Achieving collaboration in distributed systems deployed over selfish peers

Tobias René Mayer

Ordering N° ....

Year 2013

THESIS

COTUTELLE-DE-THÈSE

ACHIEVING COLLABORATION IN DISTRIBUTED SYSTEMS DEPLOYED OVER SELFISH PEERS

AN ILLUSTRATIVE CASE STUDY WITH PUBLISH/SUBSCRIBE

Submitted to

National Institute of Applied Sciences (INSA de Lyon) Lyon, France	University of Passau Passau, Germany
Doctoral School of Computer Science and Mathematics (InfoMaths)	Department of Informatics and Mathematics

in fulfillment of the requirements for

DOCTORAL DEGREE

Prepared by

Tobias Rene Mayer

Defended on ....

Examination Committee

Prof. Harald Kosch	Université de Passau, Allemagne	Rapporteur
Prof. Gordon Blair	Université de Lancester, Royaume-Uni	Rapporteur
Prof. Jean-Marc Nicod	Université de Franche-Comté, France	Rapporteur
Prof. Sylvie Calabretto	INSA de Lyon, France	Examinateur
Prof. Michael Granitzer	Université de Passau, Allemagne	Examinateur
Prof. Aris Ouksel	Université d'Illinois (Chicago), États-Unis	Examinateur
Prof. Dietmar Jannach	Université de Dortmund, Allemagne	Examinateur
Prof. Lionel Brunie	INSA de Lyon, France	Directeur de thèse
Dr. David Coquil	Université de Passau, Allemagne	Invité

ABSTRACT

Today's networks are often characterized by a free aggregation of independent nodes. Thus, the possibility increases that a selfish party operates a node, which may violate the collaborative protocol in order to increase a personal benefit. If such violations differ from the system goals they can even be considered as attack. Current fault-tolerance techniques may weaken the harmful impact to some degree but they cannot always prevent them. Furthermore, the several architectures differ in their fault-tolerance capabilities. This emphasizes the need for approaches able to achieve collaboration from selfish nodes in distributed systems.

In this PhD thesis, we consider the problem of attaining a targeted level of collaboration in a distributed architecture deployed over rational selfish nodes. They have interest in deviating from the collaborative protocol to increase a personal benefit. In order to cover a broad spectrum of systems, we do not modify the collaborative protocol itself. Instead, we propose to add a monitoring logic to inspect the correctness of a node's behaviour. The designer of the monitoring system is faced with a complex and dynamic situation. He needs to consider aspects such as the specific circumstances (e.g. message traffic), the inspection effort or the node's individual preferences. Furthermore, he should consider that each agent could be aware of the other agents' preferences as well as selfishness and perform strategic choices consequently. This complex and interdependent setup corresponds to a class of Game Theory (GT) known as *Inspection Games* (IG). They model the general situation where an *inspector* verifies through inspections the correct behaviour of another party, called *inspectee*. However, inspections are costly and the inspector's resources are limited. Hence, a complete surveillance is not possible and an inspector will try to minimize the inspections.

In this thesis, the initial IG model is enriched by the possibility that a violation is not detected during an inspection (false negatives). Applied to distributed systems, the IG is used to model the network participants' strategy choice. As outcome, it enables to calculate system parameters to attain the desired collaboration value. The approach is designed as generic framework. It can be therefore applied to any architecture considering any selfish goal and any reliability technique. For the sake of concreteness, we will discuss the IG approach by means of the illustrative case of a Publish/Subscribe architecture.

The IG framework of this thesis secures the whole collaborative protocol by a monitoring approach. This represents a new way in terms of reliability mechanisms. The applicability is furthermore supported by the software library RCourse. Simplifying robustness evaluations of distributed systems, it is suitable for model verification and parameter calibration.

KEYWORDS: Reliability, availability, and security · Network management · Modeling of distributed systems · Game Theory · Inspection Games

RÉSUMÉ

Les réseaux actuels sont souvent caractérisés par une intégration dynamique de nœuds étrangers. La possibilité qu'une entité dissidente égoïste exploite un nœud augmente alors, ce qui peut constituer une violation du protocole de collaboration en vue d'accroître un avantage personnel. Si de telles violations diffèrent des objectifs du système, elles peuvent même être considérées comme une attaque. Si des techniques de tolérance aux fautes existent pour affaiblir l'impact sur le système, celui-ci ne peut pas totalement se prémunir de ce type d'attaque. Cela justifie la nécessité d'une approche pour maintenir un degré de collaboration nœuds égoïstes dans les systèmes distribues.

Dans cette thèse, nous considérons le problème d'atteindre un niveau ciblé de collaboration dans une architecture répartie intégrant des nœuds égoïstes, qui ont intérêt à violer le protocole de collaboration pour tirer parti du système. L'architecture et le protocole seront modifies le moins possible. Un mécanisme d'inspection de chaque nœud sera mis en place pour décider de la légitimité de ses interactions avec ses voisins. Le concepteur du système d'inspection est confronté avec une situation complexe. Il doit corréler plusieurs aspects tels que les circonstances particulières de l'environnement ou des préférences individuelles du nœud. En outre, il doit tenir compte du fait que les nœuds peuvent connaitre l'état de ses voisins et construire ses décisions en conséquence. La surveillance proposée dans cette thèse correspond à une classe de modèles de la théorie des jeux connus sous le nom « Inspection Game » (IG). Ils modélisent la situation générale où un « inspecteur » vérifie par des inspections du comportement correct d'une autre partie, appelée « inspectee ». Toutefois, les inspections sont coûteuses et les ressources de l'inspecteur sont limitées. Par conséquent, une surveillance complète n'est pas envisageable et un inspecteur tentera de minimiser les inspections.

Dans cette thèse, le modèle initial IG est enrichi par la possibilité d'apparition de faux négatifs, c'est à dire la probabilité qu'une violation ne soit pas détectée lors d'une inspection. Appliqué sur des systèmes distribués, cette approche permet de modéliser les choix collaboratifs de chacun des acteurs (violer le protocole ou pas, inspecter ou pas). Comme résultat, le modèle IG retourne les paramètres du système pour atteindre le niveau de collaboration souhaité. L'approche est conçue comme un « framework ». Elle peut donc s'adapter à toutes les architectures et toutes les techniques de fiabilité. Cette approche IG sera présentée à l'aide d'un exemple concret d'architecture Publish/Subscribe.

L'approche du jeu d'inspection de cette thèse pour objectif de sécuriser l'ensemble du protocole de collaboration. Ceci constitue un nouveau concept de mécanisme de fiabilité. Afin de permettre une large application, la généralité de cette approche est renforcée par la contribution RCourse. En simplifiant les évaluations de la robustesse des systèmes, elle permet la vérification de l'approche IG et le calibrage des paramètres du système.

MOT-CLÉS : Fiabilité, disponibilité et sécurité · Gestion de réseaux · Modélisation des systèmes répartis · Théorie des jeux · Inspection Games

ZUSAMMENFASSUNG

Heutige Netzwerke entstehen häufig durch einen dynamischen Zusammenschluss von Knoten. Dabei steigt die Wahrscheinlichkeit, dass ein Knoten von egoistischen Individuen betrieben wird welche bewusst das Protokoll verletzen um ein persönliches Ziel zu verfolgen. Dieses Verhalten kann nicht nur als schädlich sondern auch als Angriff betrachtet werden. Die Fehlertoleranz aktueller Systeme kann die negativen Auswirkungen abschwächen jedoch nicht vollständig verhindern. Aktuelle Systeme unterscheiden sich in diesem Punkt zum Teil dramatisch. Dies verdeutlicht den Bedarf von Systemen, die aktiv eine Befolgung des kollaborativen Protokolls aufrechterhalten und damit die korrekte Funktion eines verteilten Systems.

In dieser Dissertation betrachten wir das Problem ein bestimmtes Kooperationsniveau von egoistischen Knoten eines verteilten Systems zu erreichen. Für eine möglichst hohe Anwendbarkeit wird das System selbst so wenig wie möglich verändert. Stattdessen überwacht ein Monitoring-Ansatz die Korrektheit des Verhaltens der einzelnen Knoten. Der Designer des Monitoring-Systems ist mit einer komplexen und dynamischen Situation konfrontiert. Er muss Aspekte berücksichtigen wie den aktuellen Systemzustand (z.B. Nachrichtenverkehr), Inspektionsaufwand oder individuelle Ziele eines egoistischen Knotens. Dies führt zu einer komplexen, gegenseitigen und möglicherweise interaktiven Entscheidungslandschaft für die Monitoring-Logik und Knoten. Dies entspricht exakt einer Klasse der Spieltheorie: *Inspection Games* (IG). IGs modellieren die generelle Situation wo ein *inspector* durch Inspektionen das korrekte Verhalten eines anderen Individuums, dem *inspectee*, überprüft, welches vom vorgegebenen Protokoll abweichen möchte um einen individuellen Nutzen zu erhöhen. Aufgrund begrenzter Ressourcen versucht der inspector die Anzahl der Inspektionen zu reduzieren.

In der Dissertation wurden IGs als Lösungsvorschlag für das gegebene Problem auf verteilte Systeme adaptiert. Dazu wurde das initiale IG Modell zunächst durch die Möglichkeit erweitert, dass Protokollverstöße während einer Inspektion nicht entdeckt werden (false negatives). Der IG Ansatz ermöglicht die Modellierung des Kooperationsverhaltens in verteilten Systemen und die Berechnung von Systemparametern um das anvisierte Kooperationsniveau zu erreichen. Der Ansatz ist als generisches Framework konzipiert und kann individuellen Bedürfnissen angepasst werden. Dies umfasst insbesondere die Systemarchitektur, das individuelle Ziel und verwendete Techniken. Die gesamte Thematik wird in der Dissertation anhand eines konkreten Anwendungsbeispiels mit Publish/Subscribe Systemen diskutiert.

Der IG Ansatz überwacht die Korrektheit des gesamten Kooperationsprotokolls. Dies stellt hinsichtlich der Funktionsfähigkeit eines Systems einen neuen Ansatz dar. Die Anwendbarkeit wird weiterhin durch RCourse unterstützt, einer Software-Bibliothek, welche Simulationen hinsichtlich der Robustheit verteilter Systeme vereinfacht. Daher eignet es sich insbesondere zur Bewertung des IG Ansatzes und Kalibrierung von Parametern.

STICHWÖRTER: Zuverlässigkeit, Verfügbarkeit, und Sicherheit · Netzwerkmanagement · Modellierung von Computerarchitekturen · Spieltheorie · Inspection Games

CONTENTS

ACRONYMS

ALM	Application Layer Multicast
BAR	Byzantine, Altruistic, Rational
BO	Broker Overlays
BRS	Best Response Strategy
CPU	central processing units
DSR	Dynamic Source Routing
DoS	Denial-of-Service
EIP	Effective Inspection Probability
FIFO	First-In-First-Out
GT	Game Theory
IDE	Integrated Development Environment
IV	Inspection View
LRV	Loss Rate Variation
MAC	Media Access Control
MANET	Mobile Ad-Hoc Network
MEIP	Maximal Effective Inspection Probability
MQ	Median Quartile
NS	Notification Service
NHT	Next Hop Table
NPVR	Non-Permitted Violation Range
P2P	peer-to-peer
PoM	Proof-of-Misbehaviour
pub/sub	Publish/Subscribe
PS	Population Strategy
PVR	Permitted Violation Range

QoS	Quality of Service
RPC	Remote Procedure Call
SOA	Service Oriented Architectures
sCRE	Simple Comprehensive Reliability Evaluation
S-P2P	Structured Peer-to-Peer
TMS	Trust Management Systems
TTL	Time-To-Live
U-P2P	Unstructured Peer-to-Peer
VD	Value Distribution

LIST OF FIGURES

LIST OF TABLES

LISTINGS

Part I

RESEARCH CONTEXT & CHALLENGES

This first part serves as overview to the thesis' research domain and objectives. At first, the thematic context is introduced and some terminology clarified. Then, the thesis' vision and methodology are presented and the corresponding research challenges afterwards.

Overview

1 INTRODUCTION

This chapter gives a thematic introduction and an overview to the thesis' contributions.

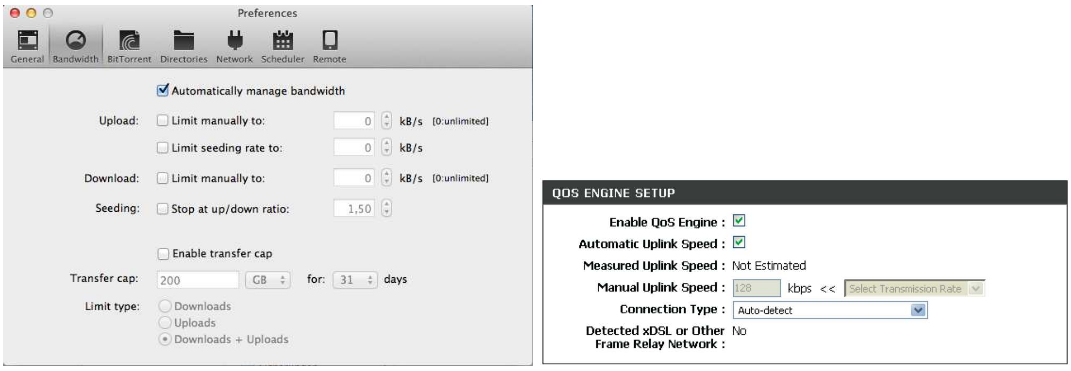
1.1 MOTIVATION

The today's distributed systems are realized as a mere enactment of a collaborative protocol. This protocol is typically implemented in the application logic and represents the system's objective. Violations of the protocol are considered as harmful due to the reduction of the system's performance. However, this is possible by selfishness-driven individuals, who follow personal objectives. With full administrative power over their machines, also called *peers* in the remainder, they are able to modify the application logic or encumber its execution. Direct physical access is not needed. Administrative power can also be obtained by malicious code (worms, viruses etc.) that take advantage of a system's vulnerability[1]. In fact, even software can operate selfishly. An example is a peer with limited energy resources, where a scheduler reduces the performance of specific applications to reduce the energy consumption. In general, selfishness-driven protocol violations are even considered as system attack [4, 1].

Faults (e.g. caused by hard-/software) can result to failures of collaborative interactions in distributed systems. These are also denoted as *Byzantine failure* [5]. In a larger scale, the probability that one of the peers is faulty increases strongly [6]. This may be exploited by selfish individuals by trying to hide their harmful behaviour behind Byzantine failures. Then, the probability of undetected selfishness-driven protocol violations is increased. Furthermore, tolerating Byzantine failures and selfishness plays an increasing role to achieve interoperability of the heterogeneous system landscape, a key aspect of future middeware systems [7]. Fault-tolerance techniques can reduce the impact of selfish-driven actions. However, they cannot fully avoid or tolerate them. This emphasizes the need for reliability approaches that take the possibility of selfishness-driven peers into account.

Aiyer et al. [8] introduced under the name *BAR model* – Byzantine, Altruistic, Rational (BAR) an abstract development concept. Selfish (rational) peers are considered already during the system design among cooperative (altruistic) peers and Byzantine failures. Such property is also denoted as *BAR toleran*ce. A full collaboration is not necessarily needed by the selfish peers. The system inherent reliability (e.g. redundancy) can be leveraged, which reduces the needed resources for the BAR tolerance mechanism. To this end, the system designer considers a targeted collaboration level. It is meant to be an average since selfishness-driven peers cannot be forced to collaborate. They may even sacrifice themselves for the personal goal accepting any consequences up to an expulsion from the system. Recent research activities addressed the problem of selfish behaviour in general (see for instance [9, 10, 11]). However,

[1] The amount of threats given by malicious code is continuously growing. See for example the Kaspersky Security Bulletin 2012 & Malware Evolution Report [3].



☑ Automatically manage bandwidth

Upload: □ Limit manually to: 0 kB/s [0:unlimited]

□ Limit seeding rate to: 0 kB/s

Download: □ Limit manually to: 0 kB/s [0:unlimited]

Seeding: □ Stop at up/down ratio: 1,50

□ Enable transfer cap

Transfer cap: 200 GB for: 31 days

Limit type: ○ Downloads
○ Uploads
● Downloads + Uploads

QOS ENGINE SETUP

Enable QoS Engine : ☑

Automatic Uplink Speed : ☑

Measured Uplink Speed : Not Estimated

Manual Uplink Speed : 128 kbps << Select Transmission Rate

Connection Type : Auto-detect

Detected xDSL or Other Frame Relay Network : No

Figure 1.1: Limiting bandwidth is a realistic issue. The functionality is integrated in the settings of the P2P software BitTorrent (left) as well as in the low-budget router D-Link DIR-615 (right).

there is no work existent for a BAR tolerance design that takes also a behaviour analysis into account to achieve a targeted collaboration level.

Approaches to avoid or tolerate selfish behaviour are manifold. They comprise for example trust- (e.g. [12, 13]) and reputation-based contributions (e.g. [14]) or introduce detection mechanisms (e.g. [15, 16]). In contrary, approaches for modelling systems with selfish peers are similar and typically done with Game Theory (GT). It enables to model the complex situations with circular dependencies. Hence, it is suitable for a behaviour analysis as done in this thesis. To this end, a concrete system is used as exemplary use case. Possible candidates are for example real world applications or mere communication models. An interesting compromise is Publish/Subscribe (pub/sub). It combines a communication paradigm (high generality) with the functioning of the BitTorrent system (real world relation). Thus, pub/sub represents an ideal representative of distributed systems for the thesis' studies.

EXAMPLE SCENARIO: DISTRIBUTED VIDEO STREAMING WITH BITTORRENT Let us consider now a scenario that illustrates the problem of selfishness-driven individuals in distributed systems. The scenario consists of distributed video streaming based on the peer-to-peer (P2P) application BitTorrent. The content dissemination (or a good portion of it) is done by the peers itself, which reduces the expenses for a complex server infrastructure. This scenario corresponds to an existing service, known under the name BitTorrent Live[2]. However, such video streaming systems are still under active research. In order to improve the throughput in presence of failures, they address among others general reliability approaches (see for example [17, 18, 19]) or the workload balancing and optimization (e.g. [20, 21]).

In this scenario, a selfish user has the personal goal of reducing the utilization of communication resources by limiting the outgoing bandwidth. This represents the problem of message loss since some network messages are omitted if the limit is reached. BitTorrent provides bandwidth limitation even as a settings feature in the application user interface (see figure 1.1 left). Furthermore, it can be realized at the network side without special knowledge. As an example, figure 1.1 shows on the right side the control panel of the router D-Link DIR-615 (about 20 €). These examples show that message drops is a realistic issue for P2P systems.

[2] http://live.bittorrent.com/

1.2 THEMATIC BACKGROUNDS

For a better understanding of the remainder, some thematic context is shortly clarified here.

1.2.1 *A Clarification of Terminology*

This section targets to unify the reader's interpretation of important terms used in this work.

DEPENDABILITY, RELIABILITY OR ROBUSTNESS? Dependability has several characterizations in literature. Definitions of two works are shown below. These are the one of Laprie [22] (upper quotation) and the IFIP WG 10.4 on dependable computing [23] (lower quotation):

> DEPENDABILITY
> *Computer system dependability is the quality of the delivered service such that reliance can justifiably be placed on the service.*
>
> *... the trustworthiness of a computing system which allows reliance to be justifiably placed on the service it delivers.*

Further taxonomies that structure terms such as dependability, reliability, fault-tolerance or survivability were proposed by Al-Kuwaiti et al. [24] and Avizienis et al. [25]. Based on the definitions of the literature we interpret the term dependability as the following characterization: *A service should – to some qualitative degree – exactly fulfil the service as it is specified and therefore expected.* The term dependability is not further discussed since it is considered as too abstract for concrete evaluations in this thesis. Instead, we focus on the other two notions – reliability and robustness – for the study of BAR tolerance in the remainder of the thesis. To this end, we rely on the definitions of the IEEE standard glossary for software engineering terminology [26]:

> RELIABILITY
> *The ability of a system or component to perform its required functions under stated conditions for a specified period of time.*
>
> ROBUSTNESS
> *The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions.*

Three major differences can be identified in these definitions. First, reliability focuses on the *ability to perform* specified operations, while robustness considers the *degree of correct system functioning*. Second, the notion of reliability assumes stated conditions. This specifies a broad range such as the ability to tolerate failures, system dynamics or to guarantee a specified amount of service quality (e.g. latencies). In contrast, robustness rather considers fault-tolerance issues, which may lead to stressful working conditions for the system. Finally, the definitions differ in the considered duration. Reliability takes account of stated conditions only for a specified period of time, while robustness assumes the stressful conditions as not limited in time. To summarize, reliability has a qualitative character. It considered as the *ability* to perform the system operation "to some degree". In contrary, robustness has a quantitative character and denotes the *degree* of system functioning under stressful conditions.

Reliability is also explained by means of safety and liveness properties. They were initially introduced by Lamport [27] as an abstract approach to characterize correct operation of distributed systems. They are outline in the following or the sake of completeness. However, analogue to dependability, they will not be further used in the thesis due to their abstractness.

> *Safety/Liveness Properties*
> *A safety property is one which states that something will not happen. … A liveness property is one which states that something must happen.*

FAULTS AND (BYZANTINE) FAILURES In contrast to the IEEE standard glossary for software engineering terminology [26], we do not consider that a fault may only be caused by a defect in hardware or an incorrect logic. Instead, we follow for this thesis the comprehensive taxonomy of faults of Azinienis et al. [25], who state that *faults* actually cause any type of failures. Hence, this taxonomy comprises all faults, that may affect a system during its life. For example, a fault may – among others – not only be caused by hard- or software but also by human actions or natural phenomena. In this context, we denote *failures* as the direct result of a fault, i.e. the system performed an erroneous action, is in an erroneous state or even not able to fulfil the functions within specified requirements. Occurring in distributing systems, failures may harmfully affect the mutual collaborative interaction and thus, the functioning of the whole distributed system itself. This is denoted as *Byzantine failure*. This term is well-known in the computer science community and relies on the *Byzantine Generals Problem* [5]. For ease of comprehension, we will consider also the term failure as synonym for Byzantine failures int the remainder of the thesis.

SELFISH BEHAVIOUR In order to characterize the notion selfish behaviour, we follow the definition of *rational* behaviour from Nielson et al. [1], whose classification of peers in a distributed system is based on game theoretic model and mechanism design [4]. Their classification, visualized in figure 1.2, distinguishes between peers that *do strategize* and those that *do not strategize*. Peers that do strategize can be further divided into *rational* (with system

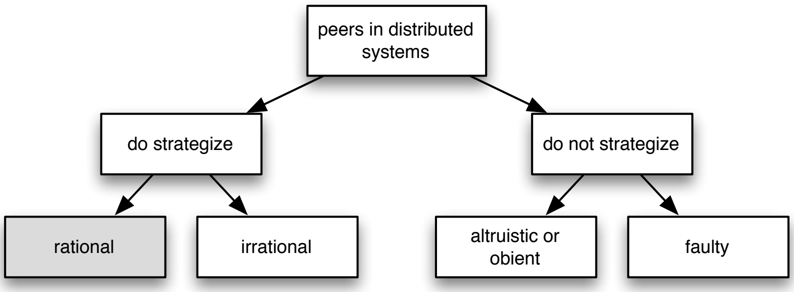


Figure 1.2: The figure illustrates a classification from Nielson et al. [1], which is used in the thesis. In this context, selfish behaviour is considered as synonym to rational behaviour, assuming further a personal goal that diverges from the system goal(s).

knowledge) and *irrational* ones (without or only incomplete system knowledge). Rational peers are therefore strategizing peers, which use their knowledge about the system to act in a selfish way (increase personal benefit). In contrary, irrational peers affect the system without being aware of the complete system mechanics, e.g. a Denial-of-Service (DoS) attack.

In this thesis, a peer's selfish behaviour considered as the behaviour of a rational peer that follows personal objectives. Hence, a peer is considered as selfish if its behaviour differs from the expected one specified by the collaborative protocol. To be more precise, this means the case where a Proof-of-Misbehaviour (PoM) is detected.

A selfish peer's objective is assumed to be divergent from the system goals. Hence, it is harmful for the system functioning and can be considered as an attack [1, 4]. Ideally, it should be always possible to distinguish between selfish behaviour and a failure. This is not always possible due to the peer's ability to hide the malicious actions behind a failure (e.g. pretending message losses). However, if some degree is exceeded – being a real failure or not – a peer can be considered malicious just as a selfish one. Selfish behaviour is not necessarily caused by human beings but can also be caused software logic. One example has already been mentioned in the introduction, where a scheduler affects the functioning (e.g. in terms of communication resources) in order to reduce the energy consumption.

Nielsen et al. discussed [1] three general mechanisms against selfishness. *Eliminating selfish behaviour as a concern* (e.g. out-of-band trust, trusted software) is not always applicable. *Genuine incentives* give peers collaboration incentives by the system design. An example is the encryption of message content for a data storage service. Being unable to distinguish the owner, a peer will abstain from deleting others' data to save on its resources. *Artificial incentives* always make use of auditing to detect misbehaviours and give some kind of collaboration incentive. Typical examples are the collection of proofs of misbehaviours (PoM) or reputation tables in combination with punishments (e.g. excluding a malicious peer from the system) when a misbehaviour is detected.

BAR TOLERANCE The BAR model [8] was introduced as abstract development model. It specifies that rational (selfish) behaviour should be considered already during the system design process among Byzantine failures and altruistic (collaborative) peers. In other words, a BAR tolerant system is assumed to implement fault-tolerance techniques as well as techniques against selfish behaviour in order to tolerate or prevent violations from the collaborative protocol. The BAR model increases the general reliability of a system but it lacks in preciseness due to its abstract definition. Therefore, the thesis intends to fill this gap by providing a structured approach to maintain BAR tolerance mechanisms. This vision is detailed hereafter.

1.2.2 *The Publish/Subscribe Paradigm*

The Publish/Subscribe (pub/sub) paradigm realizes application-layer multicast (ALM) communication. It provides an event-based interaction-style characterized by full decoupling in time, space and synchronization [28]. Thus, it is also considered as event-based middleware for the development of distributed applications (see for instance [29, 30, 31]). Pub/sub sys-

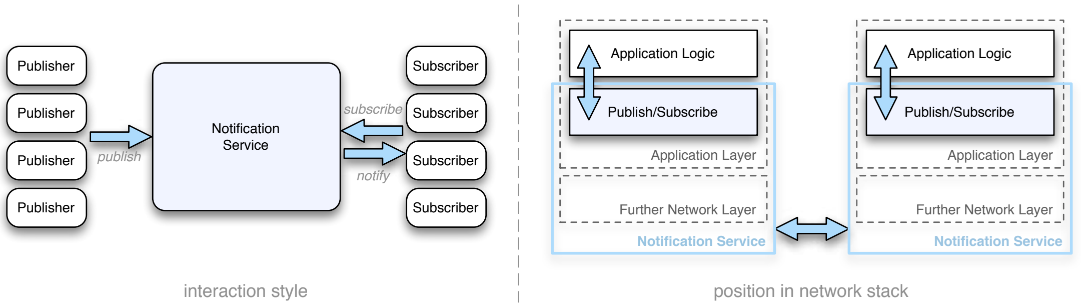


Figure 1.3: Graphical overview to the Publish/Subscribe communication paradigm showing the interaction-style (left) and the position in the network stack (right).

tems realize the decoupling between a set of content producers, called *publishers*, and a set of content consumers, called *subscribers*, by means of an intermediate logical component, called Notification Service (NS).

In pub/sub systems, the subscriber can subscribe to a multicast group of interest by informing the NS. Similarly, the publisher can publish an *event message* (or just *event*) by transferring it to the NS, which is then responsible to the delivery to interested subscribers. The process of identifying the interested set of peers is called *subscription matching*. The actual NS functioning remains a black box for the interacting publishers and subscribers.

The simplest NS realization is a single component or peer, which represents a client/server architecture. However, more advanced system consider usually a distributed architecture where the notification responsibility is distributed over different *notifier* peers. Indeed, they may also hold different roles. A peer may be publisher, subscriber and notifier at the same time, depending on the specific implementation. Due to the pub/sub logic at the application layer, overlay networks is maintained being independent from the implementation details of underlying network layers. Therefore, applications over networks such as the internet are possible without requiring specialized hardware or protocols. The pub/sub paradigm is illustrated in figure 1.3 from the interaction-style (left in figure) as well as technical point of view (left in figure), i.e. position in the network stack.

The pub/sub paradigm has attracted much research effort over the last decade, leading to a large number of heterogeneous systems. The first systems followed a deterministic dissemination[3] structure and targeted mainly core aspects such as subscription type (e.g. topic-, content-based) or overlay organization. Later on, unstructured approaches [32, 33, 34] introduced systems with non-deterministic dissemination[4] schemes. Among general architectural developments, researchers focussed also on other aspects such as scalability [35], reliability [36], quality of service [37] or security [38]. Furthermore, distributed optimization algorithms have been applied to Publish/Subscribe [39]. Worth to mention is also the PSIRP project[5], which tries to redesign the internet architecture from a pub/sub point of view. Several works reviewed the research (e.g. [40, 41, 42]) and outlined architectural specifications.

[3] In deterministic systems, the communication relations are calculated by a deterministic algorithm. Hence, the dissemination structure is fixed and can be reproduced.

[4] Non-deterministic dissemination makes always use (to some degree) of randomness such as random walks resulting in a dynamic dissemination structure.

[5] http://psirp.hiit.fi/

1.3 VISION & METHODOLOGY

VISION Several works considered selfish behaviour and introduced systems that are BAR tolerant to some degree. Typically, selfishness aspects focus only on specific selfish goals and circumstances (see for example [43, 9, 10]). Other threats are ignored and they are not considered to be coupled with other techniques or reliability mechanisms. Furthermore, implementing subsequently mechanisms with regard to other selfish goals becomes complicated due to the deep protocol modifications. To summarize, there is a design gap in terms of reliability mechanism considerations between the high level BAR model and the low level system implementations. This is illustrated in figure 1.4 left.

The *vision of this thesis* (figure 1.4 right) is to fill this gap by providing a design concept that supports the development of a system's BAR tolerance capabilities. To this end, two requirements are given and the corresponding

1. A solution approach shall be abstract enough to cover *any* Byzantine failure, *any* selfish goal and *any* system architecture. At the same time, it shall be precise enough to achieve specific implementations.

2. It should be able to leverage given reliability mechanisms or technique by attaining a targeted collaboration level of selfishness-driven peers.

METHODOLOGY In order to cover the vision's first requirement, a solution approach will be formulated as framework to achieves a flexibility in terms used mechanisms. In this context, genuine mechanisms are not possible. They require deep adaptions of system internal processes and are therefore in conflict with the abstractness of the vision. Hence, a distributed systems' functioning shall be affected as little as possible to increase the applicability to different system architectures. To this end, an artificial mechanism, a system monitoring approach will be used, which controls the peers' interactions in combination with giving collaboration incentives. The monitoring of all interactions is considered as too costly for the general case and performed by *sampling*. The principle of this monitoring approach is shown in figure

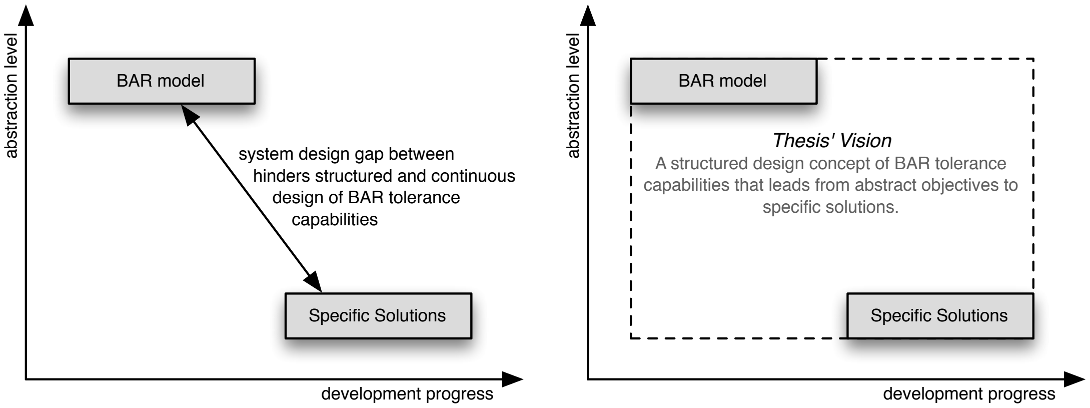


Figure 1.4: *Vision*: The objective is a design concept to enable a system deployment over selfish peers to attain a targeted collaboration level.

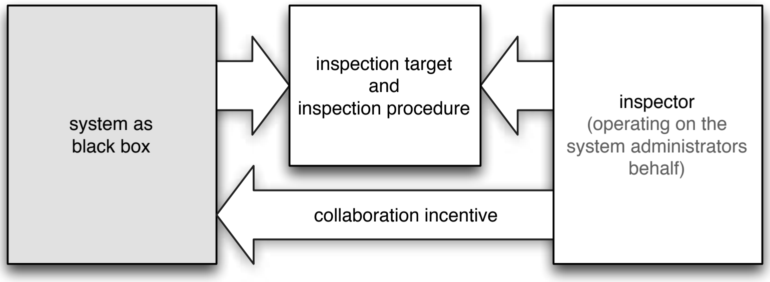


Figure 1.5: *General approach*: An inspection target maintains a peer's collaboration proofs. It is controlled by an inspector, who gives collaboration incentives (punishment) in case of detected PoM. The desired collaboration level is reached by an appropriate inspection rate.

1.5. The system itself is considered as a black box and its original functioning is kept basically unchanged. Instead, a data structure denoted as *inspection target* maintains proofs of a peer's behaviour. It is controlled during an inspection by an *inspector*, which works on behalf of the system administrator. He has the power to stimulate a collaboration by positive incentives (rewards) for collaboration or negative incentives (punishment) in case of detected misbehaviours. The latter one is a common technique (see for instance [44, 45]) and thus used in this thesis. Finally, a desired collaboration level is reached by an appropriate rate of inspections.

The determination of the inspection rate is a challenging task due to the behavioural interdependencies of the peer (violating or not) and inspector (inspecting or not). However, the monitoring approach corresponds directly to a class of Game Theory (GT): *Inspection Games* (IG). Hence, it makes sense to leverage the power of GT to model the complex and interdependent decision landscape. The modelling of the players' (the peer and the inspector) behavioural choice as IG enables a behaviour analysis of the players. As outcome, a system designer is able to calculate an inspection rate to reach the desired collaboration value.

During the thesis, the IG approach is applied to an illustrative use case. To this end, we consider the aforementioned selfishness-driven message drops scenario and Publish/Subscribe

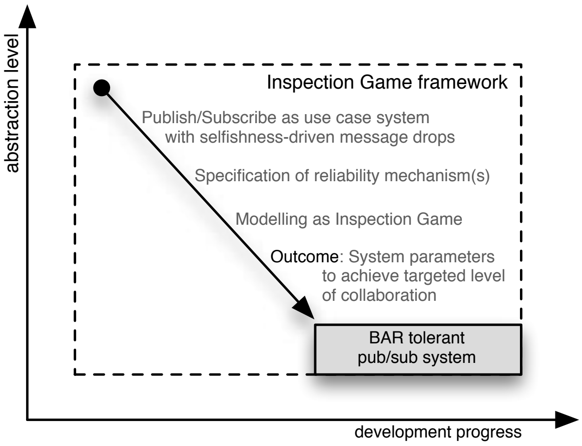

abstraction level
Inspection Game framework
Publish/Subscribe as use case system
with selfishness-driven message drops
Specification of reliability mechanism(s)
Modelling as Inspection Game
Outcome: System parameters
to achieve targeted level
of collaboration
BAR tolerant
pub/sub system
development progress

Figure 1.6: *Methodology*: The IG is considered as framework for the system design. The application to an illustrative use case shows the utilization in detail and how collaboration is achieved.

as system type. As a communication paradigm, it provides naturally a broad applicability. Furthermore, it realizes Application Layer Multicast (ALM) communication. This represents directly the BitTorrent application of the scenario with the multicast groups as the several video streams. Thus, pub/sub is considered as ideal system candidate. The IG application in the thesis stays general but provides at the same time a relation to an illustrative real world application. The methodology to meet the vision is illustrated in figure 1.6.

1.4 research challenges & contributions

research challenges In the context the thesis' vision, the main research challenge can be formulated as in the following:

> Main Research Challenge
> *Enable the deployment of a distributed system over selfish peers and obtain a targeted collaboration level in average.*

This challenge is similarly abstract as the vision itself and opens a bunch of further research questions. For example, what types of distributed system architectures are available and what are there general reliability capabilities. A crucial aspect concerns the selfishness-driven violations and the impact on the collaborative protocol or, in other words, on the whole functioning of a specific system. Further aspects to examine are how to integrate the specific mechanisms in a theoretic model, how does it perform under real world conditions and how could it be realized in current applications. The possible research question are manifold. However, there is a general *design principle* recognizable. During the top-down design it is important to have a mutual calibration feedback between the system behaviour and the theoretical model. For example, determine a system's BAR tolerance capabilities has influence on the targeted collaboration level. At the same time, it must be examined if the theoretical model's outcome corresponds the expectations, possibly resulting in further calibrations and evaluations.

In order to meet the broad main challenge, it is split into three straightforward research challenges that are listed hereafter. The first challenge (A) evaluates the reliability of current systems as part of the state-of-the-art. As we will see later, some preparations are needed, resulting in two further tasks (A.1) and (A.2). Challenge (B) refers to quantitative evaluations such as the system impact of selfishness-driven violations. It represents the practical side of the aforementioned principle, the mutual feedback during system design. Challenge (C) comprises not only the (generic) theoretical framework but also an application to distributed systems. It represents therefore the theoretical side of the design principle and the main research contribution.

(A) *Evaluate the BAR tolerance capabilities of current distributed systems.*

  (A.1) *Provide an architectural classification during the evaluation.*

  (A.2) *Specify possible failures that can be used as evaluation metrics.*

(B) *Develop a tool that supports the simulative evaluation of selfishly operating peers' impact on the system functioning.*

(C) *Develop an approach to attain a targeted (average) level of collaboration if a distributed system is deployed over selfish peers.*

CONTRIBUTIONS OF THE THESIS The thesis' contributions are directly related to the research challenges. They interact as described before in the top-down design principle, which is illustrated in figure 1.7. The interactions of the contributions are outlined here and will be detailed during the conclusion of the thesis (chapter 10).

- *Preparations for comprehensive evaluation*
  Related to the tasks (A.1) and (A.2), this contribution serves as preparations to the BAR tolerance evaluation of challenge (A). It is presented in part II and comprises of an architectural classification and a taxonomy of elementary failures. Thus, this contribution discusses the lacks of current system and introduces preparations for a BAR tolerance evaluation. It was published in [46] and is part of [47].

- *Reliability evaluation of pub/sub systems*
  Also presented in part II as part of the state-of-the-art, this contribution evaluates the BAR tolerance capabilities of current pub/sub systems. It is related to challenge (A) and performs the evaluation in a qualitative way based on the prior work, the architectural classification and a taxonomy of failures. This contribution was published in [47].

- *RCourse: An extension library for peersim to robustness evaluations*
  RCourse represents the practical contribution in terms of the aforementioned design principle. It simplifies simulative studies with a special focus on robustness capabilities to enable quickly launched evaluations. Hence, it serves as tool to evaluate the impact of selfishness-driven violations and to verify the IG approach or parameter calibrations. Furthermore, it provides simulation scenarios that are based on the BAR tolerance evaluation. Thus, RCourse is suitable to enrich it by quantifying information. Published in [48], RCourse addresses challenge (B) and is introduced in part III.

- *Inspection Game approach to deploy distributed systems over selfish* peers
  The whole part IV is related to research challenge (C) and introduces with the enhanced IG approach (chapter 8) a solution approach. It is preceded among others by GT foundations and is followed by a discussion with regard to real world applications. During

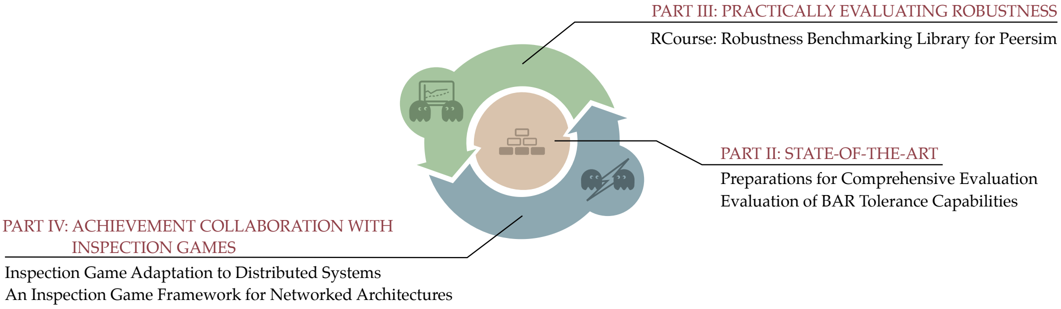

PART III: PRACTICALLY EVALUATING ROBUSTNESS
RCourse: Robustness Benchmarking Library for Peersim
PART II: STATE-OF-THE-ART
Preparations for Comprehensive Evaluation
Evaluation of BAR Tolerance Capabilities
PART IV: ACHIEVEMENT COLLABORATION WITH INSPECTION GAMES
Inspection Game Adaptation to Distributed Systems
An Inspection Game Framework for Networked Architectures

Figure 1.7: Thematical overview to the main contributions of the thesis

the GT considerations, the initial IG model is extended by false negatives and Nash equilibria calculations are introduced for all games. The extended game types are appropriate to model and analyze the interdependent strategy choice of selfish players with resource limitation constraints. By applying the IG approach to distributed systems, it represents the theoretic counterpart to RCourse. In combination they are able to realize the thesis' vision and to achieve BAR tolerant system implementations beginning from an abstract starting point. The GT foundations are published in [49, 50].

1.5 STRUCTURE OF THE THESIS

The current part introduced among others the thematic context and the thesis' research challenges. The remainder structured as indicated in the following.

PART II presents the current state-of-the-art in terms tolerating selfishness-driven harmful impact on the system functioning. The following two chapters are related to this objective:

- Chapter 2 discusses the related work of the scientific community with regard to the research challenges.
- Chapter 3 reviews BAR tolerance capabilities of 25 systems. This chapter is dedicated to meet challenges (A), (A.1) and (A.2).

PART III introduces an approach for practically evaluating the robustness of distributed systems with Publish/Subscribe in particular:

- Chapter 4 details the RCourse library, which intends to meet challenge (B).

PART IV is related to research challenge (C), i.e. attaining a specific degree of collaboration for distributed systems being deployed over selfish peers. This part consists of several considerations as outlined in the following:

- Chapter 5 provides theoretic foundations used by the IG approach in the remainder.
- Chapter 6 discusses several aspects that need to be considered for the application of the IG to distributed systems.
- Chapter 7 introduces the IG approach as possible solution for research challenge (C). All game and implementation details are provided that are needed for a realization.
- Chapter 8 presents an enhanced dynamic IG approach, which is based on the one of the foregoing chapter 7. Hence, this chapter focuses on the differences to the initial one.
- Chapter 9 discusses the applicability of the IG to real world systems as well as the meaning for user and administrator.

PART V finalizes the thesis with two chapters:

- Chapter 10 concludes the thesis by discussion the meaning of the contributions in terms of the research challenges.

- Chapter 11 provides some possible future work related the main contribution, i.e. the IG approach to attain a targeted collaboration level.

Part II

STATE-OF-THE-ART IN DEALING WITH SELFISHNESS-DRIVEN PEERS

The previous part served as thematic introduction. This part is determines now the current state-of-the-art in terms of BAR tolerance capabilities of distributed systems. At first, the related work is discussed. Then, a set of works are reviewed to evaluate their reliability capabilities.

Overview

2 RELATED WORK AROUND THE RESEARCH CHALLENGES

This chapter introduces related work around the research challenges stated before and also for Inspection Games due to its critical role for the thesis' solution approach.

2.1 CHALLENGE A: RELIABILITY & BAR TOLERANCE EVALUATIONS

We discuss now the related work for research challenge (A), the BAR tolerance evaluation of distributed systems. This section covers also the tasks (A.1) and (A.2) since, as we will see, a BAR tolerance evaluation is not available. Several reliability related works are presented before as an outline to the vast progress in this field.

RELIABILITY Reliability has been a design issue from the beginning of the distributed systems development. In fact, Freeman [51] discussed already in the year 1976 – far before modern and flexible networked architectures – design concepts to achieve reliable software. Lamport formulated in 1977 safety/liveness properties[1] [27] as abstract requirement for reliable systems and a variety of works addressed further foundations or reliable systems (see for instance [52, 5, 53, 54, 55, 56, 57, 58]). In the following decades, the research community developed several practical approaches to reach reliability in distributed systems. General fault-tolerance [59, 60, 61, 62, 58, 57, 52] was addressed (typically using redundancy and cryptography) but also specific issues such as recovery from failures [63, 64], security [65, 25, 66], trust and reputation [67, 68, 17, 13], accountability [69] and Quality of Service (QoS) [70, 71]. The contributions are manifold but can roughly be separated into general reliability concepts (e.g. [72, 73, 74]) and specific approaches or implementations (e.g. [75, 76]). During the last years, the scientific community draw increasing attention the problem of selfish peers – in addition to (Byzantine) fault-tolerance – for system reliability and security. Nielson et al. characterized in [1] the notion of rationality in the context of selfishness driven attacks. The BAR model was proposed by Aiyer et al. [8] as an abstract development concept to deal with selfishness already during systems design. It was applied to several systems (e.g. [43, 77]) and a broad range of works considered selfishness. A closer look will be given in section 2.3.

BAR TOLERANCE EVALUATIONS Several surveys reviewed the numerous contributions in the domain of reliability and BAR tolerance. In the beginning, they focused on general methodologies and mechanisms [78, 79, 80]. More recent overviewing works aggregate the foregoing research works and discuss rather general shortcomings, analysis models or reliability concepts. In other words, they concentrate on the review of reliability evaluation

[1] *"A safety property is one which states that something will not happen. ... A liveness property is one which states that something must happen."*

models. This corresponds also to a recent survey [81] of Isa et al. The authors argue the need for an intermediation approach to correlate reliability models with performance aspects. In their work, they review existing metamodels and discuss differences in terms of concepts, modelling and analysis. A further survey of Tyagi and Sharma [82] estimated the reliability of component based systems in terms of their

- scope, e.g. component based systems, Service Oriented Architectures (SOA),
- models, e.g. path or state based models,
- technique, e.g. algebraic methods, mathematical formulas,
- validation scheme, e.g. fully validated, validation through experiments,
- and noticeable features.

This framework was used to review 13 approaches but specific reliability details (e.g. evaluation results) are not mentioned. In this context, a recent survey of Pai [83] reviews software reliability models. He observed among others the common assumption for white-box models that component reliability is available while they are actually still under research. Another work of Golash [84] reviews the reliability mechanisms that are specially tailored to ethernet based systems. He discusses among others appropriate approaches for specific scenarios as well as different network structures such as point-to-point, Frame Relay or Multiprotocol Label Switching (MPLS). Further surveys intend to clarify notion, terminology and interpretation [24, 25], consider shortcomings of design concepts for reliability prediction [85] or focus on specific issues or environments. The latter one comprises for example data transport reliability in wireless sensor networks [86], patterns for security and reliability standards[87] or low-level memory error correction techniques [88]. Several books review reliability in (distributed) software systems (see for instance [61, 89]). They basically present, discuss and compare general (network) reliability concepts and implementation techniques (redundancy, coding, failure recovery etc.). One further work is worth to be mentioned: Esposito et al. introduced in [36] an approach for a reliable and time-sensitive pub/sub middleware. They reviewed as preparation the reliability of 30 pub/sub systems and introduced a taxonomy of Byzantine faults. However, their review is not fully satisfying for the study of this thesis. Important failures such as packet loss or link/node crashes were considered but no selfish peers or other possibilities with negative impact. Modified event content, injection of unauthorized messages and varying publishing rates (e.g. in streaming applications) are examples for missing scenarios. In addition, the taxonomy mixes peer failures (e.g. link anomalies, node crash) with more enhanced failure scenarios such as node churn[2] or network partitioning.

To summarize, the given surveys concentrate basically on reliability evaluation concepts and models. There is currently no work available that reviews the actual reliability capabilities in a comprehensive way considering elementary failure types. Similarly, the additional possibility of selfish-driven peers and correspondingly needed mechanisms – BAR tolerance

[2] Node churn denotes the naturally dynamics of the system, i.e. entering/leaving peers. However, if some degree is exceeded, it may harmfully affect the system functioning.

Table 2.1: The heterogeneity of overviewing works hinders the architectural comparability of Publish/-Subscribe systems.

Liu and Plale [40]	Baldoni et al. [41]	Carzaniga et al. [91]
System Architecture - *Client/Server Model* - *P2P Model*	Overlay Infrastructure - *Broker Overlay* - *P2P Structured Overlay* - *P2P Unstructured Overlay* - *Overlay for Mobile Networks* - *Overlay for Sensor Networks*	Architecture - *Hierarchical Client/Server* - *Acyclic P2P* - *General P2P* - *Hybrid*
Event Distribution Scheme - *Multicast*	Event Routing - *Flooding* - *Selective* - *Gossiping*	Routing Strategies - *Subscription Forwarding* - *Advertisement Forwarding*

– is not covered. This shows the lack of a comprehensive BAR tolerance evaluation and underline the need for research challenges (A) and (A.2).

Despite the extensive research in the domain of reliability and dependability, there are still ambiguities in terms of notions and interpretations. The authors of the two aforementioned works [24, 25] identified this lack and targeted to counteract this situation. They compare several attributes as well as definitions, clarify possibly overlapping interpretations determine characteristics for common concepts (e.g. fault-tolerance, security). However, they concentrate only on reliability, security and similar aspects. A heterogeneous interpretation is also given for the architectural specification of distributed systems. An example is shown in table 2.1 for three works, which are reviewing pub/sub systems. Only two comparable architectural dimensions are shown here: the overlay organization and the routing (or dissemination) technique. Nevertheless, strong distinctions are in focus and terminology are noticeable. This heterogeneity hinders architectural comparisons. Overviewing books such as Distributed Systems: Concepts and Design [31] or the Peer to Peer Handbook [90] present primarily the general architectural concepts. Other aspects (e.g. data model, techniques to adapt system dynamics). This emphasizes the need for challenge (A.1), an architectural classification, as preparation for a BAR tolerance evaluation.

2.2 CHALLENGE B: ROBUSTNESS EVALUATIONS OF DISTRIBUTED SYSTEMS

To quantify a system's robustness with regard to failures or the impact of selfish behaviour, theoretical analyses such as [92] are not completely satisfying. The modelling is challenging and possibly not appropriate for all evaluation metrics. A common alternative is benchmarking through experimental evaluations. However, basically all works in the literature considering robustness benchmarks focus on the benchmarking model and its specification. They address among others the specification of an appropriate fault-load (e.g. [93, 94]) and fault-injection (e.g. [95]). Evaluating works concern only specific aspects or circumstances. Examples are robustness evaluations of web service interfaces [96] or the routing in non-collaborative opportunistic networks [97]. An exception is the the recent work of Kim and

Anderson [98] who present an extensive robustness evaluation of a set of distributed systems. They examined 11 network types (random and hypergrid graphs, etc.) with three peer removal attacks (random, high-degree, high-centrality) and present more than 126 result graphs. This vast evaluation frames well the systems' robustness in terms of crashing peers or their communication links. However, it does not cover other systems or failures. Nevertheless, the complexity of this work emphasizes the sense of individually adapted evaluations. Industry-standard benchmarks such as the SPECjms2007[3] concentrate on performance related evaluations (see also [99]). Thus, robustness evaluations are typically done by means of network simulation environments (e.g. Omnet++[4]), which may provide some visualization capabilities. However, there is currently no work that eases robustness evaluations as a whole. Such approach should comprise among other simulation scenarios for several failures, measurement value aggregation as well as analysis. The lack of such an approach underlines the need for research challenge (B) to ease robustness evaluations with regard to selfish peers.

2.3 CHALLENGE C: TOLERATING SELFISH PEERS

We consider now the important challenge (C), i.e. achieving a targeted degree of collaboration from selfish peers in distributed systems. Due to the tremendous amount of works dealing with selfishness, this section highlights only the major approaches. They can be roughly distinguished in game theoretic and non game theoretic, each one with subdivisions, which is illustrated in figure 2.1. This separation represents also the structure of this section, while Inspection Games are discussed independently due to the similarity to the thesis vision & methodology.

Game Theoretic Approaches

The majority of works considering selfish peers are done in the context of game theoretic modeling. This frequent utilization of Game Theory (GT) can be explained by its definition: "*... the study of mathematical models of conflict and cooperation between intelligent rational decision-makers.*" [100]. Hence, GT is well-suited for the analysis of selfish peers' behaviour in collaborative environments, possibly forming complex situations with circular dependencies. Modern GT started in the context of economic analyses with von Neumann's and Morgensterns book [101]. Later on, the research community in this field achieved many classes of games (a recent overview is given in [102, 103, 100, 104]). GT found application in economics and other disciplines such as political sciences or biology. Shenker [105] discussed GT and selfishness in 1990 as one of the first for distributed systems. However, the scientific community picked it effectively up in the early 2000's, at first dealing with general challenges [106, 107, 108, 44, 4, 11].

CONTRIBUTIONS FOR SPECIFIC ISSUES The GT research community developed a set of works to face the problem of selfishness. Typically, they consider specific reliability and

[3] http://www.spec.org/jms2007/
[4] http://www.omnetpp.org

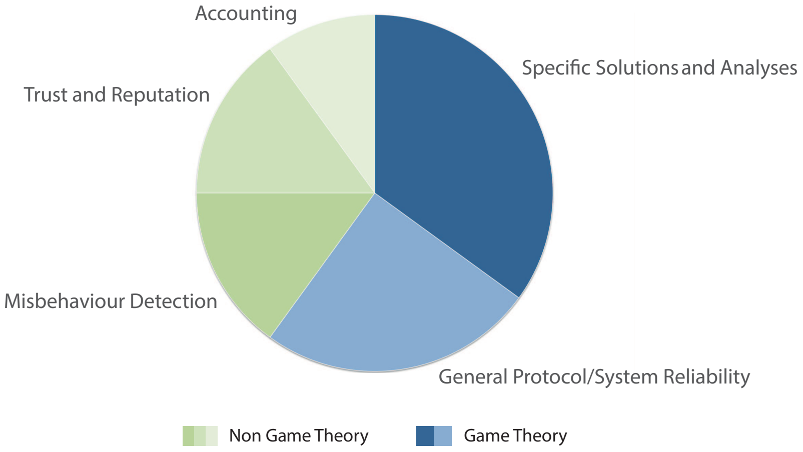


Figure 2.1: Outline of the relation of approaches suitable to face selfish peers in distributed systems. Note that this distribution outlines only the author's impression of the reviewed works.

security issues with regard to decision making in real-world situations. Several works address the exemplary scenario of the thesis, i.e. the correct packet forwarding in networks consisting of selfish peers. For example, Mei and Stefa introduce with Give2Get [10] two forwarding algorithms. Both protect cryptographically the packet transfer, which is separated in three phases: message generation, relay and test. With game-theoretical considerations they formally show that their protocols attain Nash equilibria and that no player has interest in violating, i.e. dropping messages.

Another interesting work is FlightPath [43], which addresses to reduce jitter[5] as possibility for selfishness-driven violations in the context of streaming applications. The authors examine approximate equilibria (instead of targeting strict Nash equilibria) for designing incentives to limit selfish behaviour. To this end, the message transfer between two peers are among others cryptographically secured (e.g. by encrypted promises). The FlightPath system is analyzed and evaluated used the ϵ-equilibrium[6]. The approximate equilibria approach provides some flexibility to handle dynamic situations such as node churn. The experimentations show a functioning in case of 10% malicious and 90% selfishly acting peers.

A further interesting work is the cartel maintenance framework Han et al. [109], which is based on repeated games. Here, participants of a cartel agree to some contract for a minimum degree of (average) collaboration and a punishment for non-collaboration. In the considered use case, each participant has a time slot for Media Access Control (MAC) of a wireless network. The collaboration degree is determined by observing the media access. If the collaboration falls below the agreed threshold, all noticing participants choose a non-collaboration strategy that reduces the players outcome of the game[7]. The cartel approach is an elegant way to attain a (minimum degree of) collaboration in a distributed way. However, the whole game mechanism relies on a shared resource (here: media access) for estimating collaboration

[5] In the context of internet streaming, jitter denotes the varying delivery delays such that the receiving packets do not arrive in time (too early or late). Hence, buffers are typically used as compensation.

[6] In such an equilibrium, rational selfish players deviate only if they expect to benefit by more than a factor of ϵ.

[7] Non-collaboration strategies are not further specified in the paper. An example by accessing the media during the violating peer's time slot.

and triggering punishments. Other violations (message drops, access spoofing etc.) require further modifications.

The approaches introduced here address packet forwarding (see also [10, 110, 111, 112]). Other works target further aspects such as routing [113, 114, 115], resource/bandwidth allocation [116, 117, 118, 119, 120, 121], multicast dissemination [122, 123], load-balancing [124], service orchestration [125, 126, 127] or wireless sensor networks [128, 129]. They cover several characterizations such as (non-)zero-sum, (non-)cooperative, Bayesian, differential, single-round or repeated games. Recent overviews to IT-related models and applications can be found in [130, 131, 132, 129, 103] and the references therein. Although numerous works are given in the literature, they represent only solution approaches for specific issues. Hence, they are not able to secure a whole collaborative protocol as addressed by challenge (C).

CONTRIBUTIONS FOR GENERAL PROTOCOL AND SYSTEM RELIABILITY Some contribution such as the one of Saleh and Debbabi [133] face the problem of selfish peers from a more comprehensive point of view. Their contribution is based on some prior work (e.g. [134]) and follows the idea of using games in logic specifications. This idea was initially proposed by Lorenz [135]. Saleh and Debbabi introduce a modeling framework of security protocols with concepts of game semantics. It models the specific protocol functionalities (e.g. digitally signing a message) as strategies over a game tree that represents the protocol. This approach enables a system designer to express a broad range of security properties (secrecy, authentication, fairness, etc.) and to specify and verify the resulting (cryptographic) protocols.

Proposing a general game-theoretic framework, Moscibroda et al. [136] study the efficiency reduction of a system consisting of selfish peers in case of malicious behaviour. The collaboration incentive is given by the impact of malicious peers, which reduce the selfish peers' utility. The introduce the term price of malice, which is *"... the ratio between the social welfare or performance achieved by a selfish system containing a number of malicious players and the social welfare achieved by an entirely selfish society."* Hence, in contrast to the price of anarchy the reference point is orthogonal. It is not a socially optimal welfare but the welfare of system consisting of selfish peers. This framework is applied to an abstract network graph playing a virus inoculation game (installing anti-virus software or not). The authors study the bounds of the price of malice and a *fear factor* in terms of utility loss through malicious peers.

These two approaches are examples for a large number of GT approaches that address general reliability issues. Similar works to secure collaborative network protocols are for example [137, 138, 139, 140] and several approaches for general reliability analysis of distributed systems are available (see for instance [141, 142, 143, 144, 145]). These works represent important contributions and can indeed be (partially) used as mechanism against selfish peers. However, they remain either protocols for specific issues or they represent too general analyses for concrete applications. Thus, they are interpret as not fully sufficient with regard to the thesis vision and research challenge (C) in particular.

Non Game Theoretic Approaches

Non game theoretically approaches use typically basic techniques such as redundancy and cryptography to increase the reliability/security of a system. For example, the work of Garg and Grosu [146] introduces a way to secure multicast dissemination systems that are based on the Shapley Value mechanism [147]. To this end, they implement signatures for message authentication and auditing/verification to detect cheating of the peers. As further example, Fotiou et al. [148] discuss security requirements, attacks and possible cryptographic solutions in rendezvous node based P2P systems. Such approaches are only capable to harden specific aspects of a system. Therefore, we concentrate now on three predominant approaches of the scientific literature that intends to face malicious actions, possibly caused by selfish behaviour. This is *misbehaviour detection*, *trust and reputation systems* and *accounting methods*.

MISBEHAVIOUR DETECTION Detection systems only focus on the detection of misbehaviours but also to make them known to other participants in the system. In a recent work of Serrat et al. [15] for example, so-called watchdogs at a peer can detect misbehaviours during interactions with the others. The watchdog is assumed to be collaborative to some degree, i.e. to disseminate information about selfish peers. In this context, the network is modelled with continuous time Markov chains to study the dissemination performances in terms of speed and message overhead. A work related to the vision of this thesis is the work of Cristea et al. [65]. They propose an architectural model, which combines monitoring with analysis and response mechanisms. The data are assumed to be continuously collected in real-time by an external monitoring system. The basic idea of the authors is the data analysis to detect failures by pattern matching and to predict failures, e.g. by using artificial intelligence (neural networks, moving average distributions etc.). Other works concern misbehaviour detection in terms of packet forwarding [149], backoff algorithms and media access [16, 150] in wireless networks or the detection of colluding attackers [151, 152]. More information can be found in the recent review of Raghuvanshi et al. [153]. The detection of misbehaviours is considered to be combined with further mechanisms such as failure-recovery or collaboration incentives (punishments, rewards etc.). In contrary to this concept, the thesis targets to prevent failues by giving collaboration incentives. Hence, the cannot directly serve as solution approach for research challenge (C) but could be an alternative or extension to monitoring techniques.

TRUST AND REPUTATION SYSTEMS A further approach against malicious attacks or selfish behaviour are Trust Management Systems (TMS) [154, 155], typically realizing reputation systems [156]. They aggregate and distribute feedback about the peers' behaviour to calculate reputation values. These quantify the peers' trustworthiness and serve as decision support in terms of collaboration. Thus, the collaboration is encouraged in a passive way. A peer can decide to work only with trusted peers, e.g. by denying requests from untrusted ones. A typical approach is the work of Shah and Pâris [157]. They enable trusted relations for the Bittorrent protocol by means of a local and a global score. The local score is determined by surveying the collaboration result (e.g. packet forwarding). The global score is maintained by tracker peers, which only assist in communication by providing complete membership information.

The global trust score is attached to each reply of a communication request and thus automatically disseminated to the peers in the network. The recent CAST system of Li et al. [158] introduces a context-aware approach for MANETs. By aggregating various contextual information (e.g. communication channel status, battery status), they are able to distinguish between malicious behaviour and faulty peers. With the CONFIDENT mechanism [159], Buchegger and Le Boudec introduce an extension to Dynamic Source Routing (DSR) protocols for the specific use case of correct packet forwarding. It consists of four major components[8]. They monitor the correct forwarding and exchange positive (collaboration) as well as negative (violations) feedback with other peers for a trust value calculation. Evaluations show a functioning of up to 60% malicious peers in the system. However, falsified misbehaviours (false positive) can be exchanged to blame other peers. This is solved by the similar CORE mechanism [160], which exchanges only positive feedback for the reputation calculation. Apart from that, it uses also surveillance and reputations to evaluate a peer's trustworthiness. Other approaches (see for instance [161, 162, 163, 164]) use similar mechanisms as those that were outlined here. Several surveys [165, 166, 167, 168, 169, 170] review trust and reputation systems.

Trust and reputation systems are indeed a promising approach with respect to challenge (C). This especially due to the loose coupling of the application logic and a given (passive) collaboration incentive. Reaching a targeted collaboration is generally imaginable but requires further research. Two major drawbacks are noticeable: The reputation is determined over time, possibly leading to a low reactivity, and the reputation is very simplistic (typically a collaboration or trust value). Hence, such approaches not able to take the complex decision landscape (possibly strategies over time) of selfish peers into account.

ACCOUNTING MECHANISMS Accounting methods model the interactions of peers within economic model, typically using some kind of virtual currency (other notions are credits, debts or tokens). An example is the PastryGrid system of Abbes et al. [171] in the context of grid systems. Each machine has a budget. The resource usage of other machines (in terms of computation or communication services) has a cost value and reduces the budget. Similarly, the budget can be increased by cooperating (selling own resources). Selfish behaviour, i.e. predominantly using the others' resources, is prevented by a limited budget. Being zero or a negative value, a peer must collaborate before using the others' resources. Further examples with economic models are [172, 173, 174, 123]. Accounting methods are an elegant way to reach collaboration and indeed practically used by applications such as Bittorrent clients. They are usually developed as built-in mechanisms, which hinders the adaptability and integration with other mechanisms. Furthermore, accounting mechanisms are not approaches for reliability from a comprehensive point of view. The focus typically on specific aspects relevant for a fair system functioning such as a up-/download ratio. They suitable for other threats possibly caused by selfish peers, e.g. attacks or malicious behaviour (e.g. content pollution, message drops). Thus, they are considered as not fully sufficient for challenge (C).

[8] The four components are a Monitor, a Reputation System, a Trust Manager, and a Path Manager. For further details, the interested reader is referred to the paper.

2.4 INSPECTION GAMES FOR RELIABLE DISTRIBUTED SYSTEMS

The IG was initially introduced Dresher in 1962 [175] in its simplest form: a two-player, non-zero-sum game. Further details to this class of games can be found in [176, 177]. Adaptations of the IG come under names such as Pursuit-Evasion [178] or Customs and Smuggler games [179]. The different variants can be seen as belonging to the wider class of Search Games (see [180]). In this context, Cheung et al. proposed in a recent survey of autonomous search models [181] a taxonomy structured around the three main components: the model of the searcher agent (the inspector), of the target agent (the inspectee) and of the environment. Similar to general GT, the majority of works are dedicated to IG model. For example, the inspectee can observe inspections and choose a strategy according to his self-interest [182]. In [183], the authors generalize Gale's Theorem, in which each of two players has N resources that can only be used once during N stages. Other works considering IGs are [184, 185, 186, 187, 188, 189, 190].

IGs were used for instance in industrial auditing for quality control and maintenance, tax inspection or crime control [191, 176]. Only few works are related to IT-related problems. For example, Chung et al. published recently [181] a survey of Search and Pursuit-Evasion Games in mobile robotics. They present among others fundamental results, discuss field implementations and highlight open problems. Another work of Antoniades et al. [178] targets on agent control, also formulated as Search and Pursuit-Evasion Game. In the context of digital surveillance by drones, the authors present heuristic strategies to detect and capture evaders.

Over the last years, research effort was spent to the IG models. Unfortunately, there is currently no work available with applications to communication infrastructures, ensuring the reliable functioning of communication protocols or research challenge (C) in particular. An adaptation of the basic IG model of Dresher is however reachable with some further effort. This consists basically in an extension by false negatives (non detection of misbehaviours) due to limited resources.

2.5 SUMMARY

This chapter introduced related work with regard to the thesis' research challenges. The given works related to challenge (A), a BAR tolerance evaluation, draw a clear picture. Although a multitude of works addresses reliability, evaluations or corresponding surveys, there is no work providing a comprehensive evaluation of BAR tolerance capabilities. Given works concentrate only on evaluation models/concepts, focus only on specific aspects (e.g. link/node crash) and lack in the consideration of selfishness. In addition, there are still some ambiguities in the interpretation of terminology and architecture.

The situation is similar for challenge (B), evaluating robustness capabilities of distributed systems. The given works discuss benchmarking models (e.g. fault injection) or evaluate the robustness only for specific circumstances. Only one recent contribution evaluates by a vast simulative study the capabilities of 11 network types – however, only in terms of link/node crash (with three types though). This shows the need for an approach that eases simulative robustness studies as a whole, which represents challenge (B).

Numerous works dealed with selfishness in distributed system, which is related to research challenge (C). The related work was discussed by separating game theoretic and other apraoches. The majority of works to contribute by of GT models for specific application scenarios. Only few GT works have a broader point of view as needed for challenge (C) to secure a whole collaborative protocol. The broad utilization of GT is comprehensible since this discipline deals by definition with models of cooperation between intelligent rational players. The discussed non GT works represent three predominant types of approaches: misbehaviour detection, trust and reputation as well as accounting systems. The amount of works dealing with selfishness is tremendous. However, there are several drawbacks with regard to the thesis' vision. The game models are not directly applicable, they are too specific or provide too general analyses. Hence, the given works are not suitable for the thesis' monitoring approach and not satisfying for challenge (C). In contrary to other game models, Inspection Games found only little application in distributed systems and a directly suitable work is not available. Nevertheless, the general game model fits well to the thesis' methodology and will be used, after some needed adaptation, as solution approach for challenge (C).

3 A SURVEY OF BAR TOLERANCE IN DISTRIBUTED SYSTEMS

In this chapter, we review the BAR tolerance capabilities of a set of systems as representatives for distributed systems. As discussed in the last chapter, a comprehensive comparison is hindered due to several reasons. Hence, some preparations for the evaluation are introduced beforehand, which represent at the same time the evaluation structure.

3.1 STRUCTURE FOR COMPREHENSIVE EVALUATION

The preparations for a BAR tolerance evaluation consist of an architectural classification and a taxonomy of elementary failures of a peer. They will be detailed in the following.

3.1.1 *Architectural Classification Scheme*

The classification scheme consists of nine architectural architectural dimensions, which are described in the following. They are clearly arranged in table 3.1 providing example categories for each dimension. For illustration reasons, these nine dimensions are exemplary applied to a sample of four systems in table 3.2. To design these architectural dimensions of the classification, previous surveys as well as other overviewing works around pub/sub such as [90, 192, 193] were aggregated. The following criteria were used:

1. The dimensions should not overlap.
2. The dimensions should be sufficient to fully classify a set of 25 works of the literature that considered as important.
3. The dimensions should completely cover all options for the 25 selected works, that are critical for the functioning of a system .

SUBSCRIPTION MODEL The subscription model enables the specification of interest. Common models are topic-, content- [91] and type-based [194], which mainly differ in their expressiveness. Topics represent only whole multicast groups by means of a unique identifier, while content-based models classify event messages not by an attribute but rather by the content itself. This enables a more fine grained specification of interest up to single messages at the price of higher resource usage. Type-based subscription models are rather related to (object-oriented) application developers and can easily be deployed on top of a content-based model. An event represents an object related to a specific type. Hence, it can implement attributes and methods, which brings improvements such as type-safety. Other examples are XML- [195] or context-based [192] subscription models. More details can be found in [41].

Table 3.1: Overview of the Publish/Subscribe architectural classification scheme.

Architectural Dimension	Example Categories
Subscription Model	- topic-based - content-based - type-based - context-based
Event Data Model	- serialized - tagged - untagged binary
Matching Algorithm	- Predicate indexing, e.g. - lookup tables - Hanson et al.'s algorithm [196] - Testing network, e.g. - matching trees - binary decision diagrams
Overlay Organization	- hierarchical - P2P - structured P2P (S-P2P) - unstructured P2P (U-P2P) - P2P broker overlay (BO) - clustered
Dissemination Technique	- subscription/event flooding - selective - selective filtering - rendezvous node based - basic/informed gossiping
Adaptation Technique	- active - passive
Communication Technique	- RPC - socket connection - web services
Underlay Awareness	- proximity-awareness - quality-awareness
Quality of Service	- reliability (guaranteed delivery) - delivery semantics, e.g. best-effort or exactly once - latency/bandwidth guarantees

EVENT DATA MODEL The event data model specifies the representation of event messages in the system. In systems with a high rate of published events, the data model can cause significant delays due to the amount of messages to be processed. The event data model can be classified into three general types. *Tagged* formats such as XML structure information hierarchically into fields that are accessed by an identifier (tag). On the contrary, *untagged binary* formats (e.g. IP or TCP header) provide a fixed structure, which makes processing efficient as lookup costs are constant. A comparison of tagged and untagged binary format has been done by The Gryphon Team [35]. *Serialized* formats such as the Java serialization format can represent complex information such as objects in object-oriented programming languages. However, they are computationally intensive compared to simple event messages.

MATCHING ALGORITHM Matching algorithms are used to identify the subscribers that are interested in a given event (*subscription matching*). They gain in importance in large-scale systems as the amount of subscriptions and/or processed events increase(s). Then, similar to event data processing, the subscription matching may require a significant amount of time. Subscription matching algorithms can be classified into two main groups [197], predicate indexing algorithms and testing network algorithms. Predicate indexing algorithms divide subscriptions into their elemental constraints, which are then used to identify a matching event message. Testing network algorithms use subscriptions to create efficient data structure for the matching process [198]. Examples are matching trees [199, 35] and binary decision diagrams. A formal comparison and analysis of matching algorithms is given in [200].

OVERLAY ORGANIZATION This specifies the organization of the logical overlay infrastructure. It includes among others the choice of communication partners and the group management policy. Thus, it has a strong correlation with the dissemination technique. Three common types are *hierarchical* systems, which follow the client/server principle, *P2P* systems with a flat hierarchy and *clustering* approaches. The hierarchical organization (e.g. in form of a tree) enables logarithmic hop counts on the logical overlay. P2P systems can be further divided into structured and unstructured systems [90]. Structured Peer-to-Peer (S-P2P) overlays use a deterministic approach to set up a static dissemination structure. In contrast, Unstructured Peer-to-Peer (U-P2P) overlays use non-deterministic techniques such as random walks create a dynamic dissemination structure. Clustered overlay organization defines independent domains of administrative control. They are typically used to increase performance or scalability. Another organization option are Broker Overlays (BO) [194, 35], which distinguish between peers that publish/subscribe (clients) and forward (servers). The latter type of peers are also called brokers. Client peers have exactly one broker as their access point to the overlay network (which is in turn realized by brokers). BOs can be interpret as a hybrid between a one-level-hierarchical and a P2P overlay. Therefore, they are classified here as subdivision of P2P overlays by aggregating the clients' interests into the peer (figure 3.1).

DISSEMINATION TECHNIQUE The dissemination technique specifies the routing of messages in the overlay network. It is a crucial factor of the scalability of the system. *Flooding* algorithms forward a received message to all known peers and we distinguish between *event flooding* and *subscription flooding*. *Selective algorithms* make use of a deterministic routing process. In *selective/filtering*, the subscription matching is done in a decentral way: a peer forwards events only to those outgoing links that have (matching) subscribers behind it. In

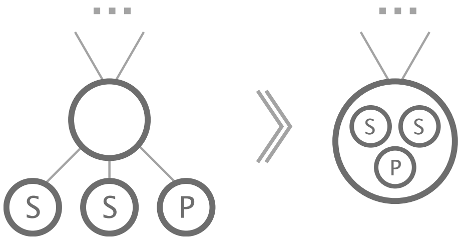


Figure 3.1: Transformation of broker overlays to P2P overlays by aggregating interests of clients.

selective/rendezvous, a specific peer (the rendezvous node or r-node) is responsible for subscription matching of an event. It is typically chosen by applying a mathematical function such as a hash functions on parameters of the event (e.g. topic identifier). All events of an event group are sent to the rendezvous node, which is aware of subscribers and forwards the event to them. Thus, a dissemination tree is realized, also known as DHT-based approach [90]. *Basic gossiping* follows the principle of epidemic algorithms [201], in which events are forwarded to a set of randomly chosen overlay links. In *informed gossiping*, this set is partially defined in a deterministic way, e.g. to form a logical ring for a guaranteed delivery.

ADAPTATION TECHNIQUE Two methods are possible to enable a system's adaption to topological changes. Here, we focus only on failure situation, omitting the case where a peers logs off correctly. The *active* way checks the state of another peer by sending periodically a heartbeat message. If the recipient fails to respond within a specified time window, it is considered as having left the system – the system must thus be adapted to this new state. With the *passive* method, a peer identifies the need for adaptation reactively when observing a deviation from standard behaviour. Common triggers are timeouts or exceeded thresholds.

COMMUNICATION TECHNIQUE This dimension defines how the communication is technically performed. It does not affect the overlay structure but may have significant influence on its characteristics such as performance or interoperability. Three common types are mentioned here. *Socket connections* are a platform-independent possibility for data exchange. They enable efficient communication by reducing the communication overhead, while the message format is not specified. A Remote Procedure Call (RPC) enables function calls in distributed environments, e.g. the event notification call at a peer. Many RPC implementations are available, which are not necessarily compatible to each other. Furthermore, *web service*s can, similar to RPC, request launching a function on a remote peer. They are standardized and enable the integration of heterogeneous programming languages and systems.

UNDERLAY AWARENESS An underlay-aware system has knowledge about the underlying physical network, which allows an overlay optimization. The underlay-awareness can be classified as *proximity-based* (e.g. hop distance, knowledge about subnets/clustering) and *quality-based* (e.g. bandwidth usage, round-trip-time). These types are not mutually exclusive and a system may combine several approaches.

QUALITY OF SERVICE Quality of Service (QoS) mechanisms target to provide guarantees for specific quality aspects, which is however hardened by the loosely coupled communication. Therefore, the operation is often done in a best-effort mode. Research efforts aimed at extending pub/sub systems to make QoS guarantees possible[71, 37]. Mahambra et al. have proposed a QoS taxonomy for event-based middleware in [70], consisting of *guarantees for latency (L), bandwidth (B), reliability (R), delivery semantics (DS)* and *message ordering (MO).* Due to space constrains, the abbreviations indicated in brackets are used for classifications in tabular form such as table 3.2. Note that in this example only R is indicated since no other QoS mechanisms are implemented.

Table 3.2: Exemplary classification of three pub/sub systems and the dissemination system basic gossiping (A=active, P=passive, R=reliability, NM=not mentioned, (..)=implem.-dependent).

Name	Subscription Model	Event Data Model	Matching Algorithm	Overlay Organisation	Dissemination Technique	Adaptation Technique	Communication Technique	Undelay Awareness	Quality of Service
Scribe [2]	topic	NM (tagged)	lookup table	S-P2P	selective (rendevous)	A	*NM*	–	R
Gryphon [35]	content	untagged binary	predicate matching	BO/S-P2P	selective (filtering)	P	*NM*	prox.	R
SpiderCast [34]	topics	*NM*	*NM*	U-P2P	gossiping (informed)	A	*NM*	–	R
Basic Gossiping *	–	–	–	U-P2P	gossiping (basic)	P	*NM*	–	R

*Basic gossiping is a pure dissemination technique and may be used to realized Publish/Subscribe systems. Thus, only general characteristics are given. Note that the classification of all dimensions depends on the specific implementation.

3.1.2 *Classifying Scribe and Gossiping*

The two systems that are used during the thesis – Scribe and (basic) gossiping – are part of the exemplary classification in table 3.2. The pub/sub system Scribe realizes a deterministic tree-based dissemination style on the network overlay, using the location-based routing service Pastry [202]. Hence, it is classified as a selective dissemination system with a rendezvous node and as S-P2P in terms of overlay organization. Scribe uses topics and for subscription matching, maintained by a subscription table. The topic id is represented by a hash value (SHA-1) of the textual topic name and the creator's name. TCP is used as message protocol, however, the actual event data model is kept implementation specific. Here and in the remainder, we assume commonly tagged formats such as XML or JSON. Although relying on TCP, Scribe implements (through Pastry) with periodic heartbeats an active adaption technique. Together with retransmissions in case of message losses (through TCP), the system provides QoS only to some degree in terms of reliability. Finally, specific communication techniques are not mentioned and Scribe provides no underlay awareness.

Since basic gossiping is not a pub/sub but a pure dissemination system, the corresponding dimension are blanked out with "–" in table 3.2. Nevertheless, to make it comparable to Scribe, we assume the same values (i.e. topic-based, tagged and a lookup table). Furthermore, it realizes a clique network and differs thus to Scribe in terms of the system architecture. By realizing a non-deterministic structure based on randomness it is classified as U-P2P. Gossiping provides a passive adaptation techniques based on timeouts.

3.1.3 *Taxonomy of Byzantine Failures*

A taxonomy of Byzantine failures is introduced as further preparation for the review. The taxonomy is listed below is distinguishes between operation and message anomalies. To support evaluations from a comprehensive point of view, the taxonomy consists of elementary failures, which may in turn lead to more enhanced failure scenarios.

- *Operation Anomalies*
  - *Link/Node unavailability*: A communication link or peer is temporarily unavailable and can no longer interact with the system.
  - *Link/Node crash*: A communication link or the whole peer crashes (with all links). It does not interact anymore with the system and needs to be restarted.
- *Message Anomalies*
  - *Delay*: The forwarding of a message is delayed, the message is delivered later than expected.
  - *Information Leak*: The message header and/or body are fully readable allow to draw conclusions as preparation for an attack.
  - *Loss*: A message is lost while being delivered to subscribers (e.g. while being transmitted through a lossy link or at the peer between transmissions).
  - *Content Pollution*: A message of arbitrary content (or type) is sent to a set of overlay neighbours, polluting the actual information in the system.
  - *Misusage*: Protocol coherent messages are used in an arbitrary way (e.g. repeated sending), possibly disturbing the system function.
  - *Tampering*: The message (header and/or body) is modified while being delivered but remains protocol coherent.

Information leaks are not directly a danger for the reliability but important with respect to privacy and security. The gained knowledge can be used to design a selfish strategy. Therefore, it should be prevented in addition to the other failures. *Message Misusage* is manifold and kept abstract here. An example is (in combination with Message Tampering) the sending of a falsified DNS response to get control over the address translation for a set of peers. These failures can cause more complex failure situations; some examples are the following:

- *Catastrophic Failures*: A specific failure appears simultaneously at a large number of peers. This may for example arrive for the failures Link/Node Crash, Message Loss or Message Tampering.
- *Node Churn Attacks*: Node churn represents actually the typical dynamics of a distributed system, i.e. joining/leaving peers. This can be exploited for system attacks by producing unnaturally high churn rates by continuously (mis-)using leave/join messages of a group of peers. Then, the system functioning is affected due to the extra resources and time needed for disseminating changes in the network.
- *Network Partitioning*: The overlay network gets split into independent networks due to crashes of a communication link or the whole peer. The events originating in a partition can no longer be transmitted to the others.
- *Message Ordering*: Messages are not received in the same order as they were sent.

3.2 BAR TOLERANCE CAPABILITIES OF PUBLISH/SUBSCRIBE SYSTEMS

After introducing the preparations, we will review some works to evaluate their BAR tolerance capabilities. To this end, the evaluation model is described in the beginning. The evaluation is then presented and finally its meaning for system development discussed.

3.2.1 *The Simple Comprehensive Reliability Evaluation Model*

The Simple Comprehensive Reliability Evaluation (sCRE) model enables a qualitative reliability evaluation considering the architectural classification and failures based on the taxonomy. This allows to draw conclusions between the system architecture and reliability aspects. Evaluations with the sCRE model are user-centric, i.e. the failures or architectural dimensions can be freely chosen. Nevertheless, the architectural dimensions *Dissemination Technique* and *Overlay Organization* should generally be considered due to their critical role in the functioning of the system. In addition, we also consider here the *Subscriptions Type*. The failures of the taxonomy of failures are adopted with the following modifications. First, the failure *Message Delays* is not considered since resulting failure scenarios (e.g. *Message Ordering*) are typically not affecting the system's general functioning. In the context of the video streaming scenario, we assume some buffers to compensate this failure. Moreover, as link and node crashes have a similar influence on the system (a node crash implies the crash of all of its links), they are considered as one failure type. The dimensions used by the sCRE model are the following, which are shown on the right-hand side in the evaluation table 3.3.

- Failure A: Link/Node Crash
- Failure B: Message Loss
- Failure C: Message Tampering
- Failure D: Content Pollution
- Failure E: Message Misuse
- Failure F: Information Leak
- Failure G: Selfish Behaviour

The evaluation with the sCRE model is done in a qualitative way and comprises "✓" (yes) and "✗" (no) as possible values. A "✓" indicates the presence of techniques against the regarding type of failure but not does not necessarily guarantee that the impact is completely avoided/tolerated. The symbol "✗" is used if mechanisms for the corresponding failure are not implemented or not mentioned. This qualitative evaluation does not reveal the robustness capabilities but enable a visualization of the systems' reliability capabilities (i.e. considered failures during system design) in a clearly arranged way. Hence, it contributes to determine the current state-of-the-art by revealing reliability issues in terms of the thesis' vision.

3.2.2 *A Qualitative Reliability Evaluation*

Several systems are evaluated that were proposed over the last ten years. The studied sample is composed of 20 pub/sub systems and 5 pure dissemination algorithms, selected based on their popularity (estimated from their number of citations), date of publication (more recent works were favored) and thematic representativeness. The results are presented in table 3.3.

Clear conclusions can be drawn from the evaluation. Almost all systems implement at best techniques against the common elementary failures link/node crash and message loss. Techniques against selfish behaviour are in general not considered. The only exception is FlightPath, a dissemination algorithm for streaming applications, thanks to its implementation of the BAR model. However, a closer look reveals a number of clear limitations. FlightPath supports only one streaming source per multicast group and uses a *tracker* that is responsible for subscription matching (each of both, source and tracker, is assumed to be deployed on a dedicated peer). The message transfers are appropriately secured, however, the behaviour of the tracker peer is not. The reliability for the multicast groups can no longer be guaranteed if the corresponding tracker itself is malicious or selfish.

The evaluation reveals a dramatic situation, which is interpret as in the following. Distributed systems provide some inherent reliability capabilities (depending on the architecture). Further functionalities, e.g. to tolerate failure type C (Message Tampering) or F (Information Leak) are considered as optional and implemented on demand. However, some techniques or failure types such as E (Message Misuse) or G (Selfish Behaviour) require deep architectural modifications. Thus, the consideration of reliability techniques as optional modules is not feasible in general. Furthermore, especially in complex systems, the interaction of the several implemented techniques can be hard to determine. This emphasizes the need for an approach that is abstract enough to cover the complexity but precise enough to take into account the specific techniques or selfish goals.

3.3 MEANING FOR THE RESEARCH CHALLENGES

The BAR tolerance evaluation developed before addresses the research challenge (A) of the thesis with the architectural classification related to (A.1) and the taxonomy of failures related to (A.2). In the thesis' research context, the evaluation (and the related work) showed that selfishness in distributed system is only slightly considered by some recent systems[1]. It showed furthermore how reliability mechanisms are typically considered for distributed systems. Basic capabilities are a result of the system architecture. In contrary, the lack of reliability is usually interpret (taking approaches of the related work discussion into account) as the consideration of mechanisms as optional modules during system design. However, this is not always feasible due to possibly deep architectural modifications. Several mechanisms in such complex systems can even interfere each other (e.g. redundancy and QoS). This emphasizes the need for more comprehensive reliability approaches as for example the one described in the thesis' vision.

[1] The system FlightPath [43] is based on preceding developments.

Table 3.3: The evaluation shows BAR tolerance capabilities of examined systems.

System Name	**Subscription Type**	**Dissemination Technique**	**Overlay Organisation**	***Failure A*** **Link/Node Crash**	***Failure B*** **Message Loss**	***Failure C*** **Message Tampering**	***Failure D*** **Content Pollution**	***Failure E*** **Message Misuse**	***Failure F*** **Information Leak**	***Failure G*** **Selfish Behaviour**
Pub/Sub Systems										
SIENA [91]	Content	Selective (Filtering)	Hierarchical & S-P2P	✗	✗	✗	✗	✗	✗	✗
Scribe [2]	Topic	Selective (Rendezvous)	S-P2P	✓	✗	✗	✗	✗	✗	✗
Bayeux [203]	Topic	Selective (Rendezvous)	S-P2P	✓	✓	✗	✗	✗	✗	✗[6]
Hermes [194]	Type	Selective (Rendezvous)	BO/S-P2P	✓	✗	✗	✗	✗	✗	✗
Narada Brokering [199]	Topic [4]	Selective (Filtering)	S-P2P (Clustered)	✓	✗	✗	✗	✗	✗	✗[5]
Gryphon [35]	Content	Selective (Filtering)	BO/S-P2P	✓	✓	✗	✗	✗	✗	✗
IndiQoS [37]	Type	Selective (Rendezvous)	BO/S-P2P	✓	✓	✗	✗	✗	✗	✗
Sub-2-Sub [32]	Content	Gossiping (Informed)	U-P2P (Clustered)	✓	✓	✗	✗	✗	✗	✗
REDS [204]	Content	Selective (Filtering)	BO/S-P2P	✓	✗	✗	✗	✗	✗	✗
Anceaume et al. [205]	Content	Sel. (Filtering) & Gossiping[14]	Hierarchical & U-P2P[14]	✓	✓[14]	✗	✗	✗	✗	✗
Baldoni et al. [206] [1]	Content	Selective (Filtering)	S-P2P	✓	✗	✗	✗	✗	✗	✗
TERA [33]	Topic	Selective (Filtering) [7]	U-P2P (Clustered)	✓	✓[7]	✗	✗	✗	✗	✗
SpiderCast [34]	Topic	Gossiping (Informed)	U-P2P	✓	✓	✗	✗	✗	✗	✗
F-A P/S [207]	Content	Selective (Filtering)	S-P2P	✗[10]	✗[10]	✗	✗	✗	✗	✗
CAPS [208] [2]	Content	Selective (Rendezvous)	S-P2P	✗	✗	✗	✗	✗	✗	✗
Jafarpour et al. [209]	Content	Selective (Filtering)	BO/S-P2P (Clustered)	✓	✓	✗	✗	✗	✗	✗
Boonma and Suzuki [210]	Topic	Selective (Filtering)	S-P2P	✓	✓[9]	✗	✗	✗	✗	✗
Esposito et al. [36] [3]	Topic	Sel. (Rendezv.) & IP Multicast[8]	S-P2P (Clustered)	✓	✓	✗	✗	✗	✗	✗
LIPSIN [211]	Topic	Selective (Filtering)	BO/S-P2P	✓	✓	✗	✗	✗	✗	✗
Kazemzadeh and Jacobsen [212]	Content	Selective (Filtering)	S-P2P	✓	✓	✗	✗	✗	✗	✗
Pure Dissemination Systems										
Peercast [62]	–	Selective (Filtering)	Hierarchical	✓	✓	✗	✗	✗	✗	✗
CREW [213]	–	Gossiping	U-P2P	✓	✓	✗	✗	✗	✗	✗
Voulgaris and van Steen [214]	–	Gossiping (Informed)	U-P2P	✓	✓	✗	✗	✗	✗	✗
FlightPath [43]	–	Gossiping	U-P2P	✓	✓[11]	✓	✓[12]	✓	✓	✓[13]
Schroeter et al. et al. [215]	–	Flooding & Gossiping (Inf.)	BO/S-P2P	✓	✓	✗	✗	✗	✗	✗

[1] Based on SIENA. [2] Based on Chord. [3] Based on Scribe. [4] Supports also JMS, JXTA and Xpath queries. [5] Monitors quality information (e.g. bandwidth) and provides a security module, both can be used as base for anti-rational-techniques. [6] Four protocols are introduced; the used one provides quality-based underlay awareness. [7] It is distinguished between outer and inner-cluster dissemination can be choosen by the administrator. [8] IP Multicast is used as inner-cluster technique to increase dissemination efficiency. [9] QoS capabilities enable to retransmit messages in case of failures. between transfers is not revealed. [12] A *tracker* manages only one subscriber per multicast group. Signed messages avoid non authorized message generation. [13] FlightPath uses proofs of misbehaviour (PoM) and punishments.
[14] This work compares a leader-based against gossip-like epidemic algorithm (may provide redundant path).

3.4 SUMMARY

This chapter evaluated BAR tolerance capabilities of a sample of current distributed systems. To this end, 20 pub/sub systems and 5 pure dissemination systems were reviewed. To enable a comprehensive evaluation, some preparations were introduced beforehand: an architectural classification scheme and a taxonomy of elementary failures of a peer. It turned out that almost no system tolerates more failures than link/node crash and message loss in the proposed form. One exception is the system FlightPath [43]. It considers the BAR concept during system design but has still important drawbacks. The scientific community started to address the problem of selfishness-driven misbehaviours. Typically, reliability techniques are proposed as optional modules (e.g. encrypted communication), which is not always reasonable or possible. An interesting less-invasive yet powerful alternative is the monitoring approach introduced in this thesis.

Part III

PRACTICALLY EVALUATING ROBUSTNESS OF PUB/SUB SYSTEMS

Addressing research challenge (B), this part introduces the RCourse benchmarking library. It serves as mean for practical evaluation of a system's capabilities to tolerate the impact of selfishness-driven violations.

Overview

4 RCOURSE: ROBUSTNESS STUDIES WITH THE PEERSIM SIMULATOR

This chapter introduces the RCourse library for the Peersim simulation environment. It targets to simplify simulative studies in terms of robustness of performance. Furthermore, RCourse enables to enrich the evaluation of the last chapter with quantifying information. Before detailing RCourse, two pub/sub algorithms are outlined. They are used for all exemplary simulations in this chapter and for the evaluations of the IG approach of part III.

4.1 A SHORT EXCURSUS TO PUB/SUB SYSTEMS

Two pub/sub systems are detailed in the following that will be used in the remainder of the thesis and in the exemplary RCourse experimentation in particular. This is Scribe and a system based on basic gossiping. They differ significantly in their dissemination style. In short, Scribe realizes an efficient dissemination but lacks in fault-tolerance mechanism. In contrary, gossiping has higher system inherent fault-tolerance capabilities but is less efficient due to some redundancy. This is illustrated in figure 4.1.

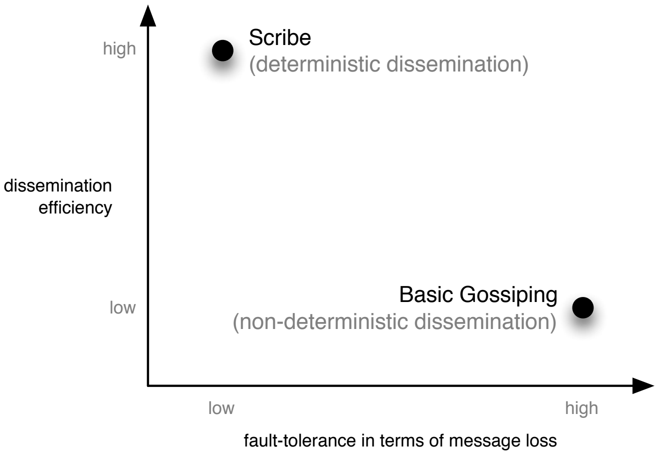
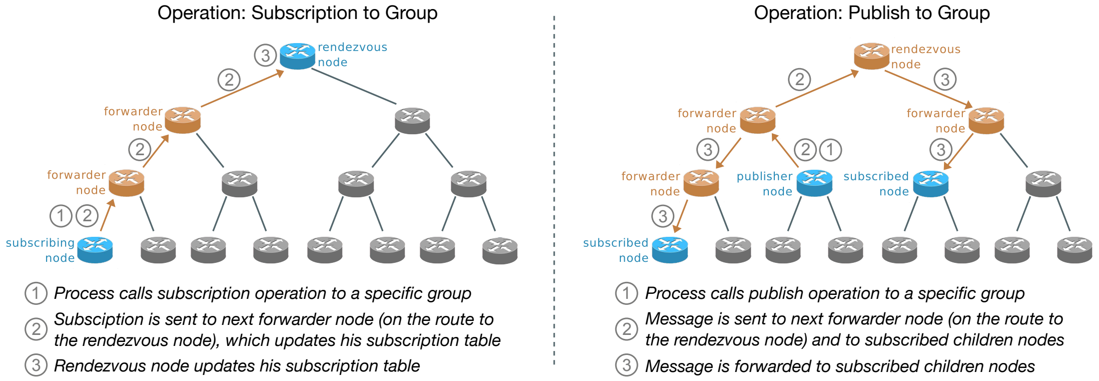

high
Scribe
(deterministic dissemination)
dissemination efficiency
low
Basic Gossiping
(non-deterministic dissemination)
low
high
fault-tolerance in terms of message loss

Figure 4.1: The two pub/sub algorithms are oppositional in terms of reliability and efficiency.

4.1.1 *Scribe*

The Scribe system [2] is one of the first pub/sub systems and realizes topic-based application layer multicast. The routing on overlay level is done by the id-based routing service Pastry [202], realizing a dissemination with $O(logN)$ forwarding steps to the target peer. Scribe is characterized by a deterministic structure, i.e. it uses deterministic algorithms to calculate the next hop on the dissemination path. This deterministic structure leads to a tree-based dissemination approach, which is also denoted as *rendezvous node* – or *r-node* – based approach.


Operation: Subscription to Group

1 Process calls subscription operation to a specific group

2 Subsciption is sent to next forwarder node (on the route to the rendezvous node), which updates his subscription table

3 Rendezvous node updates his subscription table

Operation: Publish to Group

1 Process calls publish operation to a specific group

2 Message is sent to next forwarder node (on the route to the rendezvous node) and to subscribed children nodes

3 Message is forwarded to subscribed children nodes

Figure 4.2: The Scribe system has a tree-based dissemination style. Two operations are exemplary illustrated for a multicast group: the subscription (left) and the publishing (right).

Here, publisher and subscriber send their messages to the r-node, making it to the root node of a tree for one multicast group. Thus, the r-node is responsible for subscription matching in last instance. A peer holds implicitly the role of a rendezvous node for a specific multicast group if his overlay id is equal or the closest one to an identifier, which is typically a hash value of the topic descriptor of the multicast group. Figure 4.2 shows the dissemination principle of the Scribe system exemplary for two operations, the subscription and publishing. If a peer on the route to the r-node receives a subscription, he adds the subscribing peer to his subscription list. Afterwards, he subscribes himself to the group. If a peer on the route to the r-node receives a published event message, it its further transferred to the r-node but also to interested peers in the subscription list (if any). The functioning of the Scribe system should only be outlined here; the reader is referred to [2] for further specific details.

Scribe has been chosen due to its efficient dissemination technique. In other words, it has only limited fault-tolerance capabilities (only link/node crashes) and no further redundancy techniques are implemented. Therefore, the impact of faults is directly shown without being moderated by fault-tolerance techniques.

4.1.2 *Gossiping*

Gossiping is an information dissemination protocol, which relies on the fundamental manner of epidemic algorithms [201, 216]. It is characterized by a non-deterministic structure, which makes use of some randomness during dissemination (dynamic dissemination structure). Hence, the message delivery can only probabilistically be guaranteed.

Current approaches (e.g. [34, 32, 214]) have only a *partial view* on the network containing possible candidates for a message transfer. It is maintained by membership protocols, which provide a *peer sampling service* [217, 218]). In an optimal case, all partial views should form a connected random graph. Hence, membership protocols are a critical component for the dissemination quality several approaches were developed in the in recent years. Examples are Cyclon [219], Scamp [220], Vicinity [221], HyParView [222] and X-Bot [223]. Typical partial view sizes are about 20-30, although the size has only limited impact system attributes

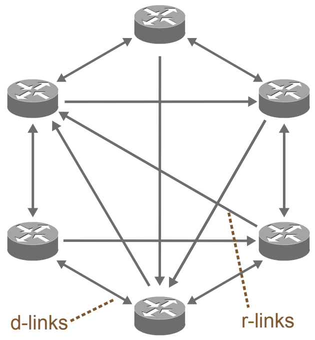


Figure 4.3: Informed gossiping realizes a deterministic dissemination ring based on the overlay ids. Further random transfers (r-links) improve the dissemination speed.

[214, 224]. The partial view typically holds additional information such as the subscriptions of a peer, which are exchanged during the procedure of the membership protocol.

The dissemination differs mainly in the way how these peers are chosen from the partial view for a message transfer. Two general types are basic and deterministic gossiping. In *basic gossiping*, the peers are randomly selected for the message transfer. In contrary, *informed gossiping* combines the probabilistic manner with a deterministic structure. Typically, the dissemination is done to randomly chosen peers (r-links) and to deterministically chosen peers (d-links), illustrated in figure 4.3. The RingCast approach [214] uses for example the d-links to create a dissemination ring (to guarantee a delivery), while r-links speed up the delivery.

Two parameters (see also [90]) are crucial for the performance of gossiping systems:

- *Fanout*: The fanout value defines the amount of peers, which are chosen from the partial view as target for the dissemination of a message. Typical fanout values are about 3-6 for informed gossiping and 9-10 in basic gossiping. Then, a delivery is probabilistically guaranteed [214]. A probabilistic calculation of the delivery is given in [224].

- *Maximum Rounds:* This value defines, similarly to the Time-To-Live (TTL) value in IP-networks, how long a message shall circulate in the network. Therefore, it means a trade-off between resource usage and delivery probability.

The random dissemination of basic gossiping results in a homogeneous load distribution. Furthermore, the redundancy in dissemination creates inherent fault-tolerance capabilities. Due to these reasons, basic gossiping is used in the remainder of the thesis for all experimentations. Cyclon [219] is used as membership protocol due to its frequent usage in research works.

4.2 UNDERLYING RESEARCH PRINCIPLES FOR PRACTICAL EVALUATIONS

In contrast to the BAR tolerance evaluation, a system designer must – due to the given diversity – concentrate on a subset of selfishness-driven violations that shall be considered. To this end, he chooses the corresponding reliability mechanisms, which are parametrized with regard to the designers' objectives, i.e. the targeted collaboration level. This is challenging since the mechanisms may interact each other and the given systems have different inherent reliability capabilities. Hence, in order to develop BAR tolerance mechanisms, a system designer must examine the system behaviour in presence of failure situations and reliability mechanisms. A typical way to do this are simulative studies.

In this context and to reach the main research challenge, a design principle was mentioned in the introduction (see section 1.4, page 11). A mutual feedback should be considered between the practical system behaviour evaluation and the theoretical model of reliability mechanisms. In other words, a simulation determines the impact of failures or selfishness-driven violations on the system behaviour and enables to adjust the mechanism parameters correspondingly. By easing simulative studies, RCourse represents the practical side of this design principle. To be more specific, the practical side of this design principle must enable the system designer to perform the following tasks:

- Evaluate the isolated impact of failures caused by specific selfishness-driven violations on the system behaviour considering or not reliability mechanisms.
- Evaluate the system behaviour under user specific circumstances (e.g. dynamics) considering or not reliability mechanisms.

The system designer uses simulations as a tool within the development process of BAR tolerance mechanisms. The following *design principles* are formulated to meet the designer's practical needs in terms of the main research challenge.

1. *Simplicity of utilization*
   Performing simulations shall be simplified as much as possible to provide time and work gains. Ideally, they should be able to be launched once the peer logic is implemented.
2. *Isolated failure evaluation*
   The impact of specific failures (comprising also selfish behaviour) on the system behaviour shall be evaluated independently from others.
3. *Adaptability to individual needs*
   Due to the diversity of selfishness-driven violations, a system designer shall be able to easily adapt the simulation to individual needs.

Several simulation environments are available to meet these principles, each one providing individual characteristics; examples are Omnet++[1], PlanetLab[2], ns-3[3] or NetSim[4]. The Peersim simulator[5] [225] has been developed for the study of overlay networks in particular. Since pub-/sub realizes overlay networks, it is a good option for the evaluations in this thesis. Peersim omits the simulation of the actual network stack and is as open-source software available for free usage. However, there is currently no extension for statistical result aggregation and analysis available for Peersim. Thus, performing network simulations requires additional work to obtain practically usable results. Furthermore, no benchmarking application for Peersim specifically targeting robustness is available. RCourse serves as an approach to fill this gap.

[1] http://www.omnetpp.org/
[2] https://www.planet-lab.org/
[3] http://www.nsnam.org/
[4] http://tetcos.com/
[5] http://peersim.sourceforge.net/

4.3 OVERVIEW TO RCOURSE

Enriching the simulation environment Peersim, RCourse contributes to meet research challenge (B) and the design principles stated before. Therefore, RCourse targets to simplify simulative studies with a special focus on robustness and performance aspects. Although it is especially prepared for studies on pub/sub systems (for example, it comes with several pre-implemented pub/sub algorithms), it is designed to be easily adaptable to evaluations of P2P systems in general. RCourse is, as Peersim, freely available under the GPLv2 open-source license and downloadable at the Sourceforge project page[6].

4.3.1 *Design Goals*

The development of RCourse strictly considered the following four design goals, which are directly based on the principles mentioned before.

1. *Time/work savings*
   As many tasks as possible shall be automated in order to spare work/time to the user.

2. *Architectural modularity*
   Each step of a simulative study shall be adaptable to specific user needs.

3. *Stand-alone simulations in terms of robustness and performance*
   The RCourse library shall enable complete simulative studies in terms of robustness and performance without requiring other software products or further calculations.

4. *Free tool for research community*
   RCourse shall be designed as a free tool and available under an open-source license.

Figure 4.4 illustrates the simulation workflow with the supported steps by RCourse (continuous lines) as well as the possibilities for individual adaptations (dashed lines). RCourse enables time/work savings (goal 1.) already beginning with the first workflow step – the development of the dissemination algorithms that shall be simulated. By means of a set of Java classes, RCourse provides among others functionalities for data aggregation and write-out. Thus, the user can concentrate on the development of the algorithm itself, all other steps may be taken over by RCourse components. Further support is given for example by means of pre-defined simulation scenarios and scripts in the R programming language. They analyze the simulation result data set and automatically generate result graphs as pdf files. The design goal 2. was reached by modularizing the RCourse components among the workflow steps. This loose coupling results in a modular architecture and facilitates the adaptation of the different steps to user specific needs. In order to fulfill goal 3., the RCourse library is complete in the sense that a simulative study can be directly launched once the algorithm development is completed. Indeed, two pub/sub systems have been developed for use with RCourse, which are included in the library. These are Scribe and gossiping that were described before. RCourse provides a basic as well as informed gossiping implementation, based on the membership protocols Cyclon [219] and also HyParView [222]. RCourse is primarily targeting the computer networks

[6] http://rcourse.sourceforge.net/

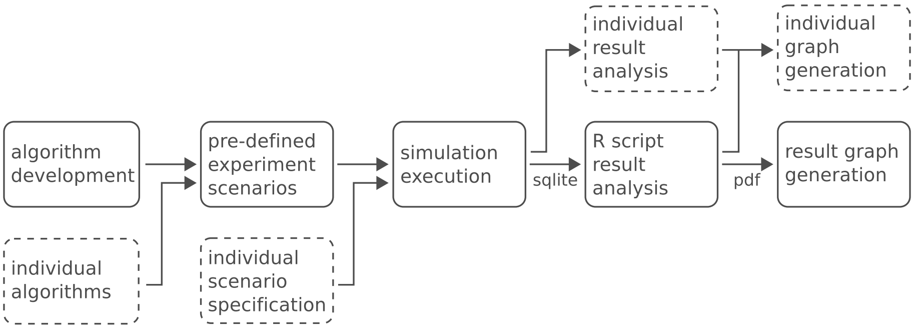

algorithm development
individual algorithms
pre-defined experiment scenarios
individual scenario specification
simulation execution
sqlite
individual result analysis
R script result analysis
pdf
individual graph generation
result graph generation

Figure 4.4: RCourse supports the user in the workflow of simulative studies by performing up to all steps after the algorithm development. Each step can be individually adapted (dashed lines).

research community. Thus, in order to fulfill design goal 4., it makes only use of free software – indeed, this is only Peersim and the programming languages Java as well as R – and is itself published as open source under the GPLv2 license. As mentioned before, the RCourse library as well as additional files (scenario/graph overview, example result files etc.) are available for download on the RCourse project homepage.

4.3.2 *RCourse Components*

RCourse is realized as a loose coupling of basically three components:

- *A set of Java classes* support the user among others in value aggregation and write out as SQLite result files.
- *Pre-defined simulation scenarios* can be directly used to simulate and evaluate a system after finishing the development of the dissemination algorithm.
- *Analysis scripts* (written in the R programming language) aggregate the SQLite result files, analyze the data and generate result graphs as pdf files.

A simulation that makes full use of these components is completely reduced to the development of the dissemination algorithm. For a given algorithm, the development concerns basically the implementation of a measurement value aggregation and write-out. Simulations can then be executed using the pre-defined scenarios and result graphs created by means of R analysis scripts. These components will now be explained in more detail.

JAVA CLASSES The Java classes of RCourse mainly provide six essential components. They are indicated in figure 4.5 and outlined in the following.

- The class *RCParamStorage* holds simulation-wide configuration parameters as well as other global values and is accessible by the peers.
- *RCTrafficGenerator* and *RCTurbulenceGenerator* are helper classes for the simulation: *RCTrafficGenerator* generates an appropriate workload and *RCTurbulenceGenerator* creates different stressful situations such as crashing nodes and node churn.

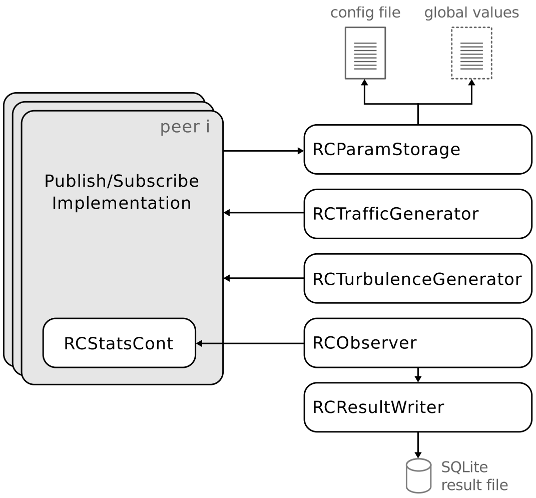


Figure 4.5: RCourse Java library consists of six main classes.

- The class *RCStatsCont* represents the container for measurement values. It must be maintained by each simulated peer in order to record measurement values.

- The *RCObserver* plays a key role in RCourse. It collects at specified time points all measurement containers and passes them to the *RCResultWriter*.

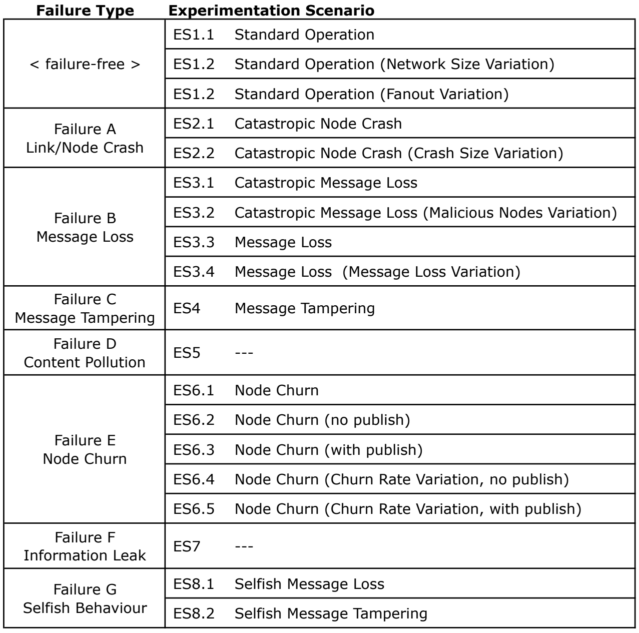
- The *RCResultWriter* class extends *RCResultWriterBase* (not shown in figure 4.5) and generates the SQLite result file (as well as the csv file output if defined in the configuration).

To make a Peersim algorithm compatible with RCourse, a user needs to maintain the *RCStatsCont* and *RCParamStorage* objects. The interaction with the other classes such as *RCTrafficGenerator* is realized through a Java interface (that also needs to be implemented) and specified in the configuration files. For more details, please refer to the reference implementations of RCourse, which are available on the project's website.

PRE-DEFINED SIMULATION SCENARIOS In order to enable quickly launched simulations, the RCourse provides a set of simulation scenarios as Peersim configuration files. For each considered failure type, they consist of one or more *Experimentation Scenarios*. Each such scenario enables the automated generation of some result graphs. The term simulation scenario is also used as synonym to Experimentation Scenario in the remainder of the thesis. An overview of these scenarios is shown in table 4.1. Appendix A (see table A.1, page 127) details these scenarios by providing all possible result graphs of RCourse in list form. The scenarios are based on the failure types of the BAR tolerance evaluation. However, the two failures D (Content Pollution) and F (Information Leak) are ignored. The reason is that they can easily be avoided by techniques such as access control or cryptography mechanisms. The scenarios are simplistic to some degree to enable drawing conclusions from the specific failure. For representative results, the use of real world workload and failure traces is recommended, e.g. those of the Failure Trace Archive[7]. This is currently not yet supported by

[7] http://fta.inria.fr/apache2-default/pmwiki/index.php

Table 4.1: The pre-defined scenarios of RCourse base on the taxonomy of failures introduced before.



Failure Type	Experimentation Scenario	
< failure-free >	ES1.1	Standard Operation
	ES1.2	Standard Operation (Network Size Variation)
	ES1.2	Standard Operation (Fanout Variation)
Failure A Link/Node Crash	ES2.1	Catastropic Node Crash
	ES2.2	Catastropic Node Crash (Crash Size Variation)
Failure B Message Loss	ES3.1	Catastropic Message Loss
	ES3.2	Catastropic Message Loss (Malicious Nodes Variation)
	ES3.3	Message Loss
	ES3.4	Message Loss (Message Loss Variation)
Failure C Message Tampering	ES4	Message Tampering
Failure D Content Pollution	ES5	---
Failure E Node Churn	ES6.1	Node Churn
	ES6.2	Node Churn (no publish)
	ES6.3	Node Churn (with publish)
	ES6.4	Node Churn (Churn Rate Variation, no publish)
	ES6.5	Node Churn (Churn Rate Variation, with publish)
Failure F Information Leak	ES7	---
Failure G Selfish Behaviour	ES8.1	Selfish Message Loss
	ES8.2	Selfish Message Tampering

RCourse but an implementation is facilitated thanks to the architectural modularity: only the *RCTrafficGenerator* as well as *RCTurbulenceGenerator* need to be adapted.

R ANALYSIS SCRIPTS RCourse provides scripts in the programming language R to process the simulation result analysis. This concerns basically the read-in of SQLite data files for an automatic graph generation. All analysis scripts have been developed corresponding to the pre-defined scenarios. One script initiates the analysis for each scenario[8], while several files are responsible for the graph generation. The analysis is be structured into four general phases, which are shown[9] in figure 4.6. In phase (I), the user initiates the analysis for a simulation scenario by calling the corresponding analysis script, which invokes further script files. Phase (II) is dedicated to the accomplishment of two important tasks. First, the simulation result files are read-in. Second, further information such as variance or standard variation are calculated for all numeric values. These information are then directly used for the graph generation in phase (III). As an example, the script for the Experimentation Scenario ES3.2 considering a Loss Rate Variation (LRV) in terms of message loss is shown in the appendix in listing A.1 (page 123). After phase (III), all result graphs are available and can directly be used for presentation purposes. As an impression, a sample result graphs are shown in the next section in figure 4.9 (page 50).

4.4 SIMULATIVE EVALUATIONS WITH RCOURSE AND PEERSIM

After an outline, we focus now on the practical utilization to launch simulations.

[8] The scripts have the naming scheme *diRgrams_*.r*, with '*' being replaced for a specific scenario.

[9] Please note that this figure is a simplified visualization to convey the logical process.

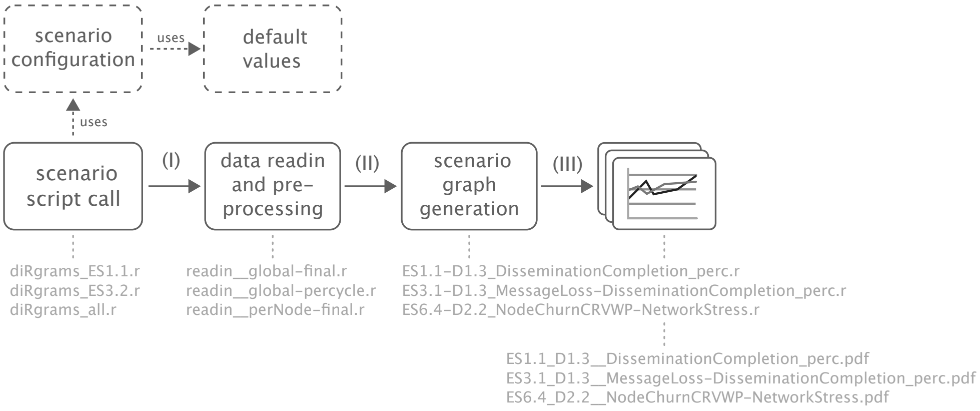
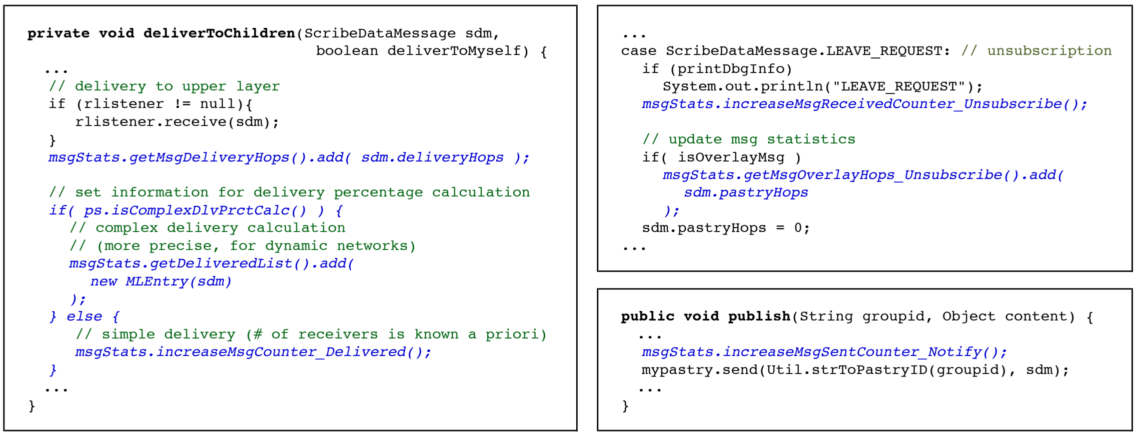

scenario configuration
uses
default values
uses
scenario script call
(I)
data readin and pre-processing
(II)
scenario graph generation
(III)
diRgrams_ES1.1.r
diRgrams_ES3.2.r
diRgrams_all.r
readin__global-final.r
readin__global-percycle.r
readin__perNode-final.r
ES1.1-D1.3_DisseminationCompletion_perc.r
ES3.1-D1.3_MessageLoss-DisseminationCompletion_perc.r
ES6.4-D2.2_NodeChurnCRVWP-NetworkStress.r
ES1.1_D1.3__DisseminationCompletion_perc.pdf
ES3.1_D1.3__MessageLoss-DisseminationCompletion_perc.pdf
ES6.4_D2.2__NodeChurnCRVWP-NetworkStress.pdf

Figure 4.6: The result analysis with R scripts is organized in 3 phases: the analysis scripts are started in phase (I), the result files are processed in phase (II) and pdf result graphs are generated in phase (III).



```
private void deliverToChildren(ScribeDataMessage sdm,
                               boolean deliverToMyself) {
  ...
  // delivery to upper layer
  if (rlistener != null){
     rlistener.receive(sdm);
  }
  msgStats.getMsgDeliveryHops().add( sdm.deliveryHops );

  // set information for delivery percentage calculation
  if( ps.isComplexDlvPrctCalc() ) {
    // complex delivery calculation
    // (more precise, for dynamic networks)
    msgStats.getDeliveredList().add(
      new MLEntry(sdm)
    );
  } else {
     // simple delivery (# of receivers is known a priori)
     msgStats.increaseMsgCounter_Delivered();
  }
  ...
}
```

```
...
case ScribeDataMessage.LEAVE_REQUEST: // unsubscription
  if (printDbgInfo)
    System.out.println("LEAVE_REQUEST");
  msgStats.increaseMsgReceivedCounter_Unsubscribe();

  // update msg statistics
  if( isOverlayMsg )
    msgStats.getMsgOverlayHops_Unsubscribe().add(
      sdm.pastryHops
    );
  sdm.pastryHops = 0;
...
```

```
public void publish(String groupid, Object content) {
  ...
  msgStats.increaseMsgSentCounter_Notify();
  mypastry.send(Util.strToPastryID(groupid), sdm);
  ...
}
```

Figure 4.7: Three examples show the utilization of RCourse components in the source code. The related code is indicated in blue/italic.

4.4.1 *Utilization and Experimentation*

The setup and utilization of RCourse is now detailed among all steps of the simulation workflow. This is done by means of the scenario *ES3.2 Message Loss* with *Loss Rate Variation*, which corresponds to the example scenario of the introduction.

PREPARATION: SETTING UP THE LIBRARY The RCourse library consists of three folders, each one providing the content for a RCourse component. The Java classes are stored in folder *src/*, the scenario definitions in *config/* and the R analysis scripts in folder *analysis/*. The Java classes need only to be referenced in the Peersim project and the scenario definitions are used by specifying them when launching a simulation. Similarly, the analysis scripts are used by calling the appropriate one within the R user interface. Thus, a typical RCourse library setup may consist of just copying the RCourse folder into the simulation project folder and calling/referencing the files when needed.

ALGORITHM DEVELOPMENT A crucial task is the source code extension to record measurement values. To this end, each peer must maintain a *RCStatsCont* container object and implement all method calls for value aggregation. Figure 4.7 shows the interaction with the data container for three examples with the Scribe system, indicating the RCourse related code in blue/italic. The shortest example (bottom-right in figure) records the amount of published messages and the medium one (up-right) the actions when an unsubscribe message is received. This is the counter of received messages and a counter for overlay hops. The biggest example (left in figure) shows the recording of the message dissemination time in terms of overlay hops as well as the delivery completion as a percentage. Please note that RCourse provides two types of calculation for the latter value. The *simple* method uses the configuration parameters to calculate the number of packets that should be delivered. Here, the message publishing must start after a subscription phase (plus some stabilization time) to provide correct values. The *complex* method calculates delivery completion based on the current situation of subscriptions and published messages in the system. Therefore, it is slower and computationally more intensive but also more suitable for dynamic workload situations such as node churn. The three examples only outline the interaction with RCourse in the Java source code. For more information, the reader is referred to the source codes of the exemplary applications to the Scribe and gossiping systems (see project website indicated in footnote before).

CONFIGURING AND PERFORMING SIMULATIONS The utilization of RCourse and Peersim is shown for the Eclipse Integrated Development Environment (IDE) in figure 4.8. In general, three information must be specified that are listed below. The configuration files contain the actual configuration and are shown in appendix A in listing A.2 (*rcourse_Base.txt*) on page 124 and listing A.3 (*ES3.2-MessageLossLRV.txt*) on page 125.

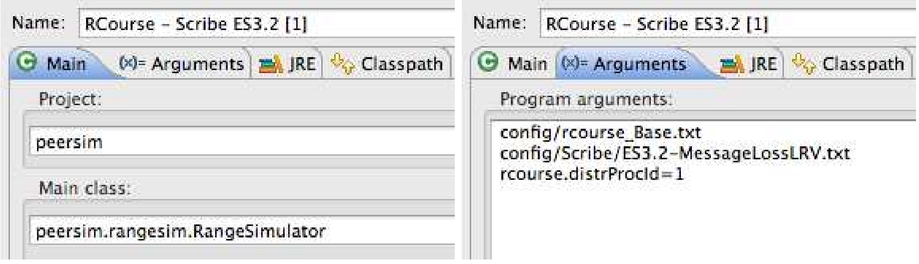
- *config/rcourse_Base.txt*
  This configuration file defines the general network setup and the structure of each peer's stack of protocols.

- *config/Scribe/ES3.2-MessageLossLRV.txt*
  The second config file represents the scenario and comprises all scenario-related parameters such as time steps for the observer execution or workload generation.

- $rcourse.distrProcId = 1$
  The *distrProcId* parameter defines the id for the current simulation in case of parallel processing. See section 4.4.2 for more information.

Important are the terms *simulation run* and *experiment repetition*. A simulation run means the execution of one simulation with a specific set of parameters. An experiment repetition represents the execution of a simulation scenario as a whole, which may consist of multiple simulation runs. The amount of simulation runs is automatically calculated based on the configuration file (experiment repetitions times the amount of parameter sets caused by possibly specified value ranges). Each run is processed independently from the others. Furthermore, a generated SQLite file encapsulates all results of one experiment repetition.

To launch simulations, RCourse must be executed with *peersim.rangesim.RangeSimulator* as main class instead of the default class *peersim.Simulator*. This is due to the fact that several



Name: RCourse - Scribe ES3.2 [1]

Project:
peersim

Main class:
peersim.rangesim.RangeSimulator

Name: RCourse - Scribe ES3.2 [1]

Program arguments:

```
config/rcourse_Base.txt
config/Scribe/ES3.2-MessageLossLRV.txt
rcourse.distrProcId=1
```

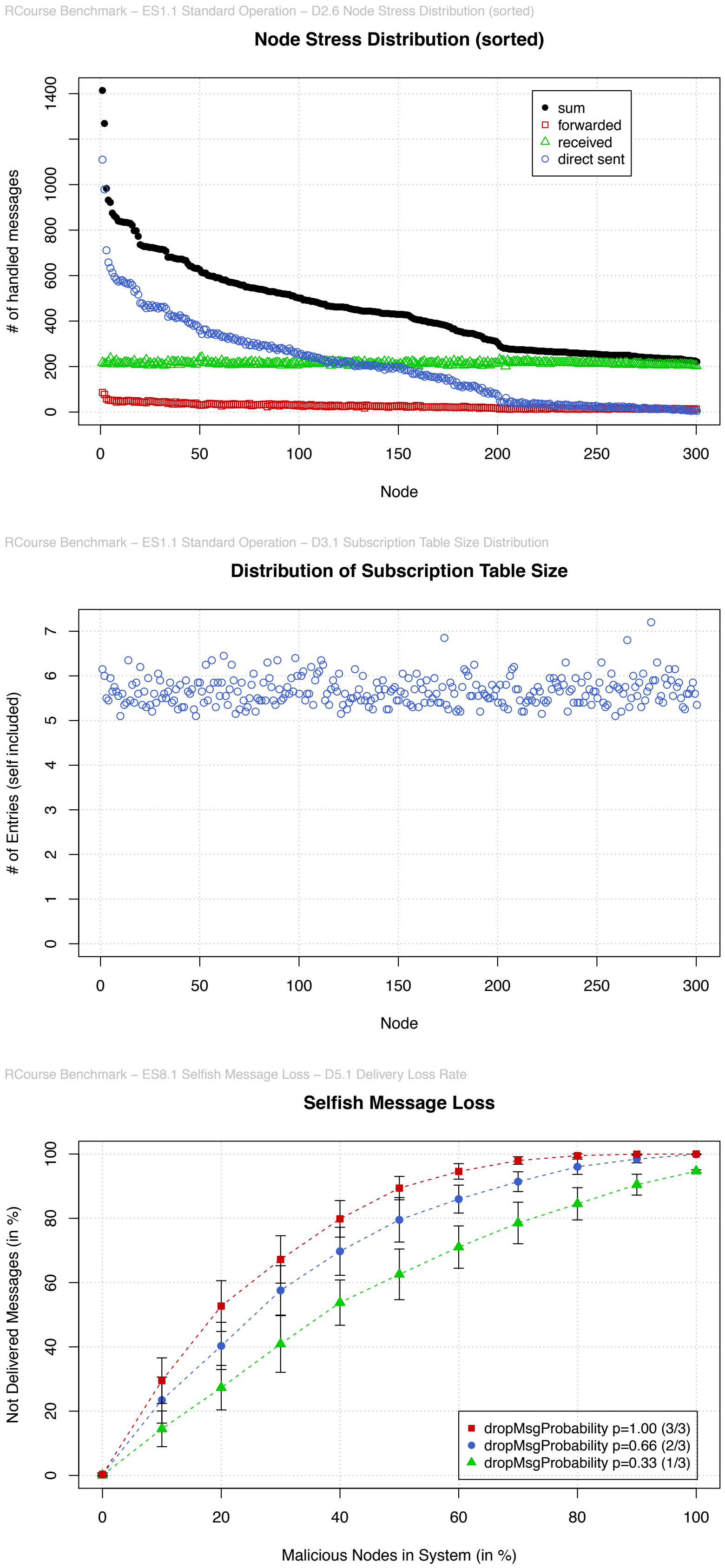
Figure 4.8: If using the Eclipse IDE for running simulations, Experimentation Scenarios may be defined as presets as shown here for ES3.2.

scenarios use value ranges for specific parameters, which is not supported by the default class. The *RangeSimulator* offers this feature for the configuration file. For example, the entry *range.malNodeP rcourse.malNodeProb;0:1|0.1* means that the value of *range.malNodeP* varies from 0 to 1 with steps of 0.1. Then, the *RangeSimulator* performs 11 simulation runs, each one with a different *range.malNodeP* value.

DATA ANALYSIS AND GRAPH GENERATION To start the analysis of simulation results, scenario scripts need to be called (stored in folder *analysis/*). Each of them is related to an Experimentation Scenario and responsible for the generation of all related graphs. Scenario scripts have the naming scheme *diRgrams_*.r*, with "*" being replaced by the corresponding scenario abbreviation. For example, the script file *diRgrams_ES3.2.r* is related to Experimentation Scenario ES3.2. Scenario scripts make use of further scenario configurations, that are also prepared as R scripts (stored in folder *analysis/config/*). Furthermore, some default values being defined in *analysis/util/default_values.r.* These files should be checked before starting the analysis process. As example, the file *diRgrams_ES3.2.r* is shown in listing A.4 on page 126. After a successful execution, several result graphs are generated as pdf files. Figure 4.9 shows three examples, which have been generated with the Scribe system [2]. These graphs serve only as an impression; graphs for all scenarios (Scribe and basic gossiping) are available on the RCourse project page. The upper graph in figure 4.9 is related to the prior discussed example scenario ES3.2, showing the graph D5.1 Delivery Loss Rate with the amount of malicious nodes in the systems on the x-axis and the delivery loss on the y-axis.

4.4.2 *A Note on Multi-Process Simulations*

Modern processors have multiple cores and/or central processing units (CPU), a feature that could be used for simulations by implementing parallel processing. However, Peersim only allows the single-threaded execution of a simulation. RCourse enables parallel processing through a parallel execution of independent simulation runs. This can even take place on different machines by means of parallelizing experiment repetitions. This workaround makes sense since a simulation should be repeated several times to achieve representative results. Let us assume for instance that we have five cores at our disposal and a simulation that should be repeated ten times, i.e. ten overall experiment repetitions. For the sake of simplicity, it is assumed that each experiment repetition consists of only one simulation run. Then, each core



RCourse Benchmark – ES1.1 Standard Operation – D2.6 Node Stress Distribution (sorted)

Node Stress Distribution (sorted)

RCourse Benchmark – ES1.1 Standard Operation – D3.1 Subscription Table Size Distribution

Distribution of Subscription Table Size

RCourse Benchmark – ES8.1 Selfish Message Loss – D5.1 Delivery Loss Rate

Selfish Message Loss

Figure 4.9: Some example graphs are shown for the Scribe [2] system, which have been generated with RCourse. This is the (sorted) node stress distribution (upper graph), the distribution of subscription table entries (middle graph) and message loss due to selfishness-driven message drops (bottom graph).

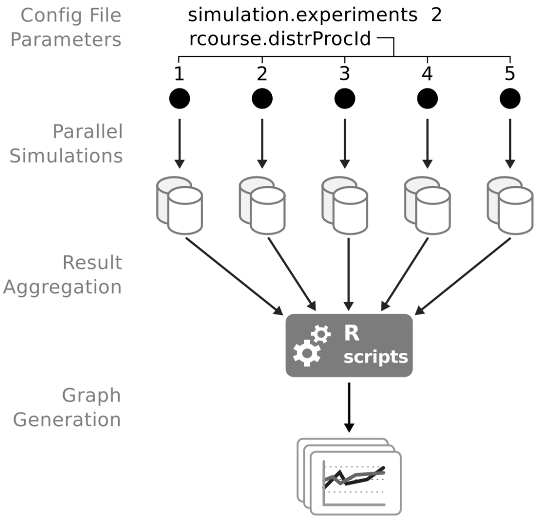


Figure 4.10: Peersim supports only single-threaded simulations. However, RCourse enables a parallel simulation to some degree and an automated result aggregation.

could execute sequentially two simulations. This is realized by setting the *rcourse.distrProcId* and *simulation.experiments* parameters in the configuration file. The first parameter only affects the name of the result file (important for R analysis scripts). Thus, each experiment repetition should have a unique value. The parameter *simulation.experiments* corresponds to a parameter that defines the number of experiment repetitions for the corresponding execution of Peersim. This results in parallel processing on all cores and in the generation of ten SQLite result files. They are then aggregated by the R scripts and used for result graph generation. The functioning of this multi-process example is shown in figure 4.10.

4.4.3 *Adaptation to Individual Measurements*

RCourse records already a wide range of measurement values, which are clearly arranged in the file *RCResultWriter.java*. In addition, it is specifically designed to be easily adaptable to individual values. To this end, one may extend the *RCStatsCont* data container. Then, only two components of RCourse have to be modified to let it record a new value. First, the *RCStatsCont* class must be extended by the new value to be recorded. This may also include appropriate methods to add data to the container object. Please note that the *RCStatsCont.mergeStats(...)* method may have to be modified as well depending on the value type. It is called by the *RCObserver* to merge the statistical values of all peers into one data container with global values (e.g. summarized counter values). Hence, it must be adapted if the new value considers network-wide values such as the average dissemination delay or even more complex data structures (linked lists, vectors etc.). The second component to modify is the *RCResultWriter* class, which is responsible for writing out all values. The required modification consists in adding the new value as an additional database column. Finally, the scripts for read-in, data analysis and result graph generation must be adapted to make use of the new value(s).

4.5 MEANING FOR THE RESEARCH CHALLENGES

RCourse meets research challenge (B) by simplifying each step in the simulation workflow. This eases simulative studies with a focus on fault-tolerance issues. Providing simulation scenarios related to specific failure types, robustness evaluations can be quickly launched. This is now illustrated by a short evaluation. As before, we consider again the simulation scenario *ES3.2 Catastrophic Message Loss*. It corresponds to the video streaming scenario with selfishness-driven message drops mentioned in the introduction. The results are shown in figure 4.11 for Scribe (upper graph) and basic gossiping (lower graph). Message loss indicates the violation probability, e.g. 0.8 means to collaborate only for 0.2 (20%). The lack of Scribe's fault-tolerance results in a fast decrease of the system functioning. In contrary, the redundant dissemination of gossiping can tolerate the failures to some degree. This changes at a loss rate of about 60% where the delivered messages decrease dramatically. This short study illustrates the benefit of RCourse to reveal the individual robustness capabilities. Furthermore, it shows the need for individual targeted collaboration levels, as defined for challenge (C).

RCourse contributes also to challenge (A) thanks to the pre-defined scenarios that base on the BAR tolerance evaluation of the last chapter. It enables to enrich the still qualitative evaluation with quantitative information. This is exemplary shown in table 4.2 using the

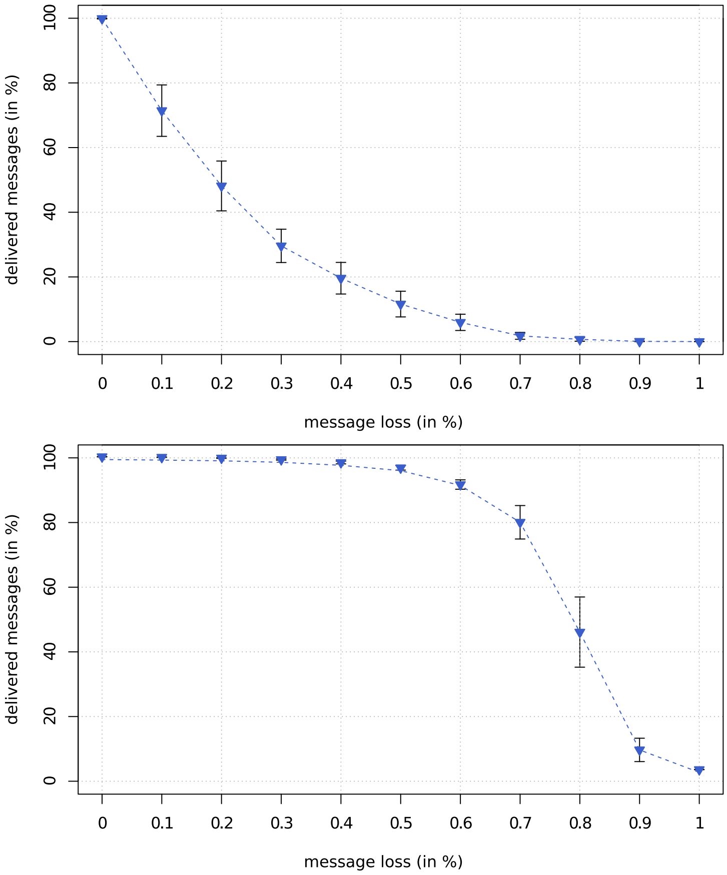


Figure 4.11: The graphs show evaluation results of selfishness-driven message drops of Scribe (upper graph) and basic gossiping (lower graph). Latter one used a fanout value of 10.

Table 4.2: RCourse enables to enrich BAR tolerance evaluations by quantitative values. This is shown (with regard to the prior simulative study) for violation probabilities 0.1 and 0.3.

System Name	**Dissemination Technique**	***Failure A*** **Link/Node Crash**	***Failure B*** **Message Loss**
Scribe [2]	Selective (Rendezvous)	✓ 0.1: 75% 0.3: 30%	✗ 0.1: 75% 0.3: 30%
–	Basic Gossiping	✓ 0.1: 100% 0.3: 98%	✓ 0.1: 100% 0.3: 98%

evaluations of figure 4.11. A selfish peer dropping messages could hide behind the Byzantine failures link/node crash and message loss and the table shows the system functioning in case of 10% and 30% lost messages, i.e. violation probability 0.1 and 0.3 respectively.

4.6 SUMMARY

This chapter introduced the RCourse library that meets research challenge (B). After a short outline, its components were described as well as the utilization to perform simulative studies. Two pub/sub algorithms were detailed in the beginning of this chapter – Scribe and basic gossiping. They were used for all exemplary result graphs and will also be used in the remainder of the thesis. The RCourse library targets to support the user in performing simulative studies of distributed systems' robustness and performance. By focusing only on the logic implementation – the rest is take over by RCourse – the user is able to quickly achieve simulation results.

Part IV

ACHIEVING COLLABORATION WITH INSPECTION GAMES

This part is dedicated to the solution approach for research challenge (C), i.e. enabling a system deployment over selfish peers. To this end, a basic two-player Inspection Game (IG) is at first adapted to the needs of distributed systems. Then, some practical issues for an application to distributed systems are discussed in chapter 6. A solution approach for challenge (C) is introduced in chapter 7, which is enhanced in chapter 8 by the capability to adapt to system dynamics. Finally, we discuss the real world applicability of the IG approach.

OVERVIEW

* Subsubsections of type "5.4.1 (...)" are not listed here. See table of contents for a full overview.

5 ADAPTING INSPECTION GAMES TO THE NEEDS OF DISTRIBUTED SYSTEMS

The utilization of Inspection Games (IG) as solution approach for distributed systems was argued in the introduction (see section 1.3 in particular). However, the related work discussion demonstrated that there is currently no appropriate IG model available. Therefore, this issue is addressed now by adapting the basic IG to our needs. To this end, the considerations comprise the extension by false negatives (non-detected violations) as well as generalizations up to games with m inspectors and n inspectees. All derivations are based on the basic two-player IG, which is introduced in the beginning.

5.1 A BASIC TWO-PLAYER INSPECTION GAME

The IG is now detailed in its initial form as introduced by Dresher [175]: a two-player zero-sum game. It is denoted as $G(1,1)$ in the remainder. An *Inspection Game* (IG) consists of two players: the *inspectee* and the *inspector*. A *strategy* for a player is a complete plan of actions throughout the game. The goal of every player consists in adopting a strategy that maximizes his own payoff, which represents the outcome of the game. Each player takes into account that it depends also upon the other players' chosen strategy. The *Nash equilibrium* (NE) is a solution that describes a steady state condition of the game. It corresponds to a combination of strategies, a *strategy profile*, such that no individual player would be better off by changing his own strategy unilaterally.

Game Setup

The inspector's set of strategies is $\{\, Inspect\, ,\, Do\ not\ inspect\, \}$, i.e. he can choose between inspecting or not. Analogue, the inspectee represents the selfish peer and can choose between violating or not with the strategy set $\{\, Violate\, ,\, Do\ not\ violate\, \}$. The players have to choose their strategy simultaneously (or equivalently such that they do not have any hint about the other's move before their own move). With real world distributed systems in mind, we do not use zero-sum payoffs but use more realistic ones. To this end, we assume for the sake of simplicity the following:

1. The case without violation and without inspection does not bring any damage nor benefit to any player.
2. Violation will bring the inspectee a positive benefit b if not detected but, if detected, it will bring him also a loss $-a$ with $|a| > |b|$.
3. The inspection has a fixed cost $-c$ for the inspector, but not detecting a violation would cost him a damage $-d$ with $|d| > |c|$.

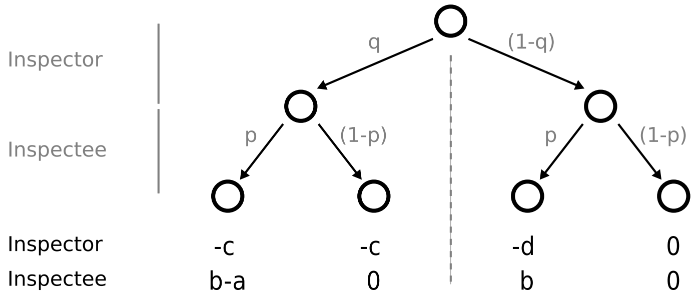


Figure 5.1: The extensive form for an IG with one inspector and one inspectee.

Table 5.1 indicates the four possible states of affairs and the corresponding payoffs for the players. In each cell, the pair (x, y) means that the first player (the inspectee) obtains a payoff x, while the second player (the inspector) obtains a payoff y. The table represents the IG in the so called *normal form*: the rows correspond to the possible moves of the inspectee (the inspectee's pure strategies), the columns correspond to the possible moves of the inspector (the inspector's pure strategies). The structure of the game can also be represented in *extensive form* as shown in figure 5.1.

Since $b > 0$ and $0 > (b - a)$, the inspectee will prefer to violate when the other does not inspect and not to violate when the other inspects. Conversely, since $-c > -d$, the inspector will prefer to inspect when the other violates and not to inspect when the other does not violate.

Game Solution: Nash Equilibrium at Indifference Strategies

Due to the circular structure of the players' preferences, they cannot determine in advance their own best pure strategy. They have to resort to a suitable randomization between the two choices so as to maximize the expected payoff, taking into account that the other will act accordingly. In other terms, each one will have to adopt a *mixed strategy* (defined by a probability distribution over the pure strategies). This mixed strategy will have to force the other party into adopting a strategy from which he has no incentives to deviate. This joint mixed strategy will represent the NE of the game. Each party's strategy at equilibrium is also called *indifference strategy*. This is due to the fact that the other party's expected payoff will not change whatever mix of his own pure strategy is adopted. For example, let us consider the situation where an inspector chooses a mixed strategy that induces indifference in the inspectee. Then, all strategies of the inspectee will result in an equal payoff for the inspector.

Table 5.1: The payoff matrix for the basic Inspection Game shows the four possible results.

		INSPECTOR	
		Inspect	*Do not inspect*
INSPECTEE	*Violate*	$(b-a, -c)$	$(b, -d)$
	Do not violate	$(0, -c)$	$(0, 0)$

Suppose now that the inspectee adopts the violation choice with probability p, and that the inspector adopts the inspection choice with probability q. Then, the solution of the game can be found by computing the pair (p,q) with is characterized by the following. Neither the inspectee can change his expected payoff by deviating from p, nor the inspector can change his expected payoff by deviating from q. If the inspectee wants to induce the indifference in the inspector, he will have to set his own parameter p so as to equalize two aspects. This is the expected inspector's payoff for an inspection and the inspector's payoff for lack of inspection. Similarly, if the inspector wants to induce the indifference in the inspectee, he will have to set his own parameter q so as to equalize the expected inspectee's payoff for a violation and the inspectee's payoff for lack of violation. Altogether we have

$$\begin{aligned} (-c) &= p(-d) \\ q(b-a)+(1-q)b &= 0 \end{aligned}$$

from which we get the simple solution (p^*, q^*) given by

$$\begin{aligned} p^* &= \frac{c}{d} \\ q^* &= \frac{b}{a} \end{aligned}$$

Notice that q^* is determined by the quantities defining the payoffs of the inspectee. In the expression, the benefit b for an undetected violation competes with the loss a for the detected one. Similarly, p^* is determined by the quantities defining the payoffs of the inspector and the cost for an inspection plays the opposite role to the avoided damage d. It is worth to remark that the expression for q^* is a legal expression for a probability only if $a \geq b$. Analogue, the expression for p^* is a probability only if $d \geq c$, which is granted by the definition of the game.

5.2 GENERALIZATION OF THE BASIC TWO-PLAYER INSPECTION GAME

Distributed systems consist of multiple participants and works in the GT literature present indeed game models with multiple inspectees. For example, Avenhaus and Kilgour [187] discussed a three-person non-zero sum IG with one inspector and two inspectees. However, the environmental setting is more complicated than the one of the thesis' monitoring approach. They study how the game's equilibrium depends on the convexity/concavity of a function representing the inspection effort. Another work worth to mention with regard to a generalization is the contribution of Hohzaki [185]. It is formulated in the context of the International Atomic Energy Agency (IAEA) and directly based on the aforementioned paper of Avenhaus and Kilgour. Hohzaki introduces an IG with n inspectees and the complex situation that an inspector is characterized by several attributes (e.g. nationality). Then, the game model aims to calculate an optimal assignment (partitioning) of the inspection effort to the inspectees, i.e. certain facilities in an inspectee's country. However, the effort cannot be partitioned in our simpler case. Due to these differences, the given models do not fulfill our needs of a mere generalization of the basic IG. These IG models G(1,n), G(m,1) and G(m,n)) are presented now with a straightforward re-derivation of the solutions.

5.2.1 *Game G(1, n) – One Inspector, n Inspectees*

Game Setup

Let us consider now a $(n+1)$-player simultaneous single-round IG, which consists of one inspector and n inspectees. It has the same three payoff assumptions as the basic one (see list in game setup section). As in the basic IG, q indicates the probability that the inspector decides to perform the inspection. Then, the inspection is done on a single randomly chosen inspectee and each inspectee has a probability $\frac{1}{n}$ to be inspected. The inspectees, from now on inspectee $1,\cdots$, inspectee n, have respectively probability $p_1, \cdots, p_n$ of violating the rule. The solution of this game is represented by the values of q^*, $p_1^*, \cdots\ p_n^*$ of the above $(n+1)$ parameters at the NE. For the sake of comprehensibility, we assume a symmetry between the inspectees, i.e. the same solution parameters $p^* = p_1^* = \cdots = p_n^*$. The tree diagram in figure 5.2 shows the different game result possibilities for $n = 2$.

Game Solution

An important point for the solution is that there is no coupling between inspectees. An inspectee's payoff does not depend on the other inspectee's choices. Furthermore, this game does not have, in general, Nash equilibria in pure strategies. Therefore, the players have to find the equilibrium in mixed strategies. The inspectees have to choose the strategy which induces indifference in the inspector. The inspector has to choose the strategy which induces indifference in the inspectees. The results can be derived through simple considerations.

INSPECTOR'S INDIFFERENCE The inspectees have to equalize the expected inspector's payoff for an inspection to the inspector's payoff for lack of inspection. This means that p^* will have to satisfy the simple equation equalizing

- the impact (value times probability) on the inspector for undetected violations due to lack of inspection
- with the balance between the impact of an unfruitful inspection and the one of a fully or partially successful inspection.

The impact for no-inspection is given by the expected number of the violations of n inspectees times the damage d created by each one: i.e. by $np \times d$. The impact for inspection is given by the constant cost c plus the impact of the undetected violations. In this case, the inspector is securing with certainty only one inspectee. Hence, the impact of the undetected violations is given by the expected number of violations p of the remaining inspectees. The inspection impact is therefore $c + (n-1)pd$. The resulting indifference equation is

$$npd = c + d(n-1)p$$

and thus

$$p^* = \frac{c}{d}$$

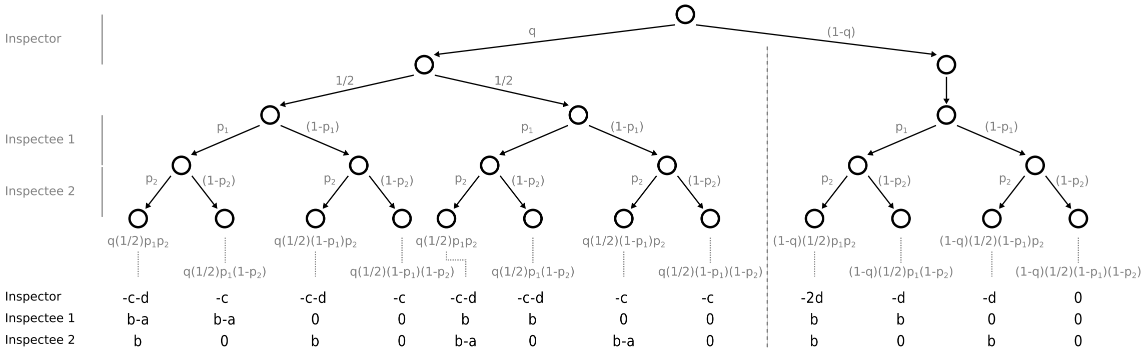


Figure 5.2: Inspection Game $G(1,2)$ in extensive form.

Notice that the optimal p is the same as the one for a single inspectee: *the presence of further inspectees does not change the best strategy of one inspectee.* This is is a natural consequence of the lack of coupling between inspectees.

INSPECTEE'S INDIFFERENCE Analogue to the inspectee, the inspector needs to make each inspectee indifferent. To this end, he equalizes the expected payoff for the inspectees' violation and the expected payoff for non-violation. Looking at the structure of the game one can observe that they consider the inspection to another inspectee as equivalent to no inspection at all. Hence, the inspector has to behave as if each of them were playing against him an effective two-player game $G(1,1)$ game with rescaled parameters. We can describe this effective game by introducing an effective probability of inspection $q_{eff} = \frac{q}{n}$. The extensive form of this game is the same as the one shown in figure 5.2 except that the probability q is substituted by $q_{eff} = \frac{q}{n}$. The inspectee's indifference is obtained equalizing the impact for non violation, which is null, to the impact of violation. The latter one is given by the balance between the detected one and the undetected one. In case of a violation, there will always be a benefit for the inspectee. The impact is therefore given by b added to the impact of the loss (loss times probability of inspection $q_{eff} = \frac{q}{n}$). The indifference equation is

$$b - a\frac{q}{n} = 0$$

hence

$$q^* = \frac{b}{\frac{a}{n}}$$

The factor $\frac{1}{n}$ results from the fact that one inspector is shared by two inspectees. As mentioned before, there is no influence on p due to the lack of coupling among the inspectees. The presence in q of a factor n represents the situation where each inspectee can see this IG as a two-player game with effective loss $\frac{a}{n}$.

5.2.2 *Game G(m,1) – m Inspectors, One Inspectee*

Game Setup

Beside the assumptions 1. to 3. already adopted so far, we are also forced to postulate some coupling between inspectors. They must *share the damage* of any occurring violation which

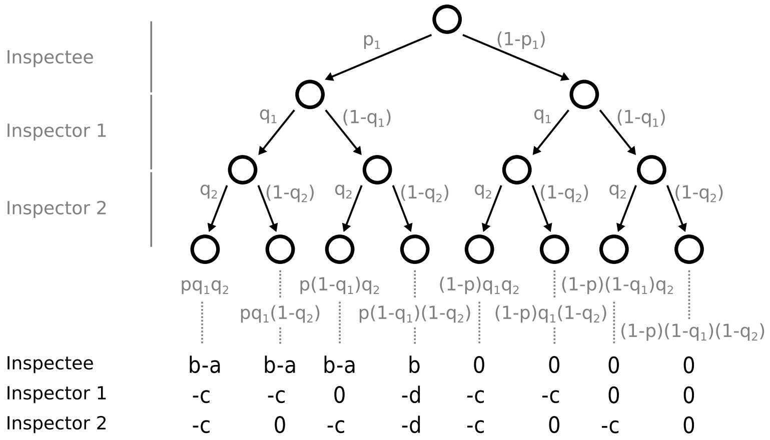


Figure 5.3: Inspection Game $G(2,1)$ in extensive form.

goes undetected (i.e. detected by none). A detailed formulation is represented in figure 5.3 in extensive form for the exemplary case of $m = 2$ inspectors and $n = 1$ inspectee.

Game Solution

We can exclude the possibility of NE in pure strategies because all the pure strategy profiles have at least one player which would benefit from switching strategy unilaterally. The inspectee would prefer to violate when no inspection occurs and each inspector would prefer to inspect when there is a violation. We will exploit the symmetry between the inspectors since we know that the equilibrium is of the form $q_1^* = \cdots = q_m^* = q^*$.

INSPECTEE'S INDIFFERENCE The equation for the inspectee's indifference should equalize the impact for no violation, which is null, to the impact for violation. This in turn is given by the balance between the impact of detection and that of non detection. Since the benefit for violation is always present (be the violation detected or not), the balance is obtained by subtracting from b the impact of loss only (probability times value of loss). The probability of detection by at least one of the inspectors is $1 - (1-q)^m$, hence the overall indifference equation is

$$b - a(1 - (1-q)^m) = 0$$

which has solution for q^* such that

$$(1-q^*)^m = 1 - \frac{b}{a}$$

and therefore

$$q^* = 1 - (1 - \frac{b}{a})^{\frac{1}{m}}$$

INSPECTOR'S INDIFFERENCE The inspector's indifference equation should equalize the impact of inspection (given by a constant cost value c) to the impact of no inspection. The latter corresponds to the expected value of the number of violations by the only inspectee when no other inspector is inspecting. Hence, the indifference equation is

$$c = dp(1-q)^{m-1}$$

which has solution for

$$p^* = \frac{\frac{c}{d}}{(1-q^*)^{m-1}}$$

or explicitly – taking into account that at the equilibrium value $(1-q^*) = (1-\frac{b}{a})^{\frac{1}{m}}$ – for

$$p^* = \frac{\frac{c}{d}}{(1-\frac{b}{a})^{\frac{m-1}{m}}}$$

5.2.3 *Game G(m,n) - m Inspectors, n Inspectees*

In the game with m uncoordinated inspectors and n (non interacting) inspectees, the presence of n inspectees reduces the probability of any inspector visiting the i-th inspectee from q_i to $q_i^{eff} = \frac{q_i}{n}$. Hereafter, exploiting the symmetry among inspectors, we will use q in place of q_i. Exploiting the symmetry among inspectees, we will use p in place of p_i.

INSPECTEE'S INDIFFERENCE The indifference equation for each inspectee, which is used here to determine q, is

$$b - a(1-(1-\frac{q}{n})^m) = 0$$

which has solution for

$$(1-\frac{q^*}{n})^m = 1-\frac{b}{a}$$

and thus

$$q^* = n(1-(1-\frac{b}{a})^{\frac{1}{m}})$$

The result is analogue to $G(m,1)$ except that q is replaced by the effective $q_{eff} = \frac{q}{n}$.

INSPECTOR'S INDIFFERENCE The inspectors' indifference equation which is used here to determine p, should equalize

- the impact of no inspection. This corresponds to d times the expected value of the number n of inspectees' violations going undetected by the other (m-1) inspectors.
- the impact of inspection. This is given by a constant cost plus the individual damage times the expected value of the number (n-1) of inspectees' violations going undetected by the other (m-1) inspectors.

In both cases, the answer depends on two aspects. This is the expected number of undetected violations when each inspectees violates the rule with probability p and that each inspector performs an inspection with probability q – a quantity which we can call $u(m,q,n,p)$. The indifference equation will equate the following two impacts

$$d\,u(m-1,q,n,p) = c + d\,u((m-1),q,(n-1),p)$$

which can be rearranged so that

$$\frac{c}{d} = u((m-1),q,n,p) - u((m-1),p,(n-1),q)$$

The difference at the second member represents the expected number of *extra* undetected violations, which occur when an inspector does not inspect. The missing inspection does not produce any extra undetected violations if the peer, which would be inspected, does not violate the rule, or if that peer is already inspected by the other inspectors. In other words, the missing inspection leaves one extra inspectee violating undetected only when that inspectee does perform the violation and the other $(m-1)$ inspectors do not detect it. The former event happens with probability p and the latter with probability $(1-\frac{q}{n})^{m-1}$ (since each inspector has probability $\frac{q}{n}$ of falling over that inspectee). Hence the indifference equation is

$$\frac{c}{d} = p(1-\frac{q}{n})^{m-1}$$

and has solution for

$$p^* = \frac{\frac{c}{d}}{(1-\frac{q}{n})^{m-1}}$$

Overall, substituting q^*, we have

$$p^* = \frac{\frac{c}{d}}{(1-\frac{b}{a})^{\frac{m-1}{m}}}$$

To summarize results of this section, the solution for all games are listed in table 5.2. Notice that the p^* of the various $G(\cdot, n)$ is equal to that of the corresponding $G(\cdot, 1)$. Adding or removing inspectees does not change the p^* because there is no coupling between inspectees. On the contrary, the q^* of the various $G(\cdot, n)$ is n times larger than that of the corresponding $G(\cdot, 1)$. Multiplying the inspectees' number by n does change q^* since it requires a proportional increase in the inspectors' effort. Notice as well that both p^* and q^* of the $G(m, \cdot)$ are reduced with respect to the corresponding $G(1, \cdot)$. This is coherent with an increased and joint inspectors' pressure.

Table 5.2: Solutions for all IGs up to m inspectors and n inspectees. p^* indicates the inspectee's equilibrium violation probability and q^* the inspector's equilibrium inspection probability.

	p^*	q^*
$G(1,1)$	$P \equiv \frac{c}{d}$	$Q \equiv \frac{b}{a}$
$G(1,n)$	P	nQ
$G(m,1)$	$\frac{P}{(1-Q)^{\frac{m-1}{m}}}$	$1-(1-Q)^{\frac{1}{m}}$
$G(m,n)$	$\frac{P}{(1-Q)^{\frac{m-1}{m}}}$	$n\left(1-(1-Q)^{\frac{1}{m}}\right)$

5.3 INSPECTION GAMES WITH FALSE NEGATIVES

In real-world applications, a system designer is faced with limited resources (e.g. limited processing or memory capabilities of machines in distributed systems). Thus, violations are possibly not detected during an inspection, denoted here as *false negatives*. This is addressed now by enriching the four IGs by a finite probability of non-detection. The enriched version of the game with one inspectee and one inspector $G(1,1)$ is exemplary shown in figure 5.4. Similar extensions can be devised for the others. We will indicate the corresponding games with false negatives by $\Gamma(\cdot,\cdot)$. Their NE can be found by straightforward considerations. Let us indicate by γ the probability that an inspection does detect a violation that actually occurred. False positives are assumed to be not possible. When an inspection detects a violation, there is no doubt that the violation occurred.

A first key observation for the development of the more general cases $G(\cdot,\cdot)$ concerns the inspectee's indifference equation used to determine q^*. Whenever an inspector sets the probability of inspection to the value q, the inspectee perceives an effective probability γq. Notice that due to this fact, all values q in the equations for the $G(\cdot,\cdot)$ are substituted by γq in the equations for the $\Gamma(\cdot,\cdot)$. Thus, the equilibrium values q^* for the inspectors in *all* the games $G(\cdot,\cdot)$ will be rescaled by a factor $1/\gamma$. For this reason we have to discuss in detail only the inspector's indifference equation in the following cases.

5.3.1 *Game* Γ*(1, 1) – One Inspector, One Inspectee*

The equilibrium equation for the inspector in $\Gamma(1,1)$ changes slightly with respect to $G(1,1)$. The payoff for the inspection is not simply $(-c)$. Instead, it is decremented by the term $(-d)(1-\gamma)p$ due to possible inspection failure. The overall indifference equation is thus $-c+(-d)(1-\gamma)p=(-d)p$ or $c=pd\gamma$. It has the solution

$$p^* = \frac{c}{\gamma d}$$

valid for $\gamma d \geq c$. As anticipated above the solution value for q, valid for $\gamma a \geq b$, is instead

$$q^* = \frac{b}{\gamma a}$$

The solutions p^* and q^* are equal to the solution values for $G(1,1)$ rescaled by a factor $1/\gamma$. This represents an increased violation rate and a correspondingly increased inspection rate.

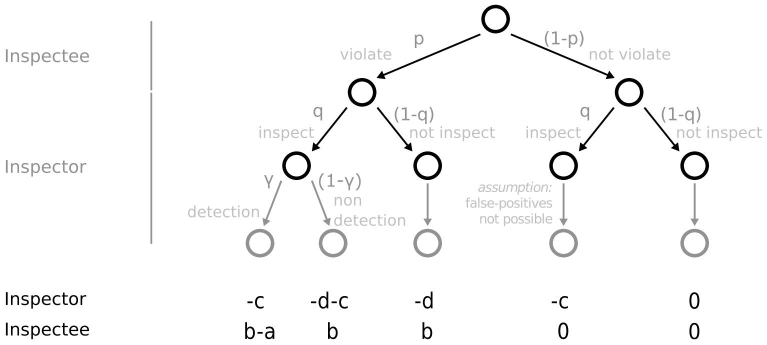


Figure 5.4: The Inspection Game $\Gamma(1,1)$ is shown in extensive form. It corresponds to $G(1,1)$ but is extended by the possibility of false negatives.

5.3.2 *Game Γ(1, n) – One Inspector, n Inspectees*

For the inspector's indifference equation in the game $\Gamma(1, n)$, we have the non-inspection side npd of the equality. This represents the expected value of the damage from a set of n independent inspectee choosing to violate with probability p. The inspection side consists of the inspection costs also present in $G(1, n)$, i.e. $c + d(n-1)p$, plus the failed inspection term $(1-\gamma)pd$. Hence, the indifference equation is

$$\begin{aligned} npd &= c + d(n-1)p + (1-\gamma)pd \\ \equiv pd &= c + (1-\gamma)pd \end{aligned}$$

which is equivalent to $\gamma pd = c$. It provides the solution

$$p^* = \frac{c}{\gamma d}$$

As anticipated above, the solution value for q is instead

$$q^* = n\frac{b}{\gamma a}$$

Again, the two solutions p^* and q^* are equal to those of $G(1, n)$ rescaled by a factor $1/\gamma$.

5.3.3 *Game Γ(m, 1) – m Inspectors, One Inspectee*

Anticipating the prior results, the solution value for q (inspectee's indifference) is such that

$$(1-\gamma q^*)^m = 1 - \frac{b}{a}$$

and its explicit form can be found in table 5.3. As for p, the inspector's indifference equation should equalize the impact of inspection to the impact of no inspection. The latter term corresponds to the expected impact of the violation (probability times impact) by the only inspectee when no other inspector performs a successful inspection. This is represented by $(-d)p(1-\gamma q)^{m-1}$, while the successful inspection has probability γq. The impact of an inspection is given by the constant cost $(-c)$ plus the expected inspection failure. For the latter one, there is only a damage $(-d)$ if the inspectee violates (probability p), the inspection fails (probability $1-\gamma$) and no other inspector performs a successful inspection (probability $(1-\gamma q)^{m-1}$). Hence, the indifference equation is

$$c + dp(1-\gamma)(1-\gamma q)^{m-1} = dp(1-\gamma q)^{(m-1)}$$

with solution

$$p^* = \frac{c}{\gamma d(1-\gamma q)^{(m-1)}}$$

In terms of the solution q^* it is

$$p^* = \frac{c}{\gamma d(1-\frac{b}{a})^{(m-1)/m}}$$

As before, the two solutions p^* and q^* are equal to those of $G(m, 1)$ rescaled by a factor $1/\gamma$.

Table 5.3: Summary of the results for the Inspection Games $\Gamma(\cdot,\cdot)$, i.e. $G(\cdot,\cdot)$ enriched by false negatives. Notice that solutions of $\Gamma(\cdot,\cdot)$ are equal to those of $G(\cdot,\cdot)$ divided by a factor γ.

	$\gamma\, p^*$	$\gamma\, q^*$
$\Gamma(1,1)$	$P \equiv \frac{c}{d}$	$Q \equiv \frac{b}{a}$
$\Gamma(1,n)$	P	nQ
$\Gamma(m,1)$	$\frac{P}{(1-Q)^{\frac{m-1}{m}}}$	$1-(1-Q)^{\frac{1}{m}}$
$\Gamma(m,n)$	$\frac{P}{(1-Q)^{\frac{m-1}{m}}}$	$n(1-(1-Q)^{\frac{1}{m}})$

5.3.4 *Game Γ(m, n) – m Inspectors, n Inspectees*

The solution for q yielded by the inspectee indifference is presented in the following equation. It is analogue to $\Gamma(m,1)$ but with q replaced by q/n. The explicit form is shown in table 5.3.

$$\left(1-\gamma\frac{q^*}{n}\right)^m = 1-\frac{b}{a}$$

The inspectors' indifference equation to determine p should equalize two information. On the one hand, this is the expected impact (on a single inspector) in case of no inspection. This corresponds to d times the expected value of the inspectees' violations going undetected by the other m-1 inspectors. On the other hand, it is the expected impact (on a single inspector) of inspection. Considering the derivation of $G(m,n)$, the value $(-c)$ of the certain cost for an inspection and needs to be equated with the expected impact of the extra detected violation. The latter one represents the damage $(-d)$ times the violation probability p times successful detection probability γ of a violation undetected by the other inspectors. This leads to

$$c \;=\; d\, p\, \gamma\, (1-\gamma\frac{q}{n})^{m-1}$$

or

$$\frac{c}{\gamma d} \;=\; p\,(1-\gamma\frac{q}{n})^{m-1}$$

and has the solution

$$p^* = \frac{\frac{c}{\gamma d}}{(1-\gamma\frac{q}{n})^{m-1}}$$

Overall, substituting q^*, we have

$$p^* = \frac{\frac{c}{\gamma d}}{(1-\frac{b}{a})^{\frac{m-1}{m}}}$$

Again, the solution values p^* and q^* are equal to those of $G(m,1)$ rescaled by a factor $1/\gamma$. The results for the games $\Gamma(\cdot,\cdot)$ are summarized in table 5.3, they are equal to the solution values for the corresponding $G(\cdot,\cdot)$ rescaled by a factor $1/\gamma$. It represents an increased violation rate and a correspondingly increased inspection rate.

5.4 A NOTE TO THE GAME SOLUTIONS

The derivation showed that the solutions of $\Gamma(\cdot,\cdot)$ are equal to those of $G(\cdot,\cdot)$ divided by a factor γ. This is due to the following reasons. The function γ has two impacts on the game $\Gamma(1,1)$. This is an increased violation rate (some violations are undetected) and an increased inspection rate. Both increases are in balance, which causes the rescaling for this game. The game with n inspectees $\Gamma(1,n)$ is played by each inspectee as two-player game due to their independence among each other. For an inspectee i, the inspection of another inspectee j is considered as no inspection. Hence, the rescaling is directly transferred to this generalization. Similarly, the game $\Gamma(m,1)$ can also be considered as a 1-1 game since the inspectors work on the administrator's behalf and share a damage d. Thus, they appear as one inspector with a correspondingly adapted inspection rate. The rescaling of $\Gamma(m,n)$ follows then straightforward considerations.

5.5 SUMMARY

This chapter introduced the game theoretic foundations for the remainder chapters. The initial two-player game $G(1,1)$ of Dresher [175] was introduced in the beginning. It was then generalized up to $G(m,n)$ with m inspectors and n inspectees. All games $G(\cdot,\cdot)$ were then extended by the possibility of false negatives (not detected violations). This is represented by function γ and the corresponding games denoted as $\Gamma(\cdot,\cdot)$. The solutions, Nash equilibria in form of indifference strategies, are given for all games. The games $\Gamma(\cdot,\cdot)$ with possibly multiple players provide an IG model that is suitable for an application to distributed systems.

6 ON THE INSPECTION GAME APPLICATION TO DISTRIBUTED SYSTEMS

This chapter discusses now some issues that arise during an application to real world systems. It focuses on two major aspects: the IG model itself and possible implementations. Please note that this discussion does not intend to be exhaustive. Instead, it targets to call the reader's attention to the most relevant issues. This is meant as a preparation for an IG application.

6.1 INSPECTION GAME MODEL

At first, we consider several aspects related to the IG model.

6.1.1 *Independency of Players and Games*

As for the the IGs of the last chapter, we assume that all players are independent: the strategy choice of one player has no influence on those of the others. Nevertheless, since all inspectors work on the system administrator behalf, there is some inherent relation among the inspectors.

The IG type such as $\Gamma(1,1)$ or $\Gamma(m,n)$ depends on the players' point of view, i.e. with who the game is actually played. Let us consider again figure 5.2 (page 61) for the game G(1,2) with one inspector and two inspectees[1]. If the inspector does not inspect, the system damage caused by the inspectees' violations (if any) is summed up. Here, it is assumed that both inspectees are controlled with only one inspection. This is not realistic: in real world distributed systems, each inspection causes some inspection costs. Hence, the game G(1,2) or in general G(1,n) can be considered for the inspector as a set of independent *1-1* games, one per inspectee. This is similar for the generalization G(m,1), shown as game G(2,1) in figure 5.3 on page 62. A violation is punished only once even if both inspectors detect the violation at the same time. In other words, the m inspectors appear for the inspectee as one inspector with a correspondingly higher inspection probability. This is also denoted as Effective Inspection Probability (EIP) in the remainder. Hence, the generalized game G(m,1) can also be considered as a *1-1* game with all m inspectors representing one inspector with the EIP value as inspection probability.

To summarize, all IG generalizations can be considered as sets of *1-1* games. From the inspector's point of view, he plays multiple games (one per inspectee) since all are independently played. Analogue, the inspectee fears to be inspected by any inspector, whose exact number affect the EIP, i.e. the violation detection probability. The EIP value is also affected by the way how inspectees are selected for an inspection. This is addressed by the next paragraph.

[1] The games $G(\cdot,\cdot)$ are only used here since the corresponding figures have already been discussed.

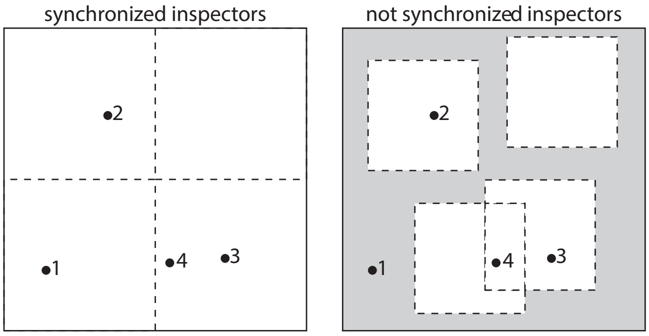


Figure 6.1: This figure illustrates synchronized (left) and not synchronized (right) inspectors in terms of the inspectee selection. The id space is represented in a two-dimensional way with dots as inspectees. The dashed squares indicate the assignment of id regions to an inspector.

6.1.2 *Synchronization of Inspectors*

In general, an inspectee may be controlled by multiple inspectors at the same time if no mechanisms are implemented to avoid it. Such *double inspections* bring – in our case with only unique punishments – no benefit for the games $G(\cdot,\cdot)$. For $\Gamma(\cdot,\cdot)$, they yield only a slight increase of the violation detection probability. Hence, double inspections are considered as a waste of resources. To avoid them, inspectors should coordinate their inspectee selection. This is denoted *synchronization of inspectors* in the remainder. It can be realized statically (e.g. assignment of id ranges) or dynamically based on some runtime criteria (e.g. hop distance).

Let us consider the term Inspection View (IV) as the set of inspectees that represent *inspection candidate*s for an inspector. Figure 6.1 illustrates the inspector synchronization for a two-dimensional id space (e.g. the IP address or overlay id) with the inspection candidates as black dots. Synchronized inspectors are shown left in the figure: the id space is separated into four equal IVs, each one assigned to exactly one inspector. On the right side, the inspectors are not synchronized and id regions not assigned to an inspector are indicated grey. Double inspections are possible when the inspectee is inside of overlapping IVs. This is shown for candidate 4, who is assigned for two inspectors. Candidate 2 and 3 are assigned to only one inspector and candidate 1 can violate without fearing to be punished (inspections will not occur). The separation of the id space is a comfortable yet efficient way to attain synchronized inspectors. It is in particular efficient for systems with id based routing such as the Scribe system. Further systems are reviewed in the survey of Lemmon et al. [226]. In general, it is not applicable in systems with non-deterministic dissemination such as gossiping. An alternative is the setup of the IV based on the network distance (e.g. hops). Then, a separation takes place in the space dimension, i.e. the network topography. In such systems, a system designer should be aware that non-assigned inspectees may also arrive through failure situations or even system dynamics (e.g. node churn). This is not the case for approaches with id space separation.

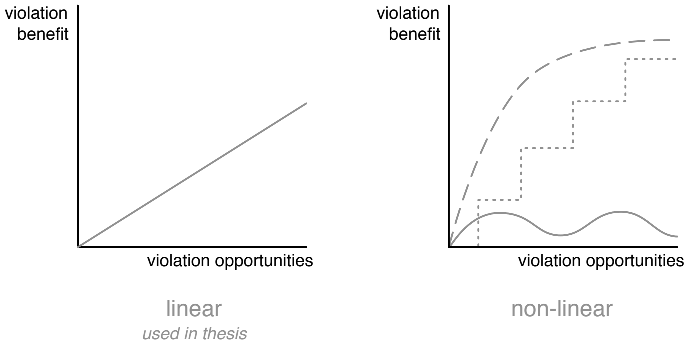


Figure 6.2: This thesis considers for the sake of simplicity a linear dependency between violation opportunities and possible violation benefit. However, non-linear relations are generally possible.

6.1.3 *Payoff in the Real World*

For an IG application, we are faced with the question *what payoff actually means in the real world?* In general, it is supposed to precisely quantify the players' outcome of a game and is established by payoff values. These values are crucial for the general IG mechanics, however, the interpretation of real world conditions and the mapping to appropriate values is a hard task due to the context dependency. This comprises several aspects such as the system state or individual preferences. Therefore, it is challenging task and an important drawback.

The complexity is now shortly outlined. The inspection procedure is known to the inspector. Hence, function γ and the inspection costs c can be evaluated in an – to some degree – objective way for the average case. To determine the system damage, the actual impact of a peer's violation and the meaning of the serviceability loss needs to be evaluated. This can be done with simulative studies. In contrary, the specification of the inspectee's benefit b and the punishment a is difficult. They depend completely on the inspectee's individual personal goal(s). The values are typically determined by the system designer's correct interpretation. This is challenging and more objective approaches should be used. Feedback mechanisms may support the designer's choice or even specify the values dynamically. A simple example is the inspection rate adaptation until a desired collaboration is reached[2].

A further aspect concerns the question how worth a violation is over time for the inspectee. Intuitively, there is a linear relation: the more violation possibilities are, the more violation benefit is possible. This is illustrated in figure 6.2. However, non-linear dependencies are also possible as indicated on the right side of the figure. The wave curve represents a relation to the time of the day. The dotted step-like curve represents a situation, where for example an internet connection contract provides a graded price in terms of data consumption. The dashed curve corresponds to a situation where especially the first violations have a high benefit. An example is a tampered response to an initial DNS request in order to redirect the further communication to a fake server.

[2] Studies of such mechanisms are actually not in the scope of this thesis. However, an exemplary simulation result of this mechanism is presented in figure 9.8 (page 110) during the discussion of chapter 9.

6.1.4 *Inspection Games for Continuous Operation*

In contrary to IGs distributed systems are supposed to operate indefinitely. An approach to cope with this problem is outlined in figure 6.3, where the several operations are split into groups. Each group is related to a specific game and each game is played independently from the other ones. Then, a game strategy is chosen for each game, which is applied to all corresponding operations.The inspectee needs to collect collaboration proofs for each operation, which are denoted as *proof messages*. To this end, we introduce a container, the *inspection target*. It stores the proof messages for a game and is controlled during an inspection by the inspector. The size of the inspection target is a game parameter and should ideally be able to store all proof messages related to a game. In order to obtain stable violation benefit values over time, a system designer should target equally composed games in terms of proof messages. One possibility is a *game splitting* method based on the amount of operations (load-driven split). Other common methods are based on time (e.g. periodical split), operation type (type-driven split) or even network message content (semantic-driven split). In addition to the game splitting method, equally composed games can also be achieved by means of the inspection target. Providing an appropriate or a dynamic size the inspection target is able to tolerate fluctuations in terms of proof messages.

In distributed systems, we cannot expect that the users are aware of playing games or that they are independently played. Hence, we specify three further assumptions:

1. For real world systems, users are informed about an IG and its mechanics (except the other player's strategy) as reliability method, for example in the settings user interface.
2. The game players are informed about possible inspections and punishments.
3. The players can draw conclusions from circumstances such as undergone inspections.

As we will see in the next paragraph, this is important for the practical objective of research challenge (C). Nevertheless, the games itself remain independent and their outcome has no influence on the other games' strategy choice. Especially strategies over multiple games remain impossible. This would change the game characterization to evolutionary games and require another IG model.

6.1.5 *General Principle for Collaboration Achievement*

IGs of the GT literature focus on the Nash equilibrium analysis in terms of payoff values. This was also done in the prior chapter 5 for $G(\cdot,\cdot)$ and $\Gamma(\cdot,\cdot)$, respectively. However, corresponding to research challenge (C), the system administrator has the practical objective to achieve a targeted collaboration $1-p$ in average. Hence, the collaboration value becomes for the inspector more important than the payoff value itself. In this context, the IG model enables to calculate an appropriate inspection strategy for the inspector. This strategy yields in a shift of the NE to a strategy combination that corresponds to the targeted level of collaboration.

For a better understanding, let us introduce the term Best Response Strategy (BRS). It denotes the strategy that results in maximal possible payoff, given the imaginary situation that

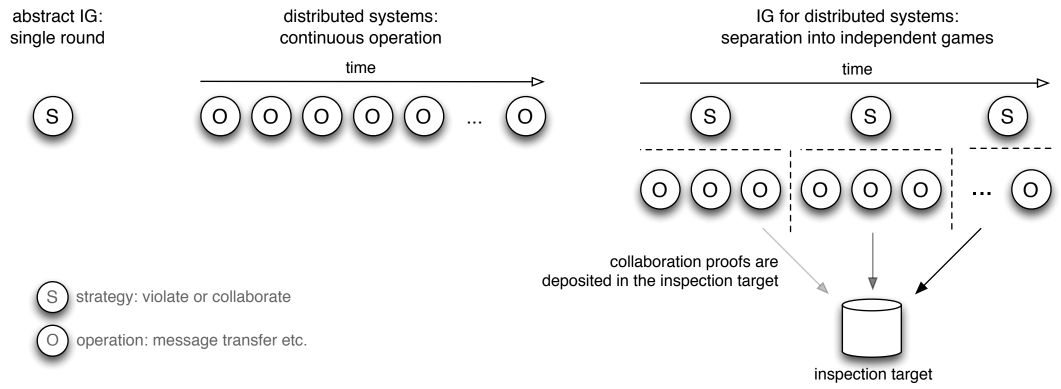

abstract IG:
single round
distributed systems:
continuous operation
time
IG for distributed systems:
separation into independent games
time
S
O
...
collaboration proofs are
deposited in the inspection target
inspection target
S strategy: violate or collaborate
O operation: message transfer etc.

Figure 6.3: The IG must be adapted to a possibly infinite operation of distributed systems. To this end, each of the the operations is mapped to single game and each game is then independently played. The inspection target holds collaboration proofs and is controlled by the inspector.

the other player's strategy is known. Considering the assumptions of the last paragraph, the players can approximate the other player's strategy – not in real time but over several games. With such approximation, the rational inspectee is able to calculate a BRS in order to increase the own payoff. This rationality, i.e. trying to optimize the payoff, is then leveraged by the inspector. By means of the IG mechanics, he is able to specify and inspection strategy that leads to the desired collaboration value $1 - p$. In more detail, the inspector choses a strategy such that the inspectee's BRS lies exactly at the target collaboration value $1 - p$.

6.2 inspection game implementation

As counterpart to the last section, we discuss now some possible implementations. This comprises the inspection architecture, the collaboration incentive and some game characteristics.

6.2.1 *Inspector & Inspection Architecture*

Still dealing with two types of individuals, the following question comes up: *How can the IG be realized in terms of the network architecture with multiple user machines?* This is answered by considering two aspects separately: the inspector and the inspection architecture.

implementation of the inspector The inspector logic can be implemented in several ways. Figure 6.4 illustrates two general approaches. On the left side, the inspector is realized as individual peer beside the inspectee peer. They are loosely coupled over the network in terms of inspections. On the right side, the inspector is realized as a network layer within the inspectee's peer. These approaches have several advantages and drawbacks, which are outlined in the following. Please note that they are only examples; other solutions (e.g. mobile software agents moving over the inspectees' peers) are possible.

Systems with independent inspector peers are very flexible. The amount of inspectors can be easily adapted to the corresponding needs. However, it requires some overhead due to the inspections over network but also in terms of inspector synchronization. The example

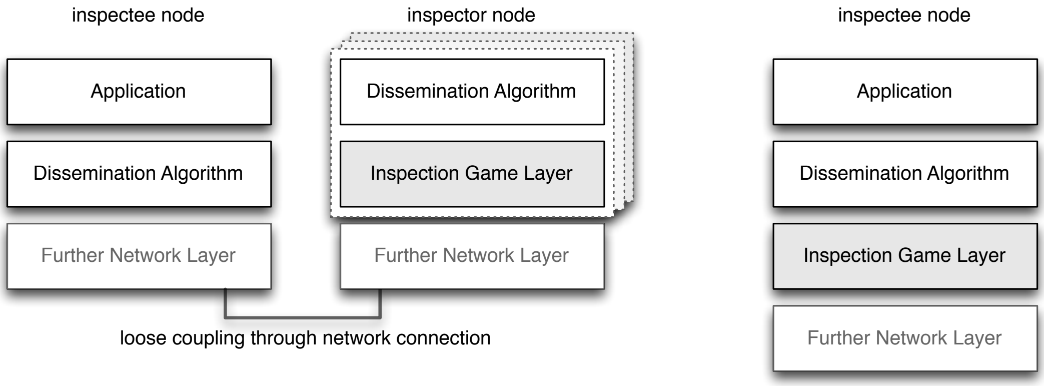


Figure 6.4: Two possible inspector implementations are illustrated: as individual peers (with possibly multiple inspector instances on the peer) and as network layer at the inspectee's peer.

with an inspector as network layer, the inspector is directly present at the inspectee. Hence, local inspections are efficiently performed since there is no overhead in terms of network communication. Even the full monitoring of all network messages passing the layer is in general possible. However, the inspector competes with the inspectee for the same limited resources possibly affecting the application's performance. In a worst case, the inspectee may even be encouraged to circumvent the reliability mechanisms.

INSPECTION ARCHITECTURE Four exemplary inspection architectures are illustrated in figure 6.5. Examples 1) and 2) represent the two discussed inspector implementations, i.e. 1) as independent node and 2) as network layer at the inspectee. They represent P2P based systems, while 3) and 4) also correspond to more centralized architectures. Example 3) provides exactly one inspector on a dedicated machine, which lies – for illustrative reasons – outside of the actual system. A realistic scenario is drawn by example 4): several dedicated inspectors (e.g. a server infrastructure) control the inspectees in the system. Heres, inspectors are synchronized, although not in an optimal way (double inspection are possible).

6.2.2 *Collaboration Proofs & Incentives*

In any case, some (secured) logic is required at the inspectee's machine in order to maintain proofs of the behaviour and to monitor system values. This is shown in figure 6.6 (left), where the logic as well as the *inspection target* is considered as part of an additional network layer. A further example is a database with some secured logic, where a table entry contains proofs for the processing of database queries. The realization as network layer of this example is also suitable to directly apply collaboration incentives. This is shown on the right side in figure 6.6 with two examples for negative incentives, i.e. punishments. The first one is a reduction of the messages passing the layer and the second one an expulsion form the system.

We consider in this thesis only negative collaboration incentives by means of punishments. Positive ones (rewards for collaboration) are also possible; however, they do not change the IG mechanics in our case. The incentive mechanisms that are artificially added (such as the

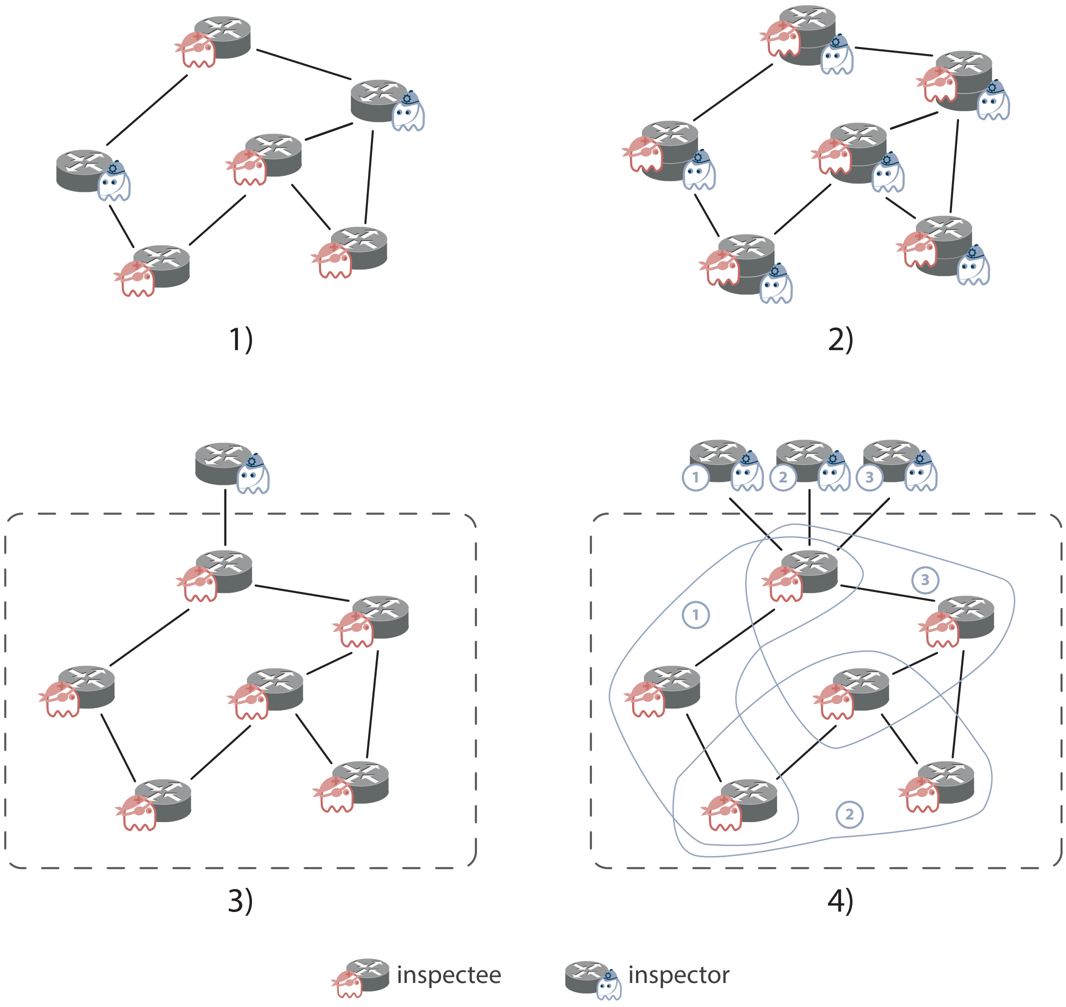


Figure 6.5: Four exemplary inspection architectures are illustrated.

network layer of figure 6.6) are denoted as *artificial incentives.* In contrary, *genuine incentive* represent mechanisms that are given by the system design itself. Another possibility is the elimination of the selfishness concern itself, e.g. by out-of-band trust mechanisms such as contracts. The interested reader is referred to the work of Nielson et al. [1] for further details about collaboration incentive classifications.

6.2.3 *Game Characteristics*

The last sections discussed some theoretic and practical aspects, however, independently from each other. We outline now the glue, which brings all together to a usable game implementation. To this end, let us consider the term *game characteristics.* It denotes the further implementation details that describe *how* the game is played. They comprise the following points, which are detailed afterwards.

- *mode* of inspection initiation
- *when* a game is initiated

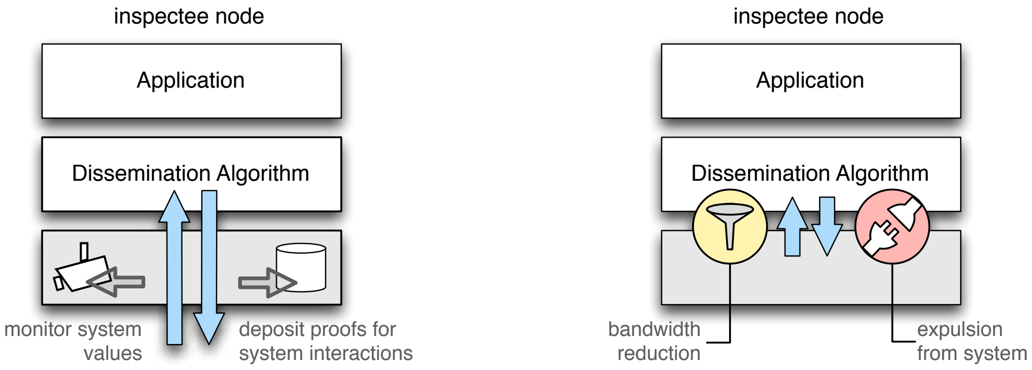


Figure 6.6: A network layer can be added to the inspectee for maintaining collaboration proofs. It may also monitor system values or impose punishments as collaboration incentive.

- *who* is actually playing a game
- *what* is controlled during an inspection

GAME CHARACTERISTIC: MODE The game mode specifies which player initiates the inspection. Two modes are in general possible, described from the viewpoint of the inspectee:

In systems with a *reactive game mode*, an inspectee waits until an inspector requests a collaboration. The inspectee can fully concentrate on the actual system operation and proves the collaboration reactively to an inspection request. In reactive mode, the power about the inspection kept by the system designer and is is initiated either *externally* or *internally*. With an external initiation, there are some external individuals such as game masters as shown in figure 6.7 (left). They do not actively participate at the system operation but induce the inspection. In case of the internal initiation, the inspectors inside the actual system decide when to initiate the inspection (figure 6.7 right).

In systems with a *proactive game mode,* the inspectees decide on their own when to transfer collaboration proofs to an inspector (see figure 6.8). The system designer imposes an *inspection policy* to specify when an inspection must be initiated or which inspector is responsible.

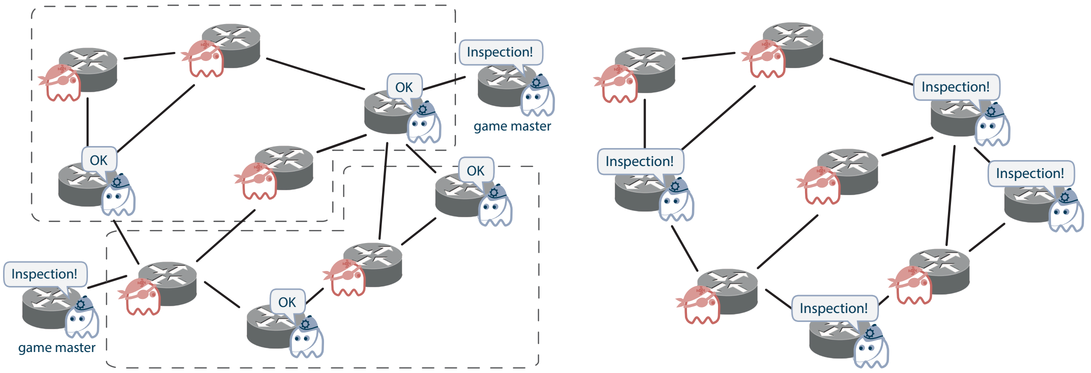


Figure 6.7: Two possibilities for a reactive game mode are shown. The left figure shows an external inspection initiation. Here, an external individual or logic (here a game master) instructs the actual inspector to perform an inspection. The right figure shows the internal initiation, where the inspectors decide on their own to induce the inspection.

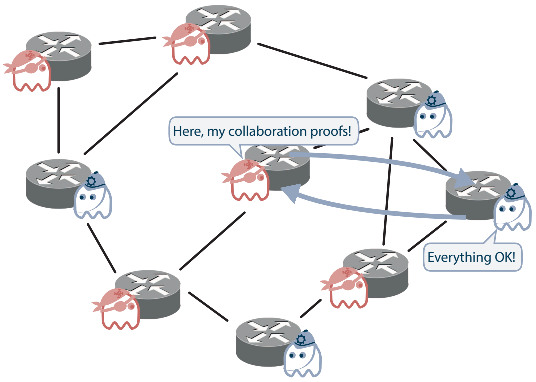


Figure 6.8: In proactive systems, the inspectees decide themselves when to prove correct behaviour in terms of the collaborative protocol. To this end, they transmit on their own – corresponding to an assigned policy – collaboration proofs to an inspector.

Inspectees in proactive systems have gain some flexibility. They can coordinate the inspections with the current work load or further interests (e.g. energy efficiency). However, a drawback is the higher maintenance overhead to verify if the inspectees adhere to the policy.

game characteristic: when The second implementation characteristic is related to the question when an inspection shall be initiated. The general possibilities are outlined in the (non-exhaustive) schema shown in figure 6.9. An inspection probability could be initiated depending on some time constraints (e.g. periodical inspections) or on the time of the day. The inspections may also depend on some values or system parameters. An example is the message load or the fluctuations of an inspectee's inspection regularity (in proactive systems). Context-driven approaches initiate an inspection depending on the given environmental circumstances. Imagine for example a Mobile Ad-Hoc Network (MANET), where an inspection could be done if an inspectee comes into the reach of an inspector.

game characteristic: who A further question to be clarified consists in the selection of the other player. This means the question which inspectee(s) is chosen by the inspector in the reactive mode and which inspector(s) is chosen by the inspectee in the proactive mode. As

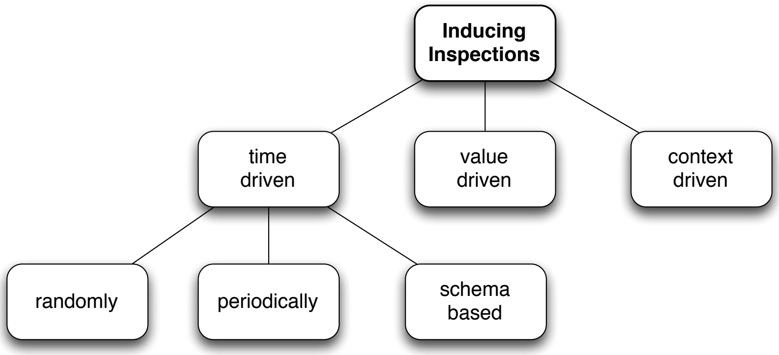


Figure 6.9: Several (not mutually exclusive) ways to initiate an inspection are possible.

a short recall, note again that all players are assumed to work independently. Implementation possibilities for this characteristic are manifold but three are shortly outlined. An intuitive approach is the randomized choice. It is fast in terms of computation and distributes the inspection load equally over all (known) players. However, it ignores the specific architectural circumstances and system states. Another possibility is to choose the other player in terms of the distance. This approach reduces the network load used for inspections by increasing the locality of the game. A further approach, applicable only for the reactive game mode, is the inspectee selection based on the amount of detected violations. Such feedback mechanism enables the IG to adapt the inspection resources to the danger for the system (in terms of protocol violations).

GAME CHARACTERISTIC: WHAT The last point is related to the question what is actually controlled during an inspection. This depends on the considered selfish goals. For example, in the use case of this thesis we consider message drops, i.e. the correct message transfer is verified. To this end, the IG approach makes use of some *secured logging* that serves as proof for the peer's behaviour. Secured logging is indeed under active research. The interested reader is referred to [227, 228, 229, 230] and the references therein.

6.3 SUMMARY

This chapter serves as connection between theory, the game theoretical foundations, and practice, an IG application to distributed systems. A possible application was discussed with regard to two aspects. At first, several aspect and modifications of the IG model were considered that arise for a practical utilization. Second, some realization details and implementation possibilities were introduced.

The discussion identified beneficial aspects but also challenges for an application. An important outcome is the insight that all IG generalizations can be considered as sets of independent two-player games. This simplifies the modelling and analysis. A significant challenge is the determination of appropriate payoff values and game parameters in general. They are crucial for the IG mechanics but depend on the individual user preferences and system states.

7 AN INSPECTION GAME FRAMEWORK FOR DISTRIBUTED SYSTEMS

This chapter introduces an application of the Inspection Game (IG) with false negatives $\Gamma(\cdot,\cdot)$ to distributed systems as solution approach for research challenge (C). In order to provide an illustrative use case, we consider the video streaming scenario of the introduction with selfishness-driven message drops. Furthermore, we use the two pub/sub systems Scribe and basic gossiping for an application.

7.1 APPROACH OVERVIEW

In order to modify a given system as less as possible, we add a network layer at the inspectee's machine. To reduce the possibility of vulnerabilities, it provides merely some secured logic responsible for two tasks. At first, it deposits proofs of the peer's system interaction into a inspection target, which has a fixed size to ease the comprehensibility. Furthermore, the logic is responsible to apply a punishment as collaboration incentive. The actual inspector is realized as independent peer, loosely coupled with the inspectees. The architectural overview of this approach is shown in figure 7.1.

The IG is operated in *(internal) reactive* mode, i.e. the inspectors initiate the inspections. This is done *time-dependently* in a *periodical* way. In other words, the system operation is split into time intervals T, each one related to one game. With all games being played sequentially, the IG can be considered as a sliding window moving over the several intervals. The length of a time interval T itself is not relevant since it serves only as point of reference. However, for the sake of simplicity, an equal interval T is assumed for all inspectees and inspectors.

The system consists of m inspectors and n inspectees. Each inspector selects randomly (covering the whole id space) n_i inspectees for an inspection per interval T to support a

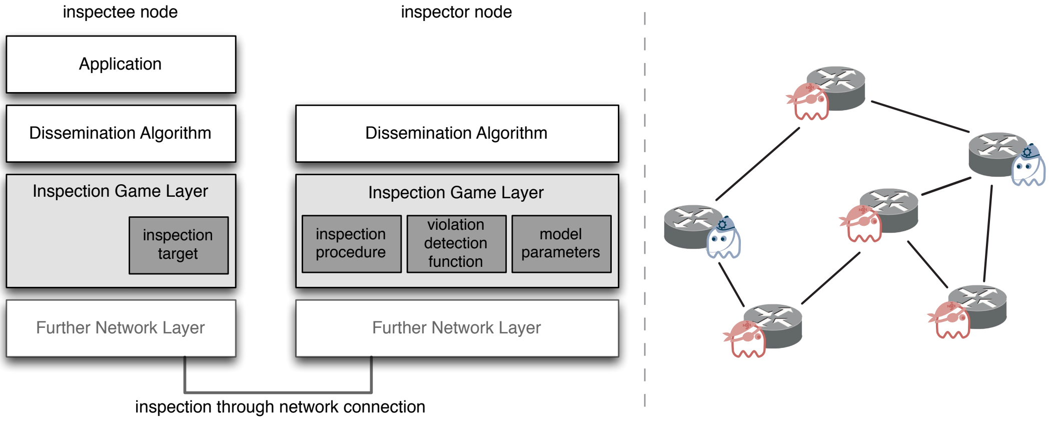


Figure 7.1: An architectural overview is shown left in the figure. Dark boxes show the most important parts to be specified during an application of the framework. The corresponding inspection architecture is shown on the right side.

broad range of systems. Hence, the inspectors are not synchronized. Furthermore, all played IGs are considered as *1-1* games as described in the previous chapter. Therefore, this approach is related to the IG as presented in the extensive form in figure 5.4 (page 65).

7.2 INSPECTION TARGET AS MONITORING INTERFACE: DESIGN DETAILS

Before providing details of the IG mechanics, we introduce a possible implementation of the inspection target and procedure. It targets to harden the system against selfishness-driven messages drops as described in the scenario of the introduction. Please note that the details presented here represent only one example for possible reliability mechanisms.

7.2.1 *Inspection Target*

The inspection target stores *proof messages* that prove a peer's behaviour in a First-In-First-Out (FIFO) manner and is also denoted as *proof history*. With an amount of r stored proof messages per message transfer and a history size h, the inspection target is able to secure $\frac{h}{r}$ of such operations. All messages are assumed to be cryptographically secured.

The inspection target is used to enable the detection of possible protocol violations in terms of two aspects. First, the hop-to-hop transfer itself is protected. Second, the existence of an omitted forwarding is checked. The hop-to-hop transfer protection is realized by a simplified version of the one presented in [43]: cryptographically secured promises and acknowledgements. A transfer from peer A to peer B consists of the following four phases. Note that all messages are digitally signed, which is omitted here for the sake of clarity if it is not essential part of the phase.

1. Peer A wants to sends a signature of the message to be sent as promise to peer B.
2. Peer B responds with to the promise.
3. Peer A sends the actual message.
4. Peer B replies the correct reception with another acknowledgement.

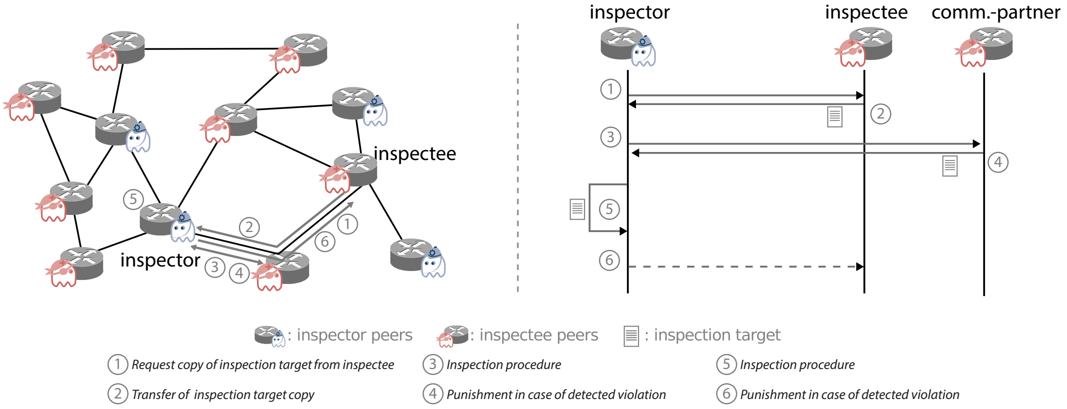

inspectee
inspector
1
2
3
4
5
6
inspector
inspectee
comm.-partner
: inspector peers
: inspectee peers
: inspection target
1 Request copy of inspection target from inspectee
2 Transfer of inspection target copy
3 Inspection procedure
4 Punishment in case of detected violation
5 Inspection procedure
6 Punishment in case of detected violation

Figure 7.2: The inspection sequence of a two player Inspection Game is shown in the context of a distributed system (left) and schematically focusing on the players' interactions (right).

Both peers store the received $r = 2$ messages (the actual message is part of a receiver's proof message). In the remainder, we consider a history size $h = 200$ and thus, $\frac{200}{2} = 100$ message transfers are secured. This value as well as other values used later, are chosen only for illustrative and comprehensibility reasons.

The second aspect checks whether a message has been correctly forwarded. An inspectee is only allowed to drop a message in each of these cases:

(a) A message with the corresponding ID has already been handled.

(b) The TTL value is zero to avoid unlimited disseminations.

(c) There are no subscribers behind the remaining outgoing links.

The TTL value is typically in the message header. To control (c), we assume the existence of a secured Next Hop Table (NHT), a data structure that maintains the possible next hops. This could be for example the routing table (Scribe) or the partial view (gossiping). Furthermore, we assume that changes in the NHT are tracked for one time interval to prevent covering violation tracks by system dynamics. Secured NHTs are under active research; see for example the secured peer sampling of Jesi et al. [66] for gossip-based systems. With the NHT information and the proof messages, a correct forwarding can be verified.

7.2.2 *Inspection Sequence & Procedure*

In the context of an inspection we denote with *inspection sequence* the several interactions with other network participants. Furthermore, we denote with *inspection procedure* those operations needed to control an inspectee's inspection target.

The inspection sequence is outlined in figure 7.2. An inspector requests at first a copy of the inspection target. After receiving it, he requests also a copy of the inspection targets of those communication partners that have proof messages in the inspectee's history. Then, they are controlled (inspection procedure). In case of a detected violation, the inspector disposes a punishment, which is applied by the secured logic at the inspectee's peer. Note that any inspectee's non-collaboration during the inspection sequence is treated as violation.

The inspection procedure consists of controlling the two aforementioned aspects as shown in figure 7.3. Each message transfer related to the inspection is secured as for the hop-to-hop transfer itself. We omit these messages for the ease of comprehensibility. To verify a hop-to-hop transfer, the inspector controls not only the r proof messages of the inspectee but also the other r messages of the corresponding communication partner. Overall, two histories are controlled during one inspection. The second part (correct message forwarding) is verified by requesting the NHT and controlling it among the proof history. For example, an inspector controls among others if the receiving peer had – at the corresponding time point – a subscription in the NHT of the sending peer. To ease the comprehensibility, we assume this forwarding verification is done during the control of the two histories and covered by the corresponding cost values. Then, the costs for the inspection of one peer consists of the control of two histories.

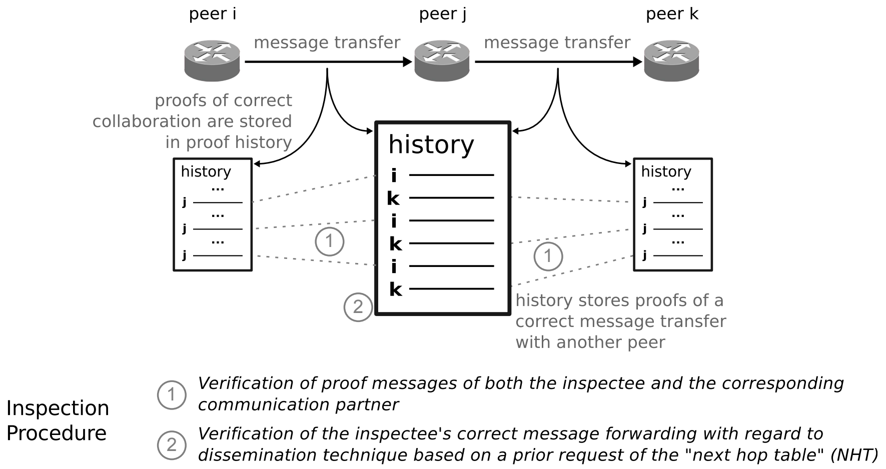

peer i
peer j
peer k
message transfer
message transfer
proofs of correct collaboration are stored in proof history
history
i
k
i
k
i
k
history
...
j
...
j
...
j
history
...
j
...
j
...
j
1
1
2
history stores proofs of a correct message transfer with another peer
Inspection Procedure
1 Verification of proof messages of both the inspectee and the corresponding communication partner
2 Verification of the inspectee's correct message forwarding with regard to dissemination technique based on a prior request of the "next hop table" (NHT)

Figure 7.3: Overview to the inspection procedure.

7.3 GAME & IMPLEMENTATION DETAILS

We introduce now all details needed for an application of the IG framework to pub/sub systems. All variables and functions as well as their values used for the simulative evaluation are clearly arranged in table 8.1. As mentioned before, the values listed in this table have been only chosen for the sake of comprehensibility and illustrative reasons.

7.3.1 *Specification of Payoff Values*

The IG payoff values are determined as detailed in the following.

- b: *violation benefit*
  As incentive for a violation, we assume the inspectee considering that violating (i.e. dropping a message) is worth twice as collaborating (i.e. forwarding a message). This is modelled by the weighting value $w_b = 2$. The benefit b depends on w_b and on the message load l and is defined as:

$$b = w_b l \tag{7.1}$$

- d: *non-detection costs*
  The damage represents the costs to the system in case of non-detected violations. It is equally shared over all m inspectors since they work on the system designer's behalf and inherently collaborate therefore to some degree. In order to cover any system architecture, we assume the simplified situation where the violating inspectee is in the middle of the dissemination route, i.e. he affects the half of all $n * g_s$ subscribers. Again, a weighting value w_d enables individual parameter adjustments. Note that, due to the message load dependency, the assumption $|d| > |c|$ of the abstract game model (see page 57) is not always guaranteed. The cost value is defined as:

$$d = \frac{w_d l n g_s}{2m} \tag{7.2}$$

- c: *inspection costs*
  The inspection costs value depends on mechanisms used for the inspection target. Considering the prior described concept of a proof history as inspection target, an inspector needs to control two proof histories (the one of the inspectee plus its corresponding entries of the communication partners) for the inspection of one inspectee. For the sake of simplicity, we assume that the inspection of one secured transfer has a unitary cost value. Then, the cost value is determined as in the following:

$$c = \frac{2h}{r} \tag{7.3}$$

- a: *punishment costs*
  The punishment costs represent the decrease of the inspectee's payoff due to a detected misbehaviour. It does not depend on the system architecture and can be specified by the system designer. As discussed before, the punishment (as well as the benefit) depends on an individual inspectee's importance and preferences. Here, this cost value is calculated relative to the non-detection costs for the whole systems. Therefore, since d represents the costs divided by m, it is multiplied again with m.

$$a = w_a dm \tag{7.4}$$

7.3.2 *EIP & MEIP: Effective Inspection Probabilities*

All inspectors operate independently but work on the system administrators behalf. Hence, they collaborate in some sense. For an inspection, each inspector selects a set of n_i inspectees. From a practical point of view, an inspectee fears to be inspected by any inspector. For the sake of comprehensibility and to cover all architectures, we consider the simple case of not synchronized inspectors. This means that all n_i inspectees are randomly chosen over the whole id space for an inspection. As a result, double inspections are possible.

Punishments are only done once per game. Thus, double inspections can be effectively represented as single inspections with a correspondingly increased inspection probability. This forms an EIP for the inspectee and is described by function σ (or EIP function) in equation 7.5. The calculated value denotes the ratio of inspectees in the system that are in average inspected per T for given q and n_i. For example, 0.25 means that a quarter of all inspectees are inspected per T and a value 2.0 that an inspection arrives twice per T.

$$\sigma(q, n_i) = 1 - \left(\frac{n - qn_i}{n}\right)^m \tag{7.5}$$

Similarly, the Maximal Effective Inspection Probability (MEIP) describes the maximal amount of the inspectees that can be inspected per T. It serves therefore as upper bound for the EIP value and is shown in equation 7.6. The MEIP requires that all inspectors are perfectly synchronized (no double inspections) and that they choose a value $q = 1$.

$$\sigma^*(q = 1, n_i) = \frac{qmn_i}{n} \tag{7.6}$$

Table 7.1: Summary of notation with values used in the example application of the Inspection Game.

Variable	System Values	Description
n	200	potentially malicious selfish peers in system (inspectees)
m	5	trusted peers in system (inspectors)
T	–	a time interval in reference to which all calculations are relatively done
T_{insp}	–	time interval between two inspections
T_{safe}	–	time interval needed for replacing all proof messages
t_i	–	multiplier for T so that $t_i * T = T_{insp}$
t_s	–	multiplier for T so that $t_s * T = T_{safe}$
q	*game strategy*	probability of an inspector to perform an inspection
p	*game strategy*	probability of an inspectee to violate
n_i	20	amount of inspectees to be inspected by an inspector
r	2	amount of proof messages to be stored in the history
h	200	entry size of the proof message history
g	30	amount of multi-cast groups
g_s	0.1 (10%)	percentaged amount of multi-cast groups a peer is subscribed to
g_p	0.25 (25%)	percentaged amount of multi-cast groups a publisher is publishing to
p_p	0.04 (4%)	probability to publish a message (one to all $g_{pub} * g$ disjoint groups)

Variable	Payoff Values		Description
	Scribe	*Gossiping*	
l	5.8	7	average transferred messages per T (network load per peer)
c	200	200	costs for the inspector performing an inspection
a	1740	2100	costs for the inspectee that occur due to a punishment
d	348	420	costs for the system for not detecting defections
b	11.6	14	personal benefit of a peer to defect (deviate from the protocol)
w_a	1	1	weighting value for a
w_d	30	30	weighting value for d
w_b	1	1	weighting value for b

Functions	Description
γ	violation detection function
γ^*	violation defection function, bounded to probability values
σ	calculates the effective inspection value (EIP)
σ^*	calculates the maximal effective inspection value (MEIP)
C	inspector payoff function
I	inspectee payoff function
ϕ	calculates the inspectee's best response strategy (BRS)

Since the inspectees are rational and have system knowledge, we assume that the EIP value is known or communicated by the administrator. Alternatively, an inspectee can monitor the inspections and calculate an approximation over time.

7.3.3 *Violation Detection Function* γ

The function γ indicates the probability that – given an inspection on a peer where a violation occurred within the last interval T – the violation is actually detected, i.e. the PoM is found. Function γ is presented after the clarification of some timing details.

With *history flushing* we denote the correct removal of all PoMs from the proof history. This is done if the peer transfers at least $\frac{h}{r}$ messages and replaces the current proof messages in the history. Let us consider $T_{safe} = T * t_s$ as the time for the history flushing relative to an interval T. Then, t_s has the following time value:

$$t_s = \frac{\frac{h}{r}}{l} = \frac{h}{rl}$$

Similarly, $T_{insp} = T * t_i$ is the average time after that a peer is periodically inspected exactly once. We specify t_i as in the following:

$$\begin{aligned} & \sigma(q, n_i)t_i & = 1 \\ \equiv \quad & t_i & = \frac{1}{\sigma(q, n_i)} \end{aligned}$$

Withe the notion of these time intervals, let us now consider the realistic situation where the inspectee target maximum payoff and assumes periodical inspections. In this case , the inspectee can violate $T_{insp} - T_{safe}$ but should collaborate afterwards for T_{safe} to flush out all violation proofs before the next inspection. Then, a violation is detected during an inspection if at least one PoM remains in the history. This happens in two cases:

1. The inspectee violates (probability p).
2. The inspectee violated but is still occupied to flush all violation proofs out of the history (by collaborating) as preparation for the next expected inspection . This happens with a probability of $\frac{T_{safe}}{T_{insp}} = \frac{t_s}{t_i}$.

Furthermore, we need to take into account that the collective of inspectors achieve an amount of $n_i * m$ inspections done per T and done relative to one inspectee (factor $\frac{1}{n}$). Then, the detection function γ is defined as in equation 7.7:

$$\begin{aligned} \gamma(n_i, q, p) &= \left(p + \frac{T_{safe}}{T_{insp}}\right)\frac{n_i m}{n} \\ &= \left(p + \frac{t_s}{t_i}\right)\frac{n_i m}{n} \\ &= \left(p + \frac{\frac{h}{rl}}{\frac{1}{\sigma(q,n_i)}}\right)\frac{n_i m}{n} \\ &= \left(p + \frac{h\sigma(q, n_i)}{rl}\right)\frac{n_i m}{n} \end{aligned} \tag{7.7}$$

Please note that function γ may exceed the probability bounds. Hence, the payoff function makes use of the adapted functions γ^*:

$$\gamma^*(n_i, q, p) = \begin{cases} 1 & if\ \gamma(n_i, q, p) \geq 1 \\ \gamma(n_i, q, p) & else \end{cases} \tag{7.8}$$

7.3.4 *Payoff Functions*

With the detection function γ^* we are now able to specify the complete payoff functions for the inspector and inspectee. They are created by directly transferring the IG in extensive form (see figure 5.4, page 65) into equations. These complete payoff functions are shown in equation 7.9 with $C(p,q)$ for the inspector and with $I(p,\sigma^*(q,p))$ for the inspectee. Please note that the inspectee needs to consider the EIP value instead of q.

$$\begin{aligned} C(p,q) &= p\Bigg(q\Big(\gamma^*(q,p)(-c)+(1-\gamma^*(q,p))(-d-c)\Big)+(1-q)(-d)\Bigg)+(1-p)\Bigg(q(-c)\Bigg) \\ I(p,q\equiv\sigma^*(p,q)) &= p\Bigg(q\Big(\gamma^*(q,p)(b-a)+(1-\gamma^*(q,p))b\Big)+(1-q)b\Bigg) \end{aligned} \tag{7.9}$$

The payoff functions are essential for the inspector's practical objective to attain a targeted degree of collaboration $1-p$. To this end, the inspector leverages the inspectee's rationality to choose a payoff maximizing strategy. In other words, the inspectee will choose the *best response strategy* (BRS) given that the inspector's strategy q is known. The inspectee's BRS calculation for a given q is formalized in equation 7.10 (the inspector's BRS is analogue but not shown here). Knowing this equation, the inspector needs to determine a value q such that the targeted inspectee's strategy p represents a BRS. The determination of such a value q can in our case be carried out by examining the whole strategy space. In more complex situations, e.g. when the strategy consists of a combination of multiple values, a solution could be found by optimization algorithms.

$$\phi(q) = \{p \mid I(p,q) = \max_{p\in[0,1]} I(p,q)\} \tag{7.10}$$

7.4 BRS GRAPHS FOR BEHAVIOUR ANALYSIS

Payoff matrices (see for instance table 5.1, page 58) are a common mean to outline and evaluate two player games. This is visualized with *best response strategy* (BRS) graphs to provide a mean for graphical system analysis. The players' BRS, i.e. the strategy with maximal possible payoff, is indicated with regard to a given strategy of the other player. The curves are calculated as defined in equation 7.10 for the inspector and in an analogue way for the inspectee. BRS graphs are outlined in figure 7.4. The continuous line shows the inspector's BRS for a given p while the dashed line represents the inspectee's BRS for a given q. Finally, a NE is given at the strategy combination where both lines are crossing. Please note that the inspectee (fearing to be inspected by any inspector) must consider the EIP value instead of q.

The strength of BRS graphs lies in illustrating the players' (cost-optimal) strategy decision landscape in a comprehensible way. This is shown in figure 7.5 outlining the players' behaviour for the IG approach with the values indicated in the notation summary (table 8.1). The two used pub/sub architectures have only small influence on the resulting payoff and are visually identical. Therefore, only the graphs for the Scribe system are presented. The

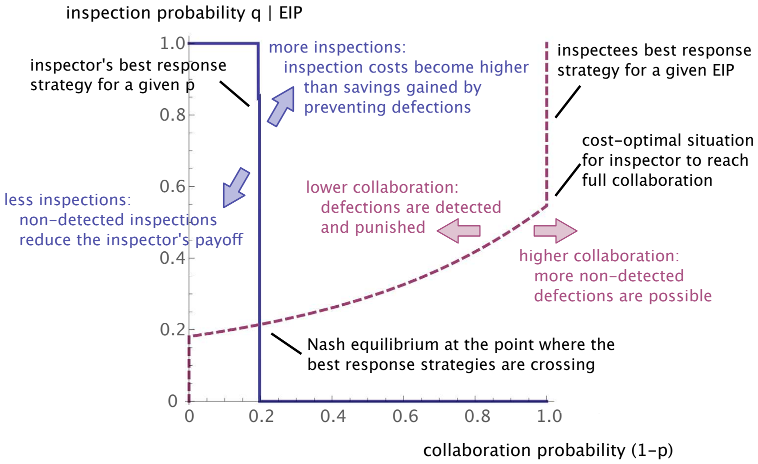


Figure 7.4: Best response strategy (BRS) graphs present the game result by visualizing the player's behaviour in an easy comprehensible way.

players' strategies are clearly visible in the left graph of figure 7.5 by a vertical and horizontal run of the curves. The NE is indicated at $p \approx 0.95$ ($1 - p \approx 0.05$) for the inspectee and at $q \approx 0.14$ for the inspector. Due to the high inspection intensity of $n_i = 20$, an inspector is able to detect an inspectee's violation for several strategy values p. Hence, it is only worth to violate for low EIP values. Here, full collaboration is reached at $EIP \approx 0.14$ for Scribe and $EIP \approx 0.155$ for the gossiping architecture. Both is reached with $q \approx 0.3$. On the right hand side, figure 7.5 shows an example for the suitability of BRS graphs as technique for evaluating parameter adjustments. Here, a reduced history size shifts the NE point correspondingly.

Note again that the used values (e.g. payoff) were chosen for comprehensibility reasons.

A NOTE TO THE BRS CURVES The BRS curves in figure 7.5 have apparently different characteristics: the inspectee's (dashed) curve is increasing while the inspector's curve has a binary nature. Considering the game mechanics (see figure 5.4, page 65), we can determine the reason. It is obvious that a higher inspection probability increases the violation detection probability. Thus, an inspectee is encouraged to collaborate for a higher value q. The actual

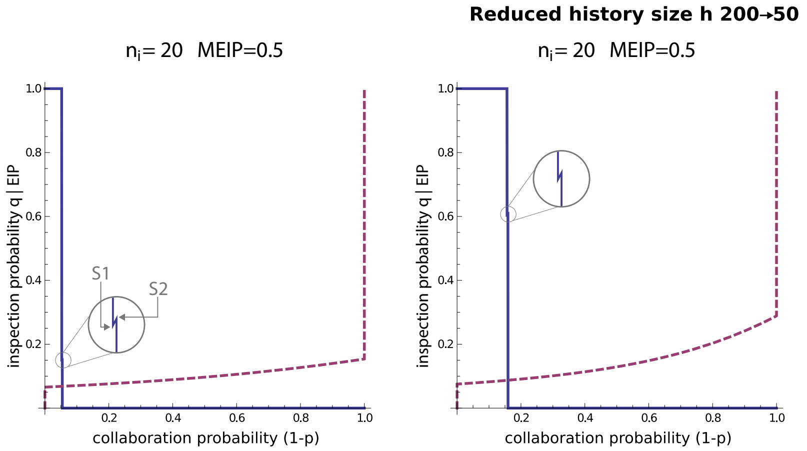


Figure 7.5: The left BRS graph illustrates the specified IG and the right possible parameter adjustments.

collaboration value is affected by function γ, producing an increasing curve. In contrary, the inspector perform a quasi binary strategy: fully inspecting or fully not inspecting. For situations $p > 0$ with $d > c$, an inspector is encouraged to inspect as much as possible to increase the violation detection probability. For $d < c$ (e.g. through high collaboration), he prefers to not inspect. Intuitively, he would switch to fully not inspecting when the expected damage pd equals the inspection costs. This is indeed the case but in detail slightly more complex. At the transition point (inspecting to not inspecting), the possibility of false negatives produces a small anomaly: the inspection probability is increasing for an increasingly collaborating inspectee. This is shown in the circle(s) in figure 7.5, indicating also two situations S1 and S2. At situation S1, the expected payoff gains by partially not inspecting equals the correspondingly increased losses. The inspector can now improve his payoff by intensifying inspections and reduce damage caused by non detection. The yielded payoff outbalances the additional inspection costs. At situation S2, the expected damage pd equals the inspection costs and the inspector changes to not inspecting for higher collaboration values.

7.5 SIMULATIONS

The IG approach introduced before is now evaluated by a simulative study. This is done straightforward way beginning with the network setup.

7.5.1 *Network Setup*

The system comprises of $m + n$ peers with $m = 5$ trusted peers (inspectors) as well as $n = 200$ selfish peers (inspectees). We assume furthermore that each peer has a unique role, either as inspector or as inspectee. Each inspectee subscribes in $T = 20$ to $g_s * g = 0.1 * 30$ groups and publishes each T (beginning in $T = 40$) with a probability of $p_p = 0.04$ one message to $g_p * g = 0.25 * 30$ groups. Hence, there are $n * g_s * g = 600$ subscriptions in the system and $p_p * n * g_p * g = 60$ new published messages per T. The actual interval length is not relevant for the IG evaluation. An interval represents a game here but does

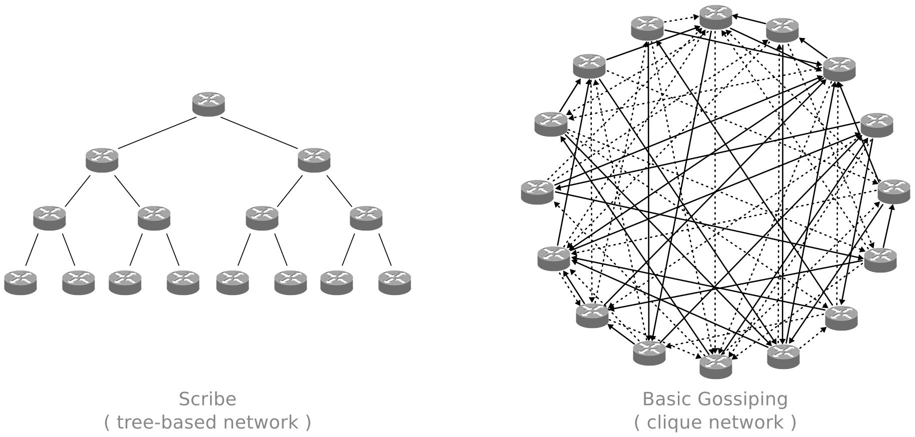


Figure 7.6: Two networks are used for the application of the Inspection Game: the Scribe system disseminates by means of a tree-based network and basic gossiping creates a clique network.

not affect the game mechanics itself. Instead, an appropriate amount of games should be considered for representative results. This is the case with a simulation duration of 500 T and correspondingly 30.000 published messages.

As mentioned before, the two pub/sub systems Scribe and basic gossiping are used for the evaluation. The latter one makes use of Cyclon [219] as membership protocol with a fanout of 10 and a partial view size of 20. These system realize different network structures, which are illustrated in simplified form in figure 7.6.

7.5.2 *Parameter Calibration*

Though the two systems realize different architectures, they differ only in the message load parameter l for the IG. As an important value for the payoff calculation, it is essential for the IG mechanics. In the given case of message drops, the message load is not only affected by the architecture but also by some game parameters such as n_i. For example, a low effective inspection leads to higher violation rates with correspondingly reduced load value.

In order to specify an appropriate value, we omit a theoretical load analysis for the sake of comprehensibility. Instead, we use load values obtained from a calibration procedure. The values l occurring in the explicit expressions of the payoffs are obtained by a Monte Carlo simulation (using $q = 1$ and $n_i = 20$) and they indicate the average message load of a peer. These are a $l = 5.8$ for the Scribe system and $l = 7$ for gossiping. An overview is presented within the simulation results figure 7.8 for gossiping in figure 7.9 for Scribe. These load values lead to payoff values that are listed in table 7.1. To summarize them, an inspectee in the Scribe system needs to violate at least $\frac{1740}{11.6} = 150$ times to balance a possible punishment of 1740 with a violation benefit of 11.6 (analogue in the gossiping architecture with $\frac{1740}{11.6} = 150$). Similarly, the system damage of 348 is almost 1.5 times the inspection costs.

7.5.3 *Simulation Setup*

The simulative evaluation of the IG approach is done with the Peersim simulator [225] and an adapted version of the RCourse library. Both systems are simulated under the conditions stated before and evaluate the IG approach in terms of the interesting point of cost-optimal full collaboration. To this end, all inspectors adjust an inspection probability $q \approx 0.3$ ($EIP \approx 0.14$), which finally results in full collaboration according the IG model. This is shown in the BRS graph (a) of figure 7.5 since it has been generated with the parameters used for the simulation. The IG is played from the beginning of the simulation. In contrary, the subscription to multicast groups starts in $T = 20$. The publishing of information starts in $T = 20$ and lasts until the end of the simulation with a duration of 500 time intervals T.

7.5.4 *Simulation Results*

At first, the simulation graphs for the EIP values are shortly presented. The main simulation results concerning the inspectees' collaboration value are presented afterwards. The graphs present the inspectees' values as median with quartiles and are therefore denoted Median

Quartile (MQ) graphs. Result graphs that show the value distributions of all inspectees (with arithmetic mean) are denoted Value Distribution (VD) graphs in the remainder. They are placed in the appendix and only presented here for the sake of clarity.

EFFECTIVE INSPECTION PROBABILITY The simulation results concerning the EIP value are shown in figure 7.7. In addition to the network setup of with $n_i = 20$ and $q = 0.3$ (centre) and $q = 1$ (right), a simulation was also done for the sake of comparison with $n_i = 1$ and $q = 1$ (left in figure). The EIP function (equation 7.5, page 83) predicts an effective inspection probability of $\sigma(1, 1) = 0.025$, $\sigma(0.3, 20) = 0.141$ and $\sigma(1, 20) = 0.41$. The MQ result graphs show that the predicted values are verified with only low variances. However, the VD graphs (see figure B.1 in the appendix on page 131) show that some values slightly differ up to about 0.1 (10%) from the predictions.

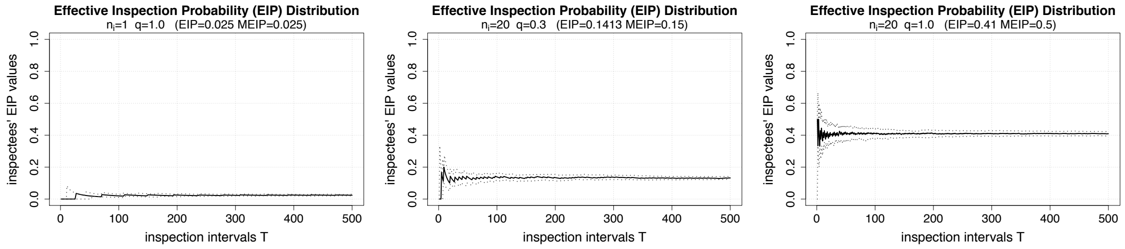


Figure 7.7: These MQ graphs verify the prediction of the EIP function, which are 0.025, 0.141 and 0.41 for the given parameters. The predictions are precise although some outliers exist with up to about 0.1 (10%) from the arithmetic mean (see VD graphs in figure B.1, page 131).

COLLABORATION ACHIEVEMENT IN PUBLISH/SUBSCRIBE ARCHITECTURES The simulation results for gossiping are shown in figure 7.8 with the payoff distribution in the first row, the message load distribution in the second row and the collaboration values $(1 - p)$ in the third row. The dissemination style leads to a homogeneous load distribution for the peers. Hence, the quartile curves are almost equal to the median. Thus, the prediction of $q = 0.3$ is precise and sufficient to reach full collaboration at all inspectees. There are some outliers during a short stabilization period as can be seen in the VD graphs given in the appendix (figure B.3 on page 132).

The left column in figure 7.9 simulation results for Scribe using the same parameters and graph placement as for gossiping. The additional fourth row shows the collaboration values $(1 - p)$ as VD graph for comprehensibility reasons. Due to Scribe's to the tree-based dissemination, the load distribution differs strongly to gossiping. There are several fluctuations since the actual load depends on the position in the tree. Nevertheless, Scribe provides an efficient dissemination relative to gossiping (see y-axis scales). The desired collaboration value is exactly reached as shown in the MQ graph. This is verified by the VD graph (bottom in figure): the mean hits almost exactly the targeted full collaboration. However, the VD graph reveals several outliers due to the heterogeneous load distribution. These variances are not visible in the MQ graphs due to the median's statistical robustness. Thus, the predicted inspection value $q = 0.3$ (left column), resulting in $EIP = 0.14127$, is considered not completely sufficient.

As a comparison, the right column of figure 7.9 shows the results with the inspection probability being dramatically increased from $q = 0.3$ to $q = 1$. Here, the amount of inspections is high enough to prevent violations from the early beginning. This outlines the difficulty in terms of specifying an appropriate inspection probability value. In systems such as Scribe, the message load value depends on the position in the dissemination tree. However, this is typically not known to the IG model (e.g. the root node of a tree is typically elected by a hash value of the group id and the inspectee's overlay id). As a result, the tree-based dissemination such as Scribe always provides a variance in terms of load. Thus, precise predictions of the needed inspection probability remain difficult for this IG model.

7.6 SUMMARY

In this chapter, the IG approach was applied to distributed systems using two Publish/Subscribe systems. The scenario of the introduction with selfishness-driven message drops was considered as illustrative use case and details for an application were introduced. The simu-

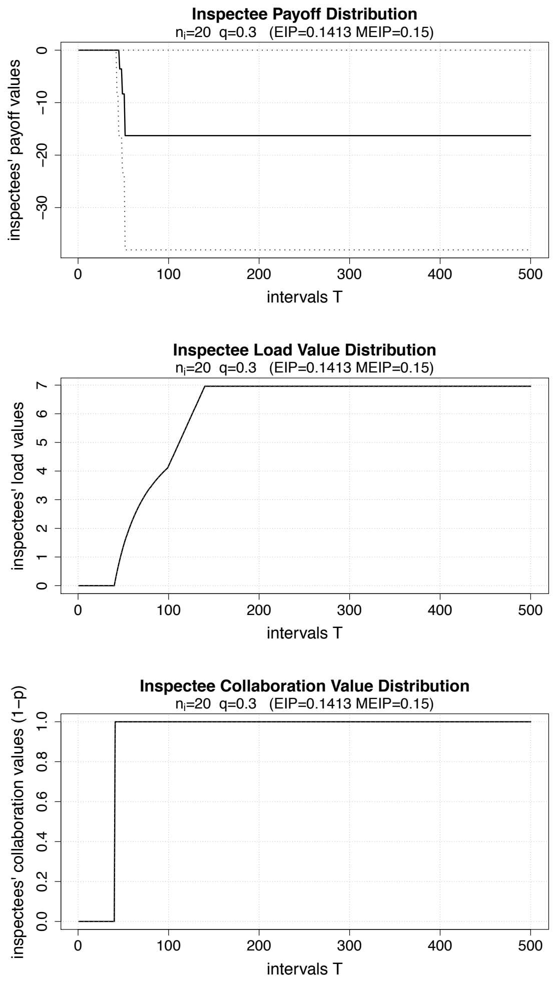


Figure 7.8: The result graphs for the gossiping system show the inspectees' payoff (first row), message load (second row) and collaboration values (third row).

lation addressed among others the inspectees' collaboration degree and verified the general functioning of the IG approach. It can already be considered as a promising approach that reaches the thesis' vision. The evaluation outlined also some drawbacks caused by the IG model's stiffness: it is not yet able to deal with dynamic conditions. This is addressed in the next chapter.

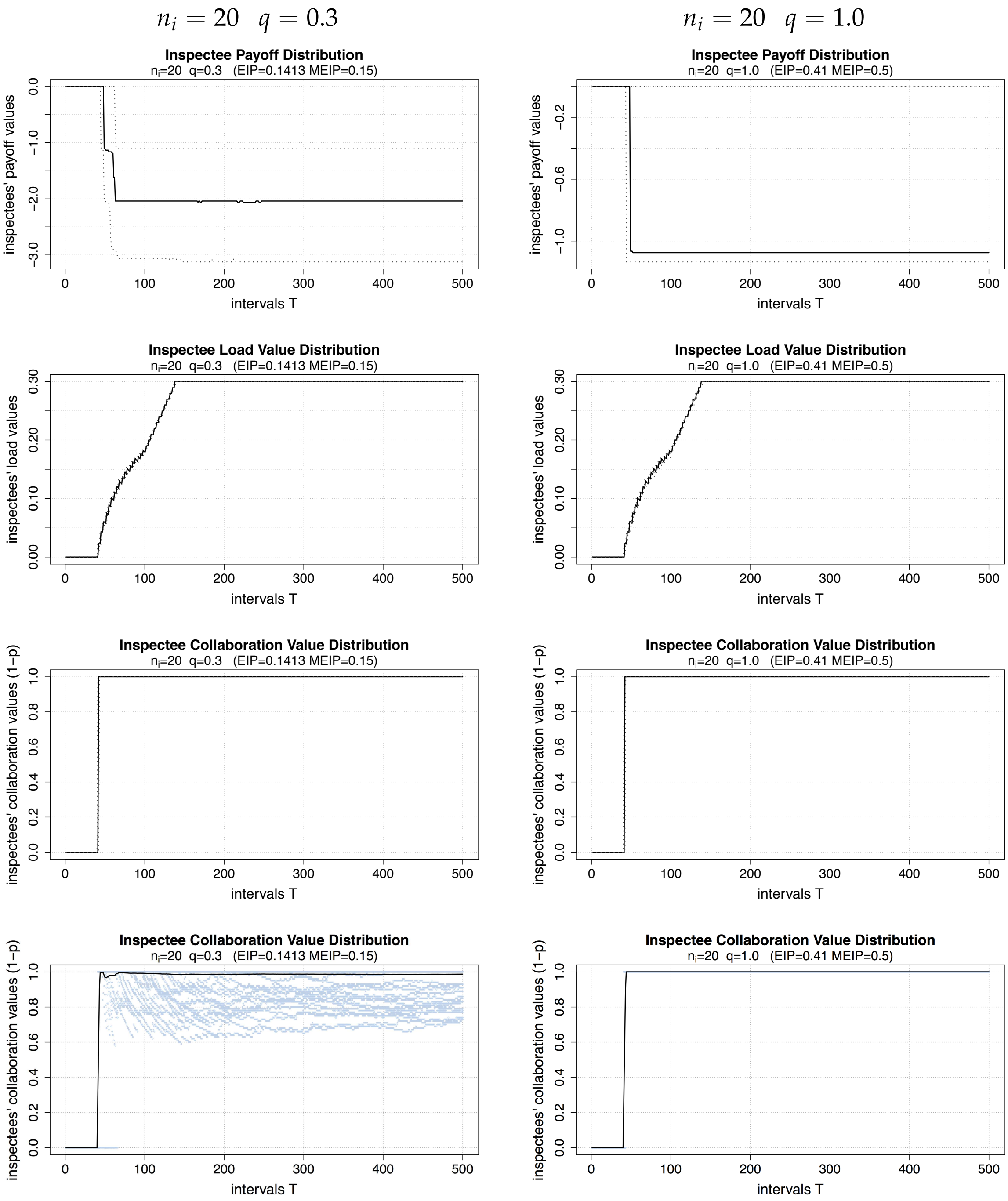


Figure 7.9: The graphs show the simulation results for the Scribe system (all corresponding VD graphs are presented figure B.2 in the appendix on page 131). The results comprise the inspectees' payoff (first row), message load (second row) and collaboration values (third, fourth row).

8 AN ENHANCED DYNAMIC INSPECTION GAME FRAMEWORK

This chapter presents the enhanced Inspection Game (IG) approach that enriches the initial one of the foregoing chapter by some degree of dynamics. This chapter introduces basically the several differences to the initial approach. For specific details (e.g. IG fundamentals, inspection target details, payoff functions), the reader is referred to the foregoing chapter.

8.1 APPROACH OVERVIEW & DIFFERENCES TO THE INITIAL APPROACH

The improvements address two important shortcomings of the initial version: varying message load as well as varying inspection rates. The enhanced IG approach considers not only the same scenario as the initial version, it has also basically the same characterization. It as played in *(internal) reactive* mode, i.e. the inspectors initiate the inspections. The continuous operation is also faced with *periodical game splittings*, i.e. the operation time is equally splitted into time intervals T. However, there are two major differences, which are detailed afterwards. These are the adaptation to system dynamics (covering the aforementioned shortcomings) and a change of the inspection architecture.

8.1.1 *Inspection Architecture as Additional Network Layer*

The initial IG approach has some secured logic as network layer at the inspectee's machine. Hence, it makes sense to deploy directly the full inspector logic, e.g. as part of the application. The resulting inspection architecture is illustrated in figure 8.1. With this architecture, each inspectee has inherently an inspector assigned. Furthermore, it makes sense to inspect

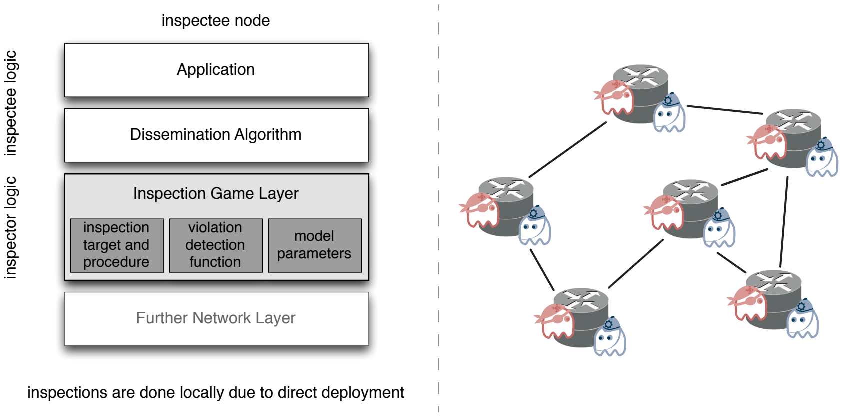


Figure 8.1: In the enhanced IG approach, an inspector is completely deployed at the inspectee's peer (left in figure). The resulting inspection architecture is shown right in the figure. Inspectors control only the local inspectee, which creates sets of two-player games.

always the local inspectee due to the efficient local inspection. Then, the whole IG logic is encapsulated into each peer. In other words, we deal with sets of two-player games.

This architectural type has several advantages compared to the initial approach. In addition to the efficient inspection at least of the local inspectee's inspection target, the inspectors uniquely assigned to an inspectee in terms of inspections. Hence, they are inherently synchronized and double inspections are not possible. This simplifies also the IG model: the EIP and MEIP values can be omitted. Now, only the one assigned inspector's strategy q must be considered by an inspectee during the own strategy choice. This supports the system designer applying the IG to individual needs and simplifies BRS graphs as mean for system analysis.

8.1.2 *Adaptation to System Dynamics*

In order to attain a dynamic IG model, the enhanced version has the following objective: *a whole interval T shall be secured by providing proofs for all operations within such time interval.* To this end, the inspection target will be – in contrary to the initial IG approach – dynamic in its size. This normalizes the meaning of a peer's position in the dissemination structure. Those with a high message load need to maintain a correspondingly large inspection target. As we will see in the next section, this objective has also a simplifying effect on the violation detection function γ.

The enhanced version does not use a parameter calibration. Instead, all payoff values are calculated in a dynamic way by adapting the current system state. This is done by monitoring the message load value. Furthermore, a more stable approximation is calculated over time. The monitored message load is in the following denoted as l' and the approximation l^+. Similarly, the players monitor the other player's behaviour in order to approximate the strategy over time. This is wanted by the system designer to sharpen the inspectee's possible calculations. The inspectee is informed when an inspection occurs, while the inspector approximates by means of detected violations. This information is denoted as q^+ and p^+, respectively. The calculation of these values will be detailed in the next section.

8.2 SPECIFICATION OF GAME DETAILS

The details of the enhanced IG approach's major differences to the initial one are introduced now. As in the last chapter, all variables and their values are clearly arranged in table 8.1.

8.2.1 *Inspection Target*

The enhanced IG version uses the same inspection target mechanism as the initial version; it is thus not detailed here. However, the different inspection architecture causes a slight change of the inspection sequence, which is visualized in figure 8.2. Being physically present at the inspectee, an inspector makes only a copy of the local inspection target. Then, he requests copies of those communication partners indicated in the local one. The actual control of the inspectee (inspection procedure) is done and a punishment induced in case of a detected PoM. A small implementation detail shall be mentioned here. The delay of requesting and

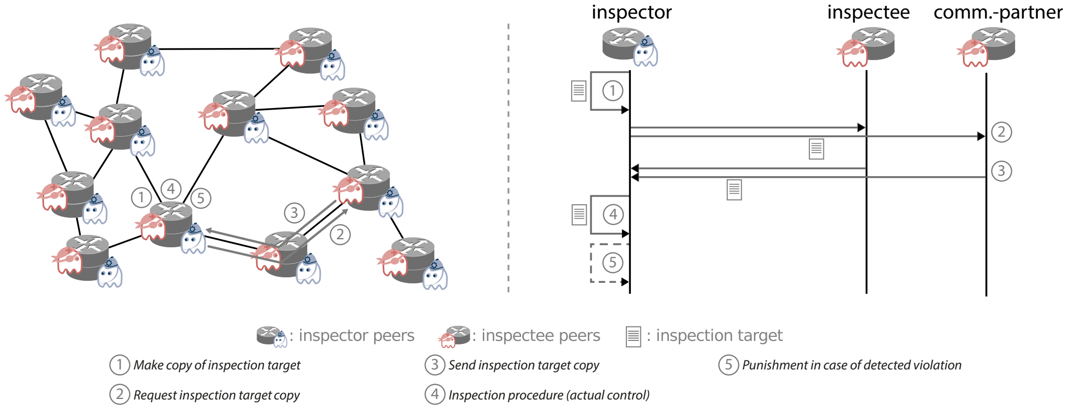

inspector
inspectee
comm.-partner
1
2
3
4
5
: inspector peers
: inspectee peers
: inspection target
1 Make copy of inspection target
2 Request inspection target copy
3 Send inspection target copy
4 Inspection procedure (actual control)
5 Punishment in case of detected violation

Figure 8.2: The inspection sequence of the enhanced IG is shown in the context of a inspection architecture (left) and schematically focusing on the players' interactions (right).

transmitting the copy of the others may lead to a loss of the last recent proof messages and thus possibly a PoM. To cope with this problem, the inspection target contains a further backup history. Proof messages that should actually be removed from the history will be deposited in the backup history, again in a FIFO manner. However, this additional history serves only as mean to make the corresponding messages accessible. Proof messages that are not related to the corresponding IG are ignored.

The proof history is dynamic in size. It can be considered as a data structure such as a linked list, which grows with the message load that actual occurred. However, for the payoff calculation, a history size must be determined for a game, whose exact message load value is not yet known. Therefore, an expected history size is used as shown in equation 8.1. The expected history size h^+ considers an expected load value l^+ multiplied with the amount of proof messages r per message transfer. The value h^+ will be used during the payoff value specification, while the expected load l^+ is detailed afterwards in section 8.2.3.

$$h^+ = rl^+ \tag{8.1}$$

8.2.2 *Payoff Values*

The payoff values differ from those of the last chapter in terms of two aspects: the monitoring of the message load value (represented by l') and the the omission of the EIP value. Here, we concentrate rather on these differences in the explanations. For further information about their composition, the reader is referred to the initial IG (section 7.3.1, page 82).

- b: *violation benefit*
  As for the initial IG, violating is considered worth twice as collaborating, designed by means of a weighting value $w_b = 2$. With the monitored message load, the benefit calculated as in the following:

$$b = l'w_b$$

- *d: non-detection costs (damage)*
  For the system damage, we assume again that the violating peer is in the middle of the dissemination route to all $n * g_s$ subscribers. In the initial IG model, m inspectors were equally sharing the damage, thus having the value m in the denominator. This is not the case here due to the fixed assignment of an inspector to an inspectee. Hence, m is omitted and the cost value is defined as:
  $$d = \frac{w_d l' n g_s}{2} \tag{8.2}$$
- *c: inspection costs*
  This cost value depends on mechanisms used for the inspection target and is equal to the initial version. However, please remember that the history size h is actually a dynamic value, depending on the message load. Since actual message load is not yet known for the current game, the cost calculation uses the expected history size of equation 8.1:
  $$c = \frac{2h^+}{r} \tag{8.3}$$
- *a: punishment costs*
  The punishment costs are again calculated relative to the non-detection costs. Here, the variable m is – analogue to damage d – omitted due to the pure two-player game:
  $$a = w_a d \tag{8.4}$$

8.2.3 *Value Approximations for System Dynamics*

As mentioned the rational players may draw conclusions from the circumstances. Therefore, they monitor some values and calculate approximation to support their game strategy decision. To this end, several possibilities are available such as the arithmetic mean, a median calculation or *exponential smoothing*. The latter one is a well-known technique for time series of data. Hence, it is appropriate for value approximations over time and will be used here.

The load value approximation l^+ for the next IG related to interval T is calculated based on three values. This is the monitored load value l' (amount of transferred messages) for $T-1$, its prior approximation and a smoothing factor α_l. The message load approximation is shown in equation 8.5. For the simplicity's sake, we omitted the index T and use l^+ instead of l^+_T.

$$l^+ = \alpha_l l'_{T-1} + (1-\alpha_l) l^+_{T-1} \tag{8.5}$$

Similar to the message load for the inspector, the inspection probability represents an important value for the inspectee, which possibly dynamic. The inspectee approximates for the BRS calculation a smoothed inspection probability q^+. To this end, the inspectee takes into account if he suffered or not from an inspection during the last interval, which is represented by $q'_{T-1} \in \{0,1\}$. The approximation is done with the same principle as for the message load and shown in equation 8.6 (analogue to l^+ we omit index T for q^+_T).

$$q^+ = \alpha_q q'_{T-1} + (1-\alpha_q) q^+_{T-1} \tag{8.6}$$

Table 8.1: Notation summary with values used during the enhanced Inspection Game application.

Variable	Values	System Value Description
n	200	amount of peers in system
T	–	time interval used for calculations
T_{insp}	–	average time interval between two inspections
T_{safe}	–	average time interval for replacing all proof messages
t_i	–	multiplier for T so that $t_i * T = T_{insp}$
t_s	–	multiplier for T so that $t_s * T = T_{safe}$
q	*game strategy*	probability of an inspector to perform an inspection
p	*game strategy*	probability of an inspectee to violate
l'	*dynamic value*	monitored message load value of an interval T
q'	*dynamic value*	monitored value if an inspection happened or not ($q' \in \{0, 1\}$)
p'	*dynamic value*	monitored value if a violation happened or not ($p' \in \{0, 1\}$)
l^+	*dynamic value*	approximation to the message load (# of transferred messages)
q^+	*dynamic value*	approximation to the inspector's strategy q
p^+	*dynamic value*	approximation to the inspectee's strategy p
h^+	*dynamic value*	approximation to the expected history size (used for payoff calculation)
h	*dynamic value*	size of the proof history (inspection target)
α_q	0.1	smoothing factor for the inspection probability approximation
α_p	0.1	smoothing factor for the violation probability approximation
α_l	0.5	smoothing factor for the message load approximation
r	2	amount of proof messages to secure one hop-to-hop transfer
g	30	amount of multi-cast groups in the system
g_s	0.1 (10%)	percentaged amount of groups a peer is subscribing to
g_p	0.25 (25%)	percentaged amount of groups a publisher is publishing to
p_p	0.04 (4%)	probability to publish one message to $g_p * g$ groups (*basic load*)

Variable	Values	Payoff Values Description
b	*dynamic value*	violation benefit of a peer deviating from the protocol
d	*dynamic value*	costs for the system for not detecting violations
c	*dynamic value*	costs for the inspector performing an inspection
a	*dynamic value*	costs for the inspectee that occur due to a punishment
w_a	0.5	weighting value for a
w_d	2	weighting value for d
w_b	2	weighting value for b

Functions	Description
γ	violation detection function
γ^*	violation detection function, bounded to probability values

In the same way, the inspector may approximate the inspectee's chosen strategy, the violation probability p. This is shown in equation 8.7 with $p'_{T-1} \in \{0, 1\}$ for detected violations. To to take also non-detected violations into account, the factor $\frac{1}{q\gamma}$ is added (again, omitting index T for p^+_T).

$$p^+ = \frac{1}{q\gamma}(\alpha_p p'_{T-1} + (1 - \alpha_p) p^+_{T-1}) \tag{8.7}$$

Approximating the inspectee's strategy is introduced here for the sake of completeness. Although not used in the remainder, the inspectee's strategy is possibly dynamic, too. Then, such approximation may be used to discover the individual preferences or as basis for feedback mechanisms in order to adapt the inspection intensity.

The low smoothing factors for the strategy approximations $\alpha_q = \alpha_q = 0.1$ provide some smoothness, i.e. statistical robustness against fluctuations. In contrary, the factor $\alpha_m = 0.5$ is higher to adapt faster to the message load.

8.2.4 *Violation Detection Function γ*

The violation detection function γ is substantially affected by the modification that the inspection target is dynamic in size. In order to make the consequences comprehensible, we will shortly recall the timing details that were discussed for the initial IG in section 7.3.3 (page 84).

To flush the proof history, all PoMs need to be removed from proof history, which is done if the peer transfers at least $\frac{h}{r}$ messages. As in the last chapter, the needed time can be considered as $T_{safe} = T * t_s$ with t_s being detailed in the following. Since function γ is used for the strategy choice, the expected values l^+ and h^+ are used here.

$$t_s = \frac{\frac{h^+}{r}}{l^+} = \frac{h^+}{rl^+} = \frac{rl^+}{rl^+} = 1$$

Furthermore, the average time interval between two inspection shall be denoted by $T_{insp} = T * t_i$ with t_i being detailed as in the following:

$$\begin{aligned} & t_i q &= 1 \\ \equiv \quad & t_i &= \frac{1}{q} \end{aligned}$$

PoMs are still in the history during an inspection for either of the following two cases. Being equal to those of the last chapter, they are only listed again for the sake of comprehensibility.

1. The inspectee violates (probability p).
2. The inspectee violated but collaborates now to flush all violation proofs out of the history.This happens with a probability of $\frac{T_{safe}}{T_{insp}} = \frac{t_s}{t_i}$.

With this preparation, we can define function γ as shown in equation 8.8:

$$\begin{aligned} \gamma(q,p) &= p + \frac{t_s}{t_i} \\ &= p + \frac{1}{\frac{1}{q}} \\ &= p + q \end{aligned} \tag{8.8}$$

Here, we can see the effect of the dynamic adaptation of the history size able to store all proof messages for one T. The system dynamics in terms of varying message load is compensated and the detection probability not any more depending on the proof history or message load values. Nevertheless, this function may still exceed the probability bounds. Analogue to the initial IG approach, this will be considered in the adapted function γ^*, which is used for the payoff functions.

$$\gamma^*(q,p) = \begin{cases} 1 & if\ \gamma(q,p) \geq 1 \\ \gamma(q,p) & else \end{cases} \tag{8.9}$$

8.3 PAYOFF FUNCTIONS & BRS GRAPHS

Since the EIP value is not used anymore, the inspectee considers the inspector's strategy q instead of $\sigma^*(p,q)$ for payoff functions. These are $I(p,q)$ for the inspectee and $C(p,q)$ for the inspector. Both have the same structure as those of initial IG approach (see section 7.9, page 86) but make use of the formalizations of this chapter (e.g. payoff values, function γ). Please not that, for an implementation, the inspector would replace p by the approximation value $p^+$and the inspectee q by q^+.

The BRS graph based on the payoff functions of the enhanced IG approach is shown in figure 8.3. The omission of the EIP values increases its comprehensibility: the dashed purple curve indicates the inspectee's BRS only with respect to the inspector's game strategy q. The inspector's curve remains without changes: the continuous blue curve indicates his BRS for a given inspectee's game strategy p.

Three targeted collaboration values are indicated in this BRS graph together with the corresponding inspection probabilities. These are the collaboration value $(1-p) = 0.6$ with the predicted inspection probability of $q = 0.2$, a collaboration value $(1-p) = 0.8$ with the corresponding inspection probability $q = 0.29$ and finally a full collaboration $(1-p) = 1.0$, which is reached at $q = 0.45$ according to the IG mechanics.

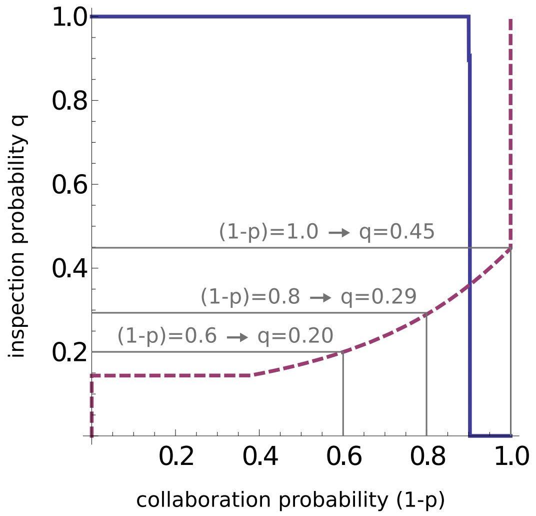


Figure 8.3: The graph indicates the players' best response strategies (BRS) for the enhanced IG approach and the given parameters (see notation summary). Three inspection probabilities are indicated with the resulting collaboration degrees according the the IG model.

8.4 SIMULATIONS

After presenting all game details, the enhanced IG is evaluated by a simulation. Again, this is done with Peersim and an adapted version of RCourse using Scribe and basic gossiping. All parameter values used during the simulation are indicated in the notation summary.

8.4.1 *Network & Simulation Setup*

The system consists of $n = 200$ inspectees but no dedicated inspector peers since the logic is integrated as IG network layer to the inspectee peers. In order to verify some dynamics, the simulation lasts 1000 time intervals T. The subscriptions as well as generated traffic is equal to the initial IG of the last chapter. The 60 published messages per T (each one having in average 20 subscribed inspectees) is considered as *basic load*.

The evaluation especially focuses on the system dynamics. This is done by varying the message load as well as the inspection probability twice during runtime. Latter one represents a change of targeted collaboration levels through a varying inspection intensity. A simulation overview is shown in figure 8.4. The simulation starts with basic load starts in time interval $T = 40$ and lasts until the half of the simulation ($T = 500$). Then, it is increased to the triple (180 published messages per T). This high load persists until $T = 800$, where it is decreased to the double of the basic load (120 published messages per T). This persists until the end of the simulation. Altogether, the simulation consists of 105600 published messages (further message transfers not included). In addition to the changing message load, the simulation provides dynamics in terms of the targeted collaboration levels by injecting new inspection probabilities q during the simulation. The three values are used that were already indicated in figure 8.3. In the beginning, we target a collaboration value of 0.8 ($q = 0.29$). In $T = 350$, the targeted degree is changed to 0.6 ($q = 0.20$). Full collaboration is then targeted in $T = 700$, which lasts until the end of the simulation.

Let us denote the term *epoch* as a set of sequential time intervals that provide the same parameters. By changing two parameters (message load and inspection rate) twice, the simulation consists of five epochs, i.e. four parameter changes (figure 8.4). This dynamics is simplistic to some degree (e.g. no use real world work load traces). However, it enables to evaluate the general adaptability of the IG approach and to draw conclusions from its preferences.

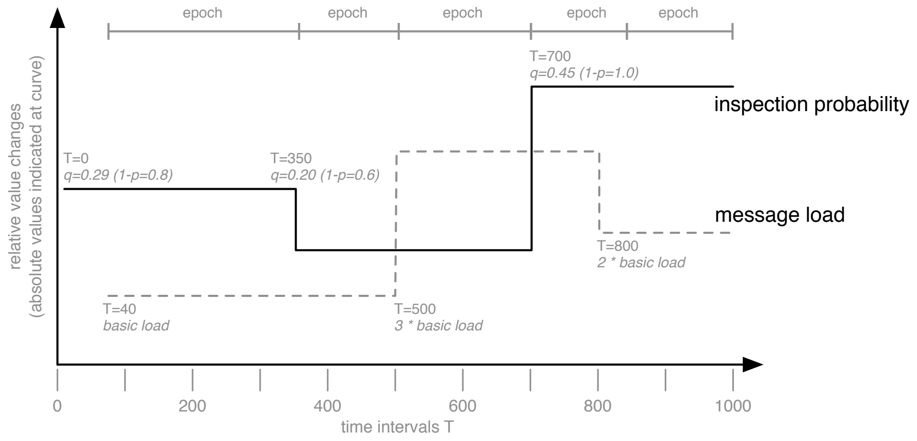


Figure 8.4: During the simulation, the message load and inspection rate, are changed twice.

8.4.2 *Simulation Results*

As in the last chapter, only the *MQ graphs* (median with quartiles) are presented here, while the corresponding *VD graphs* are available in appendix C (page 133). Please note again that the arithmetic mean in VD graphs is not equal to the median due to statistical reasons.

APPROXIMATION TO INSPECTION PROBABILITY Figure 8.5 shows the inspectees' approximations to the varying inspector game strategy. Precise approximations of the three adjusted inspection probabilities 0.29, 0.2 and 0.45 are noticeable (though small variances are existent). The graph shows also the adaptation speed of the smoothed value. The actual value is approximated within about 25 time intervals with the given parameters.

MESSAGE LOAD The further simulation results of the enhanced IG approach are shown in figure 8.6. MQ graphs are given for both the Scribe system (left column) and the gossiping system (right column). The first row shows the message load approximations as described in the game details section, the middle row the chosen collaboration values and the bottom row the inspectees' payoff value. The load graphs clearly show the three adjusted load values. Here, the Scribe's efficient but heterogeneous load distribution provides some variances. In contrary, all values of the gossiping system are close to the median but has higher load values.

COLLABORATION VALUES The middle row of figure 8.6 shows the simulation results in terms of the collaboration. Both examined systems attain – despite system dynamics and different dissemination styles – in average the three desired collaboration degrees 0.8, 0.6 and 1.0. This verifies the general functioning of the IG approach and also the ability to adapt to system dynamics. A small variance is given of about 0.1 (quartiles) from the median. The adaptation speed is with about 25 intervals analogue to figure 8.5. This makes sense due to the importance of the inspection probability approximation for the inspectee. The collaboration graphs have one conspicuous feature: there are some fluctuations for 0.6. The reason is comprehensible by means of the BRS graph (figure 8.3). For a targeted collaboration value of about $1 - p \approx 0.39$, it becomes not worth anymore for the inspectee to collaborate. The value drops directly down to zero. Hence, due to variances, some inspectees consider a tar-

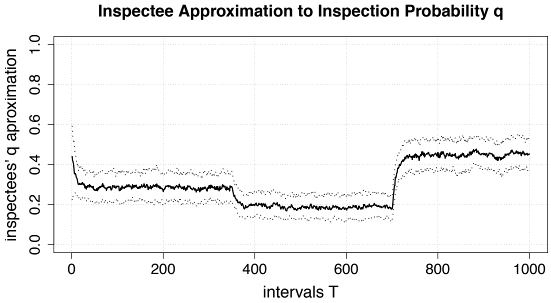
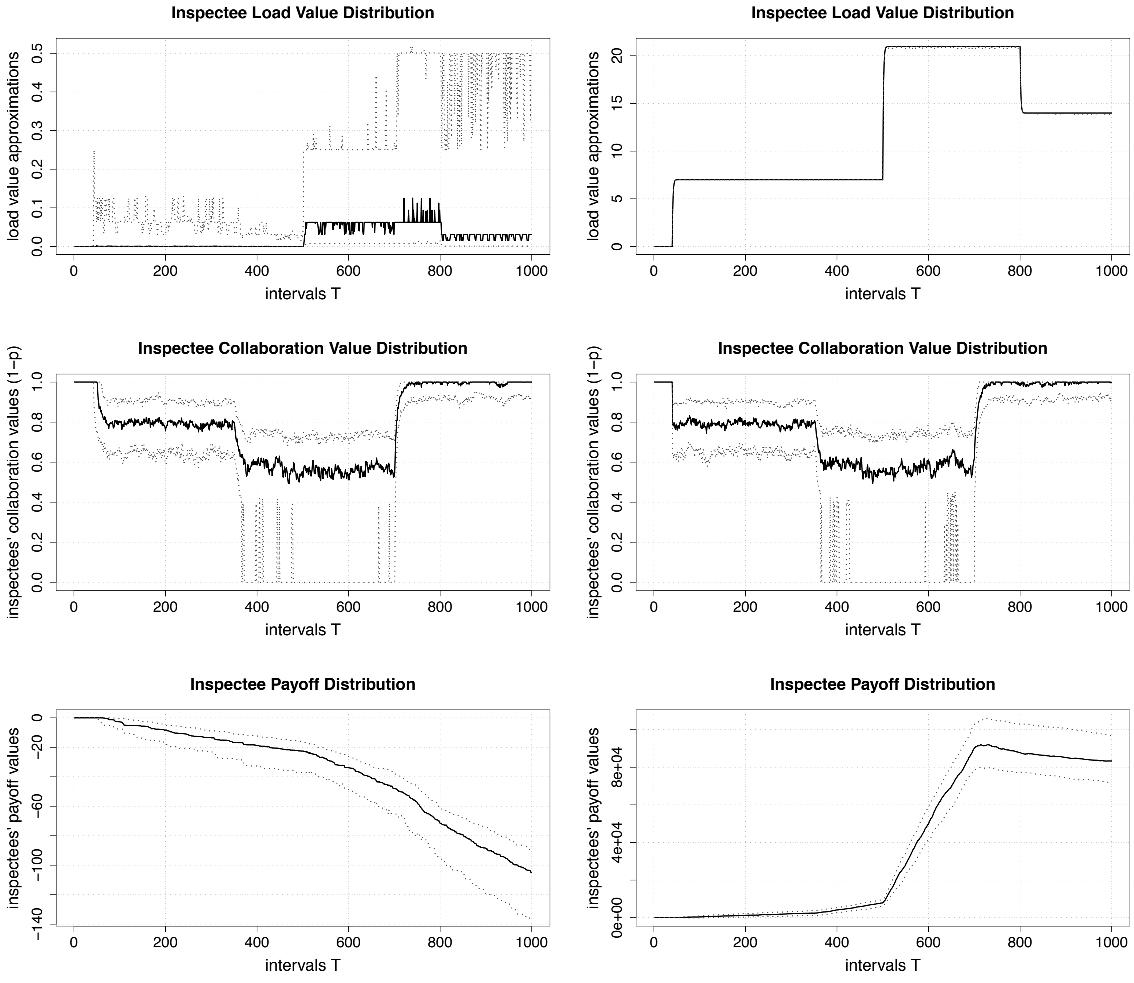


Figure 8.5: The graph shows the inspectees' approximations to the varying inspector game strategy in the Scribe system. The curves – median with quartiles – are analogue to the gossiping system (not shown here). See figure C.1 (page 133) for the corresponding VD graph.



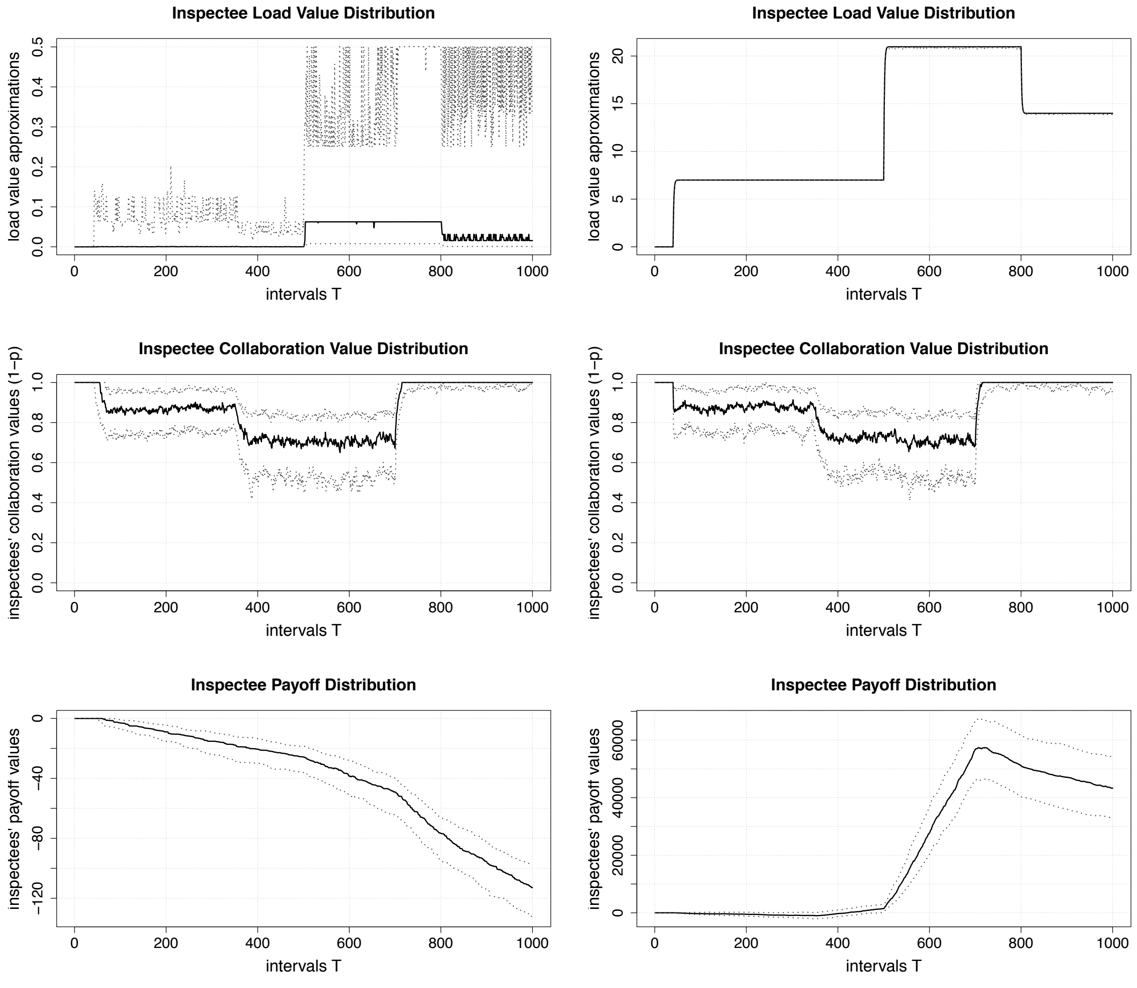
Scribe

Basic Gossiping

Figure 8.6: The graphs show the simulation results for Scribe (left column) and gossiping (right column). The corresponding VD graphs are given in the appendix (figure C.2, page 133).

geted collaboration below this threshold: they do not collaborate anymore. This is clearly shown in the corresponding VD graphs in figure C.2 (in appendix on page 133). In order to verify this aspect, a further simulation has been done with an inspection probability slightly increased by 0.05. This results in targeted collaboration values of ~ 0.87, ~ 0.7 and 1.0. The result graphs are presented in figure 8.7, arranged in the same way as in figure 8.6. Now, the fluctuation should be reduced. Furthermore, the targeted collaborations are again achieved as predicted.

PAYOFF Although using the same simulation setup, the payoff graphs (bottom row in figure 8.6) are differing significantly for both systems. Hence, the different dissemination structure must be the reason, which is interpret as in the following. In the Scribe system, the heterogeneous message load values cause more outliers in terms of the targeted collaboration value. They are in turn detected and punished and the average payoff keeps negative during



Inspectee Load Value Distribution

Inspectee Load Value Distribution

Inspectee Collaboration Value Distribution

Inspectee Collaboration Value Distribution

Inspectee Payoff Distribution

Inspectee Payoff Distribution

Scribe

Basic Gossiping

Figure 8.7: The inspection probabilities were increased by 0.05, resulting in targeted collaboration values ~ 0.87, ~ 0.7 and 1.0. Thus, the collaboration value variances are reduced for the epochs with $1 - p \approx 0.7$. The VD graphs are shown in figure C.3 (page 134).

the whole simulation. Especially for the targeted full collaboration value the punishments of detected violations affect the whole averaged payoff without being mitigated (or only of a negligible amount) by the increase through non-detected violations. The payoff values of the gossiping system are (due to the homogenous load values) more deterministic: the lower the collaboration value the higher the payoff through violation benefits. Similar to the Scribe system, some outliers are given for full targeted collaboration. The imposed punishments outbalance the increase of (also possibly not detected) violation benefits.

8.5 SUMMARY

This chapter introduced the enhanced IG approach, which is based on the initial one of the last chapter 7. It enables a system deployment over selfish peers, adapting to system dynamics such as a varying message load or targeted collaboration levels. Hence, the IG meets the research challenge (C) stated in the introduction.

The differences of the enhanced IG to the initial one consist basically in another inspection architecture and the ability to cope with system dynamics. The enhanced version realizes the inspector as network layer at the inspectee's machine. Furthermore, to cope with dynamics, it follows the general objective to secure a whole interval T. To this end, all related collaboration proofs are stored in an inspection target that is dynamic in its size.

The chapter presented all details and showed how to apply the IG framework to a specific use case. A simulative evaluation verified the functioning of the approach. The dynamics were evaluated by means of epochs, i.e. phases of not changing system parameters. Adaptations to new environmental parameters are done within about 25 time intervals, which can be adjusted by game parameters.

9 A DISCUSSION OF THE APPLICABILITY TO REAL WORLD SYSTEMS

After the introduction of the IG approach in the last chapter, we discuss now some real world aspects. This is the possible implementation into user applications and the practical meaning as reliability technique for user and administrator.

9.1 THE INSPECTION GAME AS RELIABILITY FRAMEWORK

The IG framework is not a reliability technique in itself. Instead, as a monitoring approach, the implemented mechanisms are taken into account to control the correct functioning of a peer. All information are integrated into the IG model and collaboration incentives are given as needed. It is independent from the system architecture and the considered reliability mechanisms can be chosen correspondingly to individual needs. The IG framework is versatile and can for example be applied to any system where logs are generated.

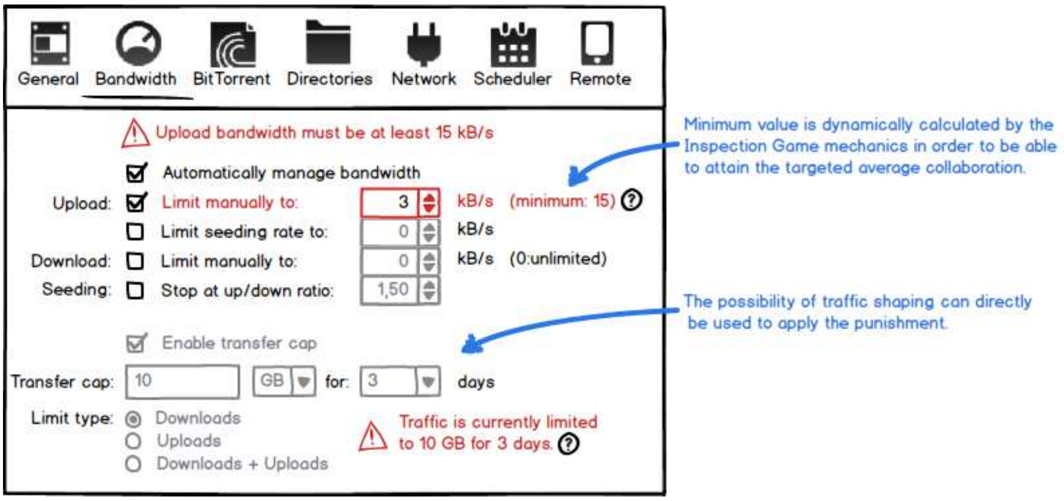
The practical objective of the IG framework is (among a behaviour analysis) to attain a targeted collaboration level. In contrary to the thesis' assumptions, real world systems provide typically altruistic peers. Hence, the actually reached collaboration will be (to some degree) higher than the targeted one. Due to this beneficial effect, the resource utilization can be further reduced if the actual amount of violations is considered.

Despite its flexibility, the framework design is not a universal remedy. The peer's goal(s) and preferences are typically not known but only how to prevent specific failures. The mechanism choice can also be challenging concerning the trade-off between reliability and resource utilization. From a comprehensive point of view, the IG framework has two major challenges:

- *Technical: A peer's misbehaviour must clearly be provable.*
  This challenge consists of creating secured (not circumventable) proof logs and to provide means for a secured and correct inspection of these logs.

- *Structural: A peer's individual preferences must be mapped to game parameters.*
  Preferences such as the violation benefit are differing along the peer's and unknown to a system designer. He must estimate them to design appropriate game mechanics.

The technical challenge are under active research and neglected for the discussion here. However, the structural challenge is an important and unsolved drawback for real world application. To demonstrate the proof-of-concept, it was compassed in the thesis by using values that are equal and supposed to be correct for all peers. This structural drawback will be in the following (among others) illustrated by means of the BitTorrent application.



⚠ Upload bandwidth must be at least 15 kB/s

☑ Automatically manage bandwidth

Upload: ☑ Limit manually to: 3 kB/s (minimum: 15) ?

☐ Limit seeding rate to: 0 kB/s

Download: ☐ Limit manually to: 0 kB/s (0:unlimited)

Seeding: ☐ Stop at up/down ratio: 1,50

☑ Enable transfer cap

Transfer cap: 10 GB for: 3 days

Limit type: ● Downloads ○ Uploads ○ Downloads + Uploads

⚠ Traffic is currently limited to 10 GB for 3 days. ?

Minimum value is dynamically calculated by the Inspection Game mechanics in order to be able to attain the targeted average collaboration.

The possibility of traffic shaping can directly be used to apply the punishment.

Figure 9.1: The IG mechanics can interact the user interface, as exemplary shown here for BitTorrent.

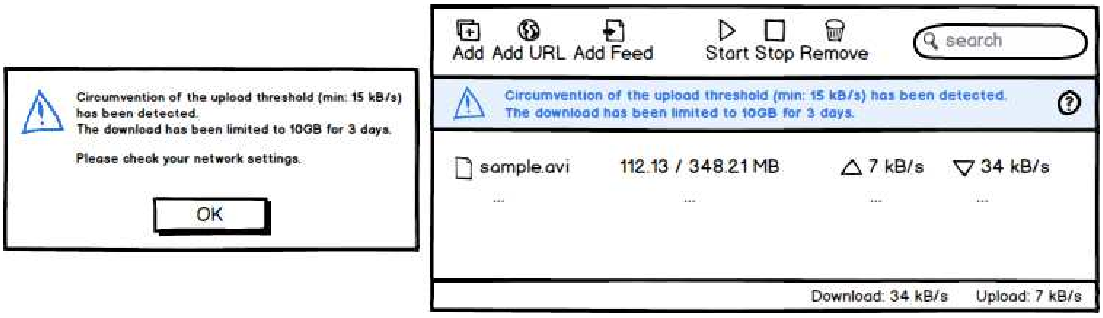
9.2 THE INSPECTION GAME IN USER APPLICATIONS

This section discusses the related between and user interfaces. To this end, the BitTorrent application is used that was presented in the introduction (see figure 1.1 left on page 4).

9.2.1 *Meaning for the user*

The majority of personal goals require modifications of the application logic itself or the infrastructure. However, some applications provide options that can be considered as selfishness-driven violations. An example is the BitTorrent user interface, shown as mockup in figure 9.1. Limiting or capping up-/download bandwidths is a feature of the application functionality. Then, only the permitted load is transferred and remaining network messages are dropped. This has direct impact on IG's collaboration value p.

However, the user interface can in turn be used to achieve collaboration as illustrated in figure 9.1. A minimum upload rate of 15 kB/s is required, while the user tries to use 3 kB/s. Such regulating mechanism can already be interpreted as punishment by the user. In fact, reducing the bandwidth has been used in the last chapters as punishment in the IG. The lower part in the figure illustrates how a punishment could be realized with quota settings: the download



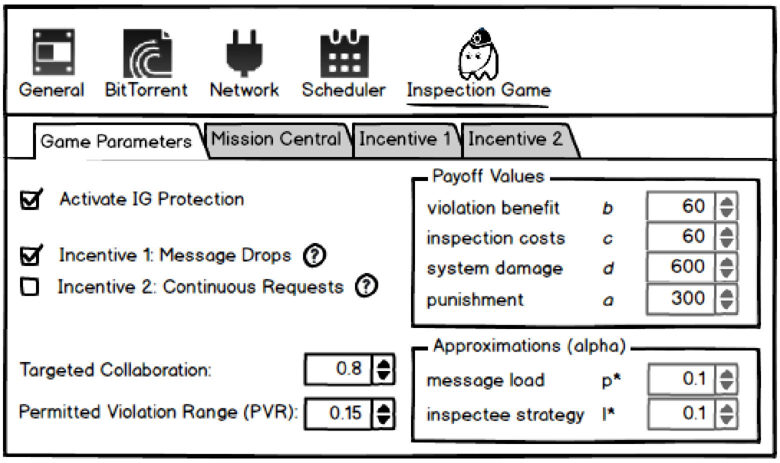
Circumvention of the upload threshold (min: 15 kB/s) has been detected.
The download has been limited to 10GB for 3 days.
Please check your network settings.

Circumvention of the upload threshold (min: 15 kB/s) has been detected.
The download has been limited to 10GB for 3 days.

sample.avi 112.13 / 348.21 MB △ 7 kB/s ▽ 34 kB/s

Download: 34 kB/s Upload: 7 kB/s

Figure 9.2: Two commonly used examples of user notifications are shown.



☑ Activate IG Protection

☑ Incentive 1: Message Drops ?

☐ Incentive 2: Continuous Requests ?

Targeted Collaboration: 0.8

Permitted Violation Range (PVR): 0.15

Payoff Values

violation benefit	b	60
inspection costs	c	60
system damage	d	600
punishment	a	300

Approximations (alpha)

message load	p*	0.1
inspectee strategy	I*	0.1

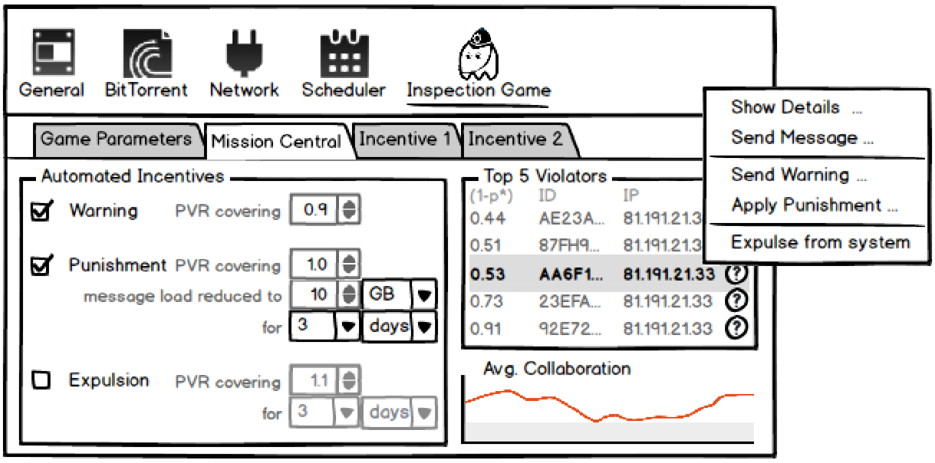
Figure 9.3: The IG configuration for the administrator could also be realized as additional area in the control interface. In this figure this is shown for the general game parameters.

rate is limited to 10 GB for 3 days. Here, the user is informed within the control interface. Figure 9.2 shows two further possibilities: message boxes and an information area in the main window. Both are commonly used in several applications (e.g. the Firefox browser).

9.2.2 *Meaning for the system administrator*

Similar to the user, the IG can be easily integrated in the administrator interface. An example is illustrated in figure 9.3. The *Game Parameters* tab uses the values of the enhanced IG approach introduced before. The second tab, *Mission Central*, contains some elements for the IG operation. Additional tabs can detail further collaboration incentives.

Figure 9.4 illustrates the Mission Central tab with practical functionalities for the IG operation. It shows an implementation aspect, denoted here as Permitted Violation Range (PVR) covering, to trigger automatic warnings and punishments. Its principle is that failures can be permitted to some degree or time. In the P2P video streaming, this corresponds to a slightly



Automated Incentives

☑ Warning PVR covering 0.9

☑ Punishment PVR covering 1.0
message load reduced to 10 GB
for 3 days

☐ Expulsion PVR covering 1.1
for 3 days

Top 5 Violators

(1-p*)	ID	IP
0.44	AE23A...	81.191.21.3
0.51	87FH9...	81.191.21.3
0.53	**AA6F1...**	**81.191.21.33** ?
0.73	23EFA...	81.191.21.33 ?
0.91	92E72...	81.191.21.33 ?

Figure 9.4: An IG control interface allows to specify manual and automatic collaboration incentives (warnings, punishment etc.). It may also give a statistical overview.

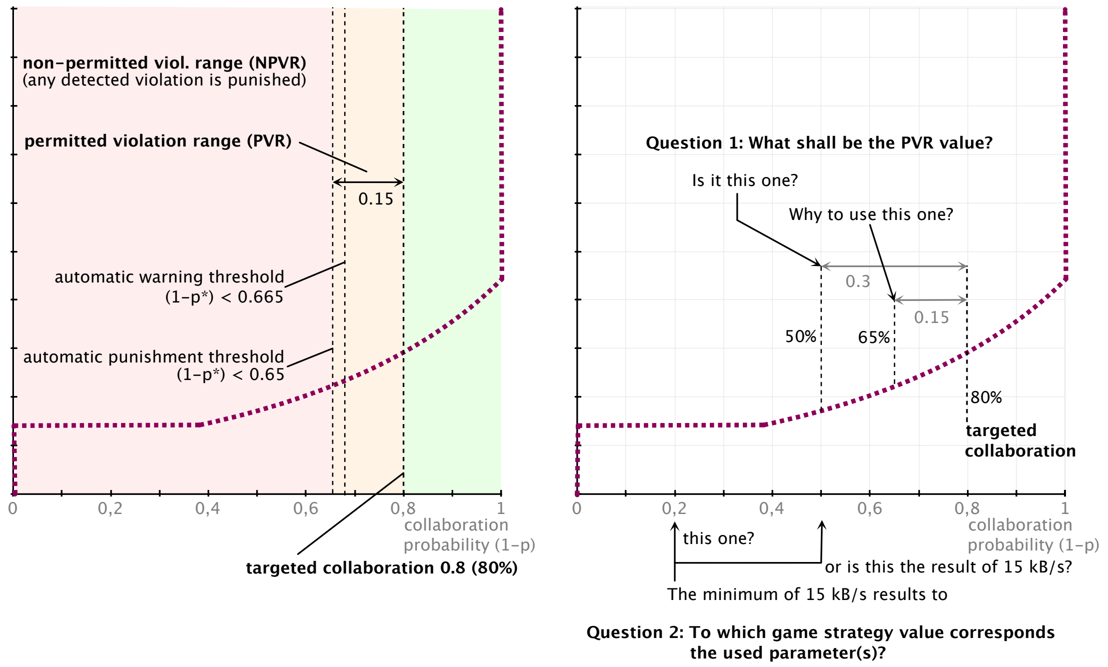


Figure 9.5: The user applications setting cause practical problems for the IG. Two are illustrated in the right BRS graph, using the values of the user interface mockups.

reduced frame rate. Collaboration values lower than the PVR are prohibited and should be directly punished; this region is also denoted as Non-Permitted Violation Range (NPVR). For example, warnings are sent if the PVR is used for at least 90%, i.e. for $0.65 \leq (1-p^*) < 0.665$. Figure 9.5 (left) illustrates the PVR (using the mockup settings) for the IG with BRS graphs. The PVR thresholds can for example be determined with simulations by evaluating the reliability capabilities.

This illustrative example shows on the one hand the simplicity of an IG integration into the application user interface. On the other hand, it brings another question up: *How correspond the user settings to the IG model with payoff and collaboration values in particular?* Practically spoken, why should we require bandwidth limitation value of 15 kB/s (e.g. to reach a PVR of 0.15) and not values such as 17 kB/s or 122 kB/s? This corresponds to the question of determining appropriate IG payoff values (and possibly other secondary parameters), which may be even dynamic. For example, a system may need temporarily a higher bandwidth. Fixed game parameters (represented through a application's configuration) would change the actual meaning of the parameters for an IG. This shows the challenging aspect since serval circumstance or the users' individual preference are not necessarily known to the administrator. A promising solution approach are feedback mechanisms to automatically determine parameter values. This is considered as possible future work but further outlined in the remainder of the thesis.

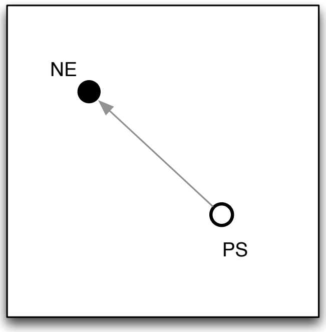


Figure 9.6: The IG principle is shown as population with the square as two-dimensional strategy space. Ideally, the population strategy (PS) moves directly to the Nash equilibrium (NE).

9.3 IMPACT OF THE USER BEHAVIOUR ON INSPECTION GAME CHALLENGES

The IG used in this thesis had the strong assumption of completely independent games. However, let us consider now the following more realistic assumptions:

- The game type is changed to a repeated or evolutionary game. The IG players' strategy may cover several games and the payoff does not necessarily depend on a single game.
- An inspectee is not anymore fully independent but takes into account the other inspectees' strategies for the own game strategy choice.

Though these assumptions make the game mechanics more complex, the resulting IG model is closer the real world. Selfish individuals may now strategize over time and they may learn from other inspectees. An example is the situation where a user asks some friends for seemingly beneficial bandwidth limitation settings of BitTorrent.

The adaption can cause further challenges. For a better understanding, let us consider the inspectees as a *population*. For the game designer, it appears as representative agent who plays the Population Strategy (PS) as mixed strategy. Then, the collaboration incentives shift the PS to the desired point, e.g. the Nash equilibrium (NE). This is visualized in figure 9.6 for a two dimensional strategy space. However, the strategy dynamics – especially in consideration of the group dynamics – may lead to other movements. Some examples are presented in figure 9.7. In the following, we denote the term *orbit* as the abstract characterization of a

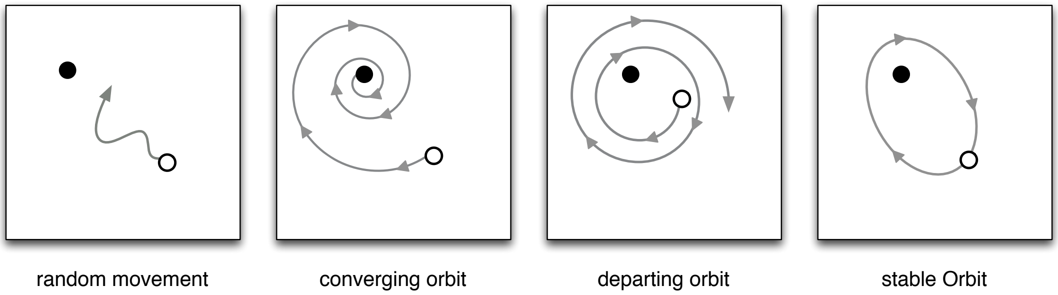


Figure 9.7: The general dynamics of the inspectees strategies may lead to several PS movements.

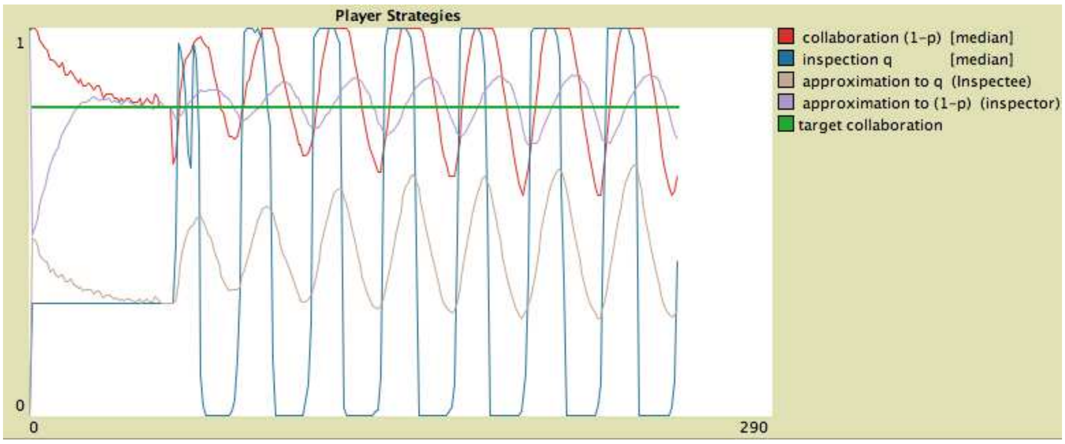


Figure 9.8: A stable orbit can also be reached by means of a dynamic inspection rate mechanism. The inspectee's collaboration value fluctuates around the target collaboration.

PS movement relative to a desired point, here represented by the NE (we omit the intuitive meaning of moving 'around' another object). Random movements will not necessarily result in a stabilized system state as the converging orbit. With a departing orbit, a stabilized situation is neither reached, while the PS is actually moving continuously away from the NE. Finally, a stable orbit is characterized by a periodical PS movement.

The PS movements can also be caused by other reasons than group dynamics. Figure 9.8 shows the simulation results of a simple feedback mechanism applied to the enhanced IG approach. Based on an approximation of the inspectee's violation strategy, the inspection rate is in-/decreased until a targeted collaboration level (straight green line) is reached. The approximation delay causes the strategy fluctuations around the target collaboration. In the context of this discussion, these fluctuations represent a stable orbit, while the movement passes even the target collaboration value itself. In addition to a dynamic PS (or inspectee strategy for the two player game), also the NE itself can move, which possibly affects the targeted collaboration level. A dynamic NE is for example given when essential game parameters (e.g. damage) change. Analogue to a moving PS, reliability mechanisms may weaken the impact or even tolerate it. This is not feasible for the general case and the dynamics should be directly taken into account during the game theoretic considerations.

To summarize, the inspectee's strategy as well as the NE (i.e. the population strategy) point should be considered as dynamic values when the IG is applied to real world systems. The system and strategy states should be evaluated over time and game parameters adapted appropriately in order to reach the NE and/or the targeted collaboration level. This requires another modified version of the IG able to handle these dynamics. Such an evolutionary game model is not in the scope of this thesis and declared here as possible future work.

Part V

CONCLUSION AND FUTURE WORK

This last part concludes the thesis by discussing the contributions in terms of the research challenges. Finally, some possible future work is presented.

Overview

10 CONCLUSION OF THE THESIS

We conclude the thesis now by reviewing the three essential contributions in terms of the research challenges stated in the introduction.

10.1 PREPARATIONS & BAR TOLERANCE EVALUATION

Due to their meaning for challenge (A), we consider the preparations for system comparison and the BAR tolerance evaluation as one contribution. This contribution outlines together with the related work discussion the state-of-the-art and justifies the thesis' main research challenge. A comprehensive system evaluation in terms of BAR tolerance capabilities was enabled. An extract is shown in table 10.1, while the full evaluation (see table 3.3 on page 35) reviews the BAR tolerance capabilities of 25 systems. The evaluation is done in a qualitative way and is based on previous preparations, an architectural classification and a taxonomy of failures. Hence, this contribution meets challenge (A) as well as (A.1) and (A.2).

The architectural classification normalizes the terms and interpretations of the literature. Thus, it enables a comprehensive architectural comparison, used by the evaluation to draw conclusions from the system architecture. The introduced taxonomy consists of elementary failures of a peer; all further failure scenarios are a direct result. This was an adequate metric for the qualitative evaluation, able to visualize the BAR tolerance capabilities in a comprehensible way. The evaluation revealed that practically all examined systems implement only basic reliability mechanisms. This is interpret as only a base for optional logic or specific adaptions to user dependent needs.

Table 10.1: The preparations enable a comprehensive reliability evaluation. This is shown here for the two systems that were used during the thesis – Scribe and Basic Gossiping.

System Name	**Subscription Type**	**Dissemination Technique**	**Overlay Organisation**	***Failure A* Link/Node Crash**	***Failure B* Message Loss**	***Failure C* Message Tampering**	***Failure D* Content Pollution**	***Failure E* Message Misuse**	***Failure F* Information Leak**	***Failure G* Selfish Behaviour**
Pub/Sub Systems										
Scribe [2]	Topic	Selective (Rendezvous)	S-P2P	✓	✗	✗	✗	✗	✗	✗
–	–	Basic Gossiping	U-P2P	✓	✓	✗	✗	✗	✗	✗

10.2 RCOURSE BENCHMARKING SUITE

RCourse is a library for the network simulation environment Peersim. As practical contribution, it supports the user in performing simulative studies with a special focus on robustness issues. All essential steps in the simulation workflow are facilitated or even completely taken over by RCourse. Meaningful simulations can be launched once the development of the peer's logic is completed. It allows to study the impact of selfishness-driven protocol violations on the system functioning and meets thus challenge (B). RCourse contributes also to challenge (C) by interacting with the IG framework. Quickly launched simulations evaluate the functioning of the IG approach and obtain desired information e.g. needed collaboration level. In addition, RCourse contributes to improve the qualitative BAR tolerance evaluation by facilitating the aggregation of more objective quantifying values. To this end, RCourse provides pre-defined simulation scenarios based on the same metrics (the taxonomy of failures). An example is shown in table 10.2 with values from the simulation results of figure 4.11 (page 52), enriching the prior table 10.1. The additional values indicate the percentage of correct system functioning in the presence of peers that violate with the probabilities 0.1 and 0.3.

During the study of the thesis, RCourse turned out to be a crucial tool in terms of the IG model calibration and verification. By starting only one script, the whole simulation workflow was taken over by RCourse, up to the result graph generation (see table A.1 in appendix A for an overview of all graphs). The design principle *simplicity of utilization* (comprising also time/work savings) is thus fully realized. The *isolation failure evaluation* is given by the pre-defined scenarios, which were helpful to identify specific reliability capabilities. Being simplistic to some degree, real world network traces would be an interesting extension. The principle *adaptability to individual needs* was reached in two aspects. A system designer can adapt the logic of a workflow phase by relying on the given files (simulation scenarios, analysis script etc.). He can furthermore, due to the modularity of the workflow itself, substitute a whole phase (e.g. result analysis) by individual tools. Overall, RCourse provides a broad flexibility and provides insights to a distributed system's behaviour with high time/work savings at the same time. Ideally, simulation results can be obtained "with one click".

Table 10.2: RCourse contributes to enrich the evaluation by quantitative information. To this end, it assists performing quickly launched simulations, also providing pre-defined scenarios.

System Name	**Subscription Type**	**Dissemination Technique**	**Overlay Organisation**	***Failure A*** **Link/Node Crash**	***Failure B*** **Message Loss**	***Failure C*** **Message Tampering**	***Failure D*** **Content Pollution**	***Failure E*** **Message Misuse**	***Failure F*** **Information Leak**	***Failure G*** **Selfish Behaviour**
Pub/Sub Systems										
Scribe [2]	Topic	Selective (Rendezvous)	S-P2P	✓ 0.1: 75% 0.3: 30%	✗ 0.1: 75% 0.3: 30%	✗	✗	✗	✗	✗
–	–	Basic Gossiping	U-P2P	✓ 0.1: 100% 0.3: 98%	✓ 0.1: 100% 0.3: 98%	✗	✗	✗	✗	✗

10.3 INSPECTION GAME FRAMEWORK TO ACHIEVE COLLABORATION

The enhanced IG framework of chapter 8 represents a solution approach to meet challenge (C). Exemplary applied to P2P based video streaming use case, it showed the ability to deploy system over selfish-driven peer and to achieve a targeted collaboration level. An evaluation verified the functioning considering some system dynamics.

Although the IG framework itself is directly related to challenge (C), the whole part IV is dedicated to IGs as mean against selfish peers. This is done straightforward from theory to practice. As theoretical contribution, the initial IG of Dresher [175] was generalized and extended by the the possibility false negatives. This enables to model the realistic situation where violations are not detected during an inspection due to limited resources. A NE calculation for all games showed furthermore that the extension shifts the NE linearly proportional to the probability of non-detection. The introduced models can be used for a broad spectrum of (possibly interdisciplinary) applications. The subsequent chapter discusses several real world implementation details such as inspection architectures. An interesting outcome was identified for the case where multiple inspections impose only one punishment. Games with multiple players can be represented as sets of 1-1 games, which simplifies the analysis. After the IG framework as practical contribution, a possible implementation was discussed by means of the example application BitTorrent. It demonstrates the simplicity of realization but outlines also challenges and possible future work; the latter one is addressed hereafter.

On the whole, the IG framework enables a new way of achieving reliability in distributed systems. It is also able to fill the BAR tolerance system design gap. This is illustrated in figure 10.1. Using a structured approach for the development process, a system designer reaches a specific implementation beginning with an abstract goal. The framework design provides a flexibility in terms of used system architecture and mechanisms to consider any selfish goal.

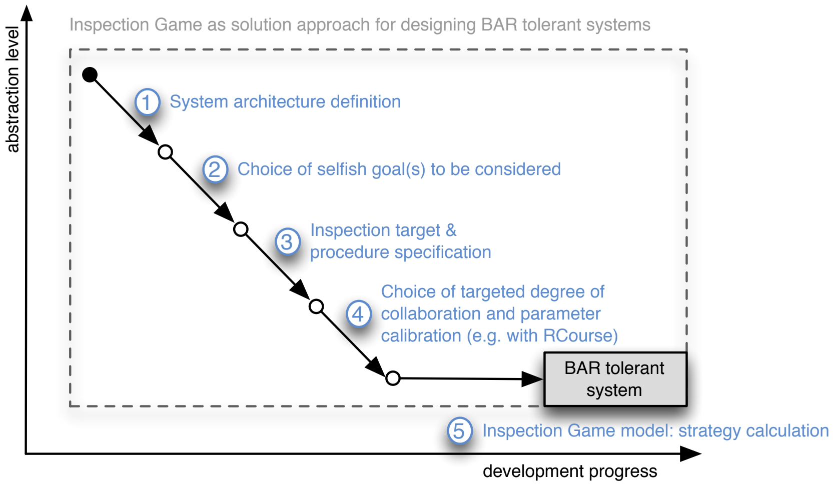

Inspection Game as solution approach for designing BAR tolerant systems
abstraction level
1 System architecture definition
2 Choice of selfish goal(s) to be considered
3 Inspection target & procedure specification
4 Choice of targeted degree of collaboration and parameter calibration (e.g. with RCourse)
BAR tolerant system
5 Inspection Game model: strategy calculation
development progress

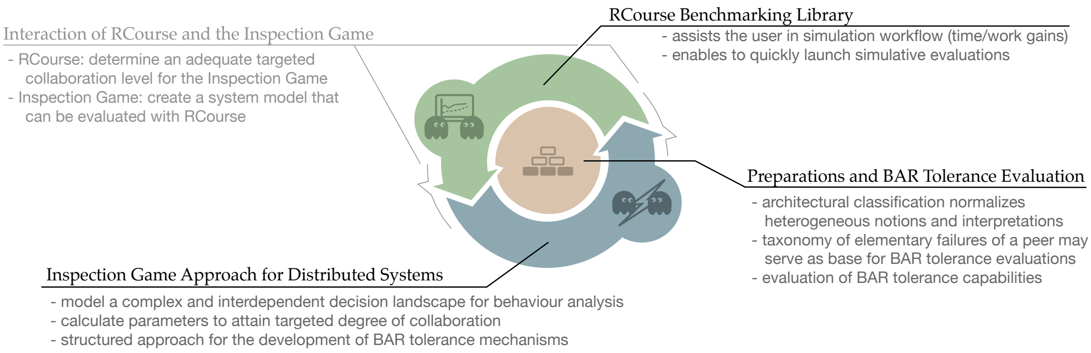
Figure 10.1: The IG approach represents a framework for the design of BAR tolerance capabilities. As outcome, it enables to calculate parameters to achieve the targeted collaboration degree.

10.4 DISCUSSION

The thesis' contributions have several relations and possible interactions. This is illustrated by figure 10.2, outlining also design principle that was mentioned in the introduction to meet the main research challenge (see chapter 1.4, page 11). A mutual exchange should be given between the actual system behaviour and the theoretical model to reach a specific implementation from an abstract starting point. This has been identified during the thesis as important aspect for the implementation of reliability mechanisms. RCourse interacts with the IG model in terms of parameter calibrations. Quickly launched simulations evaluate system properties to determine an appropriate collaboration level or payoff values. Based on these parameters, the IG approach allows a BAR tolerant system design, which can in turn be evaluated with RCourse (compare also with figure 10.1). The utilization of Publish/Subscribe was an adequate use case system. The study could take place at specific system implementations that are close to real world systems. At the same time, the pub/sub paradigm represents the basic user needs (requesting/consuming information), thus being applicable to a broad range of systems.

The IG contribution has a fundamental research character with a proof-of-concept relevance in terms of real world implementations. Nevertheless, it has a high scientific relevance. The IG approach offers a complete new way of designing BAR tolerant systems as a whole. Reliability mechanisms – system-inherent as well as artificially added – can be considered within one comprehensive model for the behaviour analysis of selfish peers. The IG approach gains a general purpose applicability and can be applied wherever entities are controlled to avoid some kind of violations. An example is the inspection of log files (commonly used router, operation system etc.) in order to detect violations.

The study demonstrated clearly the principle of the IG framework. A system designer can benefit from the IG mechanics for behaviour modelling and parameter calculations. However, appropriate payoff values must be used. This causes further challenges since selfish peers have inherently individual preferences. These are rarely known, probably not equal among the peers and could even by dynamic. "Good" and "dynamic" estimations, however these terms are determined, are thus needed to seriously pass the step to broad real world applications. Due to the heterogeneity and dynamics, only (semi-)automatic feedback mechanisms represent promising approaches.



Interaction of RCourse and the Inspection Game
- RCourse: determine an adequate targeted collaboration level for the Inspection Game
- Inspection Game: create a system model that can be evaluated with RCourse

RCourse Benchmarking Library
- assists the user in simulation workflow (time/work gains)
- enables to quickly launch simulative evaluations

Preparations and BAR Tolerance Evaluation
- architectural classification normalizes heterogeneous notions and interpretations
- taxonomy of elementary failures of a peer may serve as base for BAR tolerance evaluations
- evaluation of BAR tolerance capabilities

Inspection Game Approach for Distributed Systems
- model a complex and interdependent decision landscape for behaviour analysis
- calculate parameters to attain targeted degree of collaboration
- structured approach for the development of BAR tolerance mechanisms

Figure 10.2: The graphical overview outlines the relations among the contributions.

11 POSSIBLE FUTURE WORK

The IG approach provides several possibilities for further research. Three are introduced here representing significant improvements.

11.1 COLLUDING INSPECTEES

Colluding selfish inspectees collaborate in terms of their violations in order to reach an overall objective. In other words, they try to increase the payoff of the group as a whole. As example, the following scenario may arrive when collaborative interactions of two peers are secured. Figure 11.1 illustrates this threat at two network structures. Let us consider an optimal case for the selfish inspectees where a network segment is fully consisting of colluding peers. Then, the colluding peers form a *colluding black box*. The border peers violate by removing any PoMs and they get (probably) punished. However, due to the missing proofs, the peers' behaviour inside the colluding black box cannot necessarily be clearly proven. In other words, they could violate among the collaboration and remove or falsify the violation proofs afterwards. The problem of colluding peers in distributed systems starts to be addressed by the research community (see for instance [151, 152]). Hence, this issue provides many possibilities for subsequent research with regard to the IG approach. One example question concerns the ratio of colluding and non-colluding peers that allows to form the black box. Possible solution approaches could identify the black box frontiers and isolate it from the network.

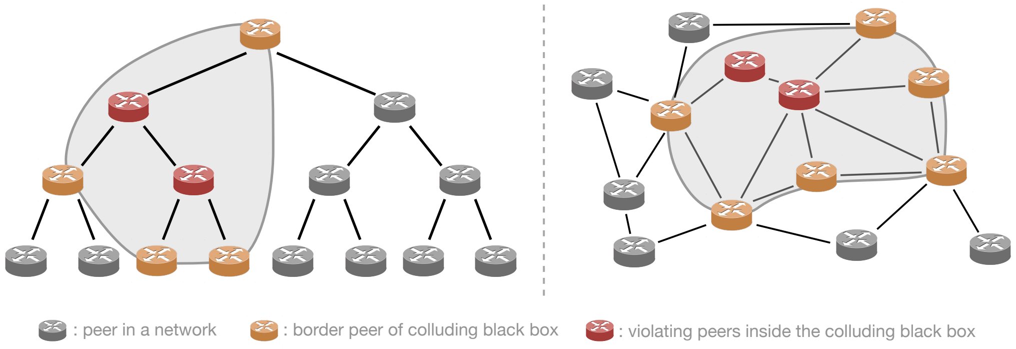

: peer in a network
: border peer of colluding black box
: violating peers inside the colluding black box

Figure 11.1: Colluding peers can create black boxes. The border peers violate and sacrifice themselves such that the colluding peers inside may perform undetected violations.

11.2 MAPPING THE REAL WORLD TO GAME PARAMETERS

Further possible future work is related to the representation of real world circumstances to game parameters. As discussed before, a general drawback of the IG approach is the determination of the payoff values. Among others, they need to represent appropriate user preferences (e.g. violation benefit) and also a suitable determination of system damage. This is a challenging task for the system designer. Especially the meaning of the violation benefit and punishment depends on the corresponding users. Furthermore, these values are presumably varying among the users and possibly even changing over time. These aspects emphasize the need of further research to support the mapping of real world circumstances to appropriate payoff values. Ideally, some feedback mechanism(s) would allow a dynamic adaptation to the individual user preferences. A simple mechanism is already shown in figure 9.8 (page 110). Here, the inspection rate is in-/decreased until the targeted collaboration is reached, while the strategy approximation delay creates fluctuations around the targeted value. However, this is only an exemplary mechanism and more elaborated ones are desired.

Another similar issue has been discussed before in this chapter and can be described by the following question: *How to map the application settings and user behaviour to game parameters such as the collaboration strategy*? Again, some feedback mechanisms are imaginable that adapt the meaning in form of game parameters to the dynamic system state. However, it has not yet been addressed and is still an open problem for IGs in particular.

11.3 EVOLUTIONARY GAME MODEL

Rational selfish individuals in the real world can strategize over time and exchange information with other individuals (see also the discussion in section 9.3). Hence, a possible improvement consists in the extension of the IG approach towards an evolutionary game model. This complicates the model itself (and thus also the analysis), however, it has several advantages. For example, it enables the analysis of player strategies across several games. Another example is related to section 9.3 (page 109) where inspections are considered as a population. Selfish individuals may be influenced by other in their strategy choice, e.g. by exchanging seemingly beneficial application settings. Then, the PS may be dynamic, possibly not reaching a targeted strategy value such as the Nash equilibrium or a collaboration level. An

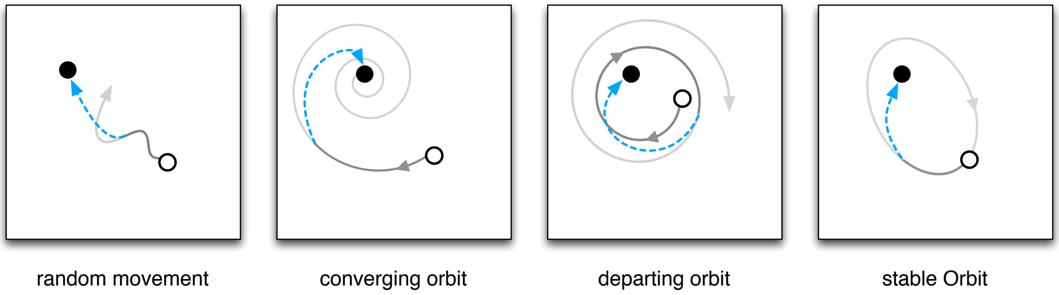


Figure 11.2: An evolutionary game model enables game modifications to redirect it movements of the population strategy to a desired point (e.g. Nash equilibrium).

evolutionary model could then identify such strategy movement. Game parameters could be dynamically adjusted such that the population's strategy is redirected to the desired strategy value. This is illustrated in figure 11.2 with regard to the original movements of figure 9.7 (page 109).

Part VI

APPENDIX

The appendix provides additional details to several contributions of the thesis.

Overview

A

DETAILS TO THE RCOURSE LIBRARY COMPONENTS

Several detail information are here given about the components of the RCourse library in order to process result data and generate graphs.

Listing A.1: Example source code of the R script to generate the graph *D5.1 Delivery Loss Rate* for *Experimentation Scenario ES3.2 Message Loss* with *Loss Rate Variation*.

```
# =============================================================
#    RCourse Analysis Script for
#               RCourse Experiment Scenario Diagram ES3.2 D5.1
#               Delivery Loss Rate - Message Loss (Loss Rate Variation)
#
#       Shows the dissemination completion in overlay hops
#       for different message loss rates.
#
#               y-axis label:   overlay hops
#               x-axis label:   lost messages (in %)
# =============================================================
library(Hmisc)
deliveryLoss_avg        <- rep(0, msgDropPropsCount)
deliveryLoss_sdev       <- rep(0, msgDropPropsCount)
deliveryLossCol         <- 17
deliveryLoss_avg        <- mean [ 1:msgDropPropsCount , deliveryLossCol ]
deliveryLoss_sdev       <- sdev [ 1:msgDropPropsCount , deliveryLossCol ]
xaxis                   <- msgDropProps

###  ============= plot diagram ==============================
pdffilename <- "ES3.2_D5.1__MessageLossLRV-DeliveryLossRate.pdf"
fullpdfpath <- sprintf("%s%s", pdffolder, pdffilename)
pdf(fullpdfpath, width=w, height=h)

# plot curves
errbar(
        xaxis,
        100 - deliveryLoss_avg,
        100 - deliveryLoss_avg + deliveryLoss_sdev,
        100 - deliveryLoss_avg - deliveryLoss_sdev,
        xlim = c( min(msgDropProps), max(msgDropProps) ), ylim = c(0, 100),
        ylab = "not delivered messages (in %)", xlab = "lost messages (in %)",
        pch = charSquare, col = colorBlue, bg = colorBlue, axes=FALSE )

points( xaxis, 100 - deliveryLoss_avg, col=colorBlue,  bg=colorBlue, pch=charSquare)
lines(  xaxis, 100 - deliveryLoss_avg, col=colorBlue,  lwd=1, lty = "dashed")

# print own xaxis labels corresponding to network size variations
axis(1, msgDropProps, msgDropProps )
axis(2)
box()

mtext(" RCourse Benchmark - ES3.2 Message Loss (Loss Rate Variation) - D5.1 Delivery Loss Rate", adj
     =0, line=-1, col="grey" , cex=0.8, outer=TRUE)
title(main="Delivery Loss Rate")

if (gossiping) {
        source("util/generate_subtitleText.r")
        mtext(t_subtitleText, line=0.2, col="darkgray", cex=0.7) }

dev.off()
print( sprintf("  Created diagram ’%s’", fullpdfpath) , quote=FALSE )
```

Listing A.2: The RCourse configuration file 'rcourse_Base.txt' for Peersim structures the simulated overlay networks in form of functional layers.

```
# =================== LAYERS ===================
protocol.0link peersim.core.IdleProtocol

protocol.1uniftr peersim.transport.UniformRandomTransport
protocol.1uniftr.mindelay MINDELAY
protocol.1uniftr.maxdelay MAXDELAY

protocol.2unreltr peersim.transport.UnreliableTransport
protocol.2unreltr.drop 0
protocol.2unreltr.transport 1uniftr

protocol.3mspastry overlay.mspastry.MSPastryProtocol
protocol.3mspastry.transport 2unreltr

protocol.4scribe application.scribe.Scribe
protocol.4scribe.transport 3mspastry


# =================== INITIALIZERS ===================
init.0randlink peersim.dynamics.WireKOut
init.0randlink.k K
init.0randlink.protocol 0link

init.1uniqueNodeID overlay.mspastry.CustomDistribution
init.1uniqueNodeID.protocol 3mspastry

init.2statebuilder overlay.mspastry.StateBuilder
init.2statebuilder.protocol 3mspastry
init.2statebuilder.transport 2unreltr

# initializer is very important and needed by scribe
init.3scribeinitializer application.scribe.ScribeInitializer
init.3scribeinitializer.protocol 4scribe


# =================== OBSERVER ===================
control.observer rcourse.RCObserver
control.observer.protocol 4scribe
control.observer.step OBSERVER_STEP
control.observer.FINAL
```

Listing A.3: The Experimentation Scenario configurations, here for ES3.2, are easy comprehensible.

```
#################################################
#
#    Peersim Configuration Script for
#
#       RCourse ES3.2 Message Loss (Loss Rate Variation)
#
#################################################

# =================== NETWORK ===================
K 1  # K wiring - 'known' connections on the overlay

NETSIZE         300
network.size    NETSIZE
MINDELAY        0       # optimal situation: no time delays
MAXDELAY        0

# =================== SIMULATION ===================
#random.seed 24680

CYCLES 500
TIME_PER_CYCLE 1000

TRAFFIC_STEP   1*TIME_PER_CYCLE
OBSERVER_STEP  5*TIME_PER_CYCLE

simulation.experiments 10
simulation.endtime TIME_PER_CYCLE*CYCLES
simulation.logtime 60*TIME_PER_CYCLE

# =================== RCOURSE ===================
#rcourse.distrProcId 1  # recommendation: define this as argument
rcourse.resultLog.filenamebase  ES3.2-MessageLossLRV
rcourse.resultLog.path          results/Scribe/ES3.2-MessageLossLRV
rcourse.resultLog.writeToDB     true

SUBSR_EXEC_CYCLES 5
rcourse.trafficGen.groupCount               100
rcourse.trafficGen.subscribeStartCycle      95
rcourse.trafficGen.subscribeExecCycles      SUBSR_EXEC_CYCLES
rcourse.trafficGen.subscriberPerGroupCount  0.05*NETSIZE/SUBSR_EXEC_CYCLES

rcourse.trafficGen.publishStartCycle        201
rcourse.trafficGen.publishExecCycles        200
rcourse.trafficGen.msgPerGroupCount         0.2

control.traffic         rcourse.RCTrafficGenerator
control.traffic.protocol 4scribe
control.traffic.step    TRAFFIC_STEP

# ========= RCOURSE TURBULENCE (message loss, loss rate variation) =========
range.1              rcourse.turbulence.dropProb;0:1|0.1
rcourse.malNodeProb  1   # malicious nodes probability: all nodes are malicious
```

Listing A.4: The analysis script *diRgrams_ES3.2.r* is called by the user and invokes all further scripts.

```r
# ============================================================
#    RCourse Analysis Script for
#
#       RCourse Scenario ES3.2 - Message Loss (Loss Rate Variation)
#
#       This script generates diagrams as pdf files
#       for the RCourse experiment scenario ES3.2
#
#       Define your simulation configuration in the
#       referenced (or individual) script file.
#
# ============================================================

print("", quote=FALSE )
print("--- RESULT GRAPH GENERATION FOR ES3.2 ---", quote=FALSE )


# simulation configuration - for usage with rcourse traffic generator
source("config/ES3.2_config.r")


# === global / final =========================================
source("util/readin__global-final.r")   # read in data files

source("global-final/ES3.2-D2.7_MessageLossLRV-NotificationOverheadRatio.r")
source("global-final/ES3.2-D5.1_MessageLossLRV-DeliveryLossRate.r")
```

Table A.1: Detailed scenario overview and result graphs generated by RCourse.

Failure Type	#	Experimentation Scenario	#	Diagram Description			Graph Axes	Result Writeout
< failure-free >	1	ES1.1 Standard Operation	1	D1.2	Dissemination Completion (absolute) (delivered message percentage per cycle)		y: delivered messages x: simulation cycles	global / per cycle
			2	D1.3	Dissemination Completion (percentage) (delivered message percentage per cycle)		y: delivered message percentage x: simulation cycles	global / per cycle
			3	D1.4	PubSub Call Overview (amount of publish/subscribe api calls per cycle)		Y : # of pubsub api calls (un-/subscribe, publish) X : simulation cycles	global / per cycle
			4	D2.3	Network Stress per Cycle (notify msg) (absolute overlay hop counts relative to simulation cycle)		y : handled messages x : simulation cycles	global / per cycle
			5	D2.4	Network Stress per Cycle (all msg) (absolute overlay hop counts relative to simulation cycle)		y : handled messages x : simulation cycles	global / per cycle
			6	D2.5	Node Stress Distribution (notify msg) (handled messages per node)		y: handled messages x: nodes in network	per node / final
			7	D2.6	Node Stress Distribution (all msg) (handled messages per node)		y: handled messages x: nodes in network	per node / final
			8	D3.1	Subscription Table Size Distribution (subscriptions at each node)		y: subscription table size x: nodes in network	per node / final
			9	D3.2	Group Subscription Distribution (subscription table entries per group)		y: # of subscription table entries x: multicast groups	global / final
			10	D10.1	View Membership Distribution (In- and out-degree of every node)	U-P2P related	y: degree of the node x: nodes in network	per node / final
			11	D4.1	Clustering Coefficient Distribution (clustering coefficient per node)		y : clustering coefficient x : nodes in network	per node / final
			12	D4.2	Network Clustering Coefficient (average clustering coefficient per cycle)		y : clustering coefficient x : simulation cycles	global / per cycle
	2	ES1.2 Standard Operation (Network Size Variation)	13	D1.1	Dissemination Delay (hop counts per network size)		y : overlay hop count x: network size (# nodes in system)	global / final
			14	D2.1	Network Stress (notifiy msg) (absolute overlay hop counts in the network – only notify and direct messages)		y : handled messages x : network size (# nodes in system)	global / final
			15	D2.2	Network Stress (all msg) (absolute overlay hop counts in the network – all forwarded and direct messages)		y : handled messages x : network size (# nodes in system)	global / final
			16	D2.7	Notification Overhead Ratio (notify message hops relative to network size)		y: notify hops per publish call x: network size (# nodes in system)	global / final
			17	D4.2	Network Clustering Coefficient (average clustering coefficient per cycle)	U-P2P related	y : clustering coefficient x : simulation cycles	global / per cycle
			18	D10.1	View Membership Distribution (In- and out-degree of every node)		y: degree of the node x: nodes in network	per node / final
	3	ES1.3 Standard Operation (Fanout Variation)	19	D1.2	Dissemination Completion (absolute) (delivered message percentage per cycle)	U-P2P related	y: delivered messages x: simulation cycles	global / per cycle
			20	D1.3	Dissemination Completion (percentage) (delivered message percentage per cycle)		y: delivered message percentage x: simulation cycles	global / per cycle
			21	D2.3	Network Stress per Cycle (notify msg) (absolute overlay hop counts relative to simulation cycle)		y : overlay hop count x : simulation cycles	global / per cycle
			22	D2.4	Network Stress per Cycle (all msg) (absolute overlay hop counts relative to simulation cycle)		y : overlay hop count x : simulation cycles	global / per cycle
			23	D2.5	Node Stress Distribution (notify msg) (handled messages per node)		y: handled messages x: nodes in network	per node / final
			24	D2.6	Node Stress Distribution (all msg) (handled messages per node)		y: handled messages x: nodes in network	per node / final

Continuation of table A.1

Failure A Link/Node Crash	4	**ES2.1 Catastrophic Node Crash** crashing nodes: 10%, 20%, 30% crashing nodes (gossiping): 30%, 50%, 80%	25	D1.2	Dissemination Completion (absolute) (delivered message percentage per cycle)		y: delivered messages x: simulation cycles	global / per cycle
			26	D1.3	Dissemination Completion (percentage) (delivered message percentage per cycle)		y: delivered message percentage x: simulation cycles	global / per cycle
			27	D2.4	Network Stress per Cycle (all msg) (absolute overlay hop counts relative to simulation cycle)		y : overlay hop count x : simulation cycles	global / per cycle
			28	D4.1	Clustering Coefficient Distribution (clustering coefficient per node)	U-P2P related	y : clustering coefficient x : nodes in network	per node / final
			29	D4.3	Partition Size Distribution (nodes per partition)		y: # nodes in partition x: # network partitions	global / final
			30	D4.4	Network Partitioning (amount of partitions per amount of crashed nodes)		y: # of partitions x: crashed nodes (in %)	global / final
	5	**ES2.2 Catastrophic Node Crash** **(Crash Size Variation)**	31	D2.7	Notification Overhead Ratio (notify message hops per publish call)		y: notify hops per publish call x: crashed nodes (in %)	global / final
			32	D5.1	Delivery Loss Rate (catastrophic node crash) (not delivered messages relative to crashed nodes)		y: not delivered messages (in %) x: crashed nodes (in %)	global / final
			33	D4.2	Network Clustering Coefficient (clustering coefficients for node crash situations)	U-P2P related	y: clustering coefficient x : crashed nodes (in %)	global / final
			34	D4.3	Partition Size Distribution (nodes per partition)		y: # nodes in partition x: # network partitions	global / final
			35	D4.4	Network Partitioning (catastrophic node crash) (amount of partitions per amount of crashed nodes)		y: # of partitions x: crashed nodes (in %)	global / final
Failure B Message Loss	6	**ES3.1 Message Loss** Loss rates: 10% , 20%, 30%	36	D1.2	Dissemination Completion (absolute) (delivered message percentage per cycle)		y: delivered messages x: simulation cycles	global / per cycle
			37	D1.3	Dissemination Completion (percentage) (delivered message percentage per cycle)		y: delivered message percentage x: simulation cycles	global / per cycle
			38	D2.4	Network Stress per Cycle (all msg) (absolute overlay hop counts relative to simulation cycle)		y : overlay hop count x : simulation cycles	global / per cycle
	7	**ES3.2 Message Loss** **(Loss Rate Variation)**	39	D2.7	Notification Overhead Ratio (notify message hops per publish call)		y: notify hops per publish call x: malicious nodes (in %)	global / final
			40	D5.1	Delivery Loss Rate (lossy nodes) (not delivered messages relative due to lossy links/nodes)		y: not delivered messages (in %) x: malicious nodes in system (in %)	global / final
Failure C Message Tampering	8	**ES4 Message Tampering** **(Malicious Nodes Variation)**	41	D6	Tampering Rate (reception rate of modified messages)		y: delivered mod. messages (in %) x: malicious nodes in system (in %)	global / final
Failure D Content Pollution	9	**ES5** ---	---	---	---		---	---

Continuation of table A.1

Failure E Message Misuse	10	**ES6.1 Node Churn** **(no publish)** churn rate: 2%	42	D2.4	Network Stress per Cycle (all msg) (absolute overlay hop counts relative to simulation cycle)	y : # of handled messages x : simulation cycles	global / per cycle
	11	**ES6.2 Node Churn** **(with publish)** churn rate: 2%	43	D1.2	Dissemination Completion (absolute) (delivered message percentage per cycle)	y: delivered messages x: simulation cycles	global / per cycle
			44	D1.3	Dissemination Completion (percentage) (delivered message percentage per cycle)	y: delivered message percentage x: simulation cycles	global / per cycle
			45	D2.4	Network Stress per Cycle (all msg) (absolute overlay hop counts relative to simulation cycle)	y : # of handled messages x : simulation cycles	global / per cycle
	12	**ES6.3 Node Churn** **(Churn Rate Variation, no publish)**	46	D2.2	Network Stress (all msg) (absolute overlay hop counts in the network – all forwarded and direct msg)	y : # of handled messages x : churn rate (in %)	global / per cycle per cycle only for calculation of appropriate final value
	13	**ES6.4 Node Churn** **(Churn Rate Variation, with publish)**	47	D2.2	Network Stress (all msg) (absolute overlay hop counts in the network – all forwarded and direct msg)	y : # of handled messages x : churn rate (in %)	global / per cycle per cycle only for calculation of appropriate final value
			48	D2.7	Notification Overhead Ratio (notify message hops per publish call)	y: notify hops per publish call x: churn rate (in %)	global / final
			49	D5.1	Delivery Loss Rate (lossy nodes) (not delivered messages relative due to lossy links/nodes)	y: not delivered messages (in %) x: churn rate (in %)	global / final

Failure F Information Leak	14	**ES7** ---	---	---	---	---	---

Failure G Selfish Behaviour	15	**ES8.1 Selfish Message Loss** drop message probabilities: 1, 0.66, 0.33	50	D5.1	Delivery Loss Rate (selfish drops) (not delivered messages relative to malicious situation)	y: not delivered messages (in %) x: malicious nodes in system (in %)	global / final
	16	**ES8.2 Selfish Message Tampering** modify message probabilities: 1, 0.66, 0.33	51	D6	Tampering Rate (reception rate of modified messages)	y: delivered modified messages (in %) x: malicious nodes in system (in %)	global / final

B

VD GRAPHS OF THE INSPECTION GAME APPROACH

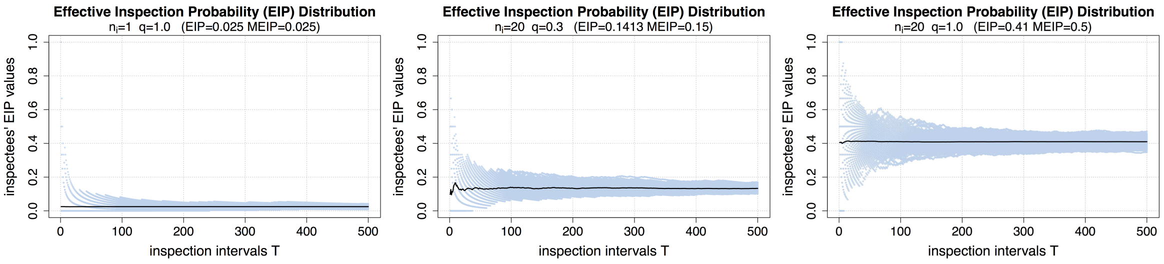


Figure B.1: The VD result graphs for the EIP value evaluation related to those of figure 7.7 (page 90).

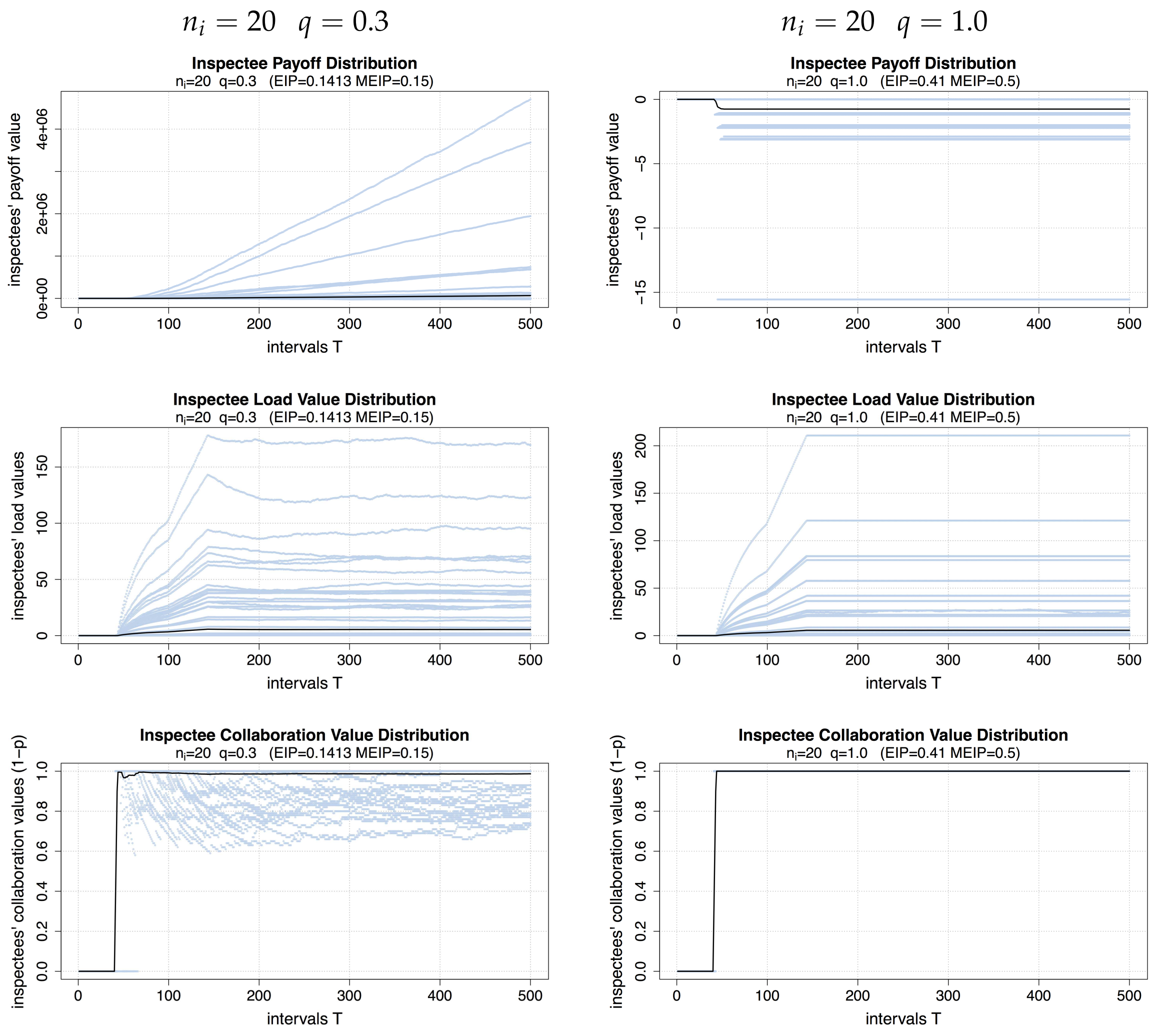


Figure B.2: The VD result graphs for the Scribe system related to those of figure 7.9 (page 92).

$n_i = 20 \quad q = 0.3$

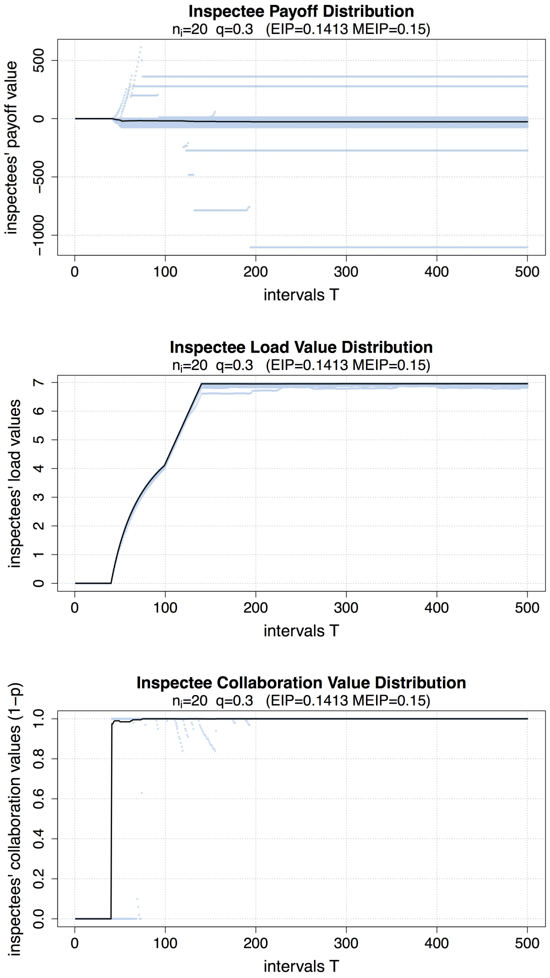


Figure B.3: The VD result graphs for the gossiping system related to those of figure 7.8 (page 91).

C

VD GRAPHS OF THE ENHANCED INSPECTION GAME APPROACH

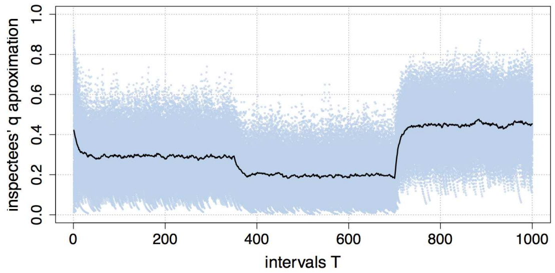
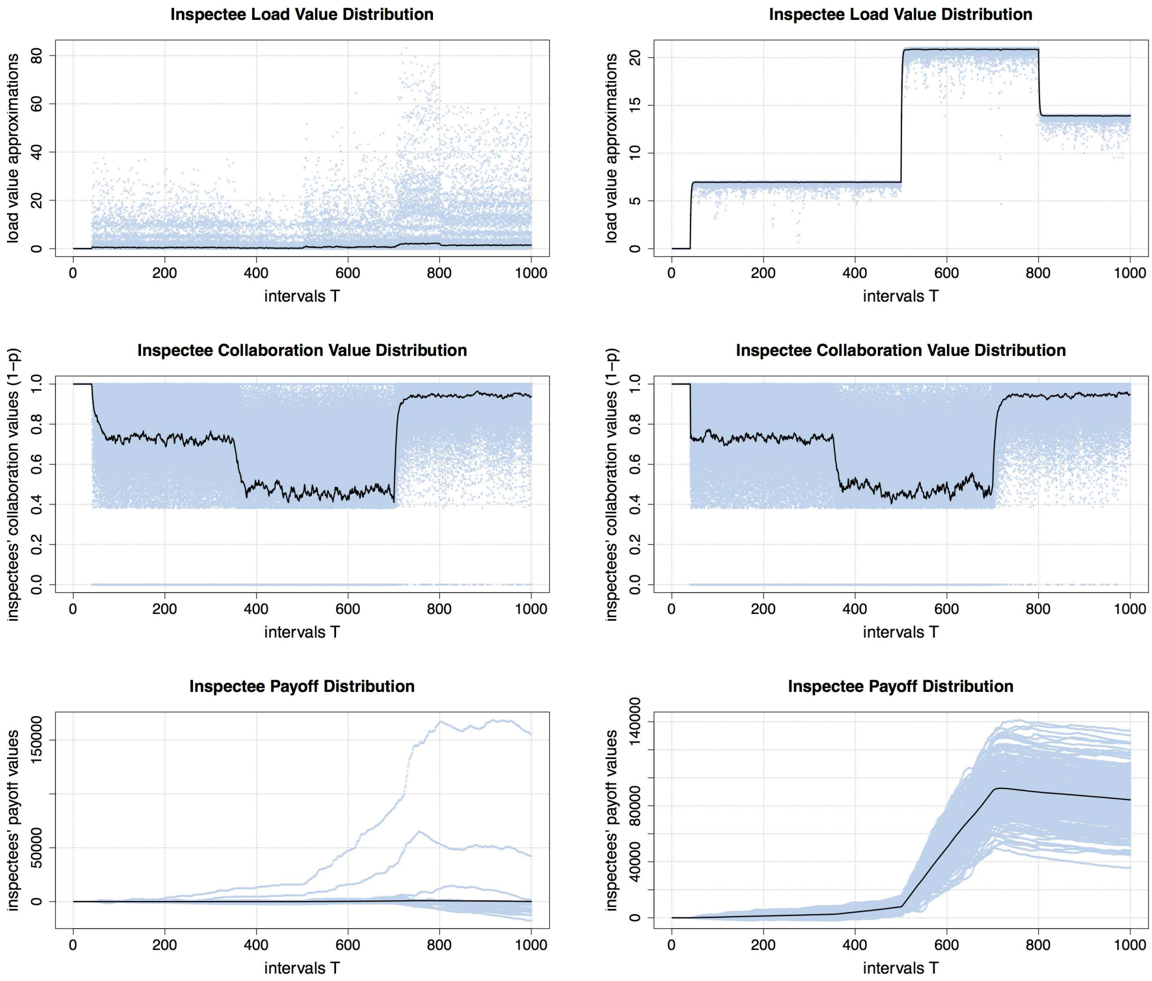


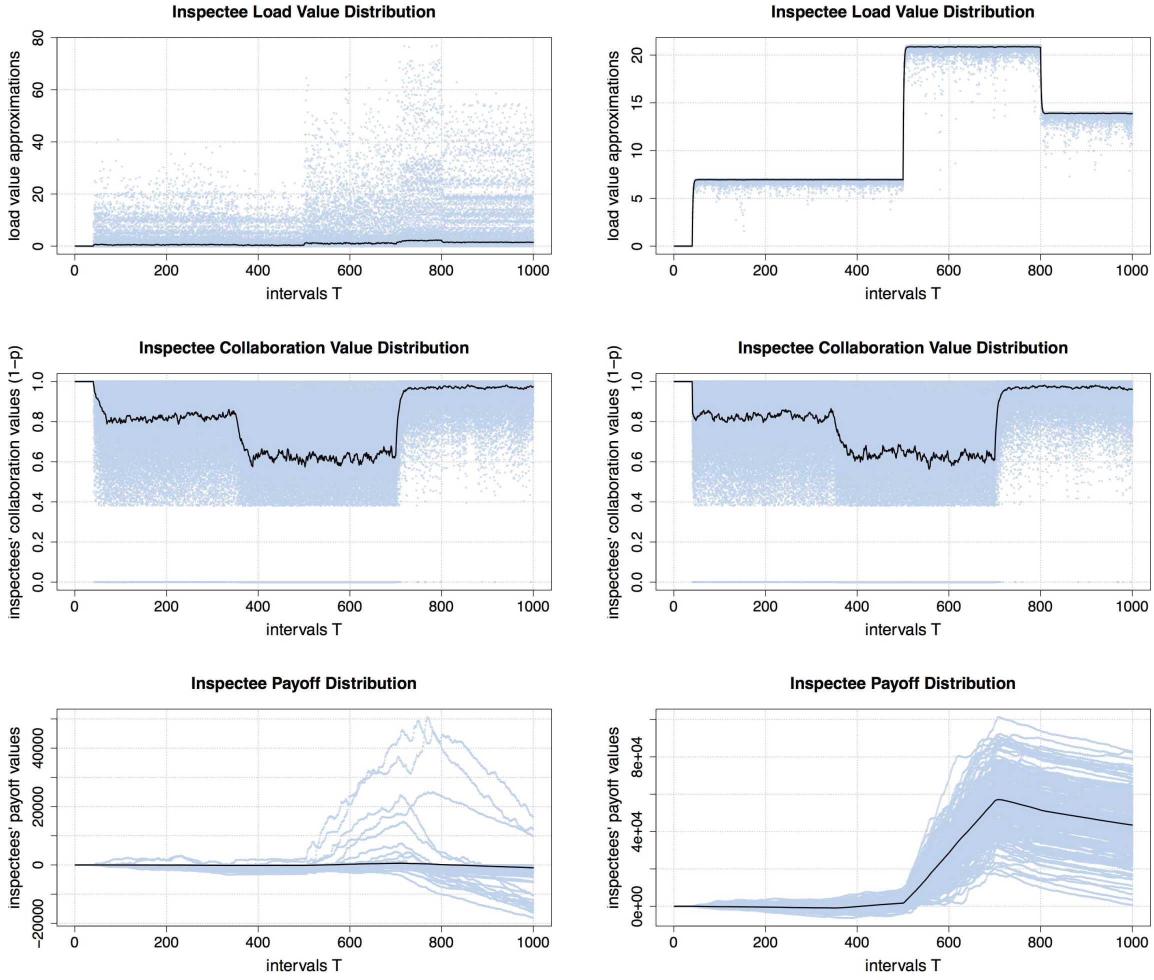
Figure C.1: The graph shows the inspectees' approximations to q and is related to figure 8.5 (page 101).



Scribe

Basic Gossiping

Figure C.2: The VD graphs correspond to those of figure 8.6 (page 102). Please note that the mean curve differs slightly to the medians of figure 8.6 with its statistically robustness.



Scribe

Basic Gossiping

Figure C.3: These results correspond to those of figure C.2 (and thus also 8.6 on page 102), however, the three inspection probabilities are increased by 0.05. As result, the variations are strongly reduced for the epochs with $1 - p \approx 0.7$.

BIBLIOGRAPHY

[1] S. Nielson, S. Crosby, and D. Wallach, "A taxonomy of rational attacks," in *Proceedings of the 4th International Workshop on Peer-to-Peer Systems (IPTPS'05)*, pp. 36–46, 2005.

[2] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron, "Scribe: a large-scale and decentralized application-level multicast infrastructure," *IEEE Journal on Selected Areas in Communications*, vol. 20, pp. 1489 – 1499, October 2002.

[3] "Kaspersky security bulletin 2012 & malware evolution." Kaspersky Labs. (Accessed: January 2013).

[4] J. Shneidman and D. Parkes, "Rationality and self-interest in peer to peer networks," in *Peer-to-Peer Systems II*, vol. 2735 of *Lecture Notes in Computer Science*, pp. 139–148, Springer, 2003.

[5] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 4, no. 3, pp. 382–401, 1982.

[6] D. Oppenheimer, A. Ganapathi, and D. Patterson, "Why do internet services fail, and what can be done about it?," in *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, USITS'03, USENIX Association, 2003.

[7] P. Grace, G. Blair, and V. Issarny, "Emergent middleware," *ERCIM News*, vol. 2012, no. 88, 2012.

[8] A. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J. Martin, and C. Porth, "BAR fault tolerance for cooperative services," in *Proceedings of the 20th ACM symposium on Operating systems principles (SOSP'05)*, pp. 45–58, ACM, 2005.

[9] Q. Li, S. Zhu, and G. Cao, "Routing in socially selfish delay tolerant networks," in *Proceedings of IEEE INFOCOM 2010*, pp. 1–9, IEEE Computer Society, March 2010.

[10] A. Mei and J. Stefa, "Give2get: Forwarding in social mobile wireless networks of selfish individuals," *International Conference on Distributed Computing Systems (ICDCS'10)*, pp. 488–497, 2010.

[11] J. Shneidman and D. Parkes, "Specification faithfulness in networks with rational nodes," in *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing (PODC'04)*, pp. 88–97, ACM, 2004.

[12] J. Li, "mssl: A framework for trusted and incentivized peer-to-peer data sharing between distrusted and selfish clients," *Peer-to-Peer Networking and Appl.*, vol. 4, no. 4, pp. 325–345, 2011.

[13] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *Proceedings of the 30th Annual Cryptology Conference (CRYPTO'10)*, vol. 6223 of *Lecture Notes in Computer Science*, pp. 465–482, Springer, 2010.

[14] J. Jaramillo and R. Srikant, "Darwin: distributed and adaptive reputation mechanism for wireless ad-hoc networks," in *Proceedings of the 13th annual ACM international conference on Mobile computing and networking*, (New York, USA), pp. 87–98, 2007.

[15] E. Hernández-Orallo, M. Serrat, J. Cano, C. Calafate, and P. Manzoni, "Improving selfish node detection in manets using a collaborative watchdog," *IEEE Communications Letters*, vol. 16, no. 5, pp. 642–645, 2012.

[16] A. Ganchev and L. Narayanan, "Selfishness detection for backoff algorithms in wireless networks," in *Proceedings of the 7th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob'11)*, pp. 517–524, 2011.

[17] T. Iyer, R. Hsieh, N. B. Rizvandi, B. Varghese, and R. Boreli, "Mobile p2p trusted on-demand video streaming," *CoRR*, vol. abs/1204.0094, 2012.

[18] Z. Wei and J. Pan, "Modeling bittorrent-based p2p video streaming systems in the presence of nat devices," in *2011 IEEE International Conference on Communications (ICC)*, pp. 1–5, June 2011.

[19] N. Ramzan, H. Park, and E. Izquierdo, "Video streaming over p2p networks: Challenges and opportunities," *Signal Processing: Image Communication*, vol. 27, no. 5, pp. 401 – 411, 2012.

[20] A. Benoit, A. Dobrila, J. Nicod, and L. Philippe, "Scheduling linear chain streaming applications on heterogeneous systems with failures," *Future Generation Computer Systems*, vol. 29, no. 5, pp. 1140–1151, 2013.

[21] A. Benoit, A. Dobrila, J. Nicod, and L. Philippe, "Workload balancing and throughput optimization for heterogeneous systems subject to failures," in *Proceedings of the 17th International Conference on Parallel Processing (EuroPar'11)*, pp. 242–254, 2011.

[22] J. Laprie, "Dependable computing and fault-tolerance: Concepts and terminology," in *'Highlights from Twenty-Five Years', 25th Intern. Symposium on Fault-Tolerant Computing*, 1995.

[23] "International Federation For Information Processing (IFIP), WG 10.4 on Dependable Computing and Fault-Tolerance." (Accessed: March 2012).

[24] M. Al-Kuwaiti, N. Kyriakopoulos, and S. Hussein, "A comparative analysis of network dependability, fault-tolerance, reliability, security, and survivability," *IEEE Communications Surveys and Tutorials*, vol. 11, no. 2, pp. 106–124, 2009.

[25] A. Avizienis, J. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. on Dependable and Secure Computing*, vol. 1, pp. 11–33, 2004.

[26] "IEEE standard glossary of software engineering terminology," *IEEE Standard*, 1990.

[27] L. Lamport, "Proving the correctness of multiprocess programs," *IEEE Transactions on Software Engineering*, vol. 3, no. 2, pp. 125–143, 1977.

[28] P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys*, vol. 35, no. 2, pp. 114–131, 2003.

[29] R. Kazemzadeh and H. Jacobsen, "Introducing publiy: a multi-purpose distributed publish/subscribe system," in *Proceedings of the Posters and Demo Track*, Middleware'12, (New York, NY, USA), pp. 1:1–1:2, ACM, 2012.

[30] H. Jacobsen and V. Muthusamy., *Green IT: Technologies and Applications*, ch. Green Middleware, pp. 341–361. Springer Verlag, 2011.

[31] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, *Distributed Systems: Concepts and Design (5th Edition).* Addison-Wesley Longman, Amsterdam, 2011.

[32] S. Voulgaris, E. Rivière, A. Kermarrec, and M. Van Steen, "Sub-2-sub: Self-organizing content-based publish subscribe for dynamic large scale collaborative networks," in *5th International Workshop on Peer-to-Peer Systems (IPTPS'06)*, 2006.

[33] R. Baldoni, R. Beraldi, V. Quema, L. Querzoni, and S. Tucci-Piergiovanni, "Tera: topic-based event routing for peer-to-peer architectures," in *Proceedings of the 2007 Inaugural International Conference on Distributed Event-based Systems (DEBS'07)*, pp. 2–13, ACM, 2007.

[34] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg, "Spidercast: a scalable interest-aware overlay for topic-based pub/sub communication," in *Proceedings of the 2007 Inaugural International Conference on Distributed Event-based Systems (DEBS'07)*, pp. 14–25, 2007.

[35] The Gryphon Team, "Achieving scalability and throughput in a publish/subscribe system," Tech. Rep. Report RC23103, IBM Research, 2004.

[36] C. Esposito, D. Cotroneo, and A. Gokhale, "Reliable publish/subscribe middleware for time-sensitive internet-scale applications," in *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems (DEBS'09)*, pp. 1–12, ACM, 2009.

[37] N. Carvalho, F. Araujo, and L. Rodrigues, "Scalable qos-based event routing in publish-subscribe systems," in *Proceedings of the Fourth IEEE International Symposium on Network Computing and Applications (NCA'05)*, pp. 101–108, IEEE Computer Society, 2005.

[38] C. Wang, A. Carzaniga, D. Evans, and A. Wolf, "Security issues and requirements for internet-scale publish-subscribe systems," in *Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)*, vol. 9, p. 303, IEEE Computer Society, 2002.

[39] I. Delamer and J. M. Lastra, "Evolutionary multi-objective optimization of qos-aware publish/subscribe middleware in electronics production," *Engineering Applications of Artificial Intelligence*, vol. 19, no. 6, pp. 593 – 607, 2006.

[40] Y. Liu and B. Plale, "Survey of publish subscribe event systems," tech. rep., Indiana University, Bloomington, USA, 2003.

[41] R. Baldoni, L. Querzoni, and A. Virgillito, "Distributed event routing in publish/subscribe communication systems: a survey," tech. rep., University of Rome, 2005.

[42] M. Hosseini, D. T. Ahmed, S. Shirmohammadi, and N. Georganas, "A survey of application-layer multicast protocols," *IEEE Communications Surveys & Tutorials*, vol. 9, no. 3, pp. 58–74, 2007.

[43] H. Li, A. Clement, M. Marchetti, M. Kapritsos, L. Robinson, L. Alvisi, and M. Dahlin, "Flightpath: Obedience vs choice in cooperative services," in *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI'08)*, December 2008.

[44] R. Dash, N. Jennings, and D. Parkes, "Computational-mechanism design: A call to arms," *IEEE Intelligent Systems*, vol. 18, pp. 40–47, November 2003.

[45] L. Chen and J. Leneutre, "Selfishness, not always a nightmare: Modeling selfish mac behaviors in wireless mobile ad hoc networks," in *27th IEEE International Conference on Distributed Computing Systems (ICDCS'07)*, 2007.

[46] T. Mayer, L. Brunie, D. Coquil, and H. Kosch, "Evaluating the robustness of publish/-subscribe systems," in *Proceedings of the Sixth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC'11)*, 2011.

[47] T. Mayer, L. Brunie, D. Coquil, and H. Kosch, "On reliability in publish/subscribe systems: a survey," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 27, no. 5, pp. 369–386, 2012.

[48] T. Mayer, D. Coquil, C. Schörnich, and H. Kosch, "Rcourse: a robustness benchmarking suite for publish/subscribe overlay simulations with peersim," in *Proceedings of the 1st EDCC Workshop on P2P and Dependability (P2PDep'12)*, 2012.

[49] G. Gianini, E. Damiani, T. Mayer, D. Coquil, H. Kosch, and L. Brunie, "Many-player inspection games in networked environments," in *Proceedings of the 7th 2013 International Conference on Digital Ecosystems and Technologies (DEST'13)*, 2013.

[50] G. Gianini, T. Mayer, D. Coquil, H. Kosch, and L. Brunie, "Inspection games for selfish network environments," No. MIP-1203, 2012.

[51] P. Freeman, "Software reliability and design: A survey," in *Proceedings of the 13th Design Automation Conference (DAC'76), San Francisco, California, USA*, pp. 484–494, 1976.

[52] B. Bhargava, "Guest editorial: Reliability issues in distributed systems," *IEEE Transactions of Software Engineering*, vol. 8, no. 3, pp. 165–167, 1982.

[53] G. Martella and B. Pernici, "A quantitative approach to evaluating reliability in distributed data bases," in *Proceedings of the 2nd International Conference on Data and Knowledge Bases (JCDKB'82)*, pp. 29–40, 1982.

[54] C. Raghavendra, M. Gerla, and A. Avizienis, "Reliability optimization in the design of distributed systems," in *Proceedings of the 3rd IEEE International Conference on Distributed Computing Systems (ICDCS'82)*, pp. 388–393, 1982.

[55] D. Eager and K. Sevcik, "Achieving robustness in distributed database systems," *ACM Transactions on Database Systems (TODS'83)*, vol. 8, no. 3, pp. 354–381, 1983.

[56] J. Laprie, "Trustable evaluation of computer systems dependability," in *Proceedings of the International Workshop of Computer Performance and Reliability, Pisa, Italy*, pp. 341–360, 1983.

[57] A. Somani and N. Vaidya, "Understanding fault tolerance and reliability," *Computer*, vol. 30, no. 4, pp. 45–50, 1997.

[58] D. Medhi, *Network Reliability and Fault-Tolerance.* John Wiley & Sons, Inc., 2001.

[59] A. Carzaniga, A. Gorla, and M. Pezzè, "Handling software faults with redundancy," in *Architecting Dependable Systems VI* (R. de Lemos, J.-C. Fabre, C. Gacek, F. Gadducci, and M. ter Beek, eds.), vol. 5835 of *Lecture Notes in Computer Science*, pp. 148–171, Springer Berlin / Heidelberg, 2009.

[60] R. Guerraoui, S. Handurukande, K. Huguenin, A. Kermarrec, F. Le Fessant, and E. Riviere, "Gosskip, an efficient, fault-tolerant and self organizing overlay using gossip-based construction and skip-lists principles," *Proceedings of the sixth IEEE International Conference on Peer-to-Peer Computing (P2P'06)*, vol. 0, pp. 12–22, 2006.

[61] M. Abd-El-Barr, *Design And Analysis of Reliable And Fault-tolerant Computer Systems.* World Scientific Pub Co Inc., 2006.

[62] K. Zerfiridis and H. Karatza, "Fault tolerant peer-to-peer dissemination network," in *Proceedings of the 9th International Euro-Par Conference (Euro-Par'03)*, pp. 1257–1264, 2003.

[63] B. Porter, F. Taiani, and G. Coulson, "Generalised repair for overlay networks," *25th IEEE Symposium on Reliable Distributed Systems (SRDS'06)*, vol. 0, pp. 132–142, 2006.

[64] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Transactions on Computer Systems (TOCS'02)*, vol. 20, no. 4, pp. 398–461, 2002.

[65] V. Cristea, C. Dobre, F. Pop, C. Stratan, A. Costan, and C. Leordeanu, "Models and techniques for ensuring reliability, safety, availability and security of large scale distributed systems," *CoRR*, vol. abs/1106.5576, 2011.

[66] G. Jesi, A. Montresor, and M. van Steen, "Secure peer sampling," *Computer Networks*, vol. 54, pp. 2086–2098, August 2010.

[67] O. Hasan, L. Brunie, and E. Bertino, "Preserving privacy of feedback providers in decentralized reputation systems," *Computers & Security*, vol. 31, no. 7, pp. 816 – 826, 2012.

[68] B. Parno, "Trust extension for commodity computers," *Communications of the ACM*, vol. 55, pp. 76–85, June 2012.

[69] A. Haeberlen, P. Kouznetsov, and P. Druschel, "Peerreview: Practical accountability for distributed systems," in *Proceedings of twenty-first ACM SIGOPS symposium on Operating Systems Principles (SOSP'07)*, pp. 175–188, ACM, 2007.

[70] S. Mahambre, M. Kumar S. D., and U. Bellur, "A taxonomy of qos-aware, adaptive event-dissemination middleware," *IEEE Internet Computing*, vol. 11, pp. 35–44, 2007.

[71] S. Behnel, L. Fiege, and G. Mühl, "On quality-of-service and publish-subscribe," *International Conference on Distributed Computing Systems Workshops*, p. 20, 2006.

[72] R. Mirandola, P. Potena, and P. Scandurra, "A reliability prediction method for abstract state machines," in *Proceedings of the Third International Conference on Abstract State Machines (ABZ'12), Pisa, Italy*, pp. 336–340, 2012.

[73] G. Kim, J. Park, and J. Baik, "An effective approach to identifying optimal software reliability allocation with consideration of multiple constraints," in *Proceedings of the 2012 IEEE/ACIS 11th International Conference on Computer and Information Science, Shanghai, China (ICIS'12)*, pp. 541–546, IEEE Computer Society, 2012.

[74] P. Gupta, R. Gupta, S. Ong, and H. Srivastava, "Reliability and non-reliability studies of poisson variables in series and parallel systems," *Applied Mathematics and Computation*, vol. 218, no. 9, pp. 5112–5120, 2012.

[75] S. Huang, H. Li, J. Wu, and V. Leung, "A reliable group-simulcasting scheme for distributed antenna systems," *IEEE Transactions on Wireless Communications*, vol. 11, no. 1, pp. 236–245, 2012.

[76] J. Hong, T. Kuo, and J. Heo, "Advanced issues of operating systems for reliable distributed sensor networks: Aim and scope," *International Journal of Distributed Sensor Networks*, vol. 2012, 2012.

[77] H. Li, A. Clement, E. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin, "Bar gossip," in *Proceedings of the 7th symposium on Operating systems design and implementation (OSDI'06)*, pp. 191–204, USENIX Association, 2006.

[78] K. Trivedi and M. Malhotra, "Reliability and performability techniques and tools: A survey," in *7. ITG/GI-Fachtagung zur Messung, Modellierung und Bewertung von Rechen- und Kommunikationssystemen (MMB), Aachen, Germany*, pp. 27–48, 1993.

[79] F. Boesch, "A survey and introduction to network reliability theory," in *Proceedings of the IEEE International Conference on Communications (ICC'88)*, vol. 2, pp. 678–682, June 1988.

[80] A. Johnson and M. Malek, "Survey of software tools for evaluating reliability, availability, and serviceability," *ACM Comput. Surv.*, vol. 20, no. 4, pp. 227–269, 1988.

[81] M. A. Isa, M. Z. M. Zaki, and D. N. A. Jawawi, "A survey of design model for quality analysis: From a performance and reliability perspective," *Computer and Information Science*, vol. 6, no. 2, pp. 55–70, 2013.

[82] K. Tyagi and A. Sharma, "Reliability of component based systems: a critical survey," *ACM SIGSOFT Software Engineering Notes*, vol. 36, no. 6, pp. 1–6, 2011.

[83] G. Pai, "A survey of software reliability models," *CoRR*, vol. abs/1304.4539, 2013.

[84] M. Golash, "Reliability in ethernet networks: A survey of various approaches," *Bell Labs Technical Journal*, vol. 11, no. 3, pp. 161–171, 2006.

[85] A. Immonen and E. Niemelä, "Survey of reliability and availability prediction methods from the viewpoint of software architecture," *Software and System Modeling*, vol. 7, no. 1, pp. 49–65, 2008.

[86] A. Willig and H. Karl, "Data transport reliability in wireless sensor networks. a survey of issues and solutions," *Praxis der Informationsverarbeitung und Kommunikation*, vol. 28, no. 2, pp. 86–92, 2005.

[87] E. Fernández, O. Ajaj, I. Buckley, N. Delessy-Gassant, K. Hashizume, and M. M. Larrondo-Petrie, "A survey of patterns for web services security and reliability standards," *Future Internet*, vol. 4, no. 2, pp. 430–450, 2012.

[88] A. Mukati, "A survey of memory error correcting techniques for improved reliability," *Journal of Network and Computer Applications*, vol. 34, no. 2, pp. 517–522, 2011.

[89] M. Shooman, *Reliability of Computer Systems and Networks: Fault Tolerance, Analysis, and Design.* John Wiley & Sons, 2002.

[90] X. Shen, H. Yu, and J. Buford, eds., *Handbook of Peer-to-Peer Networking.* Springer, Berlin/Heidelberg, Germany, 2010.

[91] A. Carzaniga, D. Rosenblum, and A. Wolf, "Design and evaluation of a wide-area event notification service," *ACM Transactions on Computer Systems*, vol. 19, no. 3, pp. 332–383, 2001.

[92] R. Samanta, J. Deshmukh, and S. Chaudhuri, "Robustness analysis of networked systems," in *Proceedings of the 14th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'13)*, pp. 229–247, 2013.

[93] P. Costa, J. G. Silva, and H. Madeira, "Dependability benchmarking using software faults: How to create practical and representative faultloads," in *Proceedings of the 15th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'09)*, pp. 289–294, 2009.

[94] J. Durães and H. Madeira, "Generic faultloads based on software faults for dependability benchmarking," in *DSN*, pp. 285–294, 2004.

[95] P. Costa, M. Vieira, H. Madeira, and J. G. Silva, "Plug and play fault injector for dependability benchmarking," in *Proceedings of the First Latin-American Symposium of Dependable Computing (LADC'03)*, pp. 8–22, 2003.

[96] M. Vieira, N. Laranjeiro, and H. Madeira, "Benchmarking the robustness of web services," in *Proceedings of the 13th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'07)*, pp. 322–329, 2007.

[97] L. Chen, C. Chiou, and Y. Chen, "An evaluation of routing reliability in non-collaborative opportunistic networks," in *Proceedings of the IEEE 23rd International Conference on Advanced Information Networking and Applications (AINA'09)*, pp. 50–57, 2009.

[98] H. Kim and R. Anderson, "An experimental evaluation of robustness of networks," *IEEE Systems Journal*, vol. 7, no. 2, pp. 179–188, 2013.

[99] K. Sachs, S. Kounev, J. Bacon, and A. P. Buchmann, "Performance evaluation of message-oriented middleware using the specjms2007 benchmark," *Performance Evaluation*, vol. 66, no. 8, pp. 410–434, 2009.

[100] R. Myerson, *Game Theory: Analysis of Conflict.* Harvard University Press, 1997.

[101] J. von Neumann and O. Morgenstern, *Theory of Games and Economic Behavior.* Princeton University Press, 1944.

[102] M. Osborne, *An Introduction to Game Theory.* Oxford University Press, 2009.

[103] N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, *Algorithmic Game Theory.* Cambridge University Press, 2007.

[104] R. Aumann and S. Hart, eds., *Handbook of Game Theory with Economic Applications*, vol. 1–3 of *Handbooks in Oper. Research and Management Science.* Elsevier Science & Techn., 1992.

[105] S. Shenker, "Efficient network allocations with selfish users," in *Proceedings of the 14th IFIP WG 7.3 International Symposium on Computer Performance Modelling, Measurement and Evaluation (Performance'90), Edinburgh, Scotland*, pp. 279–285, 1990.

[106] M. Mavronicolas and P. Spirakis, "The price of selfish routing," in *Proceedings on 33rd Annual ACM Symposium on Theory of Computing (STOC'01), Crete, Greece*, pp. 510–519, 2001.

[107] T. Roughgarden, "Designing networks for selfish users is hard," in *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS'01), Las Vegas, USA*, pp. 472–481, 2001.

[108] R. Feldmann, M. Gairing, T. Lücking, B. Monien, and M. Rode, "Selfish routing in non-cooperative networks: A survey," in *Proceedings of the 28th International Symposium, Mathematical Foundations of Computer Science (MFCS'03)*, pp. 21–45, 2003.

[109] Z. Han, Z. Ji, and K. Liu, "A cartel maintenance framework to enforce cooperation in wireless networks with selfish users," *IEEE Transactions on Wireless Communications*, vol. 7, no. 5-2, pp. 1889–1899, 2008.

[110] Y. Wang and M. Singhal, "A light-weight scalable truthful routing protocol in manets with selfish nodes," *International Journal of Ad Hoc and Ubiquitous Computing (IJAHUC)*, vol. 4, no. 3/4, pp. 210–222, 2009.

[111] M. Felegyhazi, J. Hubaux, and L. Buttyan, "Nash equilibria of packet forwarding strategies in wireless ad hoc networks," *EEE Transactions on Mobile Computing*, vol. 5, pp. 463 – 476, may 2006.

[112] M. Felegyhazi, L. Buttyan, and J. Hubaux, "Equilibrium analysis of packet forwarding strategies in wireless ad hoc networks – the static case," in *Proceedings of the IFIP-TC6 8th International Conference of Personal Wireless Communications (PWC'03), Venice, Italy*, pp. 23–25, 2003.

[113] A. Azad, E. Altman, and R. El-Azouzi, "Routing Games : From Egoism to Altruism," Rapport de recherche RR-7059, INRIA, 2009.

[114] S. Bohacek, J. Hespanha, J. Lee, C. Lim, and K. Obraczka, "Game theoretic stochastic routing for fault tolerance and security in computer networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, pp. 1227 –1240, september 2007.

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

[116] J. Choi, K. Shim, S. Lee, and K. Wu, "Handling selfishness in replica allocation over a mobile ad hoc network," *IEEE Transactions on Mobile Computing*, vol. 11, no. 2, pp. 278–291, 2012.

[117] H. Inaltekin, M. Chiang, H. Poor, and S. Wicker, "Selfish random access over wireless channels with multipacket reception," *IEEE Journal on Selected Areas in Communications*, vol. 30, no. 1, pp. 138–152, 2012.

[118] K. Akkarajitsakul, E. Hossain, D. Niyato, and D. Kim, "Game theoretic approaches for multiple access in wireless networks: A survey," *IEEE Communications Surveys Tutorials*, vol. 13, pp. 372–395, quarter 2011.

[119] Z. Han and K. Liu, *Resource Allocation for Wireless Networks: Basics, Techniques, and Applications.* Cambridge University Press, 2008.

[120] Cem U. Saraydar, N. Mandayam, and D. Goodman, "Pricing and power control in a multicell wireless data network," *Selected Areas in Communications, IEEE Journal on*, vol. 19, pp. 1883–1892, oct 2001.

[121] S. Ulukus and R. Yates, "Stochastic power control for cellular radio systems," *IEEE Transactions on Communications*, vol. 46, no. 6, pp. 784–798, 1998.

[122] I. Keidar, R. Melamed, and A. Orda, "Equicast: scalable multicast with selfish users," in *Proceedings of the 25th annual ACM Symposium on Principles of Distributed Computing (PODC'06)*, (New York, NY, USA), pp. 63–71, ACM, 2006.

[123] T. Ngan, D. Wallach, and P. Druschel, "Incentives-compatible peer-to-peer multicast." 2nd Workshop on the Economics of Peer-to-Peer Systems, 2004.

[124] S. Penmatsa and A. Chronopoulos, "Game-theoretic static load balancing for distributed systems," *Journal Parallel Distributed Computing*, vol. 71, no. 4, pp. 537–555, 2011.

[125] R. Pal and P. Hui, "Economic models for cloud service markets," in *Distributed Computing and Networking* (L. Bononi, A. Datta, S. Devismes, and A. Misra, eds.), vol. 7129 of *Lecture Notes in Computer Science*, pp. 382–396, Springer Berlin / Heidelberg, 2012.

[126] D. Ardagna, B. Panicucci, and M. Passacantando, "Generalized nash equilibria for the service provisioning problem in cloud systems," *IEEE Transactions on Services Computing*, no. 99, 2012.

[127] D. Argagna, B. Panicucci, and M. Passacantando, "A game theoretic formulation of the service provisioning problem in cloud systems," in *Proceedings of the 20th International Conference on World wide web (WWW'11)*, pp. 177–186, ACM, 2011.

[128] C. kuang Lin and Ø. Kure, "A game theoretic framework for decentralized and distributed energy utilization in wireless sensor networks," in *Proceedings of the ISCA First International Conference on Sensor Networks and Applications (SN'09)*, pp. 136–141, 2009.

[129] R. Machado and S. Tekinay, "A survey of game-theoretic approaches in wireless sensor networks," *Computer Networks*, vol. 52, no. 16, pp. 3047 – 3061, 2008.

[130] K. Apt and E. Grädel, *Lectures in Game Theory for Computer Scientists.* Cambridge University Press, 2011.

[131] Z. Han, D. Niyato, W. Saad, T. Baar, and A. Hjørungnes, *Game Theory in Wireless and Communication Networks: Theory, Models, and Applications.* Cambridge University Press, 2011.

[132] M. Debbah and H. Tembine, *Game Theory for Wireless Networks: From Fundamentals to Practice.* Academic Press, 2011.

[133] M. Saleh and M. Debbabi, "A game-theoretic framework for specification and verification of cryptographic protocols," *Formal Aspects of Computing*, vol. 22, no. 5, pp. 585–609, 2010.

[134] M. Saleh and M. Debbabi, "Verifying security properties of cryptoprotocols: A novel approach," in *Proceedings of the Fifth IEEE International Conference on Software Engineering and Formal Methods (SEFM'07)*, pp. 349–360, IEEE Computer Society, 2007.

[135] K. Lorenz, "Basic objectives of dialogue logic in historical perspective," *Synthese*, vol. 127, no. 1-2, pp. 255–263, 2001.

[136] T. Moscibroda, S. Schmid, and R. Wattenhofer, "The price of malice: A game-theoretic framework for malicious behavior in distributed systems," *Internet Mathematics*, vol. 6, no. 2, pp. 125–155, 2009.

[137] J. Halpern and R. Pass, "Game theory with costly computation: Formulation and application to protocol security," in *Proceedings of Innovations in Computer Scienc (ICS'10)*, pp. 120–142, Tsinghua University Press, 2010.

[138] S.-K. Ng and W. Seah, "Game-theoretic approach for improving cooperation in wireless multihop networks," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 40, no. 3, pp. 559–574, 2010.

[139] G. He, L. Cottatellucci, and M. Debbah, "The waterfilling game-theoretical framework for distributed wireless network information flow," *EURASIP Journal of Wireless Communication and Networking*, vol. 2010, 2010.

[140] G. Kol and M. Naor, "Cryptography and game theory: Designing protocols for exchanging information," in *Proceedings of the Fifth Theory of Cryptography Conference (TCC'08)*, Lecture Notes in Computer Science, pp. 320–339, Springer, 2008.

[141] E. Palomar, A. Alcaide, A. Ribagorda, and Y. Zhang, "The peer's dilemma: A general framework to examine cooperation in pure peer-to-peer systems," *Computer Networks*, vol. 56, no. 17, pp. 3756–3766, 2012.

[142] Z. Ma and A. Krings, "Dynamic hybrid fault modeling and extended evolutionary game theory for reliability, survivability and fault tolerance analyses," *IEEE Transactions on Reliability*, vol. 60, no. 1, pp. 180–196, 2011.

[143] Z. Chen, Y. Qiu, J. Liu, and L. Xu, "Incentive mechanism for selfish nodes in wireless sensor networks based on evolutionary game," *Computers & Mathematics with Applications*, pp. 3378–3388, 2011.

[144] Z. Rui, T. Fu, D. Lai, and Y. Jiang, "Cooperation among malicious agents: a general quantitative congestion game framework," in *Proceedings of the 2012 International Conference on Autonomous Agents and Multiagent Systems (AAMAS'12)*, pp. 1331–1332, 2012.

[145] M. Bell, "The use of game theory to measure the vulnerability of stochastic networks," *IEEE Transactions on Reliability*, vol. 52, no. 1, pp. 63–68, 2003.

[146] N. Garg and D. Grosu, "Faithful distributed shapley mechanisms for sharing the cost of multicast transmissions," in *Proceedings of the 12th IEEE Symposium on Computers and Communications (ISCC'07)*, pp. 741–747, IEEE Computer Society, 2007.

[147] J. Feigenbaum, C. Papadimitriou, and S. Shenker, "Sharing the cost of multicast transmissions," *Journal of Computer and System Sciences*, vol. 63, no. 1, pp. 21–41, 2001.

[148] N. Fotiou, G. Marias, and G. Polyzos, "Towards a secure rendezvous network for future publish/subscribe architectures," in *Proceedings of the Third Future Internet Symposium 2010 (FIS'10)*, pp. 49–56, 2010.

[149] O. Gonzalez, G. Ansa, M. Howarth, and G. Pavlou, "Detection and accusation of packet forwarding misbehavior in mobile ad-hoc networks," *Journal of Internet Engineering*, vol. 2, no. 1, pp. 181–192, 2008.

[150] S. Radosavac, A. Cárdenas, J. Baras, and G. Moustakides, "Detecting ieee 802.11 mac layer misbehavior in ad hoc networks: Robust strategies against individual and colluding attackers," *Journal of Computer Security*, vol. 15, no. 1, 2007.

[151] W. Shi, M. Yao, and J. Corriveau, "Resilient secure localization and detection of colluding attackers in wsns," in *Proceedings of the 11th 2012 International Conference on Ad-hoc, Mobile, and Wireless Networks (ADHOC-NOW'12)*, pp. 181–192, 2012.

[152] P. Gera, K. Garg, and M. Misra, "Detection and mitigation of attacks by colluding misbehaving nodes in manet," in *Recent Trends in Network Security and Applications* (N. Meghanathan, S. Boumerdassi, N. Chaki, and D. Nagamalai, eds.), vol. 89 of *Communications in Computer and Information Science*, pp. 181–190, Springer, 2010.

[153] R. Raghuvanshi, R. Kaushik, and J. Singhai, "A review of misbehaviour detection and avoidance scheme in ad hoc network," in *2011 International Conference on Emerging Trends in Networks and Computer Communications (ETNCC)*, pp. 301–306, 2011.

[154] S. Weeks, "Understanding trust management systems," in *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, pp. 94–105, 2001.

[155] M. Serva, M. Fuller, and R. Mayer, "Trust in systems development: a model of management and developer interaction research in progress," in *Proceedings of the ACM SIGCPR Conference on Computer Personnel Research (SIGCPR)*, pp. 188–191, 2000.

[156] P. Resnick, K. Kuwabara, R. Zeckhauser, and E. Friedman, "Reputation systems," *Communications of the ACM*, vol. 43, no. 12, pp. 45–48, 2000.

[157] P. Shah and J.-F. Pâris, "Incorporating trust in the bittorrent protocol," in *Proceedings of the 2007 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'07)*, 2007.

[158] W. Li, A. Joshi, and T. Finin, “Cast: Context-aware security and trust framework for mobile ad-hoc networks using policies,” *Distributed and Parallel Databases*, vol. 31, no. 2, pp. 353–376, 2013.

[159] S. Buchegger and J.-Y. Le Boudec, “Performance analysis of the confidant protocol,” in *Proceedings of the 3rd ACM Interational Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'02)*, pp. 226–236, 2002.

[160] P. Michiardi and R. Molva, “Core: a collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks,” in *Communications and Multimedia Security*, IFIP Conference Proceedings, pp. 107–121, 2002.

[161] B. Sarjaz and M. Abbaspour, “Securing bittorrent using a new reputation-based trust management system,” *Peer-to-Peer Networking and Applications*, vol. 6, no. 1, pp. 86–100, 2013.

[162] A. Vieira, R. de Almeida, J. de Almeida, and S. Campos, “Simplyrep: A simple and effective reputation system to fight pollution in {P2P} live streaming,” *Computer Networks*, vol. 57, no. 4, pp. 1019–1036, 2013.

[163] K. Walsh and E. Sirer, “Experience with an object reputation system for peer-to-peer filesharing (awarded best paper),” in *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation (NSDI'06)*, 2006.

[164] L. Xiong and L. Liu, “A reputation-based trust model for peer-to-peer ecommerce communities,” in *Proceedings 4th ACM Conference on Electronic Commerce (EC'03)*, pp. 228–229, 2003.

[165] X. Cheng, L. Macaulay, and A. Zarifis, “Modeling individual trust development in computer mediated collaboration: A comparison of approaches,” *Computers in Human Behavior*, vol. 29, no. 4, pp. 1733–1741, 2013.

[166] H. Yu, Z. Shen, C. Miao, C. Leung, and D. Niyato, “A survey of trust and reputation management systems in wireless communications,” *Proceedings of the IEEE*, vol. 98, no. 10, pp. 1755–1772, 2010.

[167] M. Louta, S. Kraounakis, and A. Michalas, “A survey on reputation-based cooperation enforcement schemes in wireless ad hoc networks,” in *Proceedings of the 2010 International Conference on Wireless Information Networks and Systems (WINSYS'10)*, pp. 90–93, 2010.

[168] M. Azer, S. El-Kassas, A. Hassan, and M. El-Soudani, “A survey on trust and reputation schemes in ad hoc networks,” in *Proceedings of the The Third International Conference on Availability, Reliability and Security (ARES'08)*, pp. 881–886, 2008.

[169] S. Ruohomaa, L. Kutvonen, and E. Koutrouli, “Reputation management survey,” in *Proceedings of the The Second International Conference on Availability, Reliability and Security (ARES'07)*, pp. 103–111, IEEE Computer Society, 2007.

[170] E. Byun, S. Choi, C. Hwang, and S. Lee, "Survey on reputation management systems in p2p network," in *Proceedings of the 2007 International Conference on Security and Management*, pp. 358–366, CSREA Press, 2007.

[171] H. Abbes, C. Cérin, and B. Oueghlani, "A decentralized model for controlling selfish use for desktop grid systems," in *Proceedings of the 13th IEEE International Conference on High Performance Computing & Communication (HPCC'11)*, pp. 814–821, 2011.

[172] A. Mondal, S. Madria, and M. Kitsuregawa, "An economic incentive model for encouraging peer collaboration in mobile-p2p networks with support for constraint queries," *Peer-to-Peer Networking and Applications*, vol. 2, no. 3, pp. 230–251, 2009.

[173] A. Mondal, S. Madria, and M. Kitsuregawa, "Abide: A bid-based economic incentive model for enticing non-cooperative peers in mobile-p2p networks," in *Advances in Databases: Concepts, Systems and Applications*, vol. 4443 of *Lecture Notes in Computer Science*, pp. 703–714, Springer Berlin Heidelberg, 2007.

[174] Z. Liang and W. Shi, "Enforcing cooperative resource sharing in untrusted p2p computing environments," *Mobile Networks and Applications*, vol. 10, no. 6, pp. 971–983, 2005.

[175] M. Dresher, "A sampling inspection problem in arms control agreements: A game theoretical analysis." Memorandum No. RM-2972-ARPA, The RAND Corporation, 1962.

[176] R. Avenhaus, B. von Stengel, and S. Zamir, "Inspection games," in *Handbook of Game Theory with Economic Applications* (R. Aumann and S. Hart, eds.), vol. 3 of *Handbook of Game Theory with Economic Applications*, ch. 51, pp. 1947–1987, Elsevier, 00 2002.

[177] T. Ferguson and C. Melolidakis, "On the inspection game," *Naval Research Logistics (NRL)*, vol. 45, no. 3, pp. 327–334, 1998.

[178] A. Antoniades, H. Kim, and S. Sastry, "Pursuit-evasion strategies for teams of multiple agents with incomplete information," in *Proceedings of the 42nd IEEE Conference on Decision and Control*, vol. 1, pp. 756 – 761 Vol.1, december 2003.

[179] V. Baston and F. Bostock, "A generalized inspection game," *Naval Research Logistics (NRL)*, vol. 38, no. 2, pp. 171–182, 1991.

[180] S. Alpern and S. Gal, *The theory of search games and rendezvous.* Springer, 2002.

[181] T. Chung, G. Hollinger, and V. Isler, "Search and pursuit-evasion in mobile robotics," *Autonomous Robots*, pp. 1–18, 2011.

[182] R. Avenhaus and M. Candy, "Playing for time: A sequential inspection game," *European Journal of Operational Research*, vol. 167, no. 2, pp. 475–492, 2005.

[183] T. Ferguson and C. Melolidakis, "Games with finite resources," *International Journal of Game Theory*, vol. 29, no. 2, pp. 289–303, 2000.

[184] N. Zoroa, P. Zoroa, and M. Fernández-Sáez, "Weighted search games," *European Journal of Operational Research*, vol. 195, pp. 394–411, June 2009.

[185] R. Hohzaki, “An inspection game with multiple inspectees,” *European Journal of Operational Research*, vol. 178, no. 3, pp. 894 – 906, 2007.

[186] R. Hohzaki, D. Kudoh, and T. Komiya, “An inspection game: Taking account of fulfillment probabilities of players’ aims,” *Naval Research Logistics (NRL)*, vol. 53, no. 8, pp. 761–771, 2006.

[187] R. Avenhaus and D. Kilgour, “Efficient distributions of arms-control inspection effort,” *Naval Research Logistics (NRL)*, vol. 51, no. 1, pp. 1–27, 2004.

[188] N. Alonso and Procopio Zoroa Terol, “Some games of search on a lattice,” *Naval Research Logistics (NRL)*, vol. 40, no. 4, pp. 525–541, 1993.

[189] B. von Stengel, “Recursive inspection games,” iasfor-bericht s-9106, Armed Forces University Munich, 1991.

[190] A. Goldman and M. Pearl, “The dependence of inspection-system performance on levels of penalties and inspection resources,” *Journal of Research of the National Bureau of Standards*, vol. 80B (Issue 2), pp. 189–236, 1976.

[191] R. Avenhaus, “Applications of inspection games,” *Mathematical Modelling and Analysis*, vol. 9, no. 3, pp. 179–192, 2010.

[192] S. Tambe, “State-of-the-art in publish/subscribe middleware for supporting mobility.” Department of Electrical Engin. and Comp. Science, Vanderbilt University, Nashville, USA, 2007.

[193] G. Mühl, L. Fiege, and P. Pietzuch, *Distributed Event-Based Systems.* Springer, 2006.

[194] P. Pietzuch and J. Bacon, “Hermes: A distributed event-based middleware architecture,” in *Proceedings of the 22nd Intern. Conference on Distributed Computing Systems*, pp. 611–618, 2002.

[195] R. Chand and P. Felber, “Xnet: A reliable content-based publish/subscribe system,” in *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems (SRDS’04)*, pp. 264–273, IEEE Computer Society, 2004.

[196] E. Hanson, M. Chaabouni, C. Kim, and Y. Wang, “A predicate matching algorithm for database rule systems,” in *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, SIGMOD ’90, (New York, NY, USA), pp. 271–280, ACM, 1990.

[197] W. Rjaibi, K. Dittrich, and D. Jaepel, “Event matching in symmetric subscription systems,” in *Proceedings of the 2002 conference of the Centre for Advanced Studies on Collaborative research (CASCON’02)*, IBM Press, 2002.

[198] A. Campailla, S. Chaki, E. Clarke, S. Jha, and H. Veith, “Efficient filtering in publish-subscribe systems using binary decision diagrams,” in *Proceedings of the 23rd International Conference on Software Engineering (ICSE’01)*, pp. 443–452, IEEE Computer Society, 2001.

[199] S. Pallickara and G. Fox, "Naradabrokering: a distributed middleware framework and architecture for enabling durable peer-to-peer grids," in *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware (Middleware'03)*, pp. 41–61, Springer, 2003.

[200] S. Kale, E. Hazan, F. Cao, and J. Singh, "Analysis and algorithms for content-based event matching," in *Proceedings of the Fourth International Workshop on Distributed Event-Based Systems (DEBS) (ICDCSW'05) - Volume 04*, pp. 363–369, IEEE Computer Society, 2005.

[201] P. Eugster, R. Guerraoui, A. Kermarrec, and L. Massoulieacute, "Epidemic information dissemination in distributed systems," *Computer*, vol. 37, no. 5, pp. 60–67, 2004.

[202] A. Rowstron and P. 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'01)*, pp. 329–350, Springer-Verlag, 2001.

[203] S. Zhuang, B. Zhao, A. Joseph, R. Katz, and J. Kubiatowicz, "Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination," in *Proceedings of the 11th international workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'01)*, pp. 11–20, ACM, 2001.

[204] G. Cugola and G. Picco, "Reds: a reconfigurable dispatching system," in *Proc. of the 6th International Workshop on Software Engineering and Middleware (SEM'06)*, pp. 9–16, ACM, 2006.

[205] E. Anceaume, M. Gradinariu, A. Datta, G. Simon, and A. Virgillito, "A semantic overlay for self-* peer-to-peer publish/subscribe," *26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)*, p. 22, 2006.

[206] R. Baldoni, R. Beraldi, L. Querzoni, and A. Virgillito, "Efficient publish/subscribe through a self-organizing broker overlay and its application to siena," *The Computer Journal*, vol. 50, pp. 444–459, 2007.

[207] Z. Jerzak, R. Fach, and C. Fetzer, "Fail-aware publish/subscribe," in *Sixth IEEE International Symposium on Network Computing and Applications (NCA'07)*, pp. 113 –125, 2007.

[208] J. Ahullo, P. G. López, and A. Gómez Skarmeta, "CAPS: Content-bAsed Publish/Subscribe services for peer-to-peer systems," in *Proceedings of 2nd International Conference on Distributed Event-Based Systems (DEBS'08). Short Paper*, July 2008.

[209] H. Jafarpour, S. Mehrotra, and N. Venkatasubramanian, "A fast and robust content-based publish/subscribe architecture," *IEEE International Symposium on Network Computing and Applications*, pp. 52–59, 2008.

[210] P. Boonma and J. Suzuki, "Self-configurable publish/subscribe middleware for wireless sensor networks," in *Proceedings of the 6th IEEE Conference on Consumer Communications and Networking Conference (CCNC'09)*, pp. 1376–1383, IEEE Press, 2009.

[211] P. Jokela, A. Zahemszky, C. Esteve Rothenberg, S. Arianfar, and P. Nikander, "Lipsin: Line speed publish/subscribe inter-networking," in *ACM SIGCOMM 2009 Conference on Data Communication*, pp. 195–206, ACM, 2009.

[212] R. Kazemzadeh and H. Jacobsen, "Reliable and highly available distributed publish/subscribe service," in *Proceedings of the 2009 28th IEEE International Symposium on Reliable Distributed Systems (SRDS'09)*, pp. 41–50, IEEE Computer Society, 2009.

[213] M. Deshpande, B. Xing, I. Lazardis, Bijit, H. N. Venkatasubramanian, and S. Mehrotra, "Crew: A gossip-based flash-dissemination system," in *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)*, p. 45, IEEE Computer Society, 2006.

[214] S. Voulgaris and M. van Steen, "Hybrid dissemination: adding determinism to probabilistic multicasting in large-scale p2p systems," in *Proceedings of the ACM/IFIP/USENIX 2007 International Conference on Middleware (Middleware'07)*, pp. 389–409, Springer, 2007.

[215] A. Schröter, D. Graff, G. Mühl, J. Richling, and H. Parzyjegla, "Self-optimizing hybrid routing in publish/subscribe systems," in *Proceedings of the 20th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM'09)*, pp. 111–122, 2009.

[216] P. Costa, M. Migliavacca, G. Picco, and G. Cugola, "Epidemic algorithms for reliable content-based publish-subscribe: An evaluation," in *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, pp. 552–561, IEEE Computer Society, 2004.

[217] M. Jelasity, R. Guerraoui, A. Kermarrec, and M. van Steen, "The peer sampling service: Experimental evaluation of unstructured gossip-based implementations," in *Proceedings of the 5th International Conference on Middleware (Middleware'04)*, pp. 79–98, Springer, 2004.

[218] Márk, S. Voulgaris, R. Guerraoui, A. Kermarrec, and M. van Steen, "Gossip-based peer sampling," *ACM Transactions on Computer Systems*, vol. 25, no. 3, pp. 8–es, 2007.

[219] S. Voulgaris, D. Gavidia, and M. van Steen, "Cyclon: Inexpensive membership management for unstructured p2p overlays," *Journal of Network and Systems Management*, vol. 13, pp. 197–217, June 2005.

[220] A. Ganesh, A. Kermarrec, and L. Massoulié, "Scamp: Peer-to-peer lightweight membership service for large-scale group communication," in *Proceedings of the 3rd Internation Workshop of Networked Group Communication (NGC'01), London, UK,*, pp. 44–55, 2001.

[221] S. Voulgaris, M. van Steen, and K. Iwanicki, "Proactive gossip-based management of semantic overlay networks: Research articles," *Concurrency and Computation: Practice & Experience*, vol. 19, no. 17, pp. 2299–2311, 2007.

[222] J. Leitao, J. Pereira, and L. Rodrigues, “Hyparview: A membership protocol for reliable gossip-based broadcast.,” in *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 419–429, IEEE Computer Society, 2007.

[223] J. ao Carlos Antunes Leitao, J. ao Pedro da Silva Ferreira Moura Marques, J. O. R. N. Pereira, and L. E. T. Rodrigues, “X-bot: A protocol for resilient optimization of unstructured overlays,” in *Proceedings of the 2009 28th IEEE International Symposium on Reliable Distributed Systems*, pp. 236–245, IEEE Computer Society, 2009.

[224] A. Kermarrec, L. Massoulié, and A. Ganesh, “Probabilistic reliable dissemination in large-scale systems,” *IEEE Transactions on Parallel Distributed Systems*, vol. 14, no. 3, pp. 248–258, 2003.

[225] A. Montresor and M. Jelasity, “PeerSim: A scalable P2P simulator,” in *Proc. of the 9th Int. Conference on Peer-to-Peer (P2P’09)*, pp. 99–100, Sept. 2009.

[226] C. Lemmon, S. Lui, and I. Lee, “Review of location-aware routing protocols,” *International Journal on Advances in Information Sciences and Service Sciences*, vol. 2, no. 2, pp. 132–143, 2010.

[227] H. van Tilborg and S. Jajodia, eds., *Encyclopedia of Cryptography and Security, 2nd Ed.* Springer, Berlin, Germany, 2011.

[228] A. Yavuz and P. Ning, “Baf: An efficient publicly verifiable secure audit logging scheme for distributed systems,” in *Proceedings of the 25th Annual Computer Security Applications Conference (ACSAC’09)*, pp. 219–228, 2009.

[229] R. Accorsi, “Safe-keeping digital evidence with secure logging protocols: State of the art and challenges,” in *Proceedings of the 5th International Conference on IT Security Incident Management and IT Forensics (IMF’09)*, pp. 94–110, 2009.

[230] R. Acorsi, “Log data as digital evidence: What secure logging protocols have to offer?,” in *Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC’09)*, pp. 398–403, 2009.

INDEX