has been presented in Section 6.6 as an example. This scenario represents a realistic scenario that would be chosen by attackers who attempt to cause the most damage. The analysis has shown the dependence of the storage trust values from the routing trust values: Only nodes that are trustworthy for routing can achieve storage ratings. Compared to the baseline, the Get success and the false positives rates have improved in almost all cases, and in many of them to a considerable extent.

So the evaluations have shown that TrustedKad is able to improve the security of a Kademlia-based P2P network. Malicious nodes are identified and avoided. The nodes are able to decide whether they trust the results of their operations and can choose to cancel operations if no trustworthy nodes can be found to interact with. This leads to improved Get success and false positives rates, so fewer Get operations end with a node choosing a fake content item.

7.3 Outlook

The feasibility and effectiveness of using trust mechanisms in a Kademlia-based P2P network have been demonstrated in this thesis. However, some of TrustedKad's features still have optimization potential: For example, the unchoking mechanism can be optimized because at the moment, a node that is unchoked during a Get operation probably cannot return a content item if it has not been stored on it before because it has not been unchoked during the corresponding Put operation. So TrustedKad could be extended by storing content items not only on trustworthy nodes, but also on non-trustworthy nodes that would be used if they were trustworthy. This way, if the nodes are unchoked during a Get operation, they would be able to respond with the correct content item to receive a positive rating. However, the implications and the attack potential of such a measure need to be analyzed thoroughly.

The decision of whether a positive or negative storage rating is given to a node group could be delegated to the application layer. In such a scenario, TrustedKad would not return a single content item version to the application layer, but a list of items. The application layer could then verify the items and determine the correct one. The node group that has delivered the correct version receives positive ratings. It would also be possible to not give any positive ratings if none of the versions is valid.

The results presented in this thesis have shown that network partitioning has a great impact. As future work, a detailed analysis of the number of partitions and how quickly malicious nodes are identified as such and therefore are discarded as bootstrap nodes could be performed.

The focus of further analyses could be set on performance evaluations and, e.g., determining a reasonable frequency of verifying the cached trust information at the trust management nodes.

For the simulations of this thesis, a central trust information store has been used instead of distributed storage because the goal of this thesis has been to demonstrate that a DHT is still functional if the nodes have a different view on the network and the nodes that are responsible for a content item. Now that this has been confirmed, further research is required to analyze the effects of the distributed storage of the trust information and attacks against that storage.

After this research has been done, TrustedKad should be implemented and analyzed in real-world networks such as Kad or trackerless BitTorrent. These networks deliberately refrain from using central entities and have known vulnerabilities that can be exploited easily. Therefore, they are ideal candidates for a live implementation of TrustedKad.

References

[1] Michael Kohnen, Mike Leske, and Erwin P. Rathgeb, "Conducting and Optimizing Eclipse Attacks in the Kad Peer-to-Peer Network," in *Proceedings of the 8th International IFIP-TC 6 Networking Conference*, Aachen, Germany, 2009, pp. 104-116.

[2] Michael Kohnen, Jan Gerbecks, and Erwin P. Rathgeb, "Applying Certificate-Based Routing to a Kademlia-Based Distributed Hash Table," in *Proceedings of the 3rd IARIA International Conference on Advances in P2P Systems (AP2PS)*, Lisbon, Portugal, 2011.

[3] Michael Kohnen, "Analysis and Optimization of Routing Trust Values in a Kademlia-Based Distributed Hash Table in a Malicious Environment," in *Proceedings of the 2nd IEEE Baltic Congress on Future Internet Communications (BCFIC)*, Vilnius, Lithuania, 2012, pp. 252-259.

[4] Michael Kohnen, "Applying Trust and Reputation Mechanisms to a Kademlia-Based Distributed Hash Table," in *Proceedings of the IEEE International Conference on Communications (ICC)*, Ottawa, ON, Canada, 2012.

[5] Skype. [Online]. http://www.skype.com/

[6] The Freenet Project. [Online]. https://freenetproject.org/

[7] Wuala - Secure Cloud Storage - Backup. Sync. Share. Access Everywhere. [Online]. http://www.wuala.com

[8] Zattoo - Live TV and more. [Online]. http://www.zattoo.com

[9] JXTA - The Language and Platform Independent Protocol for P2P Networking. [Online]. http://jxta.kenai.com/

[10] (2013, Jan.) [MS-PNRP]: Peer Name Resolution Protocol (PNRP) Version 4.0 Specification. [Online]. http://msdn.microsoft.com/en-us/library/cc239047%28prot.20%29.aspx

[11] Collanos | Think Out of the Inbox. [Online]. http://www.collanos.com/

[12] Giuseppe DeCandia et al., "Dynamo: Amazon's Highly Available Key-Value Store," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6, pp. 205-220, Oct. 2007.

[13] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," in *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, San Diego, CA, USA, 2001, pp. 149-160.

[14] Fay Chang et al., "Bigtable: A Distributed Storage System for Structured Data," in *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, vol. 7, Seattle, WA, USA, 2006, pp. 205-218.

[15] Avinash Lakshman and Prashant Malik, "Cassandra - A Decentralized Structured Storage System," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35-40, Apr. 2010.

[16] Hendrik Schulze and Klaus Mochalski. (2009) ipoque Internet Study 2008/2009. [Online]. http://www.ipoque.com/sites/default/files/mediafiles/documents/internet-study-2008-2009.pdf

[17] Sandvine Incorporated ULC. (2013, July) Global Internet Phenomena Report 1H 2013. [Online]. http://www.sandvine.com/downloads/documents/Phenomena_1H_2013/Sandvine_Global_Internet_Phenomena_Report_1H_2013.pdf

[18] (1999) Napster. [Online]. http://www.napster.com. The website of the original Napster P2P network was located at www.napster.com. However, this is now the website of the fee-based music service that is also called Napster.

[19] Janko Röttgers. (2001, July) Hundert Prozent Perfektion. [Online]. http://www.heise.de/tp/artikel/9/9064/1.html

[20] Jörg Eberspächer and Rüdiger Schollmeier, "First and Second Generation of Peer-to-Peer Systems," in *Peer-to-Peer Systems and Applications*, Ralf Steinmetz and Klaus Wehrle, Eds. Heidelberg, Germany: Springer, 2005, ch. 5, pp. 35-56.

[21] Petar Maymounkov and David Mazières, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," in *Revised Papers from the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, USA, 2002, pp. 53-65.

[22] BitTorrent. [Online]. http://www.bittorrent.com/

[23] eMule Project. [Online]. http://www.emule-project.net/

[24] Klaus Wehrle, Stefan Götz, and Simon Rieche, "Distributed Hash Tables," in *Peer-to-Peer Systems and Applications*, Ralf Steinmetz and Klaus Wehrle, Eds. Heidelberg, Germany: Springer, 2005, ch. 7, pp. 79-93.

[25] Ralf Steinmetz and Klaus Wehrle, "What Is This "Peer-to-Peer" About?," in *Peer-to-Peer Systems and Applications*, Ralf Steinmetz and Klaus Wehrle, Eds. Heidelberg, Germany: Springer, 2005, ch. 2, pp. 9-16.

[26] Ralf Steinmetz and Klaus Wehrle, "Peer-to-Peer-Networking & -Computing," *Informatik-Spektrum*, pp. 27(1):51–54, 2004.

[27] Andy Oram, Ed., *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2001.

[28] SETI@home. [Online]. http://setiathome.berkeley.edu/

[29] Diego Gambetta, "Can we trust trust?," in *Trust - Making and Breaking Cooperative Relations*, Diego Gambetta, Ed. Oxford, UK: Basil Blackwell, 1988, ch. 13, pp. 213-235.

[30] eBay - How Feedback Works. [Online]. http://pages.ebay.com/help/feedback/howitworks.html

[31] John R. Douceur, "The Sybil Attack," in *Revised Papers from the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, USA, 2002, pp. 251-260.

[32] Guido Urdaneta, Guillaume Pierre, and Maarten van Steen, "A Survey of DHT Security Techniques," *ACM Computing Surveys*, vol. 43, no. 2, pp. 8:1-8:49, Jan. 2011.

[33] Peter Bright. (2011, July) 4 million strong Alureon P2P botnet "practically indestructible". [Online]. http://arstechnica.com/security/2011/07/4-million-strong-alureon-botnet-practically-indestructable/

[34] (2012, Jan.) BitTorrent and µTorrent Software Surpass 150 Million User Milestone; Announce New Consumer Electronics Partnerships. [Online]. http://www.bittorrent.com/intl/es/company/about/ces_2012_150m_users

[35] The Annotated Gnutella Protocol Specification v0.4. [Online]. http://rfc-gnutella.sourceforge.net/developer/stable/

[36] Antony Rowstron and Peter Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," in *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, 2001, pp. 329-350.

[37] Atul Singh, Tsuen-Wan "Johnny" Ngan, Peter Druschel, and Dan S. Wallach, "Eclipse Attacks on Overlay Networks: Threats and Defenses," in *Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM)*, Barcelona, Spain, 2006.

[38] Tyson Condie, Varun Kacholia, Sriram Sank, Joseph M. Hellerstein, and Petros Maniatis, "Induced Churn as Shelter from Routing-Table Poisoning," in *Proceedings of the 13th ISOC Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, USA, 2006.

[39] Ingmar Baumgart, Bernhard Heep, and Stephan Krause, "OverSim: A Flexible Overlay Network Simulation Framework," in *Proceedings of 10th IEEE Global Internet Symposium (in conjunction with IEEE INFOCOM)*, Anchorage, AK, USA, 2007, pp. 79-84.

[40] OMNeT++. [Online]. http://www.omnetpp.org/

[41] Burkhard Stiller and Jan Mischke, "Peer-to-Peer Search and Scalability," in *Peer-to-Peer Systems and Applications*, Ralf Steinmetz and Klaus Wehrle, Eds. Heidelberg, Germany: Springer, 2005, ch. 17, pp. 269-288.

[42] Tor Klingberg and Raphaël Manfredi. (2002, June) Gnutella 0.6. [Online]. http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html

[43] (2013, May) eDonkey network. [Online]. https://en.wikipedia.org/wiki/EDonkey_network. Version of May 23rd, 2013. The original website of the eDonkey 2000 client (www.edonkey2000.com) does not exist anymore.

[44] Frank Dabek, Ben Zhao, Peter Druschel, John Kubiatowicz, and Ion Stoica, "Towards a Common API for Structured Peer-to-Peer Overlays," in *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, Berkeley, CA, USA, 2003, pp. 33-44.

[45] Ronald L. Rivest. (1992, Apr.) RFC 1321: The MD5 Message-Digest Algorithm. [Online]. http://tools.ietf.org/html/rfc1321

[46] (1995, Apr.) Federal Information Processing Standards Publication 180-1: Announcing the Standard for Secure Hash Standard. [Online]. http://www.itl.nist.gov/fipspubs/fip180-1.htm

[47] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, "Hash Functions and Data Integrity," in *Handbook of Applied Cryptography*.: CRC Press, 1997, pp. 321-383.

[48] Ingmar Baumgart, *Verteilter Namensdienst für dezentrale IP-Telefonie*. Karlsruhe, Germany: KIT Scientific Publishing, 2011, Dissertation thesis.

[49] Michael Kohnen, *Aktive Untersuchungen von DHT-Mechanismen am Beispiel eines Kademlia-basierten P2P-Netzes*. Essen, Germany: University of Duisburg-Essen, 2007, Diploma thesis.

[50] Jörg Eberspächer, Rüdiger Schollmeier, Stefan Zöls, and Gerald Kunzmann, "Structured P2P Networks in Mobile and Fixed Environments," in *Proceedings of the 2nd International Working Conference on Performance Modeling and Evaluation of Heterogeneous Networks (HET-NETs)*, Ilkley, UK, 2004.

[51] Flora Rheta Schreiber, *Sybil*.: Warner Books, 1973.

[52] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S. Wallach, "Secure Routing for Structured Peer-to-Peer Overlay Networks," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 299-314, Dec. 2002.

[53] Ingmar Baumgart and Sebastian Mies, "S/Kademlia: A Practicable Approach Towards Secure Key-Based Routing," in *Proceedings of the 13th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, Hsinchu, Taiwan, 2007, pp. 1-8.

[54] Kevin Hoffman, David Zage, and Cristina Nita-Rotaru, "A Survey of Attack and Defense Techniques for Reputation Systems," *ACM Computing Surveys (CSUR)*, vol. 42, no. 1, pp. 1:1-1:31, Dec. 2009.

[55] Mudhakar Srivatsa, Li Xiong, and Ling Liu, "TrustGuard: Countering Vulnerabilities in Reputation," in *Proceedings of the 14th International Conference on World Wide Web (WWW)*, Chiba, Japan, 2005, pp. 422-431.

[56] Loubna Mekouar, *Reputation-based Trust Management in Peer-to-Peer File Sharing Systems*. Waterloo, ON, Canada: University of Waterloo, 2010, Dissertation thesis.

[57] Sergio Marti and Hector Garcia-Molina, "Taxonomy of Trust: Categorizing P2P Reputation Systems," *Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 50, no. 4, pp. 472-484, Mar. 2006.

[58] Félix Gómez Mármol and Gregorio Martínez Pérez, "State of the Art in Trust and Reputation Models in P2P Networks," in *Handbook of Peer-to-Peer Networking*, Xuemin Shen et al., Eds.: Springer, 2010, pp. 761-784.

[59] Loubna Mekouar, Youssef Iraqi, and Raouf Boutaba, "Reputation-Based Trust Management in Peer-to-Peer Systems: Taxonomy and Anatomy," in *Handbook of Peer-to-Peer Networking*, Xuemin Shen et al., Eds.: Springer US, 2010, ch. 24, pp. 689-732.

[60] Loubna Mekouar, Youssef Iraqi, and Raouf Boutaba, "An Analysis of Peer Similarity for Recommendations in P2P Systems," *Springer Multimedia Tools and Applications*, vol. 60, no. 2, pp. 277-303, Sep. 2012.

[61] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina, "The Eigentrust Algorithm for Reputation Management in P2P Networks," in *Proceedings of the 12th ACM International Conference on World Wide Web (WWW)*, Budapest, Hungary, 2003, pp. 640-651.

[62] Nathan Griffiths, Kuo-Ming Chao, and Muhammad Younas, "Fuzzy Trust for Peer-to-Peer Systems," in *Proceedings of the 26th IEEE International Conference Workshops on Distributed Computing Systems (ICDCSW)*, Lisbon, Portugal, 2006.

[63] Karl Aberer, "P-Grid: A Self-Organizing Access Structure for P2P Information Systems," in *Proceedings of the 9th IFCIS International Conference on Cooperative Information Systems (CoopIS)*, Trento, Italy, 2001, pp. 179-194.

[64] Li Xiong and Ling Liu, "PeerTrust: Supporting Reputation-Based Trust for Peer-to-Peer Electronic Communities," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, pp. 843-857, July 2004.

[65] Karl Aberer and Zoran Despotovic, "Managing Trust in a Peer-2-Peer Information System," in *Proceedings of the 10th ACM International Conference on Information and Knowledge Management (CIKM)*, Atlanta, GA, USA, 2001, pp. 310-317.

[66] Wojciech Galuba, Karl Aberer, Zoran Despotovic, and Wolfgang Kellerer, "Authentication-Free Fault-Tolerant Peer-to-Peer Service Provisioning," in *Proceedings of the 5th International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P)*, Vienna, Austria, 2007.

[67] Sergio Marti, Prasanna Ganesan, and Hector Garcia-Molina, "DHT Routing Using Social Links," in *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS)*, San Diego, CA, USA, 2004, pp. 100-111.

[68] Ze Li, Haiying Shen, and Karan Sapra, "Leveraging Social Networks to Combat Collusion in Reputation Systems for Peer-to-Peer Networks," in *Proceedings of the 25th IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, Anchorage, AK, USA, 2011, pp. 532-543, Accepted for IEEE Journal Transactions on Computers.

[69] Nitin Chiluka, Nazareno Andrade, Dimitra Gkorou, and Johan Pouwelse, "Personalizing EigenTrust in the Face of Communities and Centrality Attack," in *Proceedings of the 26th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, Fukuoka, Japan, 2012, pp. 503-510.

[70] Aameek Singh and Ling Liu, "TrustMe: Anonymous Management of Trust Relationships in Decentralized P2P

Systems," in *Proceedings of the 3rd IEEE International Conference on Peer-to-Peer Computing (P2P)*, Linköping, Sweden, 2003, pp. 142-149.

[71] Tassos Dimitriou, Ghassan Karame, and Ioannis Christou, "SuperTrust – A Secure and Efficient Framework for Handling Trust in Super Peer Networks," in *Proceedings of the 26th ACM Symposium on Principles of Distributed Computing (PODC)*, Portland, OR, USA, 2007.

[72] Hui Xia, Zhiping Jia, Xin Li, Lei Ju, and Edwin H.-M. Sha, "Trust Prediction and Trust-Based Source Routing in Mobile Ad Hoc Networks," *Ad Hoc Networks*, vol. 11, pp. 2096-2114, Sep. 2013.

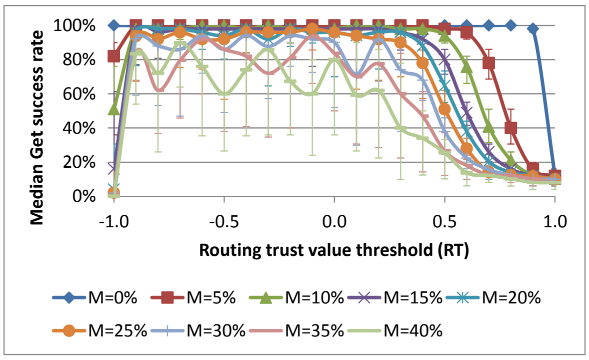
[73] Moritz Steiner, Taoufik En-Najjary, and Ernst W. Biersack, "Exploiting KAD: Possible Uses and Misuses," *SIGCOMM Computer Communication Review*, vol. 37, no. 5, pp. 65-70, Oct. 2007.

[74] INET Framework. [Online]. http://inet.omnetpp.org/

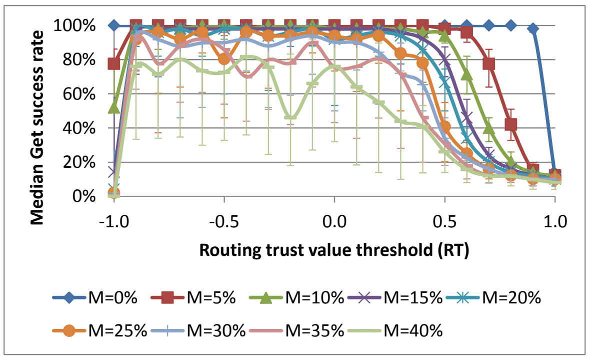
A Detailed Simulation Results

A.1 How TrustedKad Protects Against Routing Attacks

A.1.1 TrustedKad in a Partitioned Network



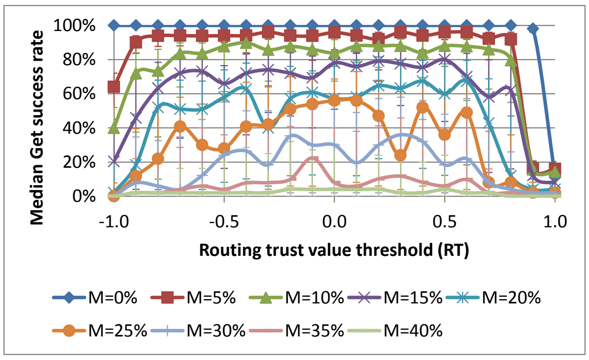
With isClosestAttack



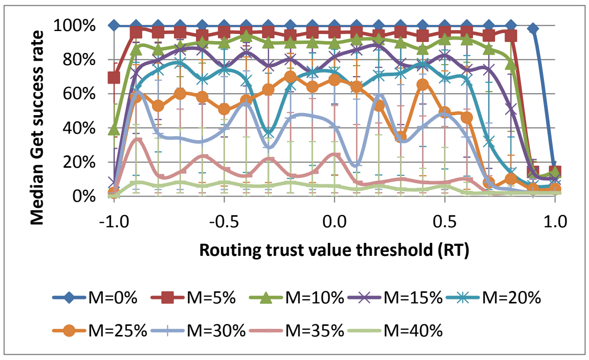
Without isClosestAttack

Figure 105: Median Get success rates

A.1.2 TrustedKad in a Partitioned Network With Successfully Faked Cached Trust Information



With isClosestAttack



Without isClosestAttack

Figure 106: Median Get success rates

A.2 How TrustedKad Protects Against Combined Routing and Storage Attacks

A.2.1 Figures for 20% Malicious Nodes

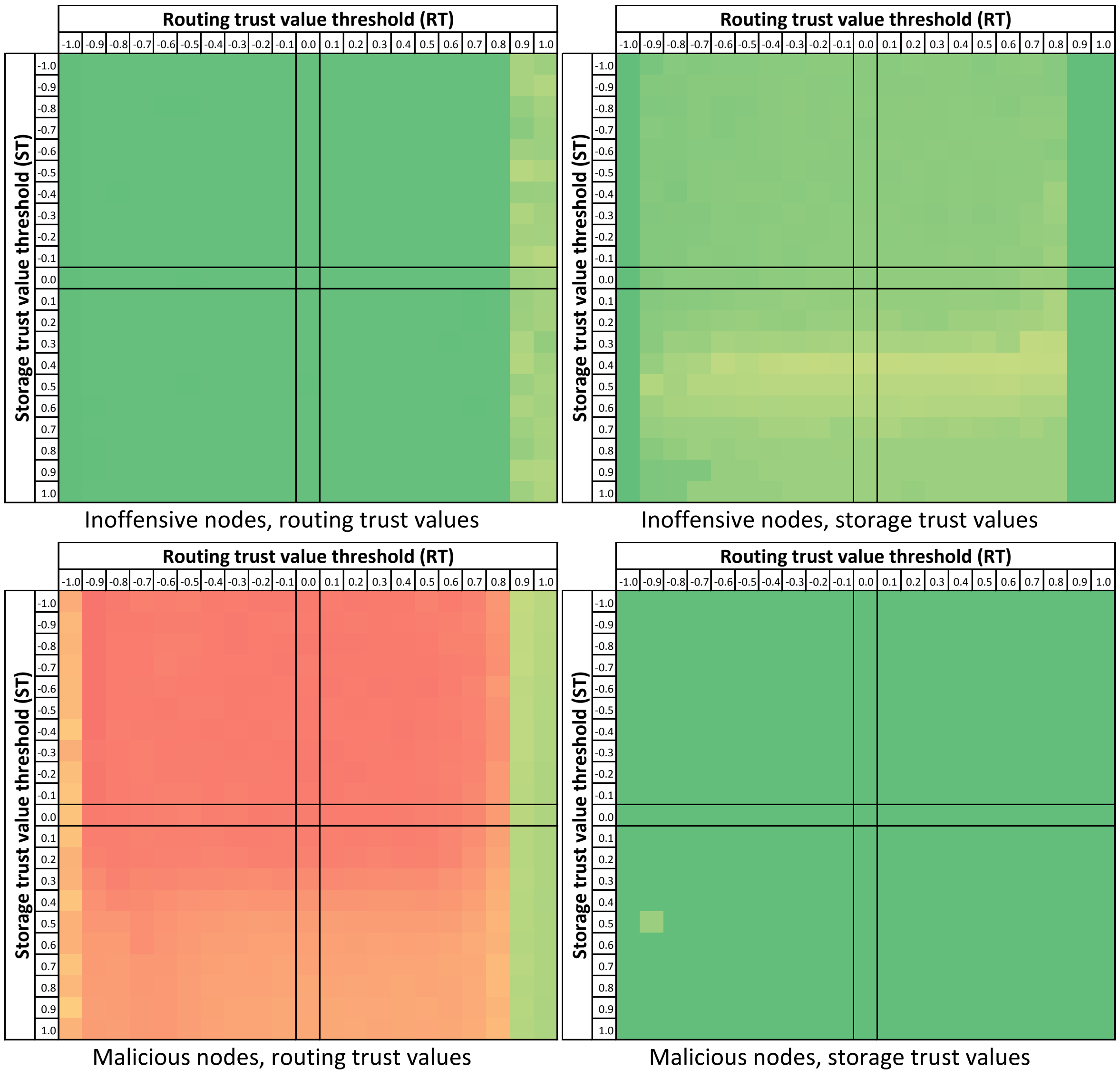


Figure 107: Median routing and storage trust values of inoffensive and malicious nodes (M = 20%)

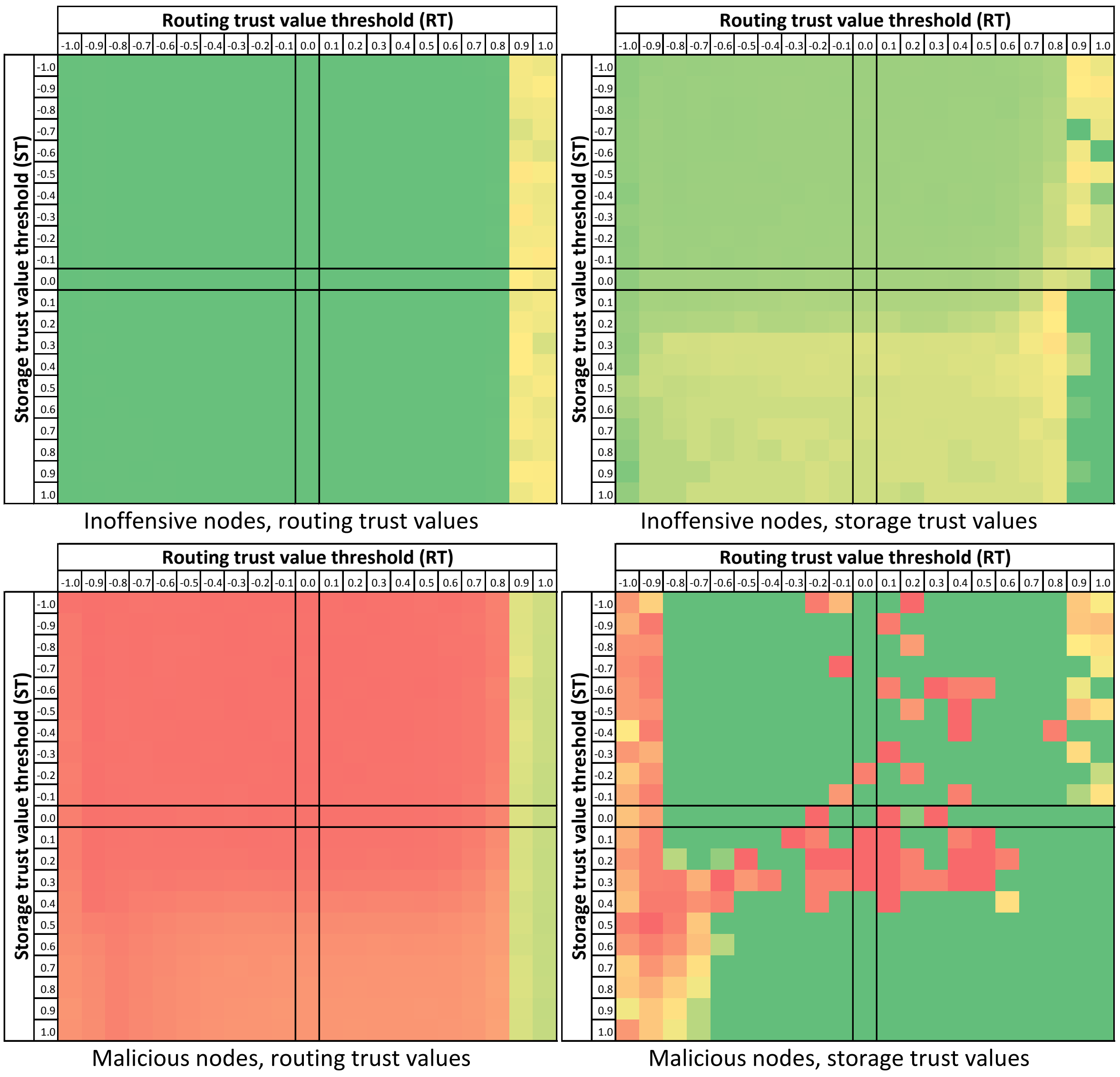


Figure 108: 25% quartiles of routing and storage trust values of inoffensive and malicious nodes (M = 20%)

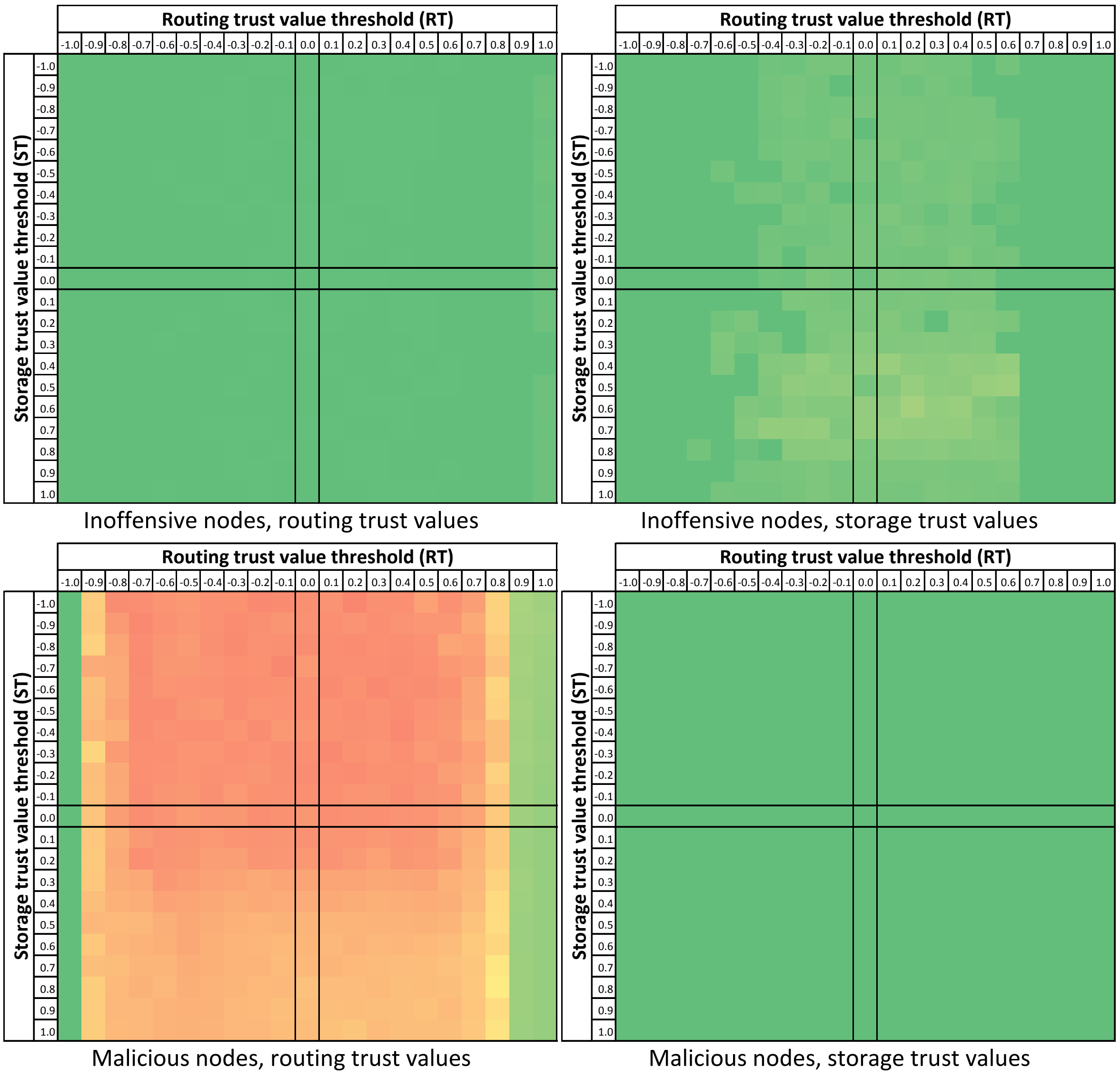


Figure 109: 75% quartiles of routing and storage trust values of inoffensive and malicious nodes (M = 20%)

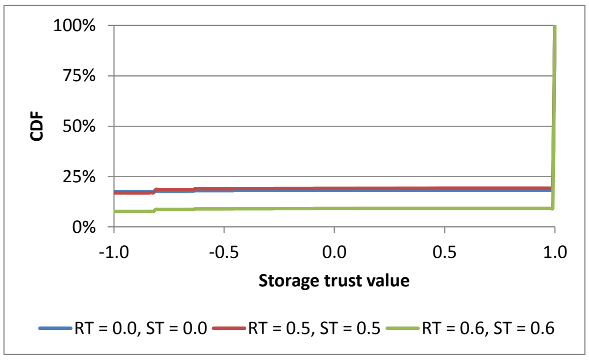


Figure 110: CDF of storage trust values of malicious nodes (M = 20%)

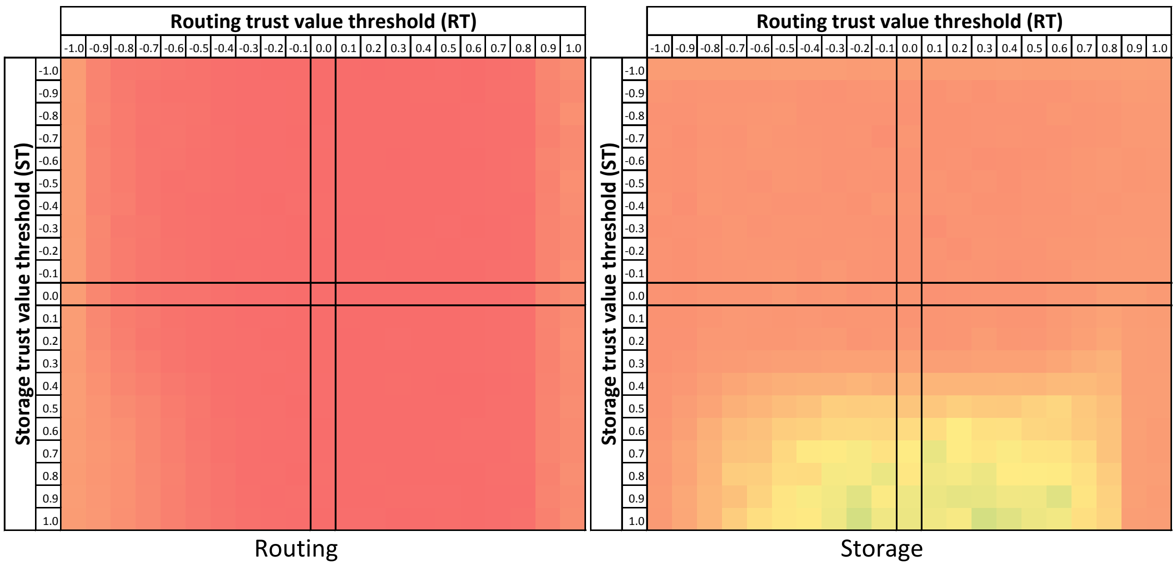


Figure 111: Fractions of trusted nodes that are malicious (M = 20%)

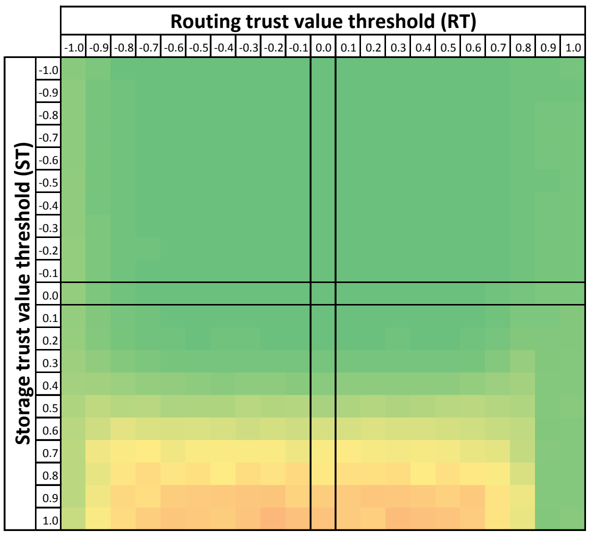


Figure 112: Median Put success rates (M = 20%)

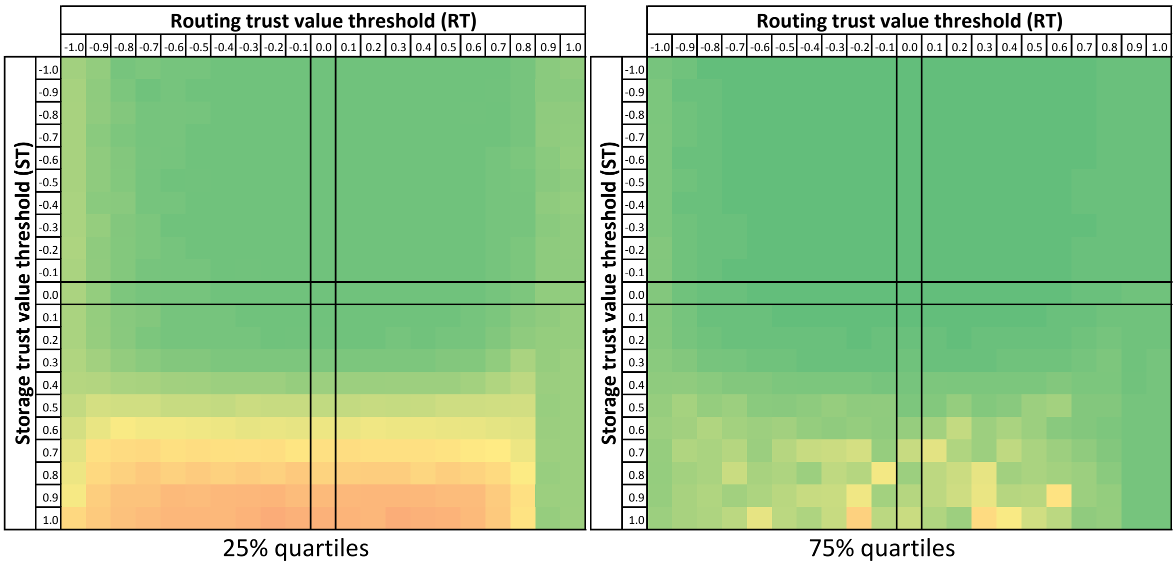


Figure 113: 25% and 75% quartiles of the Put success rates (M = 20%)

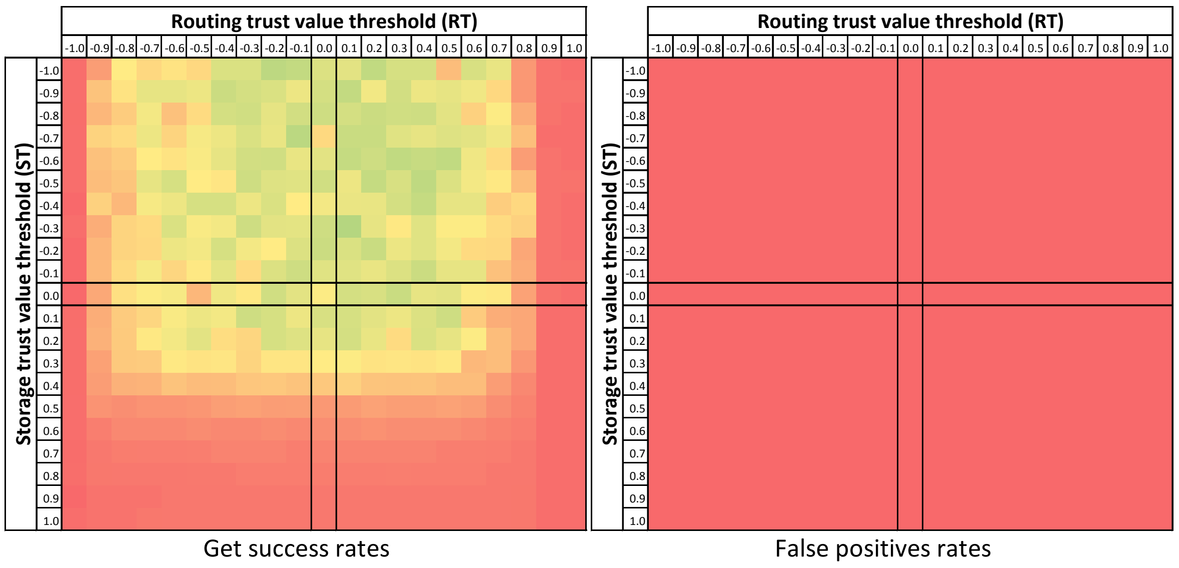


Figure 114: Median Get success and false positives rates (M = 20%)

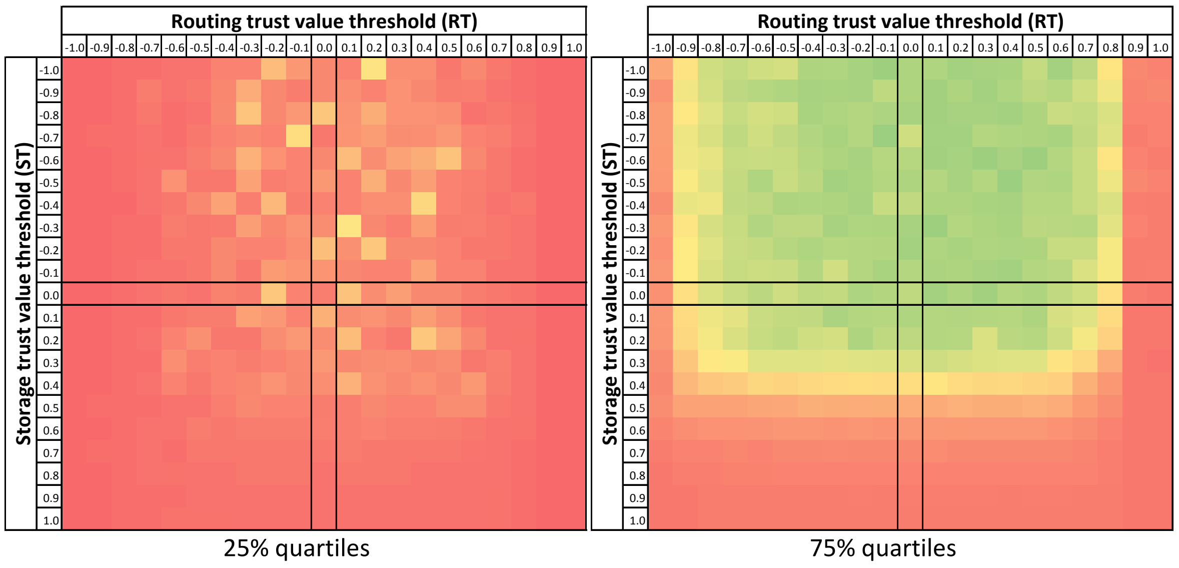


Figure 115: 25% and 75% quartiles of the Get success rates (M = 20%)

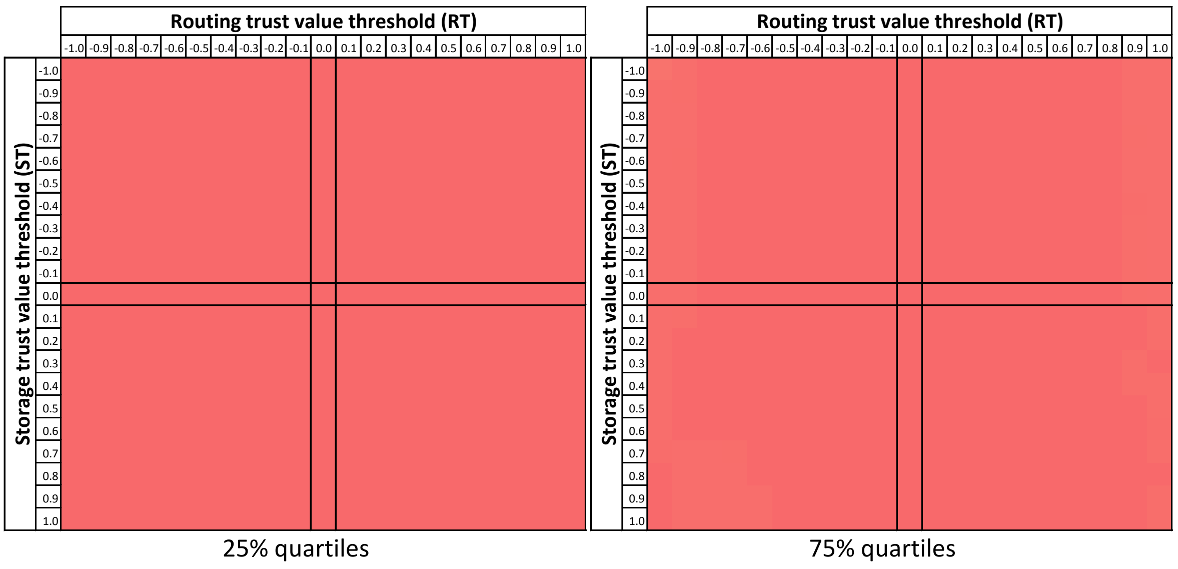


Figure 116: 25% and 75% quartiles of the false positives rates (M = 20%)

A.2.2 Quartiles for 5% Malicious Nodes

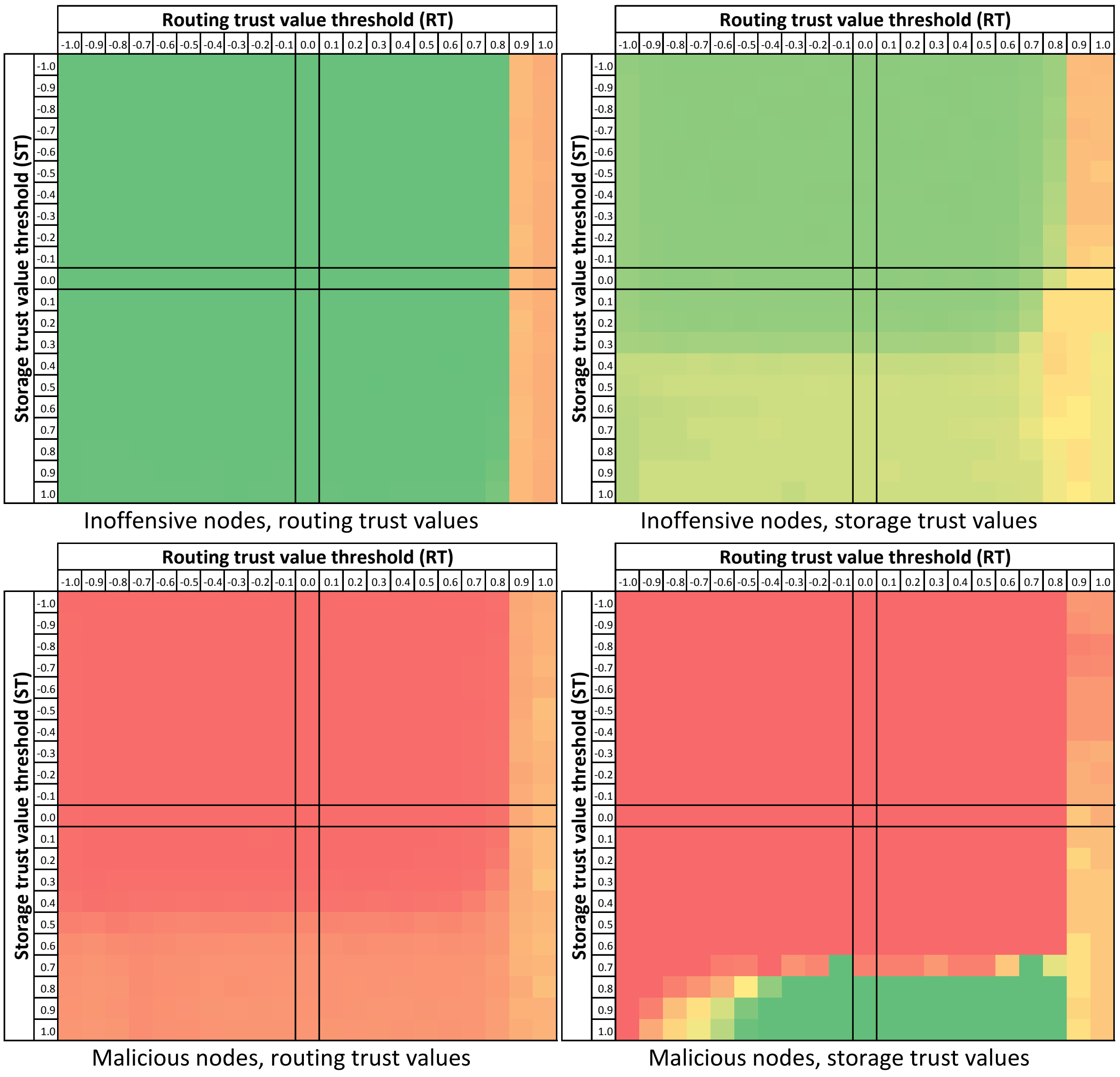


Figure 117: 25% quartiles of routing and storage trust values of inoffensive and malicious nodes (M = 5%)

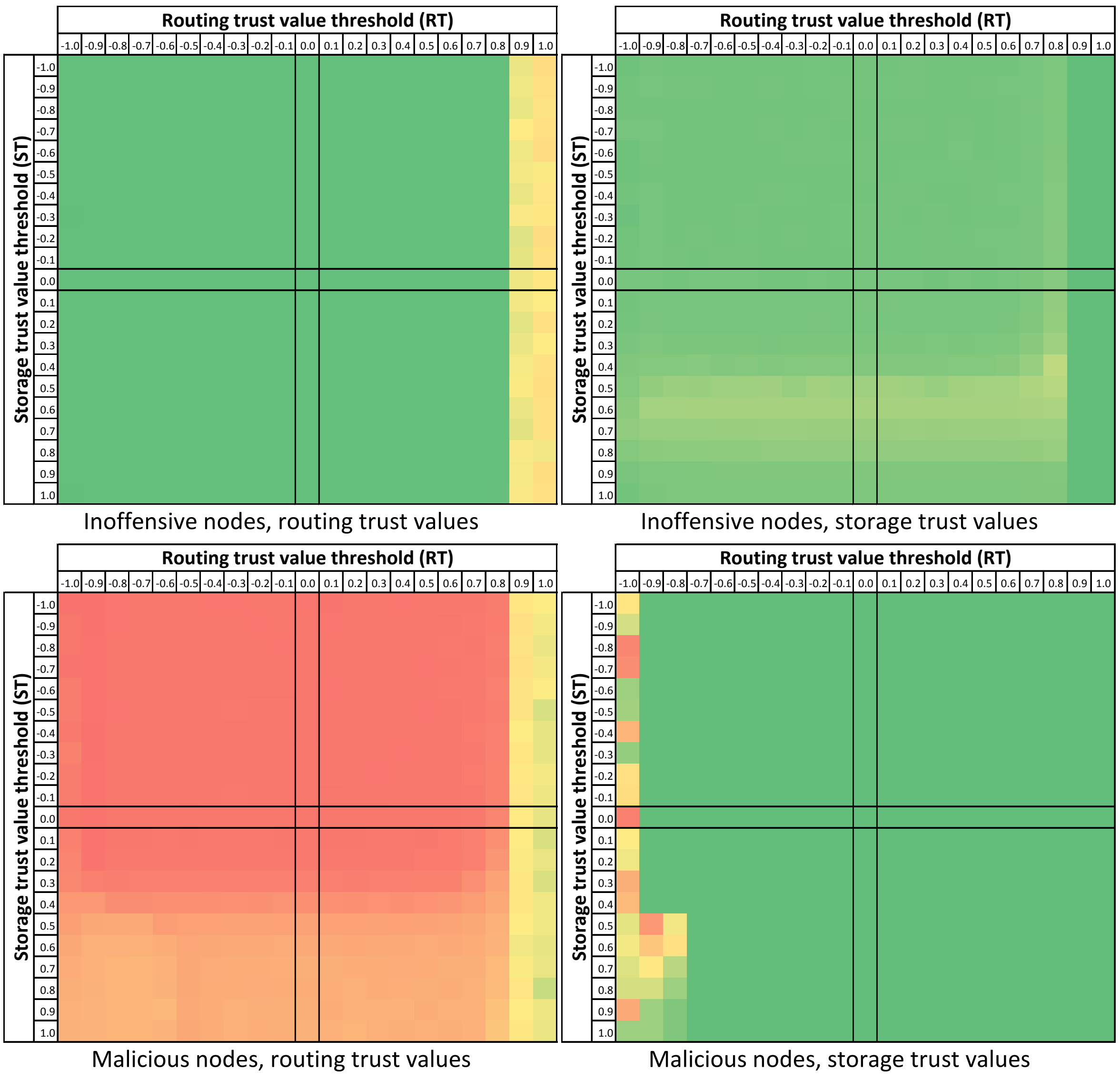


Figure 118: 75% quartiles of routing and storage trust values of inoffensive and malicious nodes (M = 5%)

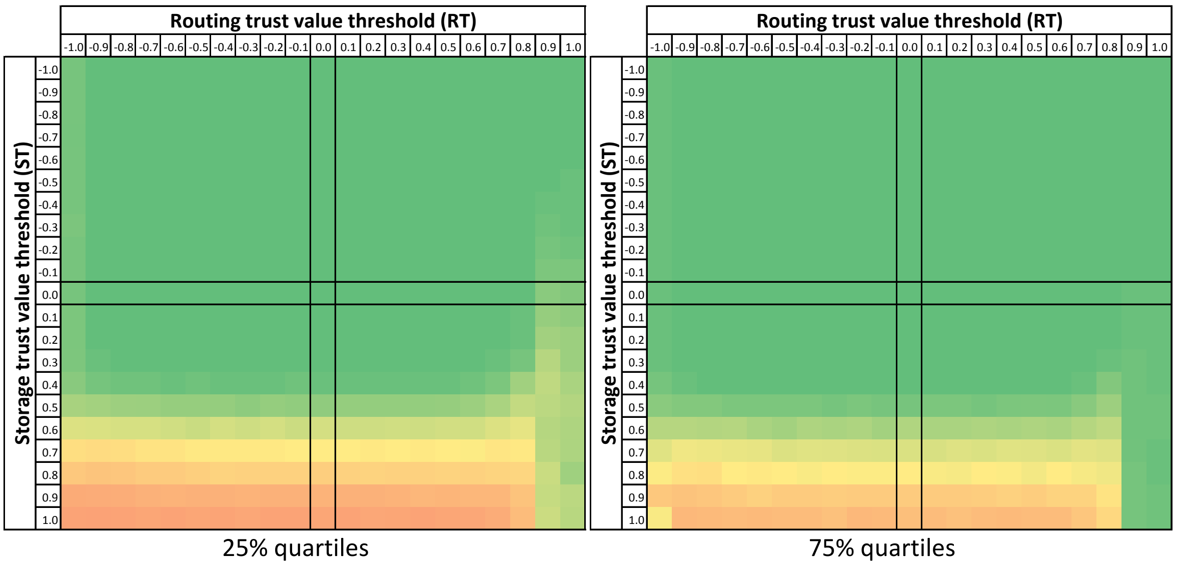


Figure 119: 25% and 75% quartiles of the Put success rates (M = 5%)

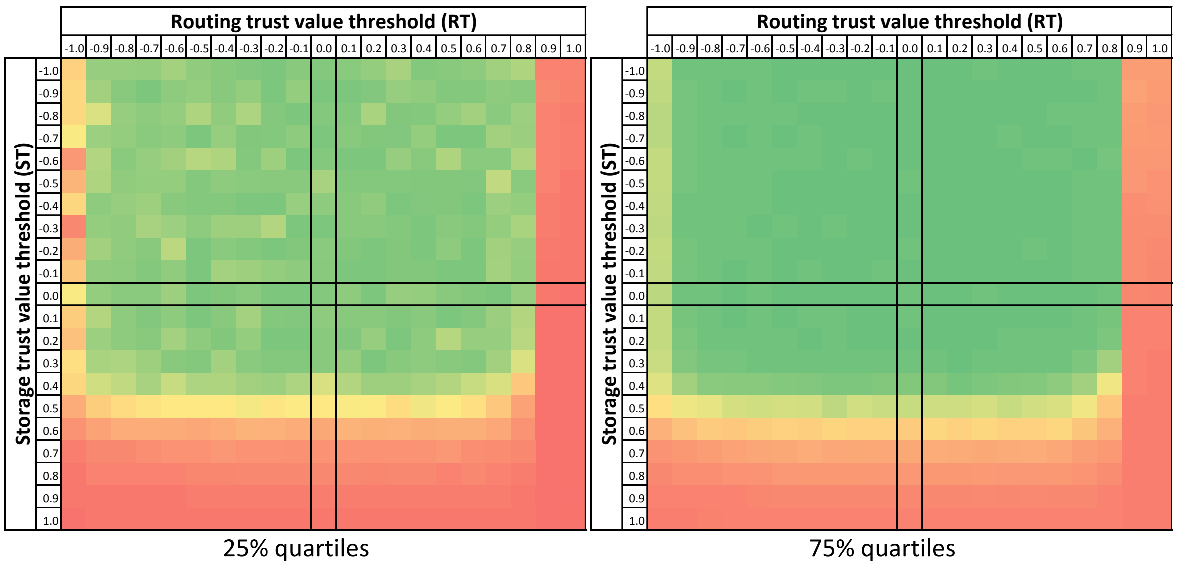


Figure 120: 25% and 75% quartiles of the Get success rates (M = 5%)

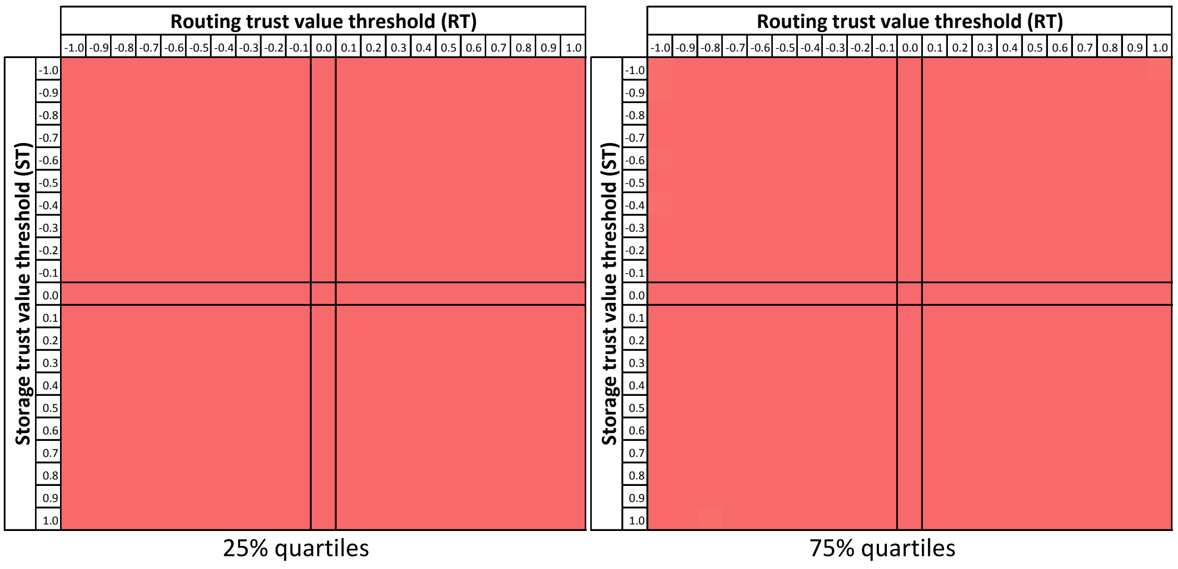


Figure 121: 25% and 75% quartiles of the false positives rates (M = 5%)